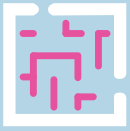


Georgios N. Yannakakis
Julian Togelius

Artificial Intelligence and Games

Second Edition

Springer

Artificial Intelligence and Games

Georgios N. Yannakakis · Julian Togelius

Artificial Intelligence and Games

Second Edition

Springer

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta

Julian Togelius
Department of Computer Science
and Engineering
New York University
Brooklyn, NY, USA

ISBN 978-3-031-83346-5 ISBN 978-3-031-83347-2 (eBook)
https://doi.org/10.1007/978-3-031-83347-2

1st edition: © Springer International Publishing AG, part of Springer Nature 2018
2nd edition: © The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Cover image by Rebecca Portelli and Daniel Mercieca

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

To our families

Foreword to the Second Edition

It's my pleasure to introduce the second edition of the Artificial Intelligence and Games book, a publication that seeks to consolidate the variety of ways artificial intelligence (AI) can be deployed in or for games. The intersection of AI and games is a rather loaded one. The field of AI has long held games in high regard, often seen as a mechanism to highlight the capabilities of intelligent algorithms. From the early days of chess and checkers in the 1950s and 1960s through to more contemporary works in Go, *StarCraft*, and *Dota 2* that you've heard about in the media. The desire to build intelligent systems that "play" has also sat alongside the growing suite of AI tools and systems deployed within commercial game development.

As such, when one says they have an interest in "AI for games", this can mean many things. You could be building opponents for board games or strategy video games, non-player characters in video games that provide a challenge or support the player in their experience. It could be dynamic difficulty systems that observe players and change the game to suit, or procedural generation algorithms designed to create everything from characters to quest lines, levels, weapons, or even entire games. But equally it can also mean building systems and tools that the player does not observe. Data analytics pipelines for modeling and quantifying player behavior are used in many popular titles to observe behavioral trends. Ranging from analyzing how you win a clutch moment, to understanding whether you figured out how to use a specific menu or interface, or detecting whether other players around you are cheating or toxic. Meanwhile, quality assurance frameworks powered by AI allow for games to be tested during—and after—release to not just reduce bugs in the product, but help developers keep the game from breaking as they're building it—a common occurrence, I can assure you.

If you're joining this space for the first time, this can all be rather intimidating. The landscape of AI algorithms, methodologies, and approaches, specifically for games, has exploded in the past 20 years. Not only as academic research began to see the value of games as a mechanism to deploy AI in a variety of scientific contexts but also as the games industry has seen the proliferation of these technologies, and recognized the opportunities that exist to integrate it into their production pipelines. Trying to get a grasp on this technological wave and communicate it to a broader audience has

long been my mission, largely via my work on the *AI and Games* YouTube channel. For 10 years, I have sought to detail the workings of dozens of commercial video games and summarize academic research for an audience that quite often does not have the same grasp of the subject as I do.

In fact, I've been asked on several occasions whether I would be interested in writing a book that consolidates this for a more scholarly audience. It's a notion I've rejected, because not only does that book already exist—spoiler alert, you're reading it—but it's written by the two people arguably best equipped to do so.

Georgios N. Yannakakis and Julian Togelius have ridden the wave of game AI research since its inception in the mid-2000s, both earning Ph.D.s in AI for games at a time where the world cared little for it. Our authors have been not just at the forefront of many of the methodologies discussed in this book—delivering many papers that are often cited either for their novelty or comprehensive surveys of the subject—but continue to guide the next generation of game/AI scholars and developers in understanding how to make these ideas tractable and practical. Both as active research supervisors and as the organizers of the *AI and Games Summer School* which welcomes hundreds of participants each year from around the world. Togelius and Yannakakis were the researchers that I was citing in my Ph.D. thesis 15 years ago, and I can't think of many individuals who equal their contributions to the field of AI research for games. So rest assured, you're in safe hands.

This second edition is a reflection of how much the field has evolved since the original release in 2018. Not only as the technology has seen significant gains but also in how these technologies can be applied in both commercial projects or scholarly pursuits that utilize games for broader research goals. We have entered a period of change and upheaval as AI begins to intersect various corners of society. At present how new AI methods and tooling will influence the development of commercial games is not readily apparent. Progress can often be as fast as it is slow. But the foundational knowledge required to get a grasp on this space can be found in these pages. And so while the field will continue to see a significant change in the years to come, this book will continue to serve as stepping stone for members of the game AI community both young and old.

London, UK
November 2024

Tommy Thompson

Foreword to the First Edition

It is my great pleasure to write the Foreword for this excellent and timely book. Games have long been seen as the perfect test-bed for artificial intelligence (AI) methods, and are also becoming an increasingly important application area. Game AI is a broad field, covering everything from the challenge of making super-human AI for difficult games such as Go or *StarCraft*, to creative applications such as the automated generation of novel games.

Game AI is as old as AI itself, but over the last decade, the field has seen massive expansion and enrichment with the inclusion of video games, which now comprise more than 50% of all published work in the area and enable us to address a broader range of challenges that have great commercial, social, economic and scientific interest. A great surge in research output occurred in 2005, coinciding with both the first IEEE Symposium (Conference) on Computational Intelligence and Games (CIG)—which I co-chaired with Graham Kendall—and the first AAAI AIIDE Conference (Artificial Intelligence in Digital Entertainment). Since then this rich area of research has been more explored and better understood. The Game AI community pioneered much of the research which is now becoming (or about to become) more mainstream AI, such as Monte Carlo Tree Search, procedural content generation, playing games based on screen capture, and automated game design.

Over the last decade, progress in deep learning has had a profound and transformational effect on many difficult problems, including speech recognition, machine translation, natural language understanding, and computer vision. As a result, computers can now achieve human-competitive performance in a wide range of perception and recognition tasks. Many of these systems are now available to the programmer via a range of so-called cognitive services. More recently, deep reinforcement learning has achieved ground-breaking success in a number of difficult challenges, including Go and the amazing feat of learning to play games directly from screen capture (playing from pixels). It is fascinating to contemplate what this could mean for games as we stumble towards human-level intelligence in an increasing number of areas. The impacts will be significant for the intelligence of in-game characters, the way in which we interact with them, and for the way games are designed and tested.

This book makes an enormous contribution to this captivating, vibrant area of study: an area that is developing rapidly both in breadth and depth as AI is able to cope with a wider range of tasks (and to perform those tasks to increasing levels of excellence). The service to the community will be felt for many years to come: the book provides an easier and more comprehensive entry point for newcomers to the field than previously available, while also providing an indispensable reference for existing AI and Games researchers wishing to learn about topics outside their direct field of interest.

Georgios Yannakakis and Julian Togelius have been involved with the field ever since its widespread expansion to video games, and they both presented papers at the first 2005 CIG. Over the years, they have made an enormous contribution to the field with a great number of highly cited papers presenting both novel research and comprehensive surveys. It is my opinion that these authors are best qualified to write this book, and they do not disappoint. The book will serve the community very well for many years to come.

London, UK
August 2017

Simon Lucas

Preface

Of all the things that wisdom provides for the complete happiness of one's entire life, by far the greatest is friendship.

Epicurus, *Principal Doctrines*, 27

Human beings, viewed as behaving systems, are quite simple. The apparent complexity of our behavior over time is largely a reflection of the complexity of the environment in which we find ourselves.

—Herbert A. Simon

It would be an understatement to say that **Artificial Intelligence** (AI) is an important topic, and it is unlikely to become any less important in the future. More researchers than ever work on AI in some form, and more non-researchers than ever are interested in the field. Some of the world's highest-valued companies have reoriented their businesses around AI, and new AI startups crop up every day. It would also be an understatement to say that **games** are a popular application area for AI research. While board games have been central to AI research since the inception of the field, during the last couple of decades video games have increasingly become the domain of choice for inventing, testing, and showcasing new algorithms. At the same time, video games themselves have become more diverse and sophisticated, and appealing to ever-broader segments of the population. More and more games incorporate advances in AI for controlling non-player characters, generating content, or adapting to players. Game developers have increasingly realized the power of AI methods to analyze large volumes of player data and optimize game designs. Finally, a small but rapidly growing community of researchers and designers experiment with ways of using AI to design and create complete games, automatically or in dialog with humans. It is indeed an exciting time to be working on AI and games!

Book Overview

This is a book about **AI and games**. As far as we know, it is the first *comprehensive* textbook covering the field. By comprehensive, we mean that it features all the major application areas of AI methods within games: gameplaying, content generation and player modeling. We also mean that it discusses AI problems in many different types of games, including board games and video games of many genres. The book is also comprehensive in that it takes multiple perspectives of AI and games: how games can be used to test and develop AI, how AI can be used to make games better and easier to develop, and how it can be used to understand players and design. While this is an academic book which is primarily aimed at students and researchers, we will frequently address problems and methods relevant for game designers and developers. We believe most of the book is relevant to game developers as well.

Our Motivation to Write This Book

We wrote this book based on our long experience doing research on AI for games, each on our own and together, and helping lead and shape the research community. We both independently started researching AI methods in games in 2004, and we have been working together since 2009. Together, we played a role in introducing research topics such as procedural content generation and player modeling to the academic research community, and created several of the most widely used game-based AI benchmarks.

This book is a natural outgrowth of the classes on AI and games we have taught at four universities, and the several survey papers of the field and of individual research topics within it that we have published over the years. Both of us have been teaching and researching game artificial intelligence at undergraduate and graduate levels in various research and educational institutions across the globe for nearly two decades. Back in 2018 (when we published the first edition of the book) we felt that a comprehensive **textbook** on game AI was necessary for our students and a service to the learning objectives of our programs. Many fellow academics felt the same way. Such a book was not available back then, and given our extensive experience with the field, we felt we were well placed to write the book we needed. Given that we have been collaborating on game AI research since 2009, and know each other since 2005, we knew our perspective was coherent enough to actually agree on what should go into the first edition of the book without undue bickering. Our aim in writing that book on the use of AI in and for games was to be widely adopted by graduate programs and used as a key reference to early-state researchers of the field. Did we manage to get there, though?

Six years have passed since the first edition of this book. As we write these lines, the first edition of the book has sold nearly 2000 copies (including eBooks), has been accessed online 53k times via Springer, and has been cited more than 800 times

including citations to individual chapters. We don't even know how many times the self-archived version of the book on our website[1] has been downloaded as we don't track that, but anecdotally it is very many times. Thanks to Jialin Liu, Junkai Lu, and Bolan Su, the book was also translated to Chinese and reached a much wider audience far beyond our initial expectations. During that period, several game AI academics (including ourselves) have used the book in their graduate courses and as part of doctoral training programs within their institutions. Some have even used the book for advanced lectures in undergraduate game AI classes. We solicited opinions from both our colleagues and our students about the book contents and its possible uses in the classroom. We also engaged with popular social media and asked readers of the book to send us suggestions for further improving its contents. Our findings converged on the following: the first edition was well received and widely adopted but there was so much more we could do to improve it.

Since the launch of the book in 2018 we also started running a series of annual summer schools on the topic, using the book as the foundation reading material. At the time of writing, the *Artificial Intelligence and Games Summer School*[2]—as we named it after the book—is in its seventh year of running. The summer school has so far trained nearly 1000 doctoral students, academics, and game industry professionals on the various game AI uses that were covered in the first edition of this book. Our participants have also made dozens of game AI prototypes showcasing the potential of AI for game development.[3] Talks and workshops by fellow researchers working in the most respected industry-based AI research labs or video game studios have complemented the academic contents of the book. Our various guest lecturers in the summer school have presented a continuous stream of new AI methods, novel application areas and unconventional use cases throughout the years. Such a creative osmosis with the AI and games ecosystem spurred ideas on how a new and improved edition of the book should be presented.

Six years ago, we also cofounded *modl.ai*, a start-up with the vision to automate the game testing process through AI, and to pioneer new ways of creating non-player characters. In the process, we have worked with dozens of game developers, from boutique outfits to global leaders and household names. This has taught us much about the challenges of bringing modern AI methods into actual games, from the time and resource pressures of the game development process and the constraints of game software architecture to the cultural differences between researchers, designers, and developers. We have learned a lot, and much of that makes it to this second edition.

Ultimately, this book is also a response to the lack of a good introductory book for junior and senior researchers entering the AI and Games research field. Early discussions on writing such a book date roughly two decades back, but no one actually wrote one, until our first edition in 2018. Taking all this into account, our goal in producing a second version of this book was to thoroughly update it with work

[1] https://gameaibook.org.

[2] https://school.gameaibook.org/.

[3] Example repository of 2023 game AI jam entries: https://github.com/GameAISchool2023members/.

that has happened in the last 6 years, and make the book even more relevant to the whole AI and games community, including researchers, students, and developers. It is important to note, however, that while we tried hard to write a book that will appeal to many and be useful for both students and researchers from different backgrounds, it ultimately reflects our perspective of what game AI is and what is important within the field.

What's New in This Edition?

This second edition attempts to capture the major changes that took place in AI and its various applications to games since the first edition in 2018. During these last 6 years, we witnessed algorithmic advancements in computer vision, reinforcement learning, and deep learning among many other AI areas. Indicatively, new AI principles such as the *attention* mechanism and the *transformer* architecture gave birth to *large language models* that currently dominate the interest of AI research, along with *diffusion models*, another recent invention. Since the first edition of this book so many new popular terms and families of methods have been introduced—*generative AI* in particular has become a popular moniker for some old and some new methods—that we felt we need to discuss in the context of games. From the perspective of the domain of games, we have experienced entirely new ways of utilizing such AI methods. Arguably, the future of video game development is tightly coupled with AI tools that will assist us creating, developing, and testing new games. Below, we list the major changes of this edition:

- We reorganized the various parts of the book and our taxonomies, offering a fresh and comprehensive overview of the field and its recent advancements.
- Our new AI background section is up to date with general AI principles and core methods that are commonly used in games.
- We have expanded and updated all three core uses of AI in games (play, generate, and model) and dedicated a book part to each.
- We expanded on the ethical considerations of AI applications to games through a new chapter given how timely and important this discussion is.
- Our online presence has been revamped. We now include new online exercises, repositories, and lecture slides as additional educational material for any graduate class on the topic.

Who Is This Book for?

This is a book for readers who already understand AI methods and concepts to the level of having taken a fundamental AI course, and the introductory computer science or engineering courses that led up to that course. We thus hope to reach readers with a

general interest in the application of AI to games, who already know at least the basics of artificial intelligence. While writing the book, however, we particularly envisioned three groups of people benefiting directly. The first group is university **students**, of graduate or advanced undergraduate level, who wish to learn about AI in games and use it to develop their career in game AI programming or in game AI research. With care and some supplementary material, it could be used for an introductory course as well. The second group is AI **researchers** and **educators** who want to use this book to inspire their research and perhaps also as a textbook for a class in AI and games. We particularly think of active researchers within some AI-related field wanting to start doing research in game AI, and new Ph.D. students in the area. The final group is computer game **programmers** and practitioners who have limited AI or machine learning background and wish to explore the various creative uses of AI in their game or software application. Here we provide a complement to the more industry-focused books that are listed in Chap. 1 by taking a broader view of what AI in and for games could be. For further fostering the learning process and widening the practical application of AI in games the book is accompanied by a website that features lectures, exercises, and additional resources such as readings and tools.

This book is written with the assumption that its readers come from a *technical background* such as computer science, software engineering, or applied math. As pointed out above, we assume that our readers have taken courses on the fundamentals of artificial intelligence (or acquired this knowledge elsewhere) as the book does not cover the algorithms in detail; our focus, instead, is on the *use* of the algorithms in games and their modification for that purpose. We assume that the reader is familiar with core concepts in tree search, optimization, supervised learning, self- and semi-supervised learning, unsupervised learning, reinforcement learning, deep learning, and preferably also state-of-the-art methods such as transformers and diffusion models.

The book also assumes that the reader is comfortable with reading a pseudocode description of an algorithm and implementing it. Our envisaged reader has implemented some basic algorithms from the categories listed above within games (or beyond) and has some basic understanding of algebra and calculus. Part I provides an overview of core methods for game AI and a refresher for the reader whose knowledge is a bit rusty. While it offers a summary of AI principles, learning paradigms, and methods used in the book, it is intended more as a reference than as an introduction. Several of our readers, however, have informed us that they enjoyed reading this background material as an introduction to the AI field—independently of games. Finally, the book assumes a basic familiarity with games, if not designing them then at least playing them.

It is also useful to point out what this book is not. It is not a hands-on book with step-by-step instructions on how to build AI for your game. It does not feature discussions of any particular game engine or software framework, and it does not discuss software engineering aspects or many implementation aspects at all. There are no code examples. It is not an introductory book, and it does not give a thorough introduction to basic AI or game design concepts. For all these roles, there are better books available.

How to Use This Book

The textbook is intended to be used for a one- or a two-semester graduate-level or advanced undergraduate-level class. This can take several different shapes to support different pedagogical practices. One way of teaching such a class would be a traditional class, with lectures covering the chapters of the book in order, a conventional pen-and-paper exam at the end, and a few programming exercises. Another way of organizing a class around this book, more in line with how we personally prefer to teach such courses, is to teach the course material during the first half of the semester and spend the second half on a group project.

For your convenience, each AI method covered in Part I is accompanied by a game example. Also for your convenience, the last chapter of each of the main parts of the book includes suggestions for exercises on AI playing games, generating content, and modeling players. Note, however, that our aim is to offer some representative ways one can use a specific AI method to perform a particular task in games. Hence, our examples and exercises offer sufficient flexibility in terms of their implementation and leave room for one's interpretation. Some examples and exercises are more detailed than others, depending on the game AI task we cover. None of our examples contains code (or even pseudocode) yet our online links to exercises contain frameworks, repositories of methods, and data one may use.

The material offered by this book can be used in various ways and support a number of different classes. In our experience, a traditional two-semester class on game artificial intelligence would normally cover Parts I and II in the first semester and then focus on alternative uses of AI in games (Parts III and IV) in the second semester. Instead, when teaching the material in a compressed (one-semester) fashion, it is advisable to skip Part I (using it as a reference when needed), and focus the majority of the lectures on Parts II, III, and IV. Chapters 13 and 14 can be used as material for inspiring advanced graduate-level projects in the area. Beyond the strict limits of game AI, Part III can complement classes with a focus on generative AI in games, game design or computational creativity whereas Part IV and Chap. 15 can complement classes with a focus on affective computing, user experience, and data analytics in games and beyond. This book can also be used for an introductory undergraduate class for students who have not taken an AI class before, but in that case, we advise the instructor to select a small subset of topics to focus on, and to complement the book with online tutorials on specific methods (e.g., best-first search, evolutionary computation) that introduce these topics in a more detailed fashion than this book does.

Notes on Terminology and Writing Style

The term "artificial and computational intelligence in games" is sometimes used to refer to the entire field covered in this book. This reflects the dual roots of the field in

artificial intelligence and **computational intelligence** (CI) research, and the use of these terms in the names of the major conferences in the field (AIIDE and CIG) and the first flagship journal (IEEE TCIAIG) explicitly targeting both CI and AI research. There is no agreement on the exact meaning of the terms AI and CI. Historically, AI has been associated with logic-based or symbolic methods such as reasoning, knowledge representation, and planning. CI has been associated with biologically-inspired or statistical learning methods such as neural networks (including what is now known as deep learning) and evolutionary computation. However, there is considerable overlap and strong similarities between these fields. Most of the methods proposed in both fields aim to make computers perform tasks that have at some point been considered to require intelligence to perform, and most of the methods include some form of heuristic search. The field of machine learning intersects with both AI and CI, and many techniques could be said to be part of either field. These days, AI is often used synonymously with machine learning, given the dominance of machine learning and particularly large autoregressive models in AI research and practice.

In the remainder of the book, we will use the terms "AI and games", "AI in games", and "game AI" interchangeably to refer to the whole research field, including those approaches that originally come from the CI and machine learning fields. There are three reasons for this: simplicity, readability, and that we think that the distinction between CI and AI is not useful for the purposes of this book or indeed the research field it describes. Our use of these terms is not intended to express any prejudice towards particular methods or research questions. Despite being a book about AI and the ways it has revolutionized how we interact with and design games it is worth noting that we did not make use of large language models to write any of its contents (besides parts of this sentence); instead, we relied on our own chimeric and perhaps idiosyncratic writing style, for better or worse. We end this preface with our rationale when it comes to the use of *italics* and **bold** text throughout the book. We use italics to put an emphasis on a particular term, method, or notion that we consider important for the reader. We use bold text, instead, to refer to important terms that are included in the index section of the book; not all indexed terms are boldfaced, however.

Sliema, Malta
New York, NY, USA
December 2024

Georgios N. Yannakakis
Julian Togelius

Acknowledgments Writing a book of this size would be impossible without the support and the contributions of a large number of people. First, we would like to thank Alexandru Ciolan (second edition) and Ronan Nugent (first edition), our editors at Springer Nature, who guided us and helped us from the book proposal phase all the way to its final production.

We also would like to thank the following persons who read all (or parts of a draft of) the book and provided useful feedback: **(first edition)** Amy Hoover, Amin Babadi, Sander Bakkes, Vadim Bulitko, Phil Carlisle, Georgios Chalkiadakis, Dave Churchill, Mike Cook, Renato Cunha, Kevin Dill, Nathaniel Du Preez-Wilkinson, Chris Elion, Andy Elmsley, David Fogel, Bernardo Galvão, Kazu-ma Hashimoto, Aaron Isaksen, Emil Johansen, Mark Jones, Niels Justesen, Graham Kendall, Jakub Kowalski, Antonios Liapis, Nir Lipovetzky, Jhovan Mauricio López, Simon Lucas, Jacek Mańdziuk, Luciana Mariñelarena-Dondena, Chris Martens, Sean Mealin, Mark Nelson, Sven Neuhaus, Alexander Osherenko, Santiago Ontañón, Cale Plut, Mike Preuss, Hartmut Procha-ska, Christoffer Holmgård, Florian Richoux, Sebastian Risi, Christoph Salge, Andrea Schiel, Jacob Schrum, Magy Seif El-Nasr, Adam Smith, Gillian Smith, Dennis Soemers, Nathan Sturtevant, Gabriel Synnaeve, Nicolas Szilas, Varunyu Vorachart, James Wen, Marco Wiering, Mark Winands, Junkai Lu, Francesco Calimeri, Diego Pérez Liébana, Corine Jacobs, Junkai Lu, Hana Rudova, Robert Zubek, **(second edition)** Antonios Liapis, Konstantinos Makantasis, Ahmed Khalifa, David Melhart, Rodrigo Cannan, Mostafa Touny, Graham Todd, Jeroen Van Goey, Kyung-Joong Kim, Sherzod Hakimov, Giovanni Paolini, Kok Keng Fai, Christian Guckelsberger, Pedro Santos, Jialin Liu, Matthew Stephenson, Manuel Guimarães, Timo Bertram, Youichiro Miyake, Matthias Müller-Brockhausen, Tiago Machado, Jack Parker-Holder, Tim Rocktäschel, Joanna J. Bryson, Zehua Jiang, Tim Merino, Eric Piette, Dennis Soemers, and Veronique Ventos. Of these, we would like to extend our gratitude to Timo Bertram for offering guidance on the collectible card games section, Matthew Stephenson and Eric Piette for their text input on AI for Angry Birds and the Ludii general game system, Manuel Guimarães for his assistance with discussions related to the relationship between game AI academia and industry, and Youichiro Miyake for kindly sending us relevant material related to Pac-Man such as his interview reflections with Toru Iwatani, the game's creator. We also wish to especially thank Simon Lucas and Tommy Thompson for writing the forewords to the first and second editions of the book, Georgios Chalkiadakis for providing substantial input on parts of the book related to game theory, and Mark Nelson, Mike Preuss, and Adam Smith for reviewing large parts of the first edition of the book and providing particularly detailed feedback. The unwavering efforts of Jialin Liu, Junkai Lu, and Bolan Su were instrumental in the creation of the Chinese version of the first edition of the book. We are deeply grateful for their invaluable support! We also want to thank all those who granted us their permission to reproduce images and figures from their papers; they are all acknowledged in the figure captions of the book. Special thanks also go to Rebecca Portelli and Daniel Mercieca for the artwork featured on the cover page of the book. Finally, we wish to thank Antonios Liapis wholeheartedly for painstakingly reviewing every tiny detail of the

book and leaving more than 3000 comments for us to address (!); we couldn't have hoped for a better and more honest reviewer!

Some chapters of this book build on papers that we have co-authored. In some cases the papers are co-authored by more than the two of us; for those papers, our co-authors graciously gave us permission to reuse parts of the material and we wish to thank them for that. In particular,

- Chapter 1: [1152, 1269].
- Part III: [365, 643 1024].
- Part IV: [178, 291, 694, 1284, 1287, 1288, 1289].
- Chapter 13: [1292].
- Chapter 14: [1173].
- Chapter 15: [746, 763].

Writing this book has been a very long journey for both of us; a challenging and overwhelming journey at times. There are a number of people who have supported this effort that we would like to thank. Georgios would like to thank the University of Malta for granting him a sabbatical leave, without which the book would not have been possible. Georgios and Julian also wish to thank each other for putting up with each other despite everything, and announce that they intend to buy each other drinks to celebrate when they meet next.

Last but not least, both of us wish to thank our families and all the people who showed us their support, care, encouragement, and love at times when they were absolutely needed. You are too many to list here, but big thanks go to you all! Georgios would not have been able to write this book without the love and support of Myrto, Amyrsa, and Stavroula; this book is dedicated to all three of you! Julian would like to thank Dylan and Raz for everything you do and everything we have. Dear family, this book is dedicated to you!

Competing Interests The authors have no competing interests to declare that are relevant to the content of this manuscript.

Website

https://gameaibook.org/

This book is associated with the above website. The website complements the material covered in the book with up-to-date exercises, lecture slides, and readings.

Contents

1 Introduction 1
- 1.1 What Is This Book About? 2
- 1.2 A Brief History of Artificial Intelligence and Games 4
  - 1.2.1 AI and Games Research in Academia 8
  - 1.2.2 Game AI Developments in Industry 9
  - 1.2.3 The "Gap" Between Academia and Industry 14
- 1.3 Why Use Games for Artificial Intelligence? 17
  - 1.3.1 Games Are Hard and Interesting Problems 17
  - 1.3.2 Rich Human-Computer Interaction 19
  - 1.3.3 Games Are Popular 20
  - 1.3.4 There Are Challenges for All AI Areas 22
  - 1.3.5 Games Best Realize Long-term Goals of AI 23
- 1.4 Why Use Artificial Intelligence for Games? 25
  - 1.4.1 AI Can Play, Test and Improve Games 26
  - 1.4.2 More Content, Better Content 26
  - 1.4.3 Player Experience and Behavior Analytics 27
- 1.5 The AI and Games Co-revolution 28
- 1.6 Structure of This Book 28
  - 1.6.1 What We (Don't) Cover in This Book 30
- 1.7 Summary 31

Part I Background

2 AI Fundamentals 35
- 2.1 General Concepts 35
  - 2.1.1 Representation 36
  - 2.1.2 Utility 37
- 2.2 Search and Optimization 38
  - 2.2.1 Tree Search 38
  - 2.2.2 Evolutionary Computation 39

2.3 Machine Learning . . . 40
2.3.1 Unsupervised Learning . . . 40
2.3.2 Supervised Learning . . . 42
2.3.3 Semi-supervised Learning . . . 45
2.3.4 Self-supervised Learning . . . 46
2.3.5 Reinforcement Learning . . . 48
2.4 AI and Human Feedback . . . 52
2.4.1 The Ordinal Nature of Human Feedback . . . 58
2.5 Summary . . . 60

3 AI Methods for Games . . . 61
3.1 Behavior Authoring . . . 62
3.1.1 Finite State Machines . . . 62
3.1.2 Behavior Trees . . . 64
3.1.3 Utility-Based AI . . . 66
3.1.4 Further Reading . . . 68
3.2 Tree Search Methods . . . 69
3.2.1 Uninformed Search . . . 69
3.2.2 Best-First Search . . . 70
3.2.3 Minimax . . . 72
3.2.4 Monte Carlo Tree Search . . . 74
3.2.5 Further Reading . . . 79
3.3 Local and Global Search Methods . . . 79
3.3.1 Local Search . . . 79
3.3.2 Evolutionary Algorithms . . . 82
3.3.3 Further Reading . . . 88
3.4 Unsupervised Learning Methods . . . 89
3.4.1 K-Means Clustering . . . 89
3.4.2 Hierarchical Clustering . . . 90
3.4.3 Apriori . . . 91
3.4.4 Generalized Sequential Patterns . . . 92
3.4.5 Further Reading . . . 93
3.5 Supervised Learning Methods . . . 93
3.5.1 Support Vector Machines . . . 93
3.5.2 Decision Tree Learning . . . 96
3.5.3 Artificial Neural Networks . . . 99
3.5.4 Further Reading . . . 105
3.6 Deep Learning Methods . . . 106
3.6.1 Deep Neural Networks . . . 106
3.6.2 Autoencoders . . . 111
3.6.3 Convolutional Networks . . . 112
3.6.4 Generative Adversarial Networks . . . 114
3.6.5 Diffusion Models . . . 116
3.6.6 Transformers . . . 118

3.6.7 Large Language Models ... 122
3.6.8 Further Reading ... 124
3.7 Reinforcement Learning Methods ... 124
3.7.1 Q-Learning ... 124
3.7.2 TD Learning with Artificial Neural Networks ... 126
3.7.3 Proximal Policy Optimization ... 129
3.7.4 Neuroevolution ... 131
3.7.5 Further Reading ... 134
3.8 Summary ... 134

Part II Play

4 Playing Games ... 137
4.1 Why Use AI to Play Games? ... 138
4.2 Taxonomy: Playing Games ... 139
4.2.1 Uses ... 140
4.2.2 Game Characteristics ... 150
4.2.3 Game AI Deployment ... 157
4.3 Further Reading ... 161
4.4 Summary ... 162

5 Methods for Playing Games ... 163
5.1 Behavior Authoring ... 163
5.2 Planning-Based Approaches ... 164
5.2.1 Classic Tree Search ... 164
5.2.2 Stochastic Tree Search ... 167
5.2.3 Evolutionary Planning ... 168
5.2.4 Planning with Symbolic Representations ... 169
5.3 Reinforcement Learning ... 170
5.3.1 Classic and Deep Reinforcement Learning ... 171
5.3.2 Evolutionary Reinforcement Learning ... 173
5.4 Supervised Learning ... 174
5.5 Large Pretrained Models ... 175
5.6 Chimeric Game Players ... 176
5.7 Learning Beyond Policies ... 177
5.7.1 Learning Representations ... 177
5.7.2 Learning Forward Models ... 179
5.8 Further Reading ... 180
5.9 Summary ... 180

6 Gameplaying AI by Game Genre ... 181
6.1 Board Games ... 182
6.1.1 Classic Board Games ... 182
6.1.2 Modern Board Games ... 184
6.2 Card Games ... 185
6.2.1 Classic Card Games ... 186

6.2.2 Collectible Card Games 187
6.2.3 Other Card Games 190
6.3 Real-time 2D Video Games 191
6.3.1 Classic Arcade Games 192
6.3.2 Pac-Man and Ms Pac-Man 192
6.3.3 Super Mario Bros 194
6.3.4 The ALE Framework 197
6.3.5 General Video Game AI 199
6.3.6 Fighting Games 201
6.3.7 Other Environments 202
6.4 Real-Time 3D Video Games 203
6.4.1 Racing Games 203
6.4.2 First-Person Shooters 206
6.4.3 Minecraft 210
6.5 Dungeon Crawlers 213
6.6 Puzzle Games 215
6.7 Strategy Games 217
6.7.1 Diplomacy 219
6.7.2 StarCraft 219
6.7.3 Team Sports Games 223
6.8 Text-Based Games 224
6.9 Social Simulation Games 226
6.10 Further Reading 229
6.11 Exercises 230
6.12 Summary 231

Part III Generate

7 Procedural Content Generation 235
7.1 Why Generate Content? 236
7.2 Taxonomy: Procedural Content Generation 237
7.2.1 Content 237
7.2.2 Methods 240
7.2.3 Roles 242
7.3 Further Reading 251
7.4 Summary 251

8 Methods for Generating Content 253
8.1 Constructive Methods 253
8.1.1 Grammar-Based Methods 254
8.1.2 Cellular Automata 256
8.1.3 Noise and Fractals 260
8.1.4 Wave Function Collapse 262
8.2 Evolutionary Algorithms and Search-Based PCG 266

8.3 Quality-Diversity Methods ... 270
8.4 Solver-Based Methods ... 273
8.5 PCG via Machine Learning ... 275
8.5.1 n-grams and Markov Models ... 276
8.5.2 Neural Networks ... 277
8.5.3 GANs ... 278
8.5.4 LLMs ... 279
8.6 Reinforcement Learning ... 282
8.7 Further Reading ... 285
8.8 Summary ... 286

9 Procedural Content Generation by Content Type ... 287
9.1 Levels ... 287
9.2 Visuals ... 289
9.3 Audio ... 292
9.4 Narrative ... 294
9.5 Rules and Mechanics ... 299
9.6 Games ... 302
9.7 Evaluating Content Generators ... 305
9.7.1 Content Evaluation: Why Is It Difficult? ... 306
9.7.2 Content Properties: Function Versus Aesthetics ... 306
9.7.3 How Can We Evaluate a Generator? ... 307
9.8 Further Reading ... 310
9.9 Exercises ... 310
9.9.1 Maze Generation ... 311
9.9.2 Platformer Level Generation ... 311
9.9.3 Level Generation via RL ... 311
9.9.4 PCG Benchmark ... 312
9.10 Summary ... 312

Part IV Model

10 Player Modeling ... 315
10.1 Why Model Players? ... 316
10.2 What Is Player Modeling? ... 317
10.3 Taxonomy: Player Modeling ... 320
10.3.1 Roles ... 321
10.3.2 Approaches ... 324
10.3.3 Input and Output ... 334
10.4 Further Reading ... 335
10.5 Summary ... 335

11 From Observations to Models of Players 337
11.1 What Is the Model's Input Like? 338
11.1.1 Gameplay Input 340
11.1.2 Objective Input 342
11.1.3 Game Context 346
11.1.4 Player Profile 348
11.2 What Is the Model's Output Like? 350
11.2.1 Learning Player Embeddings 350
11.2.2 Modeling Player Behavior 352
11.2.3 Modeling Player Experience 353
11.3 Manual Labeling: Principles and Tools 354
11.4 Methods for Player Modeling 361
11.4.1 Player Model Authoring 362
11.4.2 Supervised Learning 364
11.4.3 Reinforcement Learning 369
11.4.4 Semi-Supervised Learning 371
11.4.5 Self-Supervised Learning 372
11.4.6 Unsupervised Learning 373
11.5 Further Reading 374
11.6 Summary 375

12 Player Modeling Exemplified 377
12.1 Player Embeddings 378
12.2 Player Behavior 382
12.3 Player Experience 386
12.4 Further Reading 395
12.5 Exercises 395
12.5.1 Player Behavior 396
12.5.2 Player Experience 397
12.6 Summary 398

Part V The Road Ahead

13 Panorama and Map of Synergies 403
13.1 Panoramic Views of AI and Games 404
13.1.1 Methods (Computer) Perspective 405
13.1.2 End User (Human) Perspective 407
13.1.3 Player-Game Interaction Perspective 409
13.2 How Game AI Areas Inform Each Other 410
13.2.1 Play Games 411
13.2.2 Generate Content 414
13.2.3 Model Players 418
13.3 What's Next? 421
13.4 Summary 422

14 Frontiers of AI and Games Research 423
14.1 General General Game AI 423
14.1.1 General Play 425
14.1.2 General Game Generation and Orchestration 426
14.1.3 General Affective Game Loop 429
14.2 Game Engines for Game AI 430
14.3 Other Uses and Roles of AI in Games 431
14.4 Summary 437

15 Ethics of AI and Games 439
15.1 Overview 439
15.2 Elicitation—Boundaries of Artificially Induced Emotions 441
15.3 Sensing—Privacy and Control 443
15.4 Detection—Transparency in Limited Information Systems 445
15.5 Adaptation—Ownership of Data and Models 447
15.6 Other Issues and Concerns 448
15.7 Summary 449

References 451

Index 515

Acronyms

A3C	Asynchronous Advantage Actor-Critic
ABL	A Behavior Language
AFC	Alternative Forced Choice
AFK	Away From Keyboard
AGI	Artificial General Intelligence
AI	Artificial Intelligence
AIIDE	Artificial Intelligence and Interactive Digital Entertainment
ALE	Arcade Learning Environment
ANN	Artificial Neural Network
ASP	Answer Set Programming
BCI	Brain-Computer Interfaces
BDI	Belief-Desire-Intension
BT	Behavior Tree
BWAPI	Brood War API
CA	Cellular Automata
CCG	Collectible Card Game
CFR	Counterfactual Regret Minimization
CI	Computational Intelligence
CIG	Computational Intelligence and Games
CLIP	Contrastive Language-Image Pre-training
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
CoG	Conference on Games
CPPN	Compositional Pattern Producing Network
DDPM	Denoising Diffusion Probabilistic Model
DNN	Deep Neural Network
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
EA	Evolutionary Algorithm
ECG	Electrocardiography
EDPCG	Experience-Driven Procedural Content Generation

EEG	Electroencephalography
EMG	Electromyography
FPS	First-Person Shooter
FSM	Finite State Machine
FSMC	Functional Scaffolding for Musical Composition
GA	Genetic Algorithm
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
GDC	Game Developers Conference
GGP	General Game Playing
GRFE	Google Research Football Environment
GSP	Generalized Sequential Patterns
GSR	Galvanic Skin Response
GVGAI	General Video Game Artificial Intelligence
HCI	Human-Computer Interaction
ID3	Iterative Dichotomiser 3
IDM	Inverse Dynamics Model
JPS	Jump Point Search
LLM	Large Language Model
LMM	Large Multimodal Model
LSTM	Long Short-Term Memory
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
MMMORPG	Massively-multiplayer Online Role-playing Game
MOBA	Multiplayer Online Battle Arena
MtG	Magic: the Gathering
MUD	Multi-User Dungeon
MVC	Model-View-Controller
NEAT	NeuroEvolution of Augmenting Topologies
NES	Natural Evolution Strategy
NLP	Natural Language Processing
NPC	Non-Player Character
OCC	Ortony-Clore-Collins
PAD	Pleasure, Arousal, Dominance
PC	Player Character
PCG	Procedural Content Generation
PCGML	Procedural Content Generation via Machine Learning
PCGRL	Procedural Content Generation via Reinforcement Learning
PENS	Player Experience of Need Satisfaction
PLT	Preference Learning Toolbox
PPO	Proximal Policy Optimization
QD	Quality Diversity
RBF	Radial Basis Function
ReLU	Rectified Linear Unit

RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RPG	Role-Playing Game
RRT	Rapidly-exploring Random Tree
RTS	Real-Time Strategy
SC:BW	StarCraft: Brood War
SDT	Self-Determination Theory
SGD	Stochastic Gradient Descent
SOM	Self-Organizing Map
STRIPS	Stanford Research Institute Problem Solver
SVM	Support Vector Machine
TCIAIG	Transactions on Computational Intelligence and AI in Games
TD	Temporal Difference
TORCS	The Open Racing Car Simulator
TRU	Tomb Raider: Underworld
TSP	Traveling Salesman Problem
UPEQ	Ubisoft Perceived Experience Questionnaire
UT2k4	Unreal Tournament 2004
VGDL	Video Game Description Language
ViT	Vision Transformer
VLM	Vision-Language Model
WFC	Wave Function Collapse

Chapter 1
Introduction

Abstract This chapter introduces the main topics of the book, and the field of AI and Games. The chapter gives a short outline of AI and games, including both academic research on AI applied to games, and developments in game AI from within the game industry. We then discuss how games, including video games as well as board games and other types of games, can be useful for AI research, as well as why AI can be used in game design and development. A key takeaway of this chapter is that there are a large number of applications for AI in games, many of them which are not about playing the games. The chapter finishes by presenting the structure of the book and outlines how to use it.

Artificial Intelligence (AI) has seen immense progress in recent years. It is both a thriving research field featuring an increasing number of important research areas and a set of core technologies for an increasing number of application areas. In addition to algorithmic innovations, the rapid progress in AI is often attributed to hardware advancements and data availability. The success stories of AI can be experienced in our daily lives through its many practical applications. AI advances have enabled better understanding and generation of images and speech, emotion detection, self-driving cars, web searching, AI-assisted creative design, and gameplaying, among many other tasks; for some of these tasks machines have reached human-level status or beyond. Most recently, large language models and diffusion models have stunned the world with their ability to create and understand natural language and images.

There is, however, a difference between what machines can do well and what humans are good at. In the early days of AI, researchers envisaged computational systems that would exhibit aspects of human intelligence and achieve human-level problem solving or decision making skills. These problems were presented to the machines as a set of formal mathematical notions within rather narrow and controlled spaces, which could be solved by some form of symbol manipulation or search in symbolic space. The highly formalized, symbolic representation allowed AI to succeed in many cases. Naturally, games—especially **board games**—have been a popular domain for early AI attempts as they are formal and highly constrained, yet complex, decision making environments.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_1

Over the years the focus of much AI research has shifted to tasks that are relatively simple for humans to do but are hard for us to describe how to do, such as remembering a face or recognizing our friend's voice over the phone. As a result, AI researchers began to ask questions such as: *How can AI detect and express emotion? How can AI educate people, be creative or artistically novel? How can AI play a game it has not seen before? How can AI learn from a few trials? How can AI feel guilt?* All these questions pose serious challenges to AI and correspond to tasks that are not easy for us to formalize or define objectively. Perhaps surprisingly, tasks that require relatively low cognitive effort from us often turn out to be much harder for machines to tackle. Again, games have provided a popular domain to investigate such abilities as they feature aspects of a subjective nature that cannot be formalized easily. These include the experience of play or the creative process of game design [999].

Ever since the birth of the idea of artificial intelligence, games have been helping progress in AI research. Games not only pose interesting and complex problems for AI to solve such as playing a game well; they also offer a canvas for creativity and expression which is *experienced* by users (people or even machines!). Games are a rare domain where science (problem solving) meets art and interaction: these ingredients have made games a unique and favorite domain for the study of AI. But it is not only AI that is advanced through games; games have also been advanced through AI research. We argue that AI has been helping games across several fronts: in the way we play them, in the way we understand their inner functionalities, in the way we design and develop them, and in the way we understand play, interaction and creativity. This book is dedicated to the intersection of games and AI and the numerous ways both games and AI have been challenged, but nevertheless, advanced through this relationship. It is a book about *AI for games* and *games for AI*.

1.1 What Is This Book About?

The study of AI *in* and *for* games is what this book defines as the research field of **game artificial intelligence** (in brief **game AI**, also occasionally referred to as **AI and games**).[1] The book offers an academic perspective of game AI and serves as a comprehensive guidebook for this exciting and fast-moving research field. Game AI—in particular video game or computer game AI—has seen major advancements in the (roughly) twenty years of its existence as a separate research field. During this time, the field has seen the establishment and growth of important yearly meetings—including the IEEE Conference on Computational Intelligence and Games (CIG)—known as the IEEE Conference on Games (CoG) since 2019—and the AAAI Artificial Intelligence and Interactive Digital Entertainment (AIIDE)

[1] There are various other definitions of these terms, in particular, "Game AI" is sometimes used to refer to only NPC control using certain classical AI methods We take what we consider a more inclusive perspective.

conference series—as well as the launch of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES (TCIAIG) journal—which was renamed IEEE TRANSACTIONS ON GAMES[2] (ToG) in 2018. Since the early days of game AI we have seen numerous success stories within the several subareas of this growing and thriving research field. We can nowadays use AI to play many games better than any human, we can design AI bots that are believable and human-like, we can collaborate with AI to design better and unconventional (aspects of) games, we can better understand players and play by modeling the overall game experience, we can better analyze game design by modeling it as an algorithm, and we can improve game design and tune our monetization strategy by analyzing massive amounts of player data. This book builds on these success stories and the algorithms that took us there by exploring the different *uses* of AI for games and games for AI.

Looking at the existing literature to allocate readings for a potential course on game AI one can rely partly on a small number of relevant and recent surveys and vision papers for specific game AI research areas. Examples include papers meant to serve as general introductions to game AI [675, 1268, 1291], general game AI [1172] and the ethics of game AI [745], and papers related to specific methods as applied to games such as Monte-Carlo Tree Search [137], neuroevolution [946], deep learning for playing [520] and creating content [654], and large language models [365]. A number of survey papers focus on various methods for procedural content generation [559, 654, 1104, 1176, 1289], player modeling [1288], emotion in games [1286, 1287], computational narrative [938, 939], AI for game production [941], all the way to AI for games on mobile devices [452]. Earlier surveys similarly reflect the state of the art at the beginning of this research field, for example, on evolutionary computation in games [674] and computational intelligence in games [673]. No mere paper can however on its own cover the breadth and depth required by a full course on game AI. For this reason, the courses we have taught—before the first edition of the book came out—have generally been structured around a set of papers, some of them surveys and some of them primary research papers, together with slides and course notes.

A number of edited volumes as well as monographs on game AI research have been great assets for teaching game AI over the last decade. These are books with a focus on a particular area of game AI research such as game playing [1153], procedural content generation [419, 1023], emotion in games [544] and game data science [312, 314]. However, these specialized books cannot serve as textbooks for a complete course on game AI but rather as parts of a game AI course or, alternatively, as textbooks for independent courses on procedural content generation, affective game computing or game data mining, for instance.

Meanwhile several edited volumes or monographs covering aspects of game AI programming are edited or written by game AI experts from the game industry. These include the popular *Game AI programming wisdom* [907–910], *Game AI Pro* [917–919] and *Game AI Pro 360* [911–914] series edited by Rabin and other game AI programming volumes [12, 145, 146, 705, 949–951, 1007]. These books, however, target primarily professional or indie developers, game AI programmers

[2] https://transactions.games/.

and practitioners, and do not always fulfill the requirements of an academic textbook. Some of the earlier books (including ours [1292]) are somewhat outdated given the fast pace of game AI research [113, 191]. Among the industry-focused game AI books, a few target educators and students of game AI. They have a rather narrow scope, however, as they primarily focus on non-player character AI [766, 767], which is arguably the most important topic for game AI practitioners in the gaming industry [705, 1268] but only one of several research areas in academic game AI [1291]. In our terminology, the perspective of these industry-focused textbooks tends to focus strongly on what we call *playing for experience*, in particular generating interesting **non-player character** (NPC) behavior that looks lifelike and functions within the confines of a game design. As you will see, however, this is only one part of the field of game AI as we define it. Finally there are game AI books that are tied to a particular language or software suite such as Lua [1299] or Unity [56], which also limits their usefulness as general textbooks.

In contrast to the aforementioned resources, this book aims to present the research field as a whole and serve (a) as a comprehensive *textbook* for game artificial intelligence, (b) as a *guidebook* for game AI programming, and (c) as a *field guide* for researchers and graduate students seeking to orient themselves within this multifaceted research field. For this reason, we both detail the state of knowledge of the field and also present research and original scholarship in game AI. Thus the book can be used for both research-based teaching and hands-on applications of game AI. Before taking the reader through the current state of the art and future roadmap of AI in games, we dedicate the next section of this chapter to the exciting history of this field.

1.2 A Brief History of Artificial Intelligence and Games

Games and artificial intelligence have a long history together. Much research on AI for games is concerned with constructing agents for playing games, with or without a learning component. Historically, this has been the first and, for a long time, the only approach to using AI in games. Even since before artificial intelligence was recognized as a field, early pioneers of computer science wrote gameplaying programs because they wanted to test whether computers could solve tasks that seemed to require "intelligence". Alan Turing, arguably the principal inventor of computer science, (re)invented the Minimax algorithm and used it to play Chess [1191]. In 1951, Christoper Strachey, working on the *Manchester Mark 1* computer, coded the first computer program that was able to play a complete game of draughts (Checkers). The first software that managed to master a game, however, was programmed by Alexander S. Douglas the year after, on a digital version of the **Tic-Tac-Toe** game as part of his doctoral dissertation at Cambridge. The board of that game was displayed on a screen making it the first game with visuals. A few years later, Arthur Samuel

was the first to invent the form of machine learning that is now called **reinforcement learning** using a program that learned to play Checkers by playing against itself [987].

Most early research on gameplaying AI was focused on classic board games, such as **Checkers** and **Chess**. There was a hypothesis that these games, where great complexity can arise from simple rules and which had challenged the best human minds for hundreds or even thousands of years, somehow captured the essence of thought. Therefore, a good board game-playing program would have to actually be intelligent. After over three decades of research on tree search, in 1994, the **Chinook** Checkers player managed to beat the World Checkers Champion Marion Tinsley [994]; the game was eventually solved in 2007 [993]. For decades Chess was seen as "the drosophila of AI" in the sense of being the "model organism" that uncountable new AI methods were tested on [324]—at least until we developed software capable of playing better than humans, at which point Chess-playing AI somehow seemed a less urgent problem. The software that first exhibited superhuman Chess capability, IBM's **Deep Blue**, consisted of a **Minimax** algorithm with numerous Chess-specific modifications and a very highly tuned board evaluation function running on a custom supercomputer [168, 480]. Deep Blue famously won against the reigning grandmaster of Chess, Garry Kasparov, in a much-publicized event back in 1997. Twenty five years later, it is possible to download public domain software that will play better than any human player when running on a regular laptop.

Another milestone in AI research in games arrived a few years before the successes of Deep Blue and Chinook, namely the backgammon-playing program **TD-Gammon**, developed by Gerald Tesauro in 1992. TD-Gammon employs an artificial neural network which is trained via temporal difference learning by playing backgammon against itself a few million times [1133, 1134]. TD-Gammon managed to play backgammon at a level of a top human backgammon player. TD-Gammon is largely considered to be a crucial predecessor of **deep reinforcement learning** (DRL), another milestone family of algorithms introduced to us in 2015 [773]. It is also important to note that—to the best of our knowledge—a version of TD-Gammon was among the first appearances of a neural network in a commercial video game named *Jellyfish* (Frederik Dahl, 1994) [1136]. After Deep Blue, IBM's next success story was Watson, a software system capable of answering questions addressed in natural language. In 2011, Watson competed on the *Jeopardy!* TV game and won $1 million against former winners of the game [335].

The next board gameplaying AI milestone was reached in 2016 in the game of **Go**. Soon after Chinook and Deep Blue, the game of Go became the new benchmark for game playing AI with a branching factor around 250 and a vast search space many times larger than that of Chess. While playing Go at a human level had been expected sometime in the far future [621], already in 2016 Lee Sedol—a 9-dan professional Go player—lost a five-game match against Google DeepMind's **AlphaGo** software which featured a deep reinforcement learning approach [1040]. In 2017, AlphaGo won a three-game Go match against the world's number 1 ranking player Ke Jie, running on a single computer. With this victory, Go was the last great classic board game where computers have attained super-human performance. While it is possible

to construct classic-style board games that are harder than Go for computers to play, no such games are popular among human players.

But classic board games, with their discrete turn-based mechanics and their full game state visible to both players, are not the only games in town. And there is more to intelligence than what classic board games can challenge. In the last two decades, a research community has therefore grown up around applying AI other types of games, in particular video games. A large part of the research in this community focuses on developing AI for **playing games**—either as effectively as possible, or in the style of humans (or a particular human), or with some other property. An important milestone in video game AI playing (and arguably AI at large) was reached in 2015 when DRL algorithms developed by Google DeepMind learned to play several games from the classic **Atari 2600** video game console on a super-human skill level just from the raw pixel inputs [773]. One of the Atari games that proved to be hard to play well with that approach was *Ms Pac-Man* (Namco, 1982). The game was practically solved in 2017 by the Microsoft Maluuba team using a hybrid reward architecture reinforcement learning technique [1208].

A couple of years later, in 2019, a deep learning-based system from DeepMind named ***AlphaStar*** [1219] managed to defeat top professional players of the real-time strategy game *StarCraft II* (Blizzard Entertainment, 2010). Some core challenges of playing *StarCraft*—in comparison to games like Go—are the lack of an accurate and fast model that can be used for planning (i.e., a forward model), and the game-specific characteristics of non-determinism and imperfect information. *AlphaStar* succeeded in playing the game well via a deep neural network that is trained through combinations of supervised learning and reinforcement learning. Another much celebrated milestone for superhuman AI in games is OpenAI's *Five* AI system [77]. Similarly to *AlphaStar* the algorithm largely trains itself via self-play deep reinforcement learning. ***Five*** was the first AI algorithm to beat the world champion team of *Dota 2* (Valve, 2013), named OG, in a much publicized event in April 2019. More recent milestones for AI methods that learned to play specific games better than most (if not all) humans include the car racing driver *Sophy AI* [1258] that managed to outperform professional players of *Gran Turismo Sport* (Sony Interactive Entertainment, 2017) via deep reinforcement learning, and the *Cicero* agent [331] that reached average human-level play in the board game of *Diplomacy* through a combination of large language models and strategic reasoning. Finally, we also note a number of recent AI human-level long-term planning gameplaying performances in *Minecraft* (Mojang, 2011) through self-supervised and reinforcement learning methods trained on gameplaying videos of the game [46], or alternatively via agents that continuously explore the world and acquire diverse skills as they play [1128, 1130, 1232].

Other uses of AI in video games (as detailed in this book) have come to be very important as well. One of these is **procedural content generation** (PCG). Starting in the early 1980s, certain video games created some of their content algorithmically during runtime, rather than having it designed by humans. Two games that became very influential early on are *Rogue* (Toy and Wichmann, 1980), where dungeons and the placement of creatures and items in them are generated every time a new game starts, and *Elite* (Acornsoft, 1984), which stores a large universe as a set of

random seeds and creates star systems as the game is played. The great promise of games that can generate some of their own content is that you can get more—potentially infinite—content without having to design it by hand. Such generated content can also help you reduce storage space demands among many other potential benefits. The influence of these games can be seen in more recent successes such as *Diablo III* (Blizzard Entertainment, 2012), *No Man's Sky* (Hello Games, 2016) and the Chalice Dungeons of *Bloodborne* (Sony Computer Entertainment, 2015). Even more recent examples of commercial standard games that feature content (terrains and buildings) generation as part of their gameplay include *Bad North* (Raw Fury, 2018) and *Townscaper* (Oskar Stålberg, 2021). With the rise of Large Language Models (LLMs) and Large Multimodal Models (LMMs) we experience a stream of new games that rely on such foundation models for the creation of personalized stories and interactions with non player characters such as *AI Dungeon* (Latitude, 2019) and *AI people* (GoodAI, 2024 [expected]). There has also been recent high-profile research aiming to generate game environments from video [138], in addition to many other advances in PCG described in this book.

It has also become common practice to use AI to analyze games, and **model players** of games. This is becoming increasingly important as game developers need to create games that can appeal to diverse audiences, and increasingly relevant as most games are connected and can "phone home" to the developer's servers. Facebook games such as *FarmVille* (Zynga, 2009) were among the first to benefit from continuous data collection, AI-supported analysis of the data and semi-automatic adaptation of the game. Games such as *Nevermind* (Flying Mollusk, 2016) can track the emotional changes of the player and adapt the game accordingly. Several recent games such as those from the *Halo* series (Xbox Game Studios, 2001–2021) and the *Forza* series (Xbox Game Studios, 2005–2023) are equipped with versions of *TrueSkill* [769], a skill-based ranking system that is able to match players in such multi-player competitive games.

Over the last two decades, research on believable agents in games has opened new horizons in game AI. One way of conceptualizing believability is to make agents that can pass game-based Turing tests. A game Turing test is a variant of the Turing test in which a number of judges must correctly guess whether an observed playing behavior in a game is that of a human or an AI-controlled game bot [450, 1026]. Most notably, two AI-controlled bot entries managed to pass the game Turing test in *Unreal Tournament 2004* (Epic Games, 2004) on Turing's centenary in 2012 [1004]. Another more recent way to boost the believability of game agents is to employ LLM architectures that would control the memories, daily plans and reflections of non player characters as in the *Smallville* social simulation environment [851].

In the next section we will outline the parallel developments in both academia and industry and conclude the historical section on game AI with ways the two communities managed to exchange practices and transfer knowledge for a common two-fold goal: the advancement of AI and the improvement of games at large.

1.2.1 AI and Games Research in Academia

In **academic** game AI we distinguish two main domains and corresponding research activity: board games and video games. In what follows, we outline the two domains in a chronological sequence, even though game AI research is highly active in both of them.

Early Days on the Board

When it comes to game AI research, classic board games such as Chess, Checkers, Backgammon, and Go are clearly beneficial to work with as they are very simple to model in code and can be simulated extremely fast—one can easily make millions of moves per second on a modern computer—which is indispensable for many AI techniques. Also, board games seem to require thinking to play well, as reflected in the saying "a minute to learn, but a lifetime to master". Games indeed have a lot to do with learning, and good games are able to constantly teach us more about how to play them. To some extent, the fun in playing a game consists in learning it; when there is nothing more to learn we largely stop enjoying them [581]. This suggests that better-designed games are also better benchmarks for artificial intelligence. As mentioned earlier in this section, board games were the dominant domain for AI research from the early 1950s until quite recently. As we will see in the other parts of this book—notably in Part II —board games remain a popular game AI research domain even though the arrival of video games in the 1980s has shifted a large part of the focus since then.

The Digital Era

To the best of our knowledge, the first *video* game conference occurred at Harvard's Graduate School of Education in 1983 [156]. The core focus of the conference was on the educational benefits and positive social impact of video game playing. The birth date of the video game AI field can be considered to be 2001. The seminal article by Laird and van Lent [600], emphasizing the role of games as the **killer application** for AI, established the foundations of game AI and inspired early work in the field [351, 397, 499, 719, 794, 1142, 1144, 1161, 1271]. Beyond gameplaying agents, in those early days AI in video games was mainly concerned with pathfinding [1096] and agent architectures for NPC behavior [191, 669], sometimes within interactive drama [185, 666, 684, 718, 803]. Early work in these areas was presented primarily in the AAAI Spring Symposia on AI and Interactive Entertainment preceding the AIIDE (which started 2005) and the IEEE CIG (also started in 2005) conferences. Most of the early work in the game AI field was conducted by researchers with AI, optimization and control backgrounds and interests in adaptive behavior, robotics and multi-agent systems. AI academics used the best of their computational intelligence and AI tools to enhance NPC behavior in generally simple, research-focused, non-scalable projects of low commercial value and perspective.

The AI and games field has been largely diversified in terms of topics, methods and AI uses in games since its early days. Game AI researchers have grown from a small group of people to a community that is represented by several large-scale conferences that are centrally (e.g., IEEE CoG, AIIDE) or peripherally associated through AI (e.g., NeurIPS, IJCAI and ICML among many) and games research (e.g., Foundations of Digital Games and CHI Play among others). In 2009 the community launched its own IEEE journal, namely IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES—which was renamed IEEE TRANSACTIONS ON GAMES in 2018. We chose to dedicate several parts of this book to the core achievements of this community. Given the research focus of the remaining chapters of this book, the following section offers a more detailed presentation of the history of AI within the games industry.

1.2.2 Game AI Developments in Industry

AI in the games industry has progressed among a largely separate path from AI research in academia. Many techniques used in video games can appear simplistic and even outdated to academic AI researchers, but are in fact finely honed to specific use cases in game development. Throughout history there are many exceptions to this rule, and we are finally seeing signs that the divide between academia and games industry in terms of AI is beginning to close.

One way to cover the 40+ years of industrial game AI is to highlight seminal games with regards to their AI in a chronological order; decade by decade. The first book of Steve Rabin [907] is inspirational in that sense as it includes such a list of seminal games some twenty years ago. Our continuous observations of the game AI industry scene since then complement the list of the seminal games we discuss next. It is important to note that this is a *non-exhaustive* list of AI methods and tools that we believe advanced the game AI state-of-practice in the games industry.

The first released video games back in the **70s** included little or nothing that we would call artificial intelligence these days; NPC behaviors were scripted or relied on simple rules, partly because of the primitive state of AI research, but perhaps even more because of the primitive hardware of the time.

In the early **80s** we witnessed an AI system that fundamentally changed the way we design games up until today. That was the AI controller of the Ghosts (player opponents) in *Pac-Man* (Namco, 1980). As the game's creator Toru Iwatani put it in a recent interview: "*A way to satisfy game players and offer fun is though AI. If a player's skill can be judged by the game, the difficulty level can be dynamically adaptive to the player. I called this Level Control System, in Pac-Man.*" [500]. The game featured four core AI elements that made it quite challenging (to the majority of human players) despite being almost entirely deterministic: (1) ghosts would follow two different play modes (i.e., attack vs. don't care), (2) they would each have their own personality and would cooperate in an indirect fashion, (3) they would follow a pre-determined spawning strategy, and finally (4) they would follow a different speed

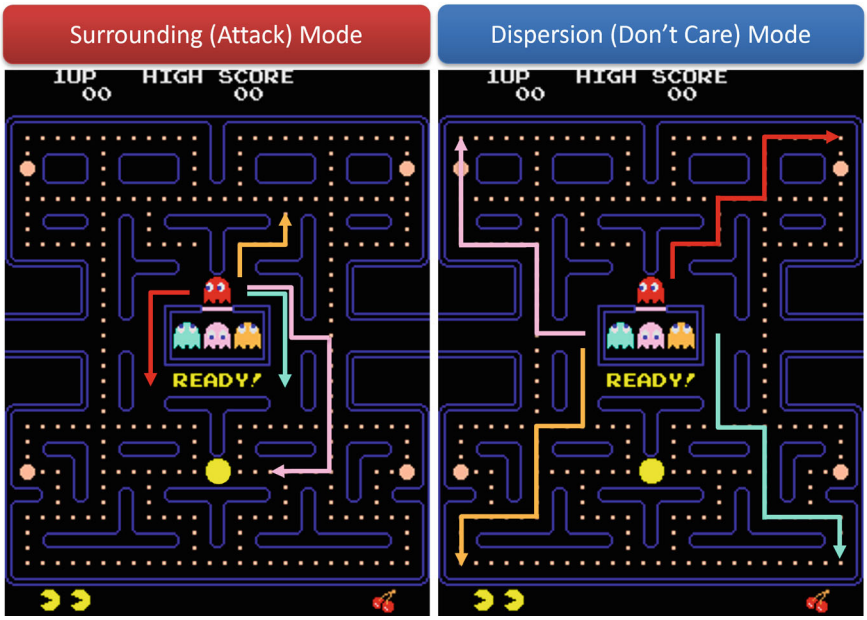


Fig. 1.1 Screenshots from *Pac-Man* (Namco, 1980), the seminal game that introduced adaptive tension levels and opponent tactics based on dissimilar ghost personalities. Ghosts alternate behaviors between two modes as detailed in [500]: the *surrounding* (attack) mode and the *dispersion* (don't care) mode. In the *surrounding* mode (see image on the left), *Blinky* (red ghost) moves towards Pac-Man, *Pinky* (pink ghost) stays 3 cells in front of Pac-Man, *Inky* (cyan ghost) takes a symmetrical position to that of *Blinky* in relation to Pac-Man's position, and *Clyde* (orange ghost) behaves as *Blinky* within 130 dots (pixels) from Pac-Man otherwise it moves randomly. In the *dispersion* mode instead (see image on the right), *Blinky*, *Pinky*, *Inky* and *Clyde* move, respectively, to the upper right, upper left, bottom right, and bottom left corner of the level. Similar adaptive tension methods can be found in games such as *Left 4 Dead* (Valve, 2008), nearly 30 years after the release of *Pac-Man* (Namco, 1980). Images inspired by [500] and repurposed with permission. Original in-game screenshot image produced by authors

pattern according to the level played [500]. Figure 1.1 illustrates the two different play models used in *Pac-Man* (Namco, 1980) as presented by Iwatani in [500]. Beyond the adaptive capacities of *Pac-Man* (Namco, 1980) opponents, the decade includes innovations in procedural content (level) generation in the games of *Rogue* (Toy and Wichmann, 1980) and *Elite* (Acornsoft, 1984) as detailed earlier.

During the **90s** we note an increasing number of games that feature intelligent agent control and some early steps towards adopting machine learning methods. Specifically, we meet the first popular application of neural networks in *Creatures* (Millennium Interactive, 1996) with the aim to model the creatures' behavior. We argue, however, that the first application of a neural network in a commercial game is the *Jellyfish* (Frederik Dahl, 1994) backgammon game published two years earlier. *Thief* (Eidos 1998) featured the first advanced sensory system of opponent agents

(guards) while the behavior-based AI of *Blade Runner* (Virgin Interactive, 1997) and the advanced opponent tactics in *Half-Life* (Valve, 1998) define two more representative examples of opponent AI in that decade. The *Civilization* game series (MicroProse, Activision, Infogrames Entertainment, SA and 2K Games, 1991–2016)—kick-starting in the 90s—offered milestone games in terms of procedural world generation.

During the early days of academic video game AI (early 00s), the game industry gradually started making steps towards integrating more sophisticated AI in games [191, 1255]. The **00s** welcomed arguably a milestone game in terms of AI innovation and in-game integration. *Black and White* (EA, 2000) manages to successfully fuse a number of different **machine learning** methods such as perceptrons, decision trees and reinforcement learning to control the behavior of an avatar creature that the player interacts with (see Fig. 1.2). The belief "desire" intention cognitive model [118] accommodates all learning and decision making processes of the creature. *Black and White* was just the beginning of a very productive and AI-friendly decade in the games industry. The design of believable agents in *The Sims* series (Electronic Arts, 2000–2024) and the team tactics and believable combat scenes in the *Halo* series (Microsoft Studios, 2011–2017) are also milestone moments for game AI in the 00s. *Halo 2* (Microsoft Game Studios, 2004) in particular introduced behavior trees for agent control and popularized their use in games since then,

Fig. 1.2 A screenshot from *Black and White* (EA, 2000), a highlight in game artificial intelligence history that successfully integrated several AI methods into its design. The game features a creature that learns through positive rewards and penalties in a reinforcement learning fashion. Further, the creature employs the belief-desire-intention (BDI) model [375] for its decision making process during the game. The desires of the creature about particular goals are modeled via simple perceptrons. For each desire, the creature selects the belief that it has formed the best opinion about; opinions, in turn, are represented by decision trees. In this in-game screenshot produced by authors the creature just learned how to perform *Miraculous Water*

overthrowing the dominant (hierarchical) finite state machines. The *Drivatar* feature that was introduced in *Forza Motorsport* (MS Game Studios, 2005) is arguably the longest-standing AI system in any commercial-standard video game. Editions of *Drivatar* have been shipped with all games of the *Forza* series (Xbox Game Studios, 2005-2023). The *Drivatar* imitates the ways you and your friends play and it can therefore assist you race better, or race even when you (or your friends) are offline. Throughout the long history of *Drivatar* a palette of AI methods have been employed to imitate human racing behavior: from simple rule-based systems in early days, to sophisticated deep learning methods nowadays. *F.E.A.R.* (Sierra Entertainment, 2005) [833] and *Left 4 Dead* (Valve, 2008) are two more games of that decade that innovated in terms of AI. *F.E.A.R.* generated context-sensitive agent behaviors via Goal Oriented Action Planning [832]—a simplified STRIPS-like planning method—which was specifically designed for that game. *Left 4 Dead* introduced the concept of the **AI director**, a hierarchical controller that considers the current tension of the game and adapts opponent and content placement to satisfy desired levels of tension for the player (see Fig. 1.3). The tension curves adopted after extensive in-house experimentation in Valve [107], unsurprisingly, appear to follow the patterns of the *attack-don't care* waves introduced in *Pac-Man* (Namco, 1980) almost three decades earlier.

The next decade saw games incorporating a wider ranger of methods, including some coming from academic AI research. For some games, these methods are placed more centrally both in the development pipeline and the marketing campaigns of the publishers. Indicative examples of important AI features in games include the

Fig. 1.3 A screenshot from *Left 4 Dead* (Valve, 2008), a milestone in AI-based game balancing. The AI director of the game spawns opponents and places items across the level so as to match a particular tension curve for the player. Game screenshot produced by the authors

realistic gunfights of *Red Dead Redemption* (Rockstar Games, 2010), the personality-based adaptation in *Silent Hill: Shattered Memories* (Konami, 2010), the affect-based cinematographic representation of multiple cameras in *Heavy Rain* (Quantic Dream, 2010), the neuroevolutionary training of platoons in *Supreme Commander 2* (Square Enix, 2010), the buddy AI (named Ellie) in *The Last of Us* (Sony Computer Entertainment, 2013), the companion character, Elizabeth, in *BioShock Infinite* (2K Games, 2013), the interactive narratives of *Blood & Laurels* (Emily Short, 2014), the alien's adaptive behavior which adjusts its hunting strategy according to the player in *Alien: Isolation* (Sega, 2014), and the antagonist named *Theodore Peterson* that learns patterns of a player's behavior and adapts to their gameplay in *Hello Neighbor* (tinyBuild, 2017). When it comes to procedural content generation applications the generated levels of *Minecraft* (Mojang Studios, 2011), *Spelunky* (Mossmouth, LLC, 2013) and the vast universes of *No Man's Sky* (Hello Games, 2016) define milestone applications of the era. The decade ends with the first integration of a **large language model** into a game: the LLM-powered text adventure game *AI Dungeon* (Latitude, 2019).

At the time of writing the second edition of this book, the early (**20s**), we experience an ongoing effort and significant investment for the adoption of state of the art **generative AI** methods—such as LLMs and LMMs—and variants of deep (reinforcement) learning in commercial-standard games. Such methods are used off-the-shelf for the creation of content (such as visuals and audio), for testing and improving the design of games, or traditionally for controlling gameplaying agents in games. Noteworthy recent examples of successful AI adoption include the use of the **wave function collapse** generative algorithm for enhancing a player's city building experience in *Townscaper* (Oskar Stålberg, 2021) and the integration of the deep RL controlled racer Sophy AI [1258] in *Gran Turismo 7* (Sony Interactive Entertainment, 2022). Beyond the integration of AI in individual games we also witness an increasing number of AI companies that build specialized AI tools for game development (e.g., modl.ai[3] or traditional video game companies hosting AI research departments (e.g., EA SEED[4]) with the aim to advance the state-of-practice of AI in the industry.

Historically, the key criterion that distinguishes successful AI in commercial-standard games had always been the level of integration and interweaving of AI in the design of the game [907, 999]. In other words, the AI must satisfy a game design need, or the game design built around the AI. Unfortunately, it's not uncommon to see AI methods inelegantly grafted onto game designs they are unsuited for. An unsuccessful coupling of game design and AI may lead to unjustifiable NPC behaviors, broken interfaces, unwanted content or other problems, reducing player immersion. A typical example of such a mismatch between AI and design is the broken navigation of bots that get stuck in a level's dead end; in such instances either the level design is not (re)considered appropriately to match the design of the AI algorithm or the AI algorithm is not tested sufficiently, or both. On the other hand,

[3] https://modl.ai/.

[4] https://www.ea.com/seed.

the successful integration of AI in the design process is likely to lead to satisfactory outcomes for the playing experience. The character design process, for instance, may consider the limitations of the AI and, in turn, absorb potential "catastrophic" failures of it. An example of such an interwoven process is the character design in *Façade* (Procedural Arts, 2005) [721] that was driven, in part, by the limitations of the natural language processing and the interactive narrative components of the game.

It is important to note that this book is not only about game AI as defined and practiced in the game industry. Instead, it is primarily an academic textbook that refers to some of the techniques that have been used in and popularized through the game industry—see for instance the *ad-hoc behavior authoring* section of Chap. 3. The reader with an interest in the AI state of practice in the game industry is referred to the several introductory articles (e.g., [283, 285, 623]) available in books such as the *game AI programming wisdom* series [907–910]. Another valuable resource is the recorded talks from top game AI developers which are hosted at the AI summit[5] of the Game Developers Conference (GDC) and are available through the GDC Vault.[6]

1.2.3 The "Gap" Between Academia and Industry

During the first decade of academic game AI research (roughly the 00s), when researchers from academia and developers from industry would meet and discuss their respective work, they would arrive at the conclusion that there exists a *gap* between them. The gap had multiple facets such as differences in background knowledge, practice, trends, and state-of-the-art solutions to important problems. Academics and practitioners would discuss ways to bridge that gap for their mutual benefit on a frequent basis [191] but that debate would persist for many years as developments on both ends were slow. The key message from academic AI was that the game industry should adopt a "high risk-high gain" business model and attempt to use sophisticated AI techniques with high potential in their games. The main response from commercial game developers was that their games were designed to work with tried-and-true classic AI methods, and designing games to work with unreliable modern AI methods was too risky. Interestingly, simple multi-layer perceptrons and decision trees were such sophisticated techniques 20 years ago.

On the other end, the central concern of industrial game AI regarding game AI academia has been the lack of domain-specific knowledge and practical wisdom when it comes to realistic problems and challenges faced during game production. Perhaps above all there was a difference in what is valued: academics valued new algorithms and new uses of algorithms that achieve superior performance or create new phenomena or experiences, and AI developers in industry valued software architectures and algorithmic modifications that reliably support specific game designs

[5] http://www.gdconf.com/conference/ai.html.

[6] http://www.gdcvault.com/.

and technologies. But what happened since then? Does this gap really exist nowadays or is it merely a ghost from the past?

It is still true that the academic game AI research community and the game industry AI development community work on different problems, using different methods. There are also some topics and methods explored by the academic community which are generally very unpopular within the game industry. Real-time adaptation and learning in NPCs is one such example; a substantial amount of academic research we cover in this book focuses on NPCs that can learn and develop from their interactions with the player and other NPCs in the game. However, AI developers in industry point out that it would be very hard to predict what these NPCs will learn, and it is very likely to "break the game" in the sense that it no longer works as designed. Conversely, there are methods and problems explored in industry which most academics do not care about, as they only make sense within the complex software architecture of a complete game.

When thinking about the use of AI within modern video games, it is important to remember that most game genres have evolved from earlier game designs. For example, the first platformers were released in the mid-1980s and the first first-person shooters and real-time strategy games in the early 1990s. At that time, the ability to build and deploy advanced AI was not even possible, so designers had to design around the lack of AI. Some of these basic design patterns have largely been inherited by today's games; many games have been designed not to need AI. For the academic who wants to build interesting AI for an in-game role, the best way forward might therefore be to create new game designs that start from the existence of modern AI methods.

On the positive side we would argue that any existing gap between academic and industrial NPC AI nowadays can be viewed as a healthy indication of a parallel progress with a certain degree of collaboration. As industry and academia do not necessarily attempt to solve the same problems with the same approaches, it is possible that NPC AI or generative AI solutions emerging from industry can inspire new approaches in academia and vice versa. In summary, the NPC AI gap is clearly smaller in tasks that both academia and industry care about. Certain aspects of NPC AI, however, are far from being solved in an ideal fashion and others—such as modeling player and NPC emotion—are still at the beginning stages of investigation. So while we can praise the NPC AI of *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011) or *Red Dead Redemption 2* (Rockstar Games, 2018) they each clearly have flaws, such as companion behavior. We can view such limitations as an opportunity that can bring industry and academia even closer to work on further improving existing NPC AI in games.

Another strong connecting link between academia and industry is the player base of a game. This is essentially the community surrounding a particular videogame which tends to influence the game's overall development through game modifications (*mods*). We argue that one of the most powerful ways of inspiring the game industry is to allow its players to modify and improve their games. In several such occasions the work of the **modding** community has even resulted in video game companies hiring members of such communities, such as *Valve* hiring Minh Le and Jess Cliffe, creators

of the original *Counter-Strike* game that was initially released as a mod of *Half-Life* (Valve, 1998) [217]. From an academic point of view, game modding provides a relatively accessible playground to explore and test new AI methods that might have a more immediate impact to the game industry. It is now easier than ever to mod or customize an already released videogame such as *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011), *Fallout* (Interplay Productions, 1997), *No Man's Sky* (Hello Games, 2016), and various games on the *Roblox* (Roblox Corporation, 2006) platform, to name a few. All of these games have vibrant modding communities and an active player base that is willing to playtest new AI ideas. In a recent example of game modding stemming from academia [413], *The Elder Scrolls V: Skyrim* was modified to employ seven different NPCs based on the *Comme il Faut* [731] social AI tool that was introduced as part of *Prom Week* [729] social simulation game. The emergent NPCs were able to create new dynamic social interactions for the players of this game. *Minecraft* (Mojang, 2011) is a particularly good example here, as it is not only one of the world's most popular games after more than a decade, but also a widely used academic research platform.

A different take on this discussion is that NPC AI has been almost solved for most production tasks; a number of studies in the early days of game AI [795, 1108] took it one step further and claimed that game AI research and development should focus solely on non-traditional uses of AI. The level of AI sophistication even in games of the previous decade(s) such as *Left 4 Dead* (Valve, 2008) and *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011) supports this argument and suggests that advances in NPC AI have reached satisfactory levels for many NPC control challenges faced during game production. Recent years, however, have seen a shift of academic and industrial interests with respect to game AI. That is due to the rise of robust and effective industrial game AI solutions, the convergence to satisfying NPC behaviors, and a more pragmatic and holistic view of the game AI challenges we face. It seems that we have reached an era where the primary focus of the application of AI in games is not on agents and NPC behaviors. At least a decade ago, the focus started to shift towards interweaving game design and game technology by viewing the role of AI holistically. AI operates in games as a content generator, a testing tool, or alternatively as a modeling process of players [1268].

The view we take in this book is that AI can help us to make better games but that won't necessarily happen through better, more human-like or believable NPCs [1268]. Notable examples of non-NPC AI that we covered earlier in this section include *No Man's Sky* (Hello Games, 2016) and its procedural generation of a quintillion different planets, *Nevermind* (Flying Mollusk, 2016) with its affective-based game adaptation via a multitude of physiological sensors and the more recent LLM-powered *AI Dungeon* (Latitude, 2019) player-adaptive text-adventure game. But there might be other AI roles with game design and game development that are still to be discovered. Beyond playing games, modeling players or generating content, AI might be able to play the role of a design assistant, a data analyst, a playtester, a game critic, or even a game director. Finally, AI could potentially play and design games as well as model their players in a general fashion. Chapter 14 in the last part of this book is dedicated to these frontier research areas for game AI.

1.3 Why Use Games for Artificial Intelligence?

There are a number of reasons why games offer the ideal domain for the study of artificial intelligence. In this section, we list the most important of them in our view.

1.3.1 Games Are Hard and Interesting Problems

Games are engaging due to the effort and skills required from people to complete them or, in the case of puzzles, solve them. It is that *complexity* and *interestingness* of games as a domain that makes them desirable for AI. Games are *hard* because their state spaces, such as the possible strategies for an agent, are often vast. Their complexity as a domain rises as their vast search spaces often feature small feasible spaces (solution spaces). Further, the goodness of any game state is very often hard (or even impossible) to assess properly.

From a computational complexity perspective, many games are **NP-hard** (NP refers to "nondeterministic polynomial time"), meaning that the worst-case complexity of "solving" them is very high. In other words, in the general case an algorithm for solving a particular game could run for a very long time. Depending on a game's properties complexity can vary substantially. Nevertheless, the list of games that are NP-hard is rather long and includes games such as the two-player incomplete information *Mastermind* game [1091, 1203], *Lemmings* (Psygnosis, 1991) [555] and *Minesweeper* (Microsoft, 1992) [552]. It should be noted that this computational complexity characterization has little to do with how hard the games are to play for humans, and does not necessarily say much about how well heuristic AI methods can play them. It is clear, however, that at least in theory, and for arbitrary-size instances, many games are very hard.

The investigations of AI capacity in playing games that are hard and complex has been benchmarked via a number of milestone games. As mentioned earlier, Chess and (to a lesser degree) Checkers have traditionally been seen as the "drosophila for AI research" even from the early days of AI [727]. After the success of Deep Blue and Chinook in these two games the AI community gradually invented and cited other more complex games as AI "drosophilae", or universal benchmarks. *Lemmings* has been characterized as such; according to McCarthy [728] it "connects logical formalizations with information that is incompletely formalizable in practice". In practice, games for which better APIs have been developed—such as *Super Mario Bros* (Nintendo, 1985),[7] *StarCraft* (Blizzard Entertainment, 1998) and *Dota 2* (Valve, 2013)—have become more popular benchmarks.

The game of computer Go has also been another core and traditional game AI benchmark with decades of active research. As a measure of problem complexity a

[7] Note that the original game title contains a dot, i.e., *Super Mario Bros.*; for practical reasons, however, we will omit the dot when referring to the game in the remainder of this book.

typical game in Go has about 10^{170} states. The first AI feature extraction investigations in Go seem to date back to the 1970s [1319]. The game received a lot of research attention during several computer Go world championships up until the recent success of AlphaGo [1040]. As covered already in Sect. 1.2, in 2016 AlphaGo managed to beat two of the best Go professional human players using a combination of deep learning and Monte Carlo tree search.

The *StarCraft* (Blizzard Entertainment, 1998)—and its successor *StarCraft II* (Blizzard Entertainment, 2010)—real-time strategy game can be viewed as one of the hardest games for computers to play well. When the first edition of this book came out in 2018, the best *StarCraft* bots could only reach the level of amateur players [188, 212]. The complexity of the game derives mainly from the multi-objective task of controlling multiple and dissimilar units in a game environment with partial information. While it is not trivial to approximate the state space of *StarCraft* a typical game has at least $10^{1,685}$ possible states [1197]. In comparison, the number of protons in the observable universe is only about 10^{80} [305]. The number of *StarCraft*'s possible states sounds huge but, interestingly enough, its search space can be of manageable size if represented by bytes. On that basis, we require about 700 bytes of information to represent the *StarCraft* search space whereas the number of protons in the known universe is equivalent to the number of configurations of about 34 bytes. One could of course design games that are harder on purpose, but there is no guarantee anyone would want to play those games. When doing AI research, working on games that people care about means you are working on relevant problems. This is because games are designed to challenge the human brain and successful games are typically good at this. *StarCraft* and *StarCraft II* are played by millions of people all over the world, with a very active competition scene of elite professional players. Because of its high complexity as a game, AI academia has already offered a rich body of algorithms for playing (parts of) *StarCraft* [214, 829, 830, 952], or generating maps for it [1166], over the last decade or more.

While writing the first edition of this book, industrial AI leader Google DeepMind announced a major new project to develop a deep learning-based system for playing *StarCraft II*, after their success at training deep networks to play Atari games in the Arcade Learning Environment (ALE) [75] framework. At the same time, DeepMind in collaboration with Blizzard Entertainment opened up *StarCraft II* to AI researchers for testing their algorithms. Meanwhile Facebook AI Research led the development of *TorchCraft* [1119]—a bridge between the deep learning Torch library and *StarCraft*—and published their first paper on using machine learning to learn to play *StarCraft* [1197], showing that they take this challenge seriously. Another industrial game AI research lab collaborating with academia on solving *StarCraft* of that era was hosted by Alibaba [862]. Back in 2018 we were arguing that it is unlikely we will conquer all of this game soon [396] but we also expected to see some AI breakthroughs in the years to come. In 2019, DeepMind's *AlphaStar* system [1219] was unveiled, and could defeat professional players of *StarCraft II* (Blizzard Entertainment, 2010) through multi-agent deep reinforcement learning. *AlphaStar* was initially seeded by neural network models that were trained to imitate those professional players, and then trained by playing against versions of itself. Soon after this

milestone for AI, game AI researchers turned to games that offer alternative playing challenges. Indicatively, the board game *Diplomacy* (Wizards of the Coast, 1959) is very hard to play (for humans and AI) as it is both an imperfect information game and it requires complicated competitive and collaborative negotiation skills from its players. Very recently, the *Cicero* agent [331] combined methods from strategic reasoning and LLMs to eventually reach average human-level play in that game.

While writing this book we observe a trend for the development of more generalized approaches for AI gameplaying agents. In other words, we note a growing interest and body of work towards **artificial general intelligence** [1154] for gameplaying agents. Indicative examples include the generalist gameplaying agent named *Gato* [930] that learns to play simple games with unseen sets of rules in a generalized fashion and the multi-game decision transformer architecture that learned to play a number of different Atari games using a shared underlying model [615].

1.3.2 Rich Human-Computer Interaction

Computer games are dynamic media by definition and, arguably, offer one of the **richest** forms of human-computer interaction (HCI). The richness of interaction is defined in terms of the available options a player has at any given moment and the ways (modalities) a player can interact with the medium. The available options for the player are linked to the game action space and the complexity associated with it in games such as *StarCraft II* and *Diplomacy*. Further, the modalities one may use to interact with games nowadays extend beyond the traditional keyboard, mouse and mobile (smartphone, smartwatch, tablet) haptics, to game controllers, physiology such as heart rate variability, eye gaze, body movement, text, and speech.

As we will see later in this book, games offer one of the best and most meaningful domains for the realization of the **affective loop** [1107]. The affective loop is a popular framework that defines the basic phases of effective affect-based interaction. The loop is closed successfully when an HCI system is able to successfully elicit, detect and respond or adapt to the cognitive, behavioral and emotive patterns of its user [1286]. One would argue that any adaptive game that considers aspects of player experience—such as the AI director of *Left 4 Dead* (Valve, 2008) or even the adaptive antagonist in *Hello Neighbor* (tinyBuild, 2017)—closes the affective loop in a successful fashion.

The emotional patterns a player goes through while playing a game are deeply interwoven in the design of any game. It is that central role of emotion-based interaction that makes games an ideal testbed for the study of HCI. Games have the potential to influence players mainly due to their ability to place the player in a continuous mode of interaction with the game which, in turn, elicits complex cognitive, affective and behavioral responses to the player. This continuous relationship is enriched by fast-paced and **multimodal** forms of user interfacing that are often possible in games. Such game-player interactions offer AI algorithms access to rich experience

stimuli and emotional manifestations from the player. Data of that type, however, pose challenges to machine learning and data science, given their subjective nature and its complex spatiotemporal characteristics [1286].

The study of players not only advances our knowledge about human behavior and emotion but also contributes to the design of better human-computer interaction. Games, however, are not merely an important domain for AI and HCI. As a matter of fact, games have shaped and advanced HCI in numerous ways given the unique challenges they pose and the opportunities they bring to AI as we outlined earlier. We argue that games significantly push the boundaries of what AI is capable of accomplishing in order to address the challenges of game-based interaction. Importantly, AI is challenged by the subjective nature of human experience, the idiosyncratic processes of game design, and the emergent human behaviors of playing alone or playing against (or with) others.

1.3.3 Games Are Popular

While video games, back in the 1980s, were introduced as a niche activity for those having access to a video arcade or to consoles such as Atari 2600, they gradually turned into a multi-billion industry. Video games generated, in the 2010s, a global market revenue higher than any other form of creative industry, including film and music. At the time of writing this book over a third of the Earth's population plays games; the projected number of gamers is expected to rise up to 3.3 billion by the end of 2024 [215]. This includes people of probably all ethnicities, nationalities, genders, ages, and occupations, in striking difference to the allegedly more homogeneous game-playing population of yore. Based on these projections we could easily argue that game playing at this massive scale is probably the largest ongoing experiment of human behavior and experience. Meanwhile the video game industry is thriving. Video games currently generate a worldwide total of almost $250 billion in revenue which is expected to rise to approximately $360 billion by 2027 (i.e., an annual growth rate of 9.83%) [808, 1083].

But why did games become so popular? Beyond the obvious argument of games being able to enhance a user's intrinsic motivation and engagement by offering interactivity capacities with a virtual environment, it was the technological advancements over the last 40 years that drastically changed the demographics of players [529]. Back in the early 1980s games were played solely in arcade entertainment machines; nowadays, however, they can be played using a multitude of devices including a PC, a mobile phone, a tablet, a handheld device, a virtual reality device, or a console, and obviously still an arcade entertainment machine! Beyond the technological advancements that fostered accessibility and democratized gameplay, it is also the culture that follows a new medium and develops it into a new form of art and expression.

The independent scene of game design and development[8] has contributed vastly to this culture [530] as games gradually found multitude of purposes and objectives beyond mere entertainment: games for art, games as art, games for a change, games as thought experiments and philosophical arguments, physical interactive games, games for education, games for training and health, games for scientific discovery, and games for cultural heritage. In brief, not only are games everywhere and present in our daily lives, they also shape our social and cultural values at large—as evidenced by the massive success of *Pokémon Go* (Niantic, 2016) which got people of all ages to go out and explore the places they lived in from a new perspective. As a byproduct of their popularity games attract people with world-class performance in the domain. World-ranked experts (i.e., professional players) of many board and video games have participated regularly in competitions against AI algorithms; examples include Garry Kasparov (Chess), Lee Sedol and Ke Jie (Go), the OG team (Dota 2), and TLO and MaNa (StarCraft).

As games become more popular, grow in quantity, and become more complex, new AI solutions are constantly required to meet the new technological challenges. This is where AI meets a domain with a strong industrial backing and a desire to support sophisticated technology for bettering player experience. Furthermore, very few domains for AI offer the privilege of daily accessibility to new **content** and **data** from their popular use. But let us look at these two aspects in more detail below.

Popular Means More Content

The more people play (more) games, the more content is required for games. Content takes effort to create but, over the years, mechanisms have been developed that allow both machines and players to design and create various forms of content in games. Games have gradually developed to be **content-intensive** software applications that demand content that is both of direct use in the game and of sufficient novelty. The overwhelming demand for new and novel gaming experiences from a massive community of users constantly pushes the boundaries of human and computational creativity to new grounds.

Content in games, beyond any other form of multimedia or software application, not only covers all possible forms of digital media content such as audio, video, image, and text but it also comes in massive numbers of different resolutions and representations. Any algorithm that attempts to retrieve and process the variety and amount of content within or across games is directly faced with the challenges of interoperability and content convergence as well as scalability caused by such big data. Given sufficient compute, however, modern generative AI methods such as LMMs can scale at that level with unprecedented ability to fuse and process multiple modalities such as text, images and sounds. Several attempts are currently made to fine tune and process existing pre-trained models for representation learning [925, 1187] or content creation [365, 1100, 1146].

[8] http://www.igf.com/.

Popular Means More Data

Massive content creation (either by algorithms or players) is one major effect of games' popularity; the other is massive data generation of game playthroughs and player behavior. Since the late 2000s, game companies have had access to accurate game **telemetry** services that allow them to track and monitor player purchases, churn and re-engagement, or the progress of play for debugging the game. The algorithmic challenges here follow the general challenges of big data and big data mining research [726], which include data filtering during data acquisition, metadata generation, information extraction from erroneous and missing data, automatic data analysis across dissimilar datasets, appropriate declarative query and mining interfaces, scalable mining algorithms, and data visualization [598]. Luckily enough some of these datasets are nowadays openly available for **game analytics** and game data mining research. Indicatively, in 2017, the OpenDota project[9]—a community-maintained open source Dota 2 data platform—released a sanitized archive of over a billion matches of *Dota 2* (Valve Corporation, 2013) that were played between March 2011 and March 2016.[10]

1.3.4 There Are Challenges for All AI Areas

Unlike other benchmarks, games challenge *all* core areas of AI. This can be verified by considering a number of broadly accepted areas of AI and discussing the challenges available for those areas in games. **Signal processing**, for starters, meets great challenges in games. Data from players, for instance, not only come in different resolutions (in-game events vs. head pose vs. the player's physiology) they also originate from multiple modalities of fast-paced interaction in an environment that elicits complex cognitive and affective patterns from the player. Multimodal interaction and multimodal fusion are non-trivial challenges when building embodied conversational agents and virtual characters. Importantly, the complexity of signal processing in games is augmented due to the spatiotemporal nature of the rich and fast-paced interaction between the player(s) and the game.

As discussed in Sect. 1.2 a number of board and video games define core milestones for **machine learning** (Go, arcade games, *Diplomacy*, *Dota 2*, *Starcraft* and *Minecraft*), **tree search** (Checkers and Chess), **knowledge representation and reasoning** (*Jeopardy!*) and **natural language processing** (*Jeopardy!*, *Diplomacy*) and, in turn, they resulted in major breakthroughs for AI. This historical association between AI accomplishments and games already provides clear evidence that all the aforementioned areas have traditionally been challenged by games. While the full potential of machine learning has been thoroughly explored in milestone games

[9] https://www.opendota.com/.

[10] https://blog.opendota.com/2017/03/24/datadump2/.

such as *StarCraft II* (Blizzard Entertainment, 2010) [1219], natural language processing (NLP) in the era of LLMs is still being challenged deeply through games involving narrative and natural language input. NLP is challenged particularly in game environments that wish to realize forms of interactive storytelling [248] or text-based adventures—e.g., in *Zork I* (Infocom, 1980) or *AI Dungeon* (Latitude, 2019). Particularly, the perception, processing, and evaluation of human feedback remains a grand challenge for text generation methods [989].

Finally when it comes to **planning** and **navigation**, games have traditionally offered environments of high and increasing complexity for algorithms to be tested in. Games such as *StarCraft* and *Minecraft* define major milestones for planning algorithms and are thus used extensively as testbeds, including for modern approaches to planning such as LLM-based agents [46, 1232]. Navigation and pathfinding have also reached a certain degree of maturity through simulated and photorealistic game environments featuring multiple entities (agents). An additional benefit of games as a domain for behavior planning is that they offer a realistic yet convenient and cheaper testbed compared to robotics. Beyond the extensive testing and advancement of variants of A* through games, popular and highly effective tree search variants such as Monte Carlo tree search [137] have been invented in response to problems posed by gameplaying.

1.3.5 Games Best Realize Long-term Goals of AI

One of the long-standing questions that AI is faced with is *what is the ultimate long-term goal for AI*? While numerous debates and books have been dedicated to this topic, the collaborative effort of Wikipedia authors addressing this question[11] reveals the areas of **social intelligence** and **affective interaction**, (computational) **creativity**, and **general intelligence** as a few of the most critical long-term goals of AI. Our reference can been critiqued for systemic bias (in any controversial question such as the one above); we argue, however, that any reference on this topic would be subjective anyhow. Without aiming to be exclusive or biased, we believe that the three aforementioned areas collectively contribute to better AI systems and we discuss why games best realize these three goals next. These three long-term goals define frontier research areas for game AI and are further elaborated on in Chap. 14.

Social and Emotional Intelligence

Affective computing [877] is the multidisciplinary area of study across computer science, cognitive science and psychology that investigates the design and development of intelligent software that is able to elicit, detect, model, and express emotion and social intelligence. The ultimate aim of affective computing is the realization of the

[11] Wikipedia (accessed: October 2024): https://en.wikipedia.org/wiki/Artificial_intelligence.

so-called *affective loop* [1107] which, as we covered earlier, defines a system that is able to successfully elicit, detect and respond to the emotions of its user. Naturally, both emotional and social aspects of intelligence are relevant for a system that realizes the affective loop.

Games can offer a highly meaningful realization of the **affective loop** and affective interaction [1286, 1287]. They are by definition both *entertaining* (whether used for pure satisfaction, training or education) and *interactive* activities that are played within fantasy worlds. Thus, any limitations of affective interaction—such as the difficulty to justify affective-based game decisions or alterations of content to the user—are absorbed naturally. For example, an erroneous affective response of a game character can still be justified if the design of the character and the game context does not break the **suspension of disbelief**—during which the player ignores the medium and the interaction for the sake of their enjoyment. Further, games are designed to offer affective experiences which are influenced by players. In turn, players are willing to go through frustrating, anxious, and fearful episodes of play to experience involvement. To that end, a player—likely more than any other user of any other form of HCI—is generally open to alterations of the interaction that influence their emotional state.

Computational Creativity

Computational creativity studies the potential of software to autonomously generate outcomes that can be considered creative or algorithmic processes that are deemed to be creative [99, 1250]. Naturally, computer games can be viewed as the killer application domain for computational creativity [643]. It is not only their unique features that we covered in earlier sections of this chapter—i.e., being highly interactive, dynamic and content-intensive software applications. Most importantly it is their *multifaceted* nature [636]. In particular, it is the fusion of the numerous and highly diverse creative domains—visual art, sound design, graphic design, interaction design, narrative, virtual cinematography, aesthetics and environment beautification—within a single software application that makes games the ideal arena for the study of computational creativity and generative AI. It is also important to note that each art form (or facet) met in games elicits different experiences to its users; their fusion in a final game software targeting a rather large and diverse audience is an additional challenge for computational creativity.

We thus argue that the study of computational creativity *within* and *for* computer games [643] advances both through the field of generative AI and the domain of games. Games can, first, be improved as products via computational creations (*for*) and/or, second, be used as the ultimate canvas for the study of computational creativity as a process (*within*). Computer games not only challenge computational creativity but they also provide a creative sandbox for advancing the field. Finally, games can offer an opportunity for computational creativity methods to be extensively assessed via a huge population of users of commercial-standard products with high impact and financial value.

General Intelligence

While there is much disagreement about exactly what *Artificial General Intelligence* (AGI) [386] is, it remains a preoccupation of and motivating ideology for many AI researchers [1154]. AI has investigated the general intelligence capacity of machines within the domain of games more than any other domain thanks to the ideal properties of games for that purpose: controlled yet interesting and computationally hard problems [998]. In particular, the capacity of AI to play unseen games well—i.e., general game playing—has seen a number of advancements in recent years. Starting with the **general game playing competition** nearly 20 years ago [374], focusing on board games and similar discrete perfect information games, we moved to the Arcade Learning Environment [75] and eventually to the General Video Game AI Competition [645, 867, 870], which offer radically different takes on **arcade video games**. Advancements vary from the efforts to create game description languages suitable for describing games used for general game playing [303, 374, 584, 667, 709, 890, 996, 1138] to the establishment of a set of general video game AI benchmarks [75, 374, 870] to more recent success of deep learning and LLM-based methods in playing arcade games with human-level performance [773, 930]. While general game playing is studied extensively and constitutes one of the key areas of game AI [1291], we argue that the focus of generality solely with regard to the performance of gameplaying agents is *narrow* with respect to the spectrum of roles for general intelligence in games. The types of general intelligence required within game development includes game and level design as well as player behavior and experience modeling. Such skills touch upon a diverse set of cognitive and affective processes which have until now been mostly ignored by general AI in games. For general game AI to be truly general and advance AI algorithmically, it needs to go beyond game playing while retaining the focus on addressing more than a single game or player [1172]. We further argue that the challenge of bringing together different types of skillsets and forms of intelligence within autonomous designers of games cannot only advance our knowledge about human intelligence but also advance the capacity of artificial general intelligence. We detail our views on AGI in games and its multiple facets in the last part of the book (see Chap. 14).

1.4 Why Use Artificial Intelligence for Games?

The various uses of AI in games are beneficial for the design of better games for a number of reasons. In this section we focus on the benefits obtained by allowing AI to play a game, to generate content and to analyze player experience and behavior.

1.4.1 AI Can Play, Test and Improve Games

AI can improve games in several ways by merely playing them. The game industry usually receives praise for the AI of their games—in particular, the non-player or opponent AI. AI may also add to the commercial value of the game, contribute to better game reviews, and enhance the experience of the player. Whether the underlying AI is based on a simple behavior tree, a utility-based AI or alternatively on a sophisticated machine learned reactive controller, or an in-house developed foundation model is of limited relevance as long as it serves the aforementioned purposes. An unconventional and effective solution to an NPC task can often be a critical factor that shapes management, marketing and monetization strategies during and after production.

As we will see in Part II, we can make AI systems that play games with two core objectives in mind: play *well* and/or play for the *experience* (believably, human-like, or interestingly). Further, AI can control either the *player* character or the *non-player* character of the game. AI that plays well as a player character focuses on optimizing the *performance* of play—performance is measured as the degree to which a player meets the objectives of the game solely. Such AI can be of tremendous importance for automatic game testing and for the evaluation of the game design as a whole. AI that plays well as a non-player character, instead, can empower dynamic difficulty adjustment and automatic game balancing mechanisms that will, in turn, personalize and enhance the experience for the player (as in [11, 875, 1075]). If the focus of AI is shifted on controlling player characters that play believably or human-like (as in [11, 165, 451, 1174]) then AI can serve as means for player experience debugging or as demonstration of realistic play for design purposes. Finally, a game that features a rich interaction with NPCs can only benefit from AI that controls NPCs which are expressive and depict human-like and believable behaviors (as in [851, 940, 1122, 1266]).

1.4.2 More Content, Better Content

There are several reasons for game designers and developers to be interested in AI and, in particular, in content generation as covered in detail in Part III. The first reason is *memory consumption*. Content can be compressed by keeping it "unexpanded" until needed. A good example is the classic space trading and adventure game *Elite* (Acornsoft, 1984), which managed to keep hundreds of star systems in a few kilobytes of memory available on the hardware. Second, content generation might foster human *creativity* and allow the emergence of completely new types of games, game genres or entirely new spaces of exploration and artistic expression [636, 643]. Third, if new content can be generated with sufficient variety, quality and quantity, then it may become possible to create truly endless games with *ultimate replay* value. Finally,

when content generation is associated with aspects of play we can expect personalized and *adaptive play* to emerge via the modification of the content generation process.

Unlike other areas of game AI, content generation is a commercial necessity [643]. Prior to the academic interest in content generation [1023, 1157, 1176, 1305], content generation systems had a long history of supporting commercial standard games in creating engaging yet unpredictable game experiences but, most importantly, in lessening the burden of manual game content creation. Games that feature sophisticated content generation systems can garner praise for their technologies or even build an entire marketing campaign on content generation—like *No Man's Sky* (Hello Games, 2016).

The procedural generation of content via machine learning [419, 654, 1104] is nowadays predominantly referred to as **generative AI**. Generative AI in games is concerned currently with the development of methods and tools, such as LLMs and diffusion models, for the creation of game content including text, visuals, audio and even gameplaying behavior [365]. Part III elaborates on the use of generative AI in games in detail.

1.4.3 Player Experience and Behavior Analytics

The use of AI for the understanding of player experience can drive and enhance the design process of games. Game designers usually explore and test a palette of mechanics that yield experience patterns they desire to put the player through. Player states such as engagement, fear and stress, frustration, anticipation, and challenge define critical aspects of the design of player experience [1286], which is dependent on the genre, the narrative and the objectives of the game. As a result, the holy grail of game design, *player experience*, can be improved and tailored to each player but also augmented via richer experience-based interaction. Further, as a direct consequence of better and faster design, the whole game development process is boosted and improved.

Beyond the experience of the player, data derived from games provide a new and complementary way of designing games, of making managerial and marketing decisions about games, of affecting the game production, and of offering a better customer service [295, 314]. One could claim that any AI-informed decision about the development of a game is based on evidence rather than merely an intuition. The popularity of game analytics and **game data mining** methods in the games industry have already showcased the potential of AI for better design, development and quality assurance procedures. In summary, as we will see in the remaining chapters of this book, AI-empowered and data-driven game design can directly contribute to better games.

1.5 The AI and Games Co-revolution

AI appears to be a key enabler of the technological advancements we witness nowadays. Particularly, AI seems to be the key driver of the current industrial change (also known as the 4th industrial revolution) that gradually merges digital technologies, robotics and even biology. Meanwhile we would also argue that games have gradually enabled and popularized highly engaging interactive experiences at a massive scale and nowadays affect most of us on a daily basis. How do these two ongoing processes that are currently shaping our lives influence each other, though? The answer is not as trivial as one would think, since the relationship between AI and games is far more impactful than a mere application of a set of methods (AI) in a domain (games). We argue that game AI is far more than the sum of its two core parts (AI and games) and is, thus, impactful across many other domains, academic fields and industrial applications.

We have so far argued that games play a critical role in advancing AI and that, in turn, AI is shaping the future of game development. Consider the breakthroughs in AI that were made possible via games: tree search, Monte Carlo tree search, deep learning and deep reinforcement learning are just a few examples. Now consider the ways AI has advanced games: play and test games effectively or in a human-like fashion, generate new content and game worlds, and model player behavior and experiences. As games get increasingly more popular but also richer and more complex through generative AI processes, AI will advance further and in turn, it will advance the environments it is trained on in a continuous **co-(r)evolutionary loop**. The methods and technology we build *through* and *for* video games today will likely run the world of tomorrow: from self-driving cars, architecture, manufacturing, and smart embedded systems such as the *internet-of-things* [37], to social media and the **metaverse**; the next generation of digital presence and interaction. Some early examples of this co-revolutionary loop include methods that explore the unknown relationships between AI decision making and game content diversity [165] or open-ended learning methods that continuously generate increasingly complex and diverse game environments for AI methods to advance from [930, 947, 1309]. Based on all the arguments and evidence provided throughout these introductory sections of the book, we firmly believe that video games are perhaps the most important domain to develop AI for, while AI is arguably the most important technological leap forward for games.

1.6 Structure of This Book

We structured this book into 15 chapters that are grouped in 5 parts. In the first **background** part (Part I) we outline the basic AI principles and a number of core learning paradigms (Chap. 2) and we then cover the main AI methods (Chap. 3) that are important for the study of AI in and for games. In the next three parts of the

book we ask the question: *how can AI be used in games*? Answers to this question define the main game AI areas identified and covered as corresponding parts: AI can **play** games (Part II, consisting of Chaps. 4–6), AI can **generate** content (Part III consisting of Chaps. 7–9) and AI can **model** players (Part IV, consisting of Chaps. 10–12). In the final part of the book, namely **the road ahead** (Part V), we attempt a synthesis of the game AI areas that make up this field and discuss the research trends in what we envisage as the **panorama and map of synergies** (Chap. 13). Building on this synthesis, we continue with a chapter dedicated to the research areas we view as largely unexplored and important for the study of game AI, namely **frontier AI and games research** areas (Chap. 14) and we conclude the book with a chapter dedicated to **ethical considerations** of AI and games (Chap. 15). An illustration of how the different chapters fit together in this book is presented in Fig. 1.4. The readers of this book may wish to skip parts that are not of interest or not appropriate given their background. For instance, readers with a background in artificial intelligence may wish to skip the first part, while readers who wish to get a rapid overview of the game AI field or a glimpse of frontier research trends in game AI can solely focus on the final part of the book.

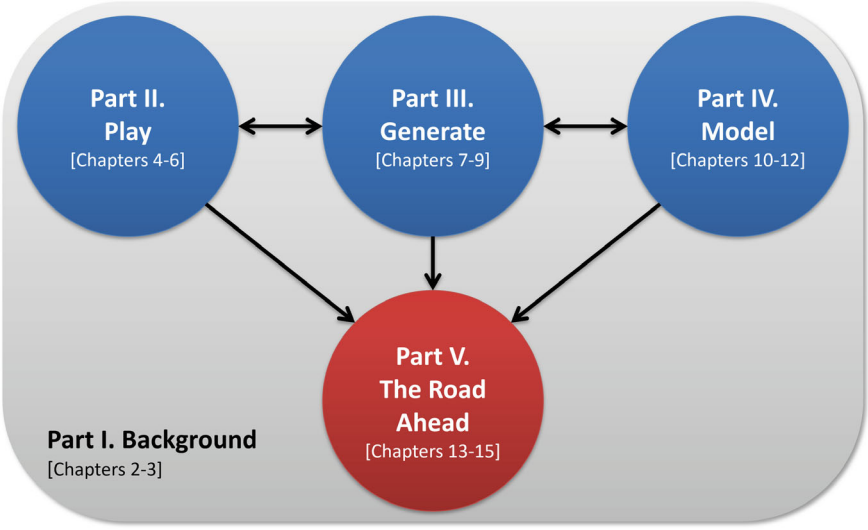


Fig. 1.4 Illustrating the associations among parts and chapters of this book. The light gray area represents the overarching chapters of the AI background covered in Part I. The three blue circles represent the three core uses of AI in games which are interconnected (illustrated with arrows) and are presented in their corresponding parts (Part II-III-IV). The red circle depicts Part V (The Road Ahead) that is informed by all three core AI uses

1.6.1 What We (Don't) Cover in This Book

The core uses of AI in games identified in this book should not be regarded as complete and inclusive of all potential areas of game AI research. It could also be argued that the list of areas we cover is arbitrary. However, this could be said of any research field in any discipline. (In software engineering, software design overlaps with software requirements analysis, and in cognitive psychology, memory research overlaps with attention research.) While it might be possible to perform an analysis of this research field so that the individual areas have minimal or no overlap, this would likely lead to a list of artificial areas that do not correspond to the areas game AI students, researchers and practitioners perceive themselves to be working in. It could also be argued that we are omitting certain areas. For example, we only briefly discuss the topic of **pathfinding** in games as, in our view, pathfinding is a relatively isolated area with restricted interaction with the other uses of AI in games. It is also quite mature. Further, pathfinding has been covered substantially in other game AI textbooks [113, 191, 767] already. Another example is the area of **computational narrative**, which is viewed as a domain for content generation and covered only relatively briefly in Part III. It would certainly be possible to write a whole textbook about text (language) generation and computational narrative, but that would be someone else's job. Beyond particular application areas of game AI, in Part I we cover a number of popular methods used in the field. The list of methods is not inclusive of all AI areas that can be applied to games. However, we consider the included methods sufficient for covering the theoretical foundations of a graduate game AI course. In that regard, we only partially cover **planning** methods and **probabilistic** methods such as Bayesian approaches and methods based on Markov chains, respectively, in Chaps. 3 and 8. It is also important to note that we do not detail behavior selection methods such as **goal-oriented action planners** and **hierarchical task networks** [266] commonly used in the video games industry. Instead, we focus on historically popular game industry methods such as finite state machines, behavior trees, and utility systems.

Another important note is the relationship of this book with the general areas of **game theory** [358, 789, 839] and other related research AI areas such as **multiagent systems** [1035]. Game theory studies mathematical models of *rational* decision makers within abstract games, for the analysis of economic or social behavior in adversarial [788] or cooperative [190] settings. Specifically, game theory focuses on characterizing or predicting the actions of rational or bounded-rational agents, and studies the related emerging "game solution concepts"—such as the celebrated Nash equilibrium [796]. While the book does not cover these areas in detail, as they are peripheral to the aims of game AI as currently envisioned, we nevertheless believe that it could be fruitful to incorporate foundational ideas and concepts from game theory research in the game AI field. Particular game AI areas such as gameplaying and player (or opponent) modeling [358] could benefit from theoretical models of gameplaying [358] and empirical implementations of agent-based systems. Similarly, we believe that game AI research and practice could help in advancing work on

theoretical game theory. Of course, interweaving these fields properly with the current stream of game AI research is a non-trivial exercise, given the different focus and paths these fields have taken. Further, there are limits to the degrees theoretical models can capture the complexity of games covered in game AI. However, game theory undeniably constitutes a key theoretical pillar for the study of rational decision making, and rational decision making is arguably key for winning in games. Some instances of economic game theory that found successful applications in game AI include the various implementations of theoretical models for playing abstract, card and board games—notably a version of Poker was used as a testbed of game theory by von Neumann and Morgenstern [1225] as far back as 1944 [674]. Some of these implementations are discussed in Chap. 4. Another example that bridged rational decision making theory with game AI is the **procedural personas** approach covered in Part II.

Finally, note that this book is the effort of two authors with a high degree of consensus between them and not an edited volume of several authors. As such, it contains *subjective views* on how the game AI field is positioned within the greater AI research scene and how the game AI areas are synthesized. Your mileage may vary.

1.7 Summary

AI has a long-standing and healthy relationship with games. AI algorithms have been advanced or even invented through games. Game design and development, in turn, has benefited largely by the numerous roles AI has taken in games. This book focuses on the main uses of AI in games: for playing games, for generating content, and for modeling players. Those are covered extensively in the following chapters. Before delving into the details of the three core AI uses, in the next part of this book (Part I) we outline the core principles, learning paradigms, methods and algorithms used in the field of game artificial intelligence.

Part I
Background

Chapter 2
AI Fundamentals

Abstract This chapter introduces core AI concepts that underlie the discussions in the rest of the book. The concepts discussed here include search, optimization, and the various types of machine learning, namely unsupervised, supervised, semi-supervised, self-supervised, and reinforcement learning. There is also a discussion on feedback offered by humans, which is a core data source for much work in games, emphasizing the ordinal nature of data solicited as human feedback. While this chapter introduces these high-level concepts, it does not discuss actual algorithms. Those are indeed discussed in Chap. 3.

The AI background part of this book presents a number of basic concepts and corresponding AI methods that are commonly used in games. The concepts and methods we cover here will be referred to in the remainder of this book, because the AI applications in games we will discuss build on some combination of them. These are **fundamental** methods that are frequently covered in introductory AI courses—if you have taken such a course, it should have exposed you to several of the paradigms and methods in this part of the book. It should also have prepared you for easily understanding the other methods discussed in the book.

Before delving into the details of particular AI methods used in games (Chap. 3) in this first chapter of Part I we take the reader through core AI concepts and paradigms. In particular, in Sect. 2.1 we go through a quick overview of two key overarching *concepts* of all methods covered in this book: representation and utility. We then present the fundamentals of *search*, *optimization* (Sect. 2.2) and *machine learning* (Sect. 2.3). The chapter ends with an introduction to *human feedback* as it relates to AI (Sect. 2.4).

2.1 General Concepts

Two overarching elements bind together all the AI methods covered in this book. The first is the algorithm's **representation** type; and the second is its **utility**. All AI algorithms somehow store and maintain knowledge obtained about a particular task (or many tasks) at hand. Most AI algorithms perform some form of search for a solution to the task. This search process takes place in a search space defined

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_2

by a representation and is driven by a utility function of some form. We should note that the utility is of no use solely in methods that employ static knowledge representations such as finite state machines, behavior trees or pre-trained models used for any zero-shot downstream task.

2.1.1 Representation

Appropriately representing knowledge is a key challenge for artificial intelligence at large and it is motivated by the capacity of the human brain to store and retrieve knowledge about the world. The key questions that drive the design of representations for AI are as follows. How do people represent knowledge and how can AI potentially mimic that capacity? What is the nature of knowledge? How generic can a representation scheme be? General answers to the above questions, however, are far from trivial at this point.

As a response to the open *general* questions regarding knowledge and its representation, AI has identified numerous and very *specific* ways to store and retrieve information which is authored, obtained, or learned. The representation of knowledge about a task or a problem can be viewed as the computational mapping of the task under investigation. On that basis, the representation needs to store knowledge about the task in a format that a machine is able to process, such as a data structure.

To enable any form of AI, knowledge needs to be represented computationally and the ways this can happen are many. Representation types include **grammars** such as in grammatical evolution, **graphs** such as finite state machines or probabilistic models, **trees** such as decision trees, behavior trees and genetic programming, real-valued **vectors** such as weights in artificial neural networks and embeddings created by such networks, and **tables** such as in temporal difference learning and Q-learning. As we will see in the remainder of this book, all aforementioned representation types find their uses in games and can be associated with various game AI tasks.

One thing is certain for any AI algorithm that is applied to on a particular task: the chosen representation has a major impact on the performance of the algorithm. Unfortunately, choosing the right representation is not a trivial task as there is no single representation type that is better than others for every problem out there. In principle we opt to scale our representations if we have all compute available out there. Recent evidence across several domains suggest that larger representations, *ceteris paribus*, will likely perform better than smaller ones; this principle is also known as *scaling maximalism* [1154, 1173]. In practice, however, there are limited resources and data available to train ever growing representations. As a rule of thumb, the representation chosen should be as *small* and as *simple* as possible given the complexity of the task at hand. Simplicity usually comes as a delicate balance between computational effort and algorithm performance as either over-detailed or over-simplistic representations might affect the performance of the algorithm. Neither simplicity nor size are trivial decisions to make with respect to the representation. As a matter of fact, good representations come with sufficient practical wisdom and empirical knowledge about the complexity and the qualitative features of the problem the AI is trying to solve.

2.1.2 Utility

Utility in game theory (and economics at large) is a measure of rational choice when playing a game. In general, it can be viewed as a function that is able to assist a search algorithm to decide which path to take. In a sense, a utility function is an approximation of how close we are to the best possible solution. In other words, it is a *measure of goodness* of the existing representation we search through.

Good ideas tend to be invented many times. There are many concepts that are essentially identical to the concept of utility. This includes the notion of a **heuristic** used in computer science and AI as an *approximate* way to solve a problem faster when *exact* methods are too slow to afford, in particular associated with the tree search paradigm. The concept of **fitness**, used to measure the quality of a solution, is mostly another name for utility. In mathematical optimization, the **objective**, **loss**, **cost**, or **error** function is the utility function to be optimized. In supervised (and semi-supervised) learning the loss function represents how well an approach maps training examples to target (desired) outputs. In reinforcement learning and Markov decision processes utility is named cumulative **reward**, which is a function an agent attempts to maximize by learning to take the right sequence of actions in particular states. In unsupervised learning, utility is often provided internally and within the representation via e.g., competitive learning or self-organization. In self-supervised learning utility is generated solely from the unlabeled data.

The utility function drives the search and is, thus, essential for **learning**. On that basis, the utility function is the *training signal* of any machine learning algorithm as it implicitly provides indications on what to do to further increase the current goodness of the representation. Systems that do not require learning (such as AI methods that are based on ad-hoc designed representations, expert systems, or pre-trained AI models) do not require a utility function. In supervised learning the utility is sampled from data—i.e., good input-output patterns. In reinforcement learning and evolutionary computation, instead, the training signal is provided by the environment—i.e., rewards for doing something well and punishments for doing something wrong. Finally, in unsupervised learning the training signal derives from the internal structure of the representation whereas in self-supervised learning, the training signal is generated from unlabeled data but without deriving the inherent structure of that data. For more details on each of these paradigms, refer to the remaining sections of this chapter, focusing on search (Sect. 2.2) and the aforementioned machine learning paradigms (Sect. 2.3).

Similarly to selecting an appropriate representation, the selection of a utility function is far from being trivial. A utility is generally difficult to design and sometimes the design task is basically impossible. The simplicity of its design pays off, but the completeness as well. The quality of a utility function largely depends on thorough empirical research and practical experience, which is gained within the domain under investigation.

2.2 Search and Optimization

Most, if not all, of artificial intelligence is really search. Any AI process that adjusts a set of parameters according to a training signal can be viewed as a search (or an optimization) process. In that regard almost every AI problem can be cast as a search problem, which can be solved by finding the best plan, path, model, function, etc, according to a utility. Search algorithms can therefore be seen as being at the core of AI, to the point that many textbooks (such as Russell and Norvig's famous textbook [975]) start with a treatment of search algorithms.

In the remainder of this section we outline two key search and optimization paradigms: tree search and evolutionary computation. The corresponding algorithms frequently used for AI applications to games are detailed in Chap. 3.

2.2.1 Tree Search

Tree search is a fundamental concept in artificial intelligence, and is usually taught in both introductory algorithms and introductory AI courses. Tree search attempts to find a sequence of actions which an agent can take in order to reach a goal or maximize an objective. At each step in the sequence one or more actions could be taken, and one action must be chosen. This sequence is found by iteratively trying to take different actions, starting with the action at the beginning of the sequence, which is called the *root* of the tree. From there, successor states are explored by taking the different actions available at each state. As most states have a multiple actions available, one can see the various states visited in a tree-like structure rooted at the start of the sequence.

Two key concepts in tree search are *states* and *nodes*. A state is a configuration of the game, or of some part of the game; perhaps only the position of a particular character, perhaps everything that goes on in the game. A node is a part of the tree that refers to a state, as well as its parent node and potentially successor nodes. Multiple nodes can refer to the same state.

Tree search is used everywhere in games. Perhaps most obviously, navigation for both NPCs and player characters is handled by tree search in almost any game. But tree search methods can also be used for planning, content generation, and many other things. The next chapter (see Sect. 3.2) describes algorithms for uniformed search, best-first search, minimax and Monte Carlo tree search as the most representative and popular in this family of methods.

2.2.2 Evolutionary Computation

While tree search algorithms start from the root node representing an origin state, and build a search tree based on the available actions, *optimization* algorithms do not build a search tree; they only consider complete solutions, and not the path taken to get there. As mentioned earlier in Sect. 2.1, all optimization algorithms assume that there is something to optimize solutions for; there must be a *utility function*, alternatively called a **fitness function** in evolutionary optimization terms, which can assign a numerical value (the *fitness*) to a solution. Given a utility function, an optimization algorithm can be seen as an algorithm that seeks in a search space solutions that have the best value of that utility.

A broad family of optimization algorithms is based on randomized variation of solutions, where one or multiple solutions are kept at any given time, and new solutions (candidates, or search points) are created through randomly changing some of the existing solutions, or maybe combining some of them. Randomized optimization algorithms which keep multiple solutions are called **evolutionary algorithms**, in an analogy with natural evolution.

As with other AI methods, optimization algorithms are sensitive to their **representation**. All solutions are represented in some way: for example, as fixed-size vectors of real numbers, or variable-length strings of characters. In other words, the same artifact can be represented in many different ways. For example, when searching for a sequence of actions that solves a maze, the action sequence can be represented as follows. In the most direct representation, the character at step t determines what action to take at time step $t + 1$. A somewhat more indirect representation would be a sequence of tuples, where the character at time step t decides what action to take and the number $t + n$ determines for how many time steps n to take that action. The choice of representation has a big impact on the efficiency and efficacy of the search algorithm, and there are several tradeoffs at play when making these choices (see Sect. 2.1).

Optimization is an extremely general concept, and optimization algorithms are useful for a wide variety of tasks in AI as well as in computing more generally. Within AI and games, optimization algorithms such as evolutionary algorithms have been used in many roles as well. In Part II we explain how optimization algorithms can be used for searching for gameplaying agents, and also for searching for action sequences (these are two very different uses of optimization that are both in the context of gameplaying). In Part III we explain how we can use optimization to create game content such as levels. In Part IV we discuss how to use optimization to create player models. The next chapter (see Sect. 3.3) details a number of local search and evolutionary search methods relevant for games including genetic and quality diversity algorithms.

2.3 Machine Learning

Machine learning (ML) is a sub-field of AI that studies the development of methods and algorithms that are able to learn from data through statistical learning processes. Generally speaking, a good ML method is able to perform well (i.e., generalize) to unseen data. ML approaches—in particular deep neural network architectures (i.e., deep learning)—have seen widespread use [390] in fields such as image recognition and video understanding, speech recognition, natural language processing, notably in large language models. Of course, there are also many applications for ML in video games. Even though not all ML methods rely strictly on statistical processes the vast majority of those are. In this section we outline the core learning paradigms of ML including *unsupervised*, *supervised*, *semi-supervised*, *self-supervised* and *reinforcement* learning. Some of the treatment in what follows might be a little too formal for some tastes; feel free to skip the formal definitions and mathematical notation if you feel like it.

2.3.1 Unsupervised Learning

Unsupervised learning is when you have a dataset, but no labels. Unsupervised learning attempts to discover associations of the input by searching for patterns among all input data attributes and without having access to a target output. Unsupervised ML is usually inspired by Hebbian learning [439] and the principles of self-organization [31]. With unsupervised learning, we focus on the intrinsic structure of and associations in the data instead of attempting to imitate or predict target values. We cover two unsupervised learning paradigms with corresponding methods in Chap. 3: *clustering* and *frequent pattern mining*.

Clustering

Clustering is the unsupervised learning task of finding unknown groups of a number of data points so that data within a group (or else, *cluster*) is similar to each other and dissimilar to data from other clusters. Clustering has found applications in detecting groups of data across multiple attributes and in data reduction tasks such as data compression, noise smoothing, outlier detection and dataset partition. Clustering is of key importance for games with applications in player modeling, game playing and content generation.

As with classification, clustering assigns data to classes; the labels of the classes, however, are unknown a priori and clustering algorithms aim to discover them by assessing their quality iteratively. Since the correct clusters are unknown, similarity (and dissimilarity) depends only on the data attributes used. Good clusters are characterized by two core properties: (1) high *intra*-cluster similarity, or else, high

compactness and (2) low *inter*-cluster similarity, or else, good separation. A popular measure of compactness is the average distance between every sample in the cluster and the closest representative point—e.g., centroid—as used in the k-means algorithm [683]. Examples of separation measures include the *single link* and the *complete link*: the former is the *shortest* distance between any sample in one cluster and any sample in the other cluster; the latter is the *longest* distance between any sample in one cluster and any sample in the other cluster. While compactness and separation are objective measures of cluster validity, it is important to note that they are not indicators of cluster meaningfulness.

Beyond the validity metrics described earlier, clustering algorithms are defined by a *membership function* and a *search procedure*. The membership function defines the structure of the clusters in relation to the data samples. The search procedure is a strategy we follow to cluster our data given a membership function and a validity metric. Examples of such strategies include splitting all data points into clusters at once (as in k-means [683]), or recursively merging (or splitting) clusters (as in hierarchical clustering [786]).

Clustering can be realized via a plethora of algorithms including hierarchical clustering [786], k-means [683], k-medoids [549], DBSCAN [328] and self-organizing maps [577]. The algorithms are dissimilar in the way they define what a cluster is and how it is formed. Selecting an appropriate clustering algorithm and its corresponding parameters, such as which distance function to use or the number of clusters to expect, depends on the aims of the study and the data available. In the next chapter (see Sect. 3.4) we outline two clustering algorithms we find to be the most useful for the study of AI in games: k-means and hierarchical clustering.

Frequent Pattern Mining

Frequent pattern mining is a set of techniques that attempt to derive frequent patterns and structures in data. Frequent pattern mining was first proposed for mining association rules [8], which aims to identify a number of data attributes that frequently associate to each other, thereby forming conditional rules among them. There are two types of frequent pattern mining that are of particular interest for game AI: **frequent itemset mining** and **frequent sequence mining**. The former aims to find structure among data attributes that have no particular internal order whereas the latter aims to find structure among data attributes based on an inherent temporal order. While associated with the unsupervised learning paradigm, frequent pattern mining aims particularly to discover frequent items or sequences in a given data set. Popular and scalable frequent pattern mining methods include the Apriori algorithm [8] for itemset mining, and SPADE [1302] and GSP [714, 1027, 1076] for sequence mining. The next chapter (see Sect. 3.4) outlines Apriori and GSP as representative algorithms for frequent itemset and frequent sequence mining, respectively.

2.3.2 Supervised Learning

Supervised learning is when you train models on labeled datasets, to learn to predict the labels of unseen instances. More formally, supervised learning is the algorithmic process of approximating the underlying function between labeled data and their corresponding attributes or features [91]. A popular example of supervised learning is that of a model[1] that is asked to distinguish between apples and pears (**labeled data**) given a set of **features** or *data attributes* such as the fruits' color and size. Initially, the model learns to classify between apples and pears by *seeing* a number of available fruit examples: the color and size of each fruit, on one hand, and their corresponding label (apple or pear) on the other. After learning is complete, the model should ideally be able to tell whether a new and *unseen* fruit is a pear or an apple based solely on its color and size. Beyond distinguishing between apples and pears, supervised learning is used in a plethora of applications including financial services, medical diagnosis, fraud detection, web page categorization, image and speech recognition, and user modeling. Supervised learning is built into so many parts of a typical application stack that it is hard to find a recently developed computer system that doesn't not employ supervised learning *somewhere*.

By definition, supervised learning requires a set of labeled training examples; hence supervised. The training signal comes as a set of supervised labels on the data (e.g., this is an apple whereas that one is a pear) which acts upon a set of characterizations of these labels (e.g., this apple has red color and medium size). Consequently, each data example comes as a pair of labels (or outputs) and features that correspond to these labels (or inputs). The ultimate goal of supervised learning is not to merely learn from the input-output pairs but to derive a function that approximates their relationship. The derived function should be able to predict well the outputs of new and *unseen* inputs (e.g., unseen apples and pears in our example), a property that is called **generalization**. Here are some examples of input-output pairs one can meet in games and make supervised learning relevant:

- {player health, own health, distance to player} $\rightarrow$ {action (shoot, flee, idle)}
- {number of kills and headshots, ammo spent} $\rightarrow$ {skill rating}
- {score, map explored, average heart rate} $\rightarrow$ {level of player frustration}
- {Ms Pac-Man and ghosts position, pellets available} $\rightarrow$ {Ms Pac-Man direction}
- {pixels of a game screen} $\rightarrow$ {player's next action}.

Formally, supervised learning attempts to derive a function $f : X \rightarrow Y$, given a set of N training examples $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$; where X and Y is the input and output space, respectively; $\mathbf{x}_i$ is the feature (input) vector of the i-th example and $\mathbf{y}_i$ is its corresponding set of labels. A supervised learning task has two core steps. In the first **training** step, the training samples—attributes and corresponding labels—are

[1] We use "model" to refer to anything that is trained by a machine learning method; this may be a neural network, or something completely different, such as a decision tree, or some induced code. We largely use the words "model" and "function" interchangeably, though technically the model represents the learned function.

presented and the function f between attributes and labels is derived. As we will see in the list of algorithms presented in Chap. 3 f can be represented as a number of classification rules, decision trees, or mathematical formulae. In the second **testing** step f can be used to predict the labels of unknown data given their attributes. To validate the generalizability of f and to avoid overfitting to the data [91], it is common practice that f is evaluated on a new independent (test) dataset using a performance measure such as accuracy, which is the percentage of test samples that are correctly predicted by our derived function. If the accuracy is acceptable, we can use f to predict new data samples.

But how do we derive this f function? In general, an algorithmic process modifies the parameters of this function (technically, the parameters of the model) so that we achieve a good match between the given labels of our training samples and the function's output, when the features of the training data are used as the function's input. There are numerous ways to find and represent that function, corresponding to different supervised learning algorithms. These include various artificial neural network architectures (shallow and deep), case-based reasoning, decision tree learning, random forests, Gaussian regression, naive Bayes classifiers, k-nearest neighbors, and support vector machines; Russell and Norvig [975], Bishop [91] and Goodfellow et al. [390] collectively detail most of these methods. The variety of supervised learning algorithms available is, in part, explained by the fact that there is no single learning algorithm that works best on all supervised learning problems out there.

Before covering the details of particular algorithms in Chap. 3 we should stress that the type of label determines the type of output and thus the type of the supervised learning approach that can be applied. We can identify three main types of supervised learning algorithms depending on the data type of the labels.

Regression

Regression is when the label is a number. More technically, if the output supervised learning is requested to predict is an interval value, the downstream task is metric regression [91]. Regression is the recommended approach when the phenomenon we wish to learn to predict can be represented—and it makes sense to be represented—as a scalar value. Under the context of AI and games such scalar values may include the completion time of a game level, the exact positions of a playtrace, or the retention time of a player. Any supervised learning method can learn to predict a scalar value as long as it is trained to minimize the distance between the output of trained f, $\hat{Y}$, and the desired Y. The Mean Squared Error (MSE) is the loss function (L_{mse}) that is commonly used to penalize this difference across N observations as follows:

$$L_{mse} = \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2 \tag{2.1}$$

The performance of a regression method is usually tested through the coefficient of determination, denoted R^2, in a test (unseen) set of data [91]. R^2, which is often

referred to as "R-squared", provides a measure of how well the observed outcomes Y can be predicted by the model f, based on the proportion of total variation of outcomes Y explained by the model f. R^2 is intuitively far more informative than alternative measures of cross-validation performance such as MSE: while the MSE has an arbitrary range, R^2 is bounded within [0, 1] and can be expressed as a percentage.

Classification

Classification is when the label is one of several categories. The label might be a nominal choice between categories that are cannot be ordered such as "apple", "orange", or "papaya". Sometimes the choice could be between discrete classes such as "novice player" and "expert player" that have an underlying order. More formally, when a ML method is tasked to predict categorical class labels (discrete or nominal) then we are faced with a classification [91] downstream task. Classification algorithms attempt to predict the category a set of observations belongs to. For example a classification ML method will try to predict the apples, pears and papayas, or the expertise level of a player of the previous examples. As with regression, any supervised learning method can learn to predict classes by minimizing distance between the desired class and the predicted probability $p_{x,i}$ that an observation x is of class i of a trained classifier. The most common error function used for that purpose is the cross-entropy loss (L_{ce}) which is expressed as follows:

$$L_{ce} = -\frac{1}{C}\sum_{i=1}^{C} c_{x,i} \log p_{x,i} \tag{2.2}$$

where C is the number of classes we consider and c is a binary indicator that is 0 if the assigned class label i is correct (or 1 if incorrect) for observation x.

Classification methods are normally evaluated for their ability to predict the correct class on a test (unseen) data set via their *accuracy*. Accuracy is usually expressed as the percentage that measures how well a classifier predicts the true classes of unseen observations and is computed as the fraction of correct classifications over all classifications.

Preference Learning

Preference learning is when the label is a preference of one choice over others. Beyond regression and classification methods, preference learning [359] algorithms learn to predict ordinal outputs, such as ranks and preferences, and attempt to derive the underlying global order that characterizes those ordinal labels. Examples of ordinal outputs include a preference of a particular sound effect over others, a player's preference of a particular navigation route over alternatives, and an increase of engagement in this particular room of the level compared to the previous room visited. The training signal in the preference learning paradigm provides information about the

relative relationship among instances of the phenomenon we attempt to approximate, whereas regression and classification provide information, respectively, about the *intensity* and the *classes* of the phenomenon. As with the regression and classification learning paradigms outlined earlier, any supervised learning method covered in Chap. 3 can be converted to a preference learner including neural network variants, random forests, transformers and so on. While there are many different methods that fall under the preference learning umbrella such as **ordinal regression** [1252] and **learning to rank** [359], most can be reduced to the *pairwise* preference learning paradigm through which a model is trained to predict the preference of a set of observation pairs in a dataset D in the following form: $\mathrm{x}_A \succ \mathrm{x}_B$ (observation A is preferred over observation B). A pairwise preference loss function (L_{pl}) can then take the general form:

$$L_{pl} = \sum_{\mathbf{x}_A, \mathbf{x}_B \in D} \phi(f(\mathbf{x}_A) - f(\mathbf{x}_B)) \tag{2.3}$$

where $\mathbf{x}_A$ and $\mathbf{x}_B$ represent a pair of training samples such that $\mathbf{x}_A$ is preferred over $\mathbf{x}_B$; ϕ can be a hinge function $\phi(z) = \max(0, 1 - z)$, an exponential function $\phi(z) = e^{-z}$ or a logistic function $\phi(z) = \log(1 + e^{-z})$ among many other options.

For each training sample, the output of the trained model represents a utility value which defines its order in terms of preference. For any two training samples, the one with the larger utility would be predicted as preferred. Many different frameworks are available for evaluating the performance of a preference learner. The outputs of the model can be viewed as equivalent to the probability distribution of a regression or a classification model and hence metrics such as R^2 or accuracy can be used, respectively. To assess the ability of a trained model to rank unseen observations one may employ the Kendall's τ coefficient [556] to measure the ordinal association between the predicted ranking and the ground truth ranking.

2.3.3 Semi-supervised Learning

Semi-supervised learning is when you have a dataset, but only a small part of it is labeled. The semi-supervised learning paradigm encompasses a family of algorithms that inherit properties of both the supervised and unsupervised learning paradigms. Semi-supervised learning becomes necessary when we need to perform a supervised learning task but we only have a limited amount of labeled data and we have large amounts of unlabeled data. The paradigm is also relevant when the cost of labeling enough data for supervised learning is prohibitively high. Semi-supervised learning is also known as **weak supervision** as the training signal relies only on a small dataset. The semi-supervised learning paradigm becomes critical particularly when training large AI models (e.g., LLMs) which normally require large amounts of labeled data that are hard to obtain.

Semi-supervised learning can be viewed intuitively as the training process to perform well on a general task, such as playing a complete game, by considering only limited samples of good playing behavior in certain parts of the game. For training a model in a semi-supervised fashion, one uses the smaller labeled dataset as a weak guide for the downstream task on the larger unlabeled dataset. One could view semi-supervised learning as the process of first clustering the data and then labeling the clusters with the available labeled data, thereby guiding training towards regions of the labeled data.

Several methods are applicable here from the family of neural networks and support vector machines (SVMs). A popular method that extends SVMs so as to cater for partially labeled data is the *transductive SVM* [1209]. While for supervised learning, SVMs attempt to find the decision boundary with maximal margin on labeled data, transductive SVMs instead label the unlabeled data so that the decision boundary yields a maximal margin over all available data.

Self-training is often used as an alternative approach to tackle the challenges of semi-supervised learning. According to this paradigm one first trains a supervised learner on the labeled data which is then employed to the unlabeled data. The supervised learner can thus generate more labeled examples for the supervised learning algorithm. The *video-pretraining* method introduced in [46] for playing *Minecraft* (Mojang Studios, 2011) is an indicative example of this process. First an *inverse dynamics model* (IDM) of the game was trained on 2, 000 h of playthrough videos labeled with player actions. Then, the trained IDM was used to label 70, 000 h of online videos for the purpose of training a foundation model of gameplay in *Minecraft*. The resulting model approaches the performance of expert players of the game as it learns to craft diamond tools within short periods of time.

2.3.4 Self-supervised Learning

Self-supervised learning is when you have a dataset with no labels, but you create labels out of the data in some way. For example, the label for an instance may be the instance itself (when learning to generate images), or a the next instance in a sequence (when learning to model language). In the self-supervised learning paradigm [49], a model is trained without depending on human feedback such as data labeling. Instead, the model uses solely the unlabeled data to generate its training signal. Self-supervised learning is applicable to tasks such as image classification where inherent structures or relationships between data (e.g., objects on images) are available and can yield appropriate training signals. As self-supervised learning does not require any human intervention (in terms of labeled data), it is easier to employ and train in an online fashion.

The training approach of the self-supervised learning paradigm is generally as follows. The input (e.g., image observations) of the model is transformed through a number of ways (e.g., through image rotations, distortions and noising). This data augmentation process generates new data samples. The transformed samples are

normally compared to the original observations to formulate the training signal of the algorithm. Self-supervised learning can be seen as a form of supervised learning since ultimately the model predicts some target outputs based on comparisons of input data. Similarly, it can be viewed as an unsupervised learning process since the model learns without labeled data. In contrast to unsupervised learning, however, the self-supervised learning model does not learn the inherent structure of the data (see Fig. 2.1).

We adopt the taxonomy of Liu et al. [656] that fits all self-supervision methods across three following general categories:

- **Generative** methods train a model to encode input x into a latent vector z and then decode or reconstruct x from z. Popular methods in this category include autoregressive models (such as Large Language Models) and autoencoders [390] (see Sect. 3.6.2 for more details). Loss functions used in this family of algorithms normally penalize the reconstruction errors generated by the decoder.
- **Contrastive** methods train a model to encode input x into latent vector z to measure similarity between input pairs. Popular methods of this category include variants

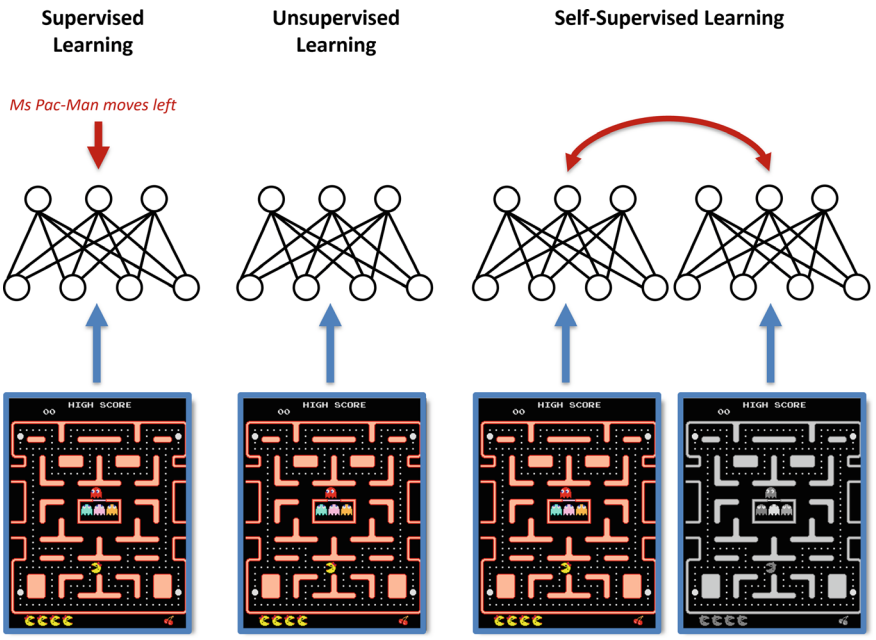


Fig. 2.1 Illustrating the differences between supervised, unsupervised and self-supervised learning in Ms Pac-Man. Blue and red arrows depict inputs and target outputs (labels), respectively. In self-supervised learning, the label is derived from co-occurring relationships of dissimilar (yet related) input. Those relationships can be obtained from a transformed input (as in the grayscale image of the game in the example provided), from another input modality (e.g., consider a text prompt "Ms Pac-Man is looking left"), or from parts of the input (e.g., think of a mask image around Ms. Pac-Man and ghosts). Illustration inspired by [656]

of contrastive learning. Contrastive learning attempts to minimize the distance between positive data samples (e.g., images of the same football game state viewed from different angles) while maximizing the distance between negative samples (e.g., images of different football game states). A recent example of contrastive learning applications in games is the series of studies by Trivedi et al. [1186, 1187] aimed at constructing generalized game representations across games of the same genre.

- **Generative-Contrastive** or **adversarial** methods train a generator model capable of generating fake data samples and a discriminator model that learns to distinguish those generated samples from real samples. The most popular method in this category is the generative adversarial network (GAN) [392].

The model architecture, the underlying objectives and the loss signal used determine the category of the self-supervised learning method [656]. The architecture may feature solely a generator (generative) or both a generator and a discriminator (contrastive methods and adversarial methods). Generators may employ only an encoder (contrastive) or both an encoder and a decoder (generative and adversarial). The generative, contrastive and adversarial models are trained, respectively, via reconstruction loss, contrastive similarity measures and measures of distributional divergence such as those from the KL-divergence [592], and Wasserstein distance [1210] families.

2.3.5 Reinforcement Learning

Reinforcement Learning (RL) [1109] is when you don't have a dataset, but you have a simulator or some other environment (perhaps a physical robot), and you learn by taking actions in that environment and observing the results. RL is a machine learning approach inspired by behaviorist psychology and, in particular, the way humans and non-human animals learn to take decisions via (positive or negative) rewards received by their environment. In reinforcement learning, samples of good behavior are usually not available (as in supervised learning); instead—similarly to evolutionary (reinforcement) learning—the training signal of the algorithm is provided by the environment based on how an agent is interacting with it. At a particular point in time t, the agent is on a particular *state* s and decides to take an *action* a from all the available actions in its current state. As a response the environment delivers an immediate **reward**, r. Through the continuous interaction between the agent and its environment, the agent gradually learns to select actions that maximize its utility (i.e., sum of rewards). RL has been studied from a variety of disciplinary perspectives [1109] including operations research, game theory, information theory, and genetic algorithms. RL has been successfully applied in problems which involve a balance between long-term and short-term rewards such as robot control and games [773, 1040]. An example of the RL problem is illustrated through a maze navigation task in Fig. 2.2.

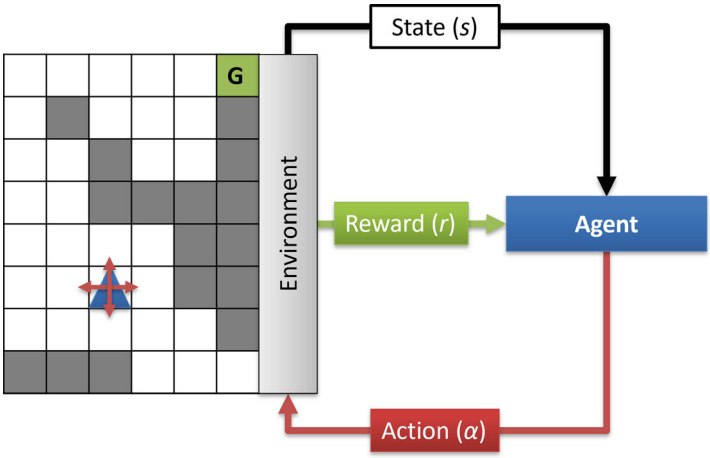


Fig. 2.2 A reinforcement learning example. The agent (triangle) attempts to reach the goal (G) by taking an action (a) among all available actions in its current state (s). The agent receives an immediate reward (r) and the environment notifies the agent about its new state after taking the action

More formally, the aim of an RL agent is to discover a **policy** (π) for selecting actions that maximize a long-term utility, such as the expected cumulative (possibly discounted) reward. A policy is a strategy that the agent follows in selecting actions, given the state it is in. If the function that characterizes the value of each action either exists or is learned, the optimal policy (π^*) can be derived by selecting the action with the highest value. The interactions with the environment occur in discrete time steps t and are modeled as a **Markov decision process** (MDP). The MDP is defined by

- S: A set of states $\{s_1, ..., s_n\} \in S$. The environment states are a function of the agent's information about the environment (i.e., the agent's inputs).
- A: A set of actions $\{a_1, ..., a_m\} \in A$ possible in each state s. The actions represent the different ways the agent can act in the environment.
- $P(s, s', a)$: The probability of transition from s to s' given a. P gives the probability of ending in state s' after picking action a in state s and it follows the **Markov property** implying that (the probability of) future states of the process depend only upon the present state, not on the sequence of events that preceded it. As a result, the Markov property of P makes predictions of 1-step dynamics possible.
- $R(s, s', a)$: The reward function on transition from s to s' given a. When the agent in state s picks an action a and moves to state s', it receives an immediate reward r from the environment.

P and R define the **world model** and represent, respectively, the environment's dynamics (P) and the long-term reward (R) for each policy. If the world model is *known* there is no need to learn to estimate the transition probability and reward

function and we thus directly calculate the optimal strategy (policy) using *model-based* approaches such as dynamic programming [83]. If, instead, the world model is *unknown* we approximate the transition and the reward functions by learning estimates of future rewards given by picking action a in state s. We then calculate our policy based on these estimates. Learning occurs via *model-free* methods such as Monte Carlo search and **temporal difference learning** [1109]. In Chap. 3 we focus on two of the most popular model-free algorithms: Q-learning (off-policy) and proximal policy optimization (on-policy). Here, we discuss a few core RL concepts and provide a high-level taxonomy of RL algorithms according to RL problems and tools used for tackling them. We will use this taxonomy to place Q-learning, Proximal Policy Optimization (PPO), and other methods with respect to RL as a whole.

Core Concepts and a High-Level Taxonomy

A central question in RL problems is the right balance between the **exploitation** of current learned knowledge versus the **exploration** of unseen territories in the search space. Randomly selecting actions (no exploitation) or always greedily selecting the best action according to a utility (no exploration) are strategies that generally yield poor results in stochastic environments. While several approaches have been proposed to address the exploration-exploitation balance issue, a popular and rather efficient mechanism for RL action selection is called ϵ-greedy, determined by the $\epsilon \in [0, 1]$ parameter [1109]. According to ϵ-greedy the RL agent chooses the action it believes will return the highest future reward with probability $1 - \epsilon$; otherwise, it chooses an action uniformly at random.

RL problems can be classified into **episodic** versus **incremental**. In episodic RL problems, algorithm training occurs offline and within a finite horizon of multiple training instances. The finite sequence of states, actions and reward signals received within that horizon is called an *episode*. Monte Carlo methods that rely on repeated random sampling, for instance, are a typical example of episodic RL [1109]. In incremental RL problems, instead, learning occurs online and it is not bounded by an horizon. TD learning is an exemplar incremental RL algorithm.

Another distinction is between **off-policy** and **on-policy** RL algorithms. An off-policy learner approximates the optimal policy independently of the agent's actions. Q-learning is an off-policy learner since it estimates the return for state-action pairs assuming that a greedy policy is followed. An on-policy RL algorithm, such as PPO, instead approximates the policy as a process tied to the agent's actions including the exploration steps.

Bootstrapping [1109] is a central notion within RL that classifies algorithms based on the way they optimize state values. Bootstrapping estimates how good a state is based on how good we think the next state is. In other words, with bootstrapping we update an estimate based on another estimate. Both TD learning and dynamic programming use bootstrapping to learn from the experience of visiting states and updating their values. Monte Carlo search methods instead do not use bootstrapping and, thus, learn each state value separately [1109].

Finally, the notion of **backup** is central in RL and acts as a distinctive feature among RL algorithms. With backup we go backwards from a state in the future, s_{t+h}, to the (current) state we want to evaluate, s_t, and consider the in-between state values in our estimates. The backup operation has two main properties: its *depth*—which varies from one step backwards to a full backup—and its *breadth*—which varies from a (randomly) selected number of sample states within each time step to a full-breadth backup. Figure 2.3 illustrates the notions of breath and depth via a two-dimensional plot as inspired by a figure from [1109]. The vertical and horizontal dimensions of that figure correspond, respectively, to the depth and width of backup. Dynamic programming methods are shown in the top right corner of the plot as they consider one-step full backup updates. The figure displays exhaustive search at the bottom-right corner as the extreme case of full-breath backup until all terminal states are visited. Methods that are placed in between dynamic programming and exhaustive search include heuristic search methods and do backup updates up to a specified depth. Along the left edge of the plot we place sample backup methods, ranging TD learning at the top left corner (one step) to Monte Carlo at the bottom left (full backup). The spectrum in between these includes methods that rely on a predetermined depth (n-step) of backup updates. The gradient colored interior of the

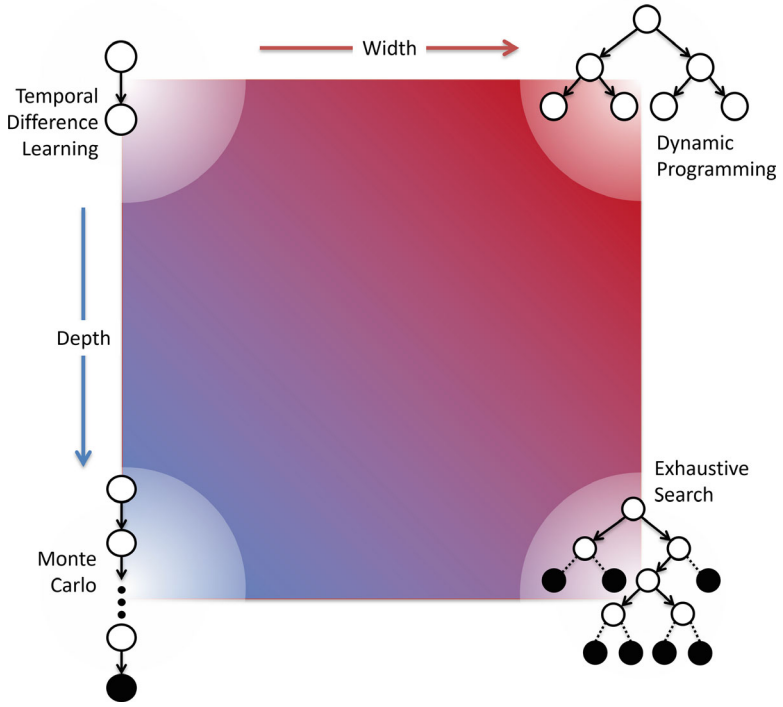


Fig. 2.3 An illustration of RL methods with respect to the *depth* (y-axis) and the *width* (x-axis) of the backup update. Figure inspired by [1109]

2D plot of Fig. 2.3 represents the space of possibilities of intermediate RL methods that vary the depth and width of their backup mechanism.

Based on the aforementioned criteria we can identify three major RL algorithm types that are also depicted at the three of the four corners of Fig. 2.3:

1. **Dynamic programming**. In dynamic programming knowledge of the world model (P and R) is required and the optimal policy is calculated via bootstrapping.
2. **Monte Carlo methods**. Knowledge of the world model is not required for Monte Carlo methods. Algorithms of this class are ideal for off-line (episodic) training and they learn via sample-breadth and full-depth backup. Monte Carlo methods do not use bootstrapping, however.
3. **TD learning**. As with Monte Carlo methods knowledge of the world model is not required and it is thus estimated. Algorithms of this type (e.g., Q-learning) learn from experience via bootstrapping and variants of backup.

2.4 AI and Human Feedback

In this last section we discuss the importance of human feedback in association with the AI paradigms we have covered. While we view human feedback from the perspective of games, the core principles we outline in this section are valid for all sorts of human feedback beyond games.

Human feedback is important and overarching across any use of AI in games because humans are involved in everything that a game is. Humans are tasked to design, play, evaluate, and test games. Therefore, their feedback is critical for the design and improvement of AI methods applied to games. Beyond data of an *objective* nature, such as play traces, humans are often requested to provide data of *subjective* nature such as their preferences, emotional reactions and opinions about a game, a playing style, a level design process, an AI-controlled agent and so on. This is the type of human feedback we are mostly concerned about in this book.

The fields of psychology, psychometrics, affective computing, and human computer interaction (among others) have given us a large volume of papers and studies dedicated to the study of humans, their perception and expression of such subjective notions [1270]. We argue that the most basic challenge in the design of human-centered AI is the human part of it, particularly the complex multimodal signals that are generated by humans. The feedback that is given to an AI algorithm is usually a response to a direct record of multimodal stimuli such as audio, video and text from the game. An annotation process would then request humans to attach labels that describe how they assess the situation. The challenge arises because the channels through which humans can express and externalize their impressions are extremely narrow [1292]. It is still largely unknown what goes on inside one's head during a play or design experience for instance. The perception and expression of those experiences are even more complex and diverse than any other kind of response that players or designers are expected to make.

At the time of writing, the design of the best possible protocol for soliciting human feedback for any AI task remains largely an open question [989]. In the remainder of the section we go through a list of critical questions often met in AI and games research regarding human feedback. Moreover we offer a set of corresponding guidelines for maximizing the validity and value of the human labels one can obtain independently of the task at hand.

Objective or Subjective Labels?

The discussion on human feedback becomes particularly relevant and important for AI and games when it is about **subjective** notions instead of **objective** phenomena. The latter includes data that can be objectively and undeniably defined such as the completion time of a level, the number of gun shots fired, the amount of time spent playing a game, the content that was bought, the in-game trajectory of an agent or a player and so on. The former, instead, considers notions related to player experience such as *engagement*, *arousal*, and *tension* [1286], designer choices, such as designer intent, goals and aims [639], user perceptions of any content type, and even gameplaying style, decision making and aesthetics of play [41]. We argue that all major uses of AI in games—gameplaying, content generation and player modeling—involve subjective notions that need to be labeled. The challenge of soliciting human feedback for such subjective notions is what we discuss in the remainder of this section.

Free or Forced Response?

Subjective labels can be based either on **free response**—retrieved via e.g., a think-aloud protocol [923] or free-form text [1323]—or on **forced responses** retrieved through questionnaires or annotation tools. Free response naturally contains richer information about the player's state, but it is often unstructured, even chaotic, and thus hard to analyze appropriately. There are however ways to process free-form text into a set of usable factors represented as labels. For that conversion one can employ humans or large foundation models. Zuniga et al. [1323], for instance, employed human annotators to review free-form responses of other human annotators and asked them to assign labels that would best characterize the free-form text. Such labels could then be used for further analysis. Alternatively, one may use human annotated text to prompt LLMs that would, in turn, process and label the free-form text. Free-form text can also be used as form of human feedback for fine-tuning such LLMs. At the time of writing this book, the challenge of utilizing human feedback for enhancing the performance of LLMs remains wide open [989]. Instead of processing and analyzing free-form responses one can force participants to report their experiences using directed questions or tasks. The forced-choice approach constrains annotators to specific questionnaire items which could vary from simple tick boxes to multiple choice items. Both the questions and the answers provided to annotators may

vary from single words to sentences, and might contain numbers, images, sounds or videos. Questionnaires may consider elements of player experience (e.g., the Game Experience Questionnaire [491]), demographic data, personality traits (e.g., a validated psychological profiling questionnaire such as the NEO-PI-R [240]), and even motivation factors [43, 740].

In summary, free-response has certain advantages when it comes to directness and expressivity, but it requires another round of either annotation (e.g., via more human feedback) or processing (e.g., fine-tuning an LLM on human preferences). Forced-choice limits the options for participants but, at the time of writing, it is a dominant approach for building models of human preferences and experiences [1270] all the way to training language models [844]. Human feedback obtained this way yields labels that are easier to collect and process.

Who Annotates?

Given the subjective nature of notions such as player experience or content aesthetics, another natural question that comes in mind is *who annotates*? Who has the right authority and the best capacity to provide us with reliable tags of any experience? We distinguish two main categories: annotations can either be self-reports or reports expressed indirectly by experts or external observers [1270, 1289].

Self-reported labels (or experience demonstrations) are provided by the humans involved in the activity themselves, namely in a **first-person** manner. For example, a player is asked to rate the level of engagement while watching their playthrough video as in [888]. In another example, a designer is asked to rank map suggestions during their level design process [639]. First-person is the most direct way to annotate someone's experience and build a model based on the solicited labels or demonstrations. We can only assume that there is some disparity between the true (inner) experience of each human and the experience as felt by themselves or perceived by others. Based on this assumption one's own annotations should normally be closer to the inner experience (**ground truth**) compared to third-person annotation. First-person annotation, however, may suffer from self-deception and memory limitations [1283]. These limitations have been attributed mainly to the discrepancies between "the experiencing self" and "the remembering self" of a person [533] which is also known as the **memory-experience gap** [770].

As a response to the aforementioned limitations expert annotators may instead be able to surpass the perception of experience and reach out to the inner experience of the player. In this second annotation category, named **third-person** annotation, an expert—such as a game designer—or an external observer labels a user's state or assesses aspects of a game and its content in a more objective manner, thereby reducing the subjective biases of first-person perceptions. For instance, experienced players of a game may provide particular labels, or experience demonstrations, while watching gameplaying videos of other players as in [59, 64, 723]. The benefit of third-person annotation is that multiple annotators can be used to assess a particular experience or label content qualities. In fact, the availability of several such subjective perceptions may allow us to approximate the ground truth better as the agreement

between many annotators enhances the validity of our data directly. A potential disagreement, on the other hand, might suggest that the subjective phenomenon we examine is non-trivial or may indicate that some of our annotators are untrained or inexperienced [64]. Disagreements among annotators might also very well indicate the existence of more than a single ground truth for a particular gameplay.

When to Annotate?

When is it best to annotate a play experience or a design session: *before*, *during* or *after* the session itself? In a **pre-experience** questionnaire we usually ask annotators to set the baseline of the annotator's state prior to the session. This state can be influenced by a number of factors such as the mood of the day, the social network activity, the caffeine consumption, earlier playing activity and so on. This is a wealth of information that can be used to enrich any model that attempts to predict the annotated labels. Again, what is worth detecting is the relative *change* [1269, 1292] from the baseline state of the user prior to playing a game, observing a session or assessing any creative outcome of a game or an algorithm.

A **during-experience** protocol, on the other hand, may involve the annotator in a first-person think-aloud setup [923, 1323] or a third-person annotation design. For the latter protocol you may think of user experience experts that observe and annotate player experience during the beta release of a game, for example. As mentioned earlier, first-person annotation during any session is a rather intrusive process that disrupts the experience and risks adding experimental noise to annotation data. In contrast, third-person annotation is not intrusive; however, there are expected deviations from the actual first-person experience, which is inaccessible to the observer.

The most popular approach for the annotation of a session's experience is after the session (or a series of sessions) is completed, in a **post-experience** fashion. Post-experience annotation is unobtrusive for the annotator and can be performed by the annotators themselves or third-party annotators. Self-reports, however, are memory-dependent by nature and memory is, in turn, time-dependent. Thus, one needs to consider carefully the *time window* between the experience and its corresponding report. For the reported post-experience to be a good approximation of the actual experience the session's time window needs to be small to minimize memory biases, yet sufficiently large to elicit particular experiences to the player. The higher the cognitive load required to retrieve the session's context, the more the reports are memory-biased and not relevant to the actual experience. The longer the time window between the real experience and the self-report, the more the annotator activates aspects of *episodic* memory associated with the experience [954]. Episodic memory traces that form the basis of self-reports fade over time, but the precise rate at which this memory decay occurs is unknown and most likely individual [954]. Ideally, memory decay is so slow that the annotator will have a clear feeling of the session when annotating it. Now, if the time window becomes substantial—on the scale of hours and days—the annotator has to activate aspects of *semantic* memory [222] such as general beliefs about a game and general aesthetic preferences of content style. In summary, the more the episodic memory, and even more so the semantic memory,

are activated during annotation, the more systematic errors are induced within the annotation data.

As a general rule of thumb the longer it takes us to evaluate an experience the larger the discrepancy between the true experience and the evaluation of the experience. It also seems that this gap between our memory of the experience and the real experience is more prominent when we report unpleasant emotions such as anger, sadness and tension rather than positive emotions [770]. Another bias that affects how we report our experience is the experience felt near the end of a session; this effect has been named *peak-end rule* [770].

An effective way to assist episodic memory and minimize post-experience cognitive load is to show annotators replay videos of their session (or the sessions of others) and ask them to annotate those. This can be achieved via crowdsourcing [165] in a third-person manner or as a first-person annotation [166, 216, 463, 664, 888]. Another notable approach is the *data-driven retrospective interviewing* method [313, 314], where a user's behavioral data is collected and analyzed to drive the construction of interview questions. These questions are then used in retrospect (post-experience) to assess the annotator's reliability.

Which Label Type?

We are often uncertain about the type of labels we wish to assign to a player state, a player or designer experience, or even a characteristic of content. Generally speaking, we can select from three data types for our annotation: ratings, classes, and ranks.

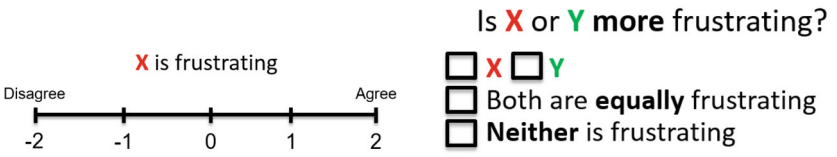
The **rating**-based format represents our subjective notion with a scalar value or a vector of values. Ratings are arguably the dominant practice for quantitatively assessing aspects of a user's behavior, experience, opinion or emotion. The vast majority of user and psychometric studies have adopted magnitude-based questionnaires to capture the opinions, preferences and perceived experiences of experiment participants—see [140, 206, 722] among many. The most popular rating-based questionnaire follows the principles of a **Likert** scale [647] in which users are asked to specify their level of agreement with (or disagreement against) a given statement—see Fig. 2.4a for an example. Other popular rating-based questionnaires for user and player experience annotation include the Geneva Wheel model [1000], the Self-Assessment Manikin [780], the Positive and Negative Affect Schedule [1067], the Game Experience Questionnaire [491], the Flow State Scale [501], the Player Experience of Need Satisfaction (PENS) survey [976] and the Ubisoft Perceived Experience Questionnaire [43].

Rating-based reporting has notable inherent limitations that are often overlooked, resulting in fundamentally flawed analyses [506, 1270, 1283]. First, ratings are analyzed traditionally by comparing their values across participants; see [395, 707] among many. While this is a generally accepted and dominant practice it neglects the existence of **inter-personal differences** as the meaning of each level on a rating scale may differ across experiment participants. For example, two participants assessing the difficulty of a level may assess it as exactly the same difficulty, but then one rates it as "very easy to play" and the other as "extremely easy to play".

It turns out that there are numerous factors that contribute to the different internal rating scales existent across participants [755, 1270] such as differences in personality, culture [1061], temperament and interests [1220]. Further, a large volume of studies has also identified the presence of primacy and recency order effects in rating-based questionnaires (e.g., [195, 1278]), systematic biases towards parts of the scale [651] (e.g., right-handed participants may tend to use the right side of the scale) or a fixed tendency over time (e.g., on a series of experimental conditions, the last ones are rated higher). Indicatively, the comparative study of [1278] between ratings and ranks showcases higher inconsistency effects and significant order (recency) effects existent in ratings.

In addition to inter-personal differences, a critical limitation arises when ratings are treated as interval values since ratings are by nature *ordinal values* [506, 1087]. Strictly speaking, any approach or method that treats ratings as numbers by, for instance, averaging their ordinal labels is *fundamentally flawed*. In most questionnaires Likert items are represented as pictures (e.g., different representations of arousal in the Self-Assessment Manikin [780]) or as adjectives (e.g., "moderately", "fairly" and "extremely"). These labels (images or adjectives) are often erroneously converted to integer numbers, violating basic axioms of statistics which suggest that ordinal values cannot be treated as interval values [1087] since the underlying numerical scale is unknown. Note that even when a questionnaire features ratings as numbers (e.g., see Fig. 2.4a), the scale is still ordinal as the numbers in the instrument represent labels. Thus, the underlying numerical scale is still unknown and dependent on the participant [602, 845, 1087]. Treating ratings as interval values is grounded in the assumption that the difference between consecutive ratings is fixed and equal. However, there is no valid assumption suggesting that a subjective rating scale is linear [506]. For instance, the difference between "fairly (4)" and "extremely (5)" may be larger than the distance between "moderately (3)" and "fairly (4)" as some experiment participants rarely use the extremes of the scale or tend to use one extreme more than the other [602]. If, instead, ratings are treated naturally as ordinal data no assumptions are made about the distance between rating labels, which eliminates noise to the analysis.

The second data type available for manual labeling via human feedback is the **class**. Classes allow annotators to select from a finite and non-structured set of options and, thus, a class-based questionnaire provides nominal data among two or more options. The questionnaire asks subjects to pick a player state from a particular representation which could vary from a simple boolean question (*was that game level frustrating or not? is this a sad facial expression? which level was the most stressful?*) to a player state selection (*is this a high- or a low-arousal game state for the player?*). The limitations of ratings are mitigated, in part, via the use of class-based questionnaires. By not providing information about the intensity of each player state, however, classes do not have the level of granularity ratings naturally have. A class-based questionnaire might also yield annotations with an unbalanced number of samples per class. A common practice in psychometrics consists of transforming sets of consecutive ratings into separate classes (e.g., see [377, 446] among many). In an example study [438], arousal ratings on a 7-point scale are transformed into



(a) Rating: A 5-point Likert item example

(b) Rank: A 4-alternative forced choice example

Fig. 2.4 Examples of rating-based (**a**) versus rank-based (**b**) questionnaires for annotating frustration. In **a** an annotator is requested to provide a *magnitude* for the frustration level of stimulus X, from −2 to 2 in this example. In **b** the annotator is requested to make a *relative* comparison between a pair of stimuli (X and Y) in terms of frustration; there are 4 options to choose from in this example

high, *neutral* and *low* arousal classes using [7–5], 4, and [3–1] ratings, respectively. While doing so might seem appropriate, the ordinal relation among classes is not taken into account. More importantly, the transformation process adds a new bias to the subjectivity bias of ratings, namely **class splitting criteria** [716].

Finally, **rank**-based questionnaires ask the annotator to provide their preference among options such as two or more sessions of the game [1267]. In its simplest form, the annotator compares two options and specifies which one is preferred under a given statement (**pairwise preference**). With more than two options, the participants are asked to provide a ranking of some or all the options. Examples of rank-based questions include: *was that level more engaging than this level? which facial expression looks happier?*). Another example of a rank-based questionnaire, *4-alternative forced choice* is illustrated in Fig. 2.4b. Being a form of subjective reporting, rank-based questionnaires (as much as rating-based and class-based questionnaires) are associated with the well known limitations of memory biases and self-deception.

Reporting about subjective constructs such as experience, preference or emotion via rank-based questionnaires has recently attracted the interest of researchers in marketing [279], psychology [130], user modeling [71, 1265], affective computing [716, 755, 1177, 1269, 1270, 1278] and artificial intelligence at large [844]. The next section details the reasons why relative (ordinal) labels should be preferred over absolute (magnitude) labels. The section also emphasizes the current paradigm shift that dominates the use of relative human feedback in games specifically and in AI generally.

2.4.1 The Ordinal Nature of Human Feedback

Describing, labeling and assigning values to subjective notions, such as player experience or designer intent, is a non-trivial task as verified by a number of disciplines including neuroscience [1013], psychology [443], economics [1044], and artificial

intelligence [531]. Annotators can attempt to assign numbers to such notions in an **absolute** manner, using for instance a rating scale, or in a **relative** fashion, using for instance a ranking. There are, however, a multitude of theoretical and practical reasons to doubt that subjective notions can be encoded as numbers in the first place [1269]. For instance, according to Kahneman [532], largely known as a founding father of **behavioral economics**, "...it is safe to assume that changes are more accessible than absolute values". His theory about judgment heuristics is built on Herbert Simon's *psychology of bounded rationality* [1044]. Further, an important thesis in psychology, named **adaptation level theory** [443], suggests that humans lack the ability to maintain a constant value about subjective notions and their preferences about options are, instead, made on a pairwise comparison basis using an internal ordinal scale [765]. The thesis claims that while we are efficient at discriminating among options, we are not good at assigning accurate absolute values for the intensity of what we perceive. For example, we are particularly bad at assigning absolute values to tension, frequency and loudness of sounds, the brightness of an image, or the arousal level of a video. These theories have also been supported by neuroscientific evidence suggesting that experience with stimuli gradually creates our own internal context, or *anchor* [1013], against which we rank any forthcoming stimulus or perceived experience. Thus, our choice about an option is driven by our internal ordinal representation of that particular option within a sample of options; not by any absolute value of that option [1088].

One may argue that the relative assessment provides less information than the absolute assessment since it does not express a quantity explicitly and only provides ordinal relations. As argued earlier, however, any additional information obtained in an absolute fashion (e.g., when ratings are treated as numbers) violates basic axioms of applied statistics. Thus the value of the additional information obtained (if any) is questioned directly [1269, 1270].

In summary, results across different domains investigating subjective assessment suggest that relative (rank-based) annotations minimize the assumptions made about experiment participants' notions of highly subjective constructs such as player experience, playstyle or design aesthetics. Annotating in a relative fashion, instead of an absolute fashion, leads to the construction of more generalizable and accurate computational models of subjective notions [716, 1269, 1270].

The use of relative labels gradually becomes the dominant approach for soliciting and processing human feedback in games. Relative labels are useful to any of the three main use cases of AI we cover in this book. Such labels may feed preference learning methods (see Sect. 2.3.2) to build surrogate models or content aesthetics [1025], game interest [1271], player believability [165], human-likeness of navigation [764], play style [41], or designer intent [639].

Beyond games, and despite their limitations [989], relative labels define a leading approach for soliciting human feedback for the majority of AI methods that consider subjective notions. Such labels are predominantly used to shape a reward model for subjective notions such as a user's preference for a text summary [1090], the aesthetics of a navigating agent [41] or the goodness of robot control tasks [210]. These reward models would, in turn, train agents to complete various tasks or fine-tune LLMs via

RL, namely **Reinforcement Learning from Human Feedback (RLHF)** [844]. Relative human feedback is also a natural choice for image generation tasks. Indicatively, Stable Diffusion [961] models seem to be able to better align with human preferences and intentions when driven by models of human preference [1257]. This gradual paradigm shift is driven by both the reported benefits of relative labels minimizing the effects of self-reporting subjectivity biases and recent findings demonstrating the advantages of ordinal annotation [755, 1269, 1270].

2.5 Summary

In this first chapter of Part I we presented core AI concepts (representation, utility), learning paradigms (search, optimization and machine learning) and principles for soliciting human feedback. Representation and utility are core concepts to almost all AI methods, as almost all such methods can be seen as search through a space of solutions for the solution with the highest utility; the solution representation is a key determinant of how that search can work and which solutions can be found. In tree search, a path is found from a given state to a state that maximizes utility, a type of problem that is ubiquitous in games and beyond. Optimization is a process for finding the best possible solution in a space defined by a representation, perhaps the most general problem formulation of all; a family of optimization algorithms called evolutionary algorithms feature prominently in much of the work covered in this book.

In machine learning, one searches for models that maximize a utility such as accuracy or cumulative reward. These models can be represented in many different ways, including as types of neural networks, or decision trees. There are various types of machine learning; two of the most prominent ones are supervised learning, where you train a model to predict a label from various features, and self-supervised learning, where you don't have given labels but you learn to predict some aspect of the data from some other aspects, or simply learn to re-produce the data. The various methods referred to as generative AI are all examples of self-supervised learning.

Human feedback is a key element of human-centered AI, and as games are very human-centered we feel that coverage of core concepts of human feedback deserve a place in a chapter about core methods in a AI and games book. There are various types of human feedback, including objective ones you can measure with machines and subjective ones that require human judgment. How to collect this feedback is a complex question, involving when to collect, how, and how to structure the feedback. A key insight is that human experience and description thereof is by its very nature ordinal, and that asking people to report experiences or judgments can easily become meaningless unless you ask them to compare about something else.

The next chapter, which is the final chapter of the first part of the book, covers the core AI methods used in games based on the AI fundamentals covered in this chapter.

Chapter 3
AI Methods for Games

Abstract This chapter presents a set of algorithms that underlie the various AI innovations and applications that are discussed in the rest of the book. This includes methods for tree search, evolutionary computation, unsupervised learning, supervised learning, deep learning, and reinforcement learning. It also gives a brief overview of behavior authoring methods, such as behavior trees, that are common for controlling agents in games. The algorithms are described with a focus on getting the key ideas across, and many of them include pseudocode, but no actual program code. However, the chapter includes ample pointers to other sources for those who wish to read more about specific algorithms. While this chapter is intended as a refresher for readers who already have a working knowledge of AI, it could be used as a mini-textbook in its own right.

From the AI fundamentals presented in the previous chapter we will move onto the specifics of AI methods commonly used in games. As noted earlier, this book assumes that the reader is already familiar with core AI methods at the level of an introductory university course in AI. Therefore, we recommend that the reader is at least cursorily familiar with the methods presented in this chapter before proceeding to read the rest of the book. The algorithm descriptions in this chapter are *high-level descriptions* meant to refresh one's memory if they have learned about the particular algorithm at some previous point, or to explain the general idea of the algorithm if they have never seen it before. Each section comes with pointers to the literature, either research papers or other textbooks, where one can find more details about each method.

Throughout the chapter we use the game of *Ms Pac-Man* (Namco, 1982) as an overarching testbed for all the algorithms we cover. For the sake of consistency, all the methods we cover are employed to *control* Ms Pac-Man's behavior even though they can find a multitude of other uses in this game such as generating content or analyzing player behavior. While a number of other games could have been used as our testbed in this chapter, we picked **Ms Pac-Man** for its popularity and its game design simplicity as well as for its high complexity when it comes to playing the game. It is important to remember that Ms Pac-Man is a *non-deterministic* variant of its ancestor *Pac-Man* (Namco, 1980) which implies that the movements of ghosts involve a degree of randomness.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_3

Behavior authoring, covered in Sect. 3.1, refers to methods employing static ad-hoc representations without any form of search or learning such as finite state machines, behavior trees and utility-based AI. **Tree search**, covered in Sect. 3.2, refers to methods that search the space of future actions and build trees of possible action sequences, often in an adversarial setting; this includes the Minimax algorithm, and Monte Carlo tree search. Section 3.3 presents **local and global search** methods, which refer to local search methods and population-based global stochastic optimization algorithms such as genetic algorithms or evolution strategies. Section 3.4 outlines **unsupervised learning** which refers to algorithms that find patterns (e.g., clusters) in datasets that do *not* have target values. This includes clustering methods such as k-means, hierarchical clustering and self-organizing maps as well as frequent pattern mining methods such as Apriori and generalized sequential patterns. **Supervised learning** (see Sect. 3.5) refers to learning a model that maps instances of datasets to target values such as classes; target values are necessary for supervised learning. Common algorithms used here are backpropagation (artificial neural networks), support vector machines, and decision tree learning. In Sect. 3.6 we present some key algorithms and architectures from the **deep learning** family that are commonly used in games: methods include autoencoders, convolutional neural networks, generative adversarial networks, diffusion models, transformers and large language models. The chapter concludes with a number of notable algorithms from **reinforcement learning**; see Sect. 3.7. Those are methods that solve reinforcement learning problems, where a sequence of actions is associated with positive or negative rewards, but not with a "target value" (the correct action). The paradigmatic algorithm here is temporal difference (TD) learning and its popular instantiation Q-learning. The section also covers methods that combine elements of the algorithms covered to yield **hybrids** such as TD learning with ANN function approximation, proximal policy optimization, and neuroevolution.

3.1 Behavior Authoring

In this section we discuss the first, and arguably most popular, class of AI methods for game development. **Finite state machines**, **behavior trees** and **utility-based AI** are ad-hoc behavior authoring methods that have traditionally dominated the control of non-player characters in games. Their dominance is evident by the fact that the term *game AI* in the game development scene is still nowadays synonymous with the use of these methods.

3.1.1 Finite State Machines

A Finite State Machine (FSM) [382] (and FSM variants such as hierarchical FSMs) is the game AI method that dominated the control and decision making processes of

non-player characters in games up until the mid-2000s. FSMs are expert-knowledge systems and are represented as graphs. An FSM graph is an abstract representation of an interconnected set of objects, symbols, events, actions or properties of the phenomenon that needs to be ad-hoc designed (represented). The FSM can only be in one state at a time; the current state can change to another if the condition in the corresponding transition is fulfilled. In a nutshell, an FSM is defined by three main components:

- A number of **states** which store information about a task—e.g., you are currently on the *explore* state.
- A number of **transitions** between states which indicate a state change and are described by a condition that needs to be fulfilled—e.g., if you hear a fire shot, move to the *alerted* state.
- A set of **actions** that need to be followed within each state—e.g., while in the *explore* state *move randomly* and *seek opponents*.

FSMs are very simple to design, implement, visualize, and debug. Further, they have proven they work well with games over the decades they have been used to implement various aspects of NPC control. However, they don't scale well. Large-scale FSMs can be very complex to design and maintain and are, therefore, computationally limited to certain tasks within game AI. An additional critical limitation of FSMs (and all ad-hoc authoring methods) is that they are not flexible and dynamic (unless purposefully designed). After their design is completed, tested and debugged there is limited room for adaptivity and evolution. As a result, FSMs are great tools for implementing simple and predictable behaviors in games, but not for more complex NPC intelligence. We can, in part, overcome such a drawback by representing transitions as fuzzy rules [889] or probabilities [191].

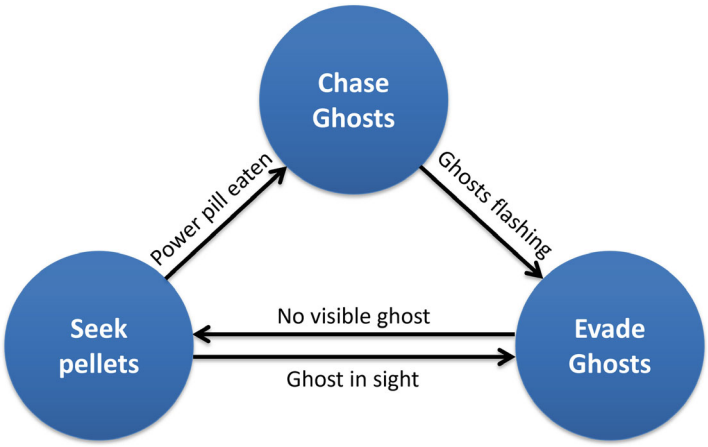


Fig. 3.1 A high-level simplified FSM example for controlling Ms Pac-Man

An FSM for Ms Pac-Man

A hypothetical FSM controller for Ms Pac-Man is illustrated in Fig. 3.1. In this example our FSM has three states (seek pellets, chase ghosts and evade ghosts) and four transitions (ghosts flashing, no visible ghost, ghost in sight, and power pill eaten). While in the *seek pellets* state, Ms Pac-Man moves randomly up until it detects a pellet and then follows a pathfinding algorithm to eat as many pellets as possible and as soon as possible. If a power pill is eaten, then Ms Pac-Man moves to the *chase ghosts* state in which it can use any tree-search algorithm to chase the blue ghosts. When the ghosts start flashing, Ms Pac-Man moves to the *evade ghosts* state in which it uses tree search to evade ghosts so that none is visible within a distance; when that happens Ms Pac-Man moves back to the *seek pellets* state.

3.1.2 Behavior Trees

A Behavior Tree (BT) [192–194] is an expert-knowledge system which, similarly to an FSM, models transitions between a finite set of tasks (or behaviors). Behavior trees were developed in response to the shortcomings of FSMs, to allow for a more structured representation of behavior which could more easily scale to complex NPC behaviors. Behavior trees exist in various versions without a canonical specification, reflecting their distributed development; a key source of inspiration was Bryson's work on robot control architectures [142, 143].

The strength of BTs compared to FSMs is their modularity: if designed well, BTs can yield complex behaviors composed of simple tasks. The main difference between BTs and FSMs (or even hierarchical FSMs) is that they are composed of *behaviors* rather than states. As with finite state machines, BTs are easy to design, test and debug, which made them dominant in the game development scene after their successful application in games such as *Halo 2* (Microsoft Game Studios, 2004) [498] and *Bioshock* (2K Games, 2007).

BTs employ a tree structure with a root node and a number of parent nodes and corresponding child nodes representing behaviors—see Fig. 3.2 for an example. We traverse a BT starting from the root. We then activate the execution of parent-child pairs as denoted in the tree. A child may return the following values to the parent in predetermined time steps (ticks): *run* if the behavior is still active, *success* if the behavior is completed, *failure* if the behavior failed. BTs are composed of three node types: the **sequence**, the **selector**, and the **decorator**, described as follows:

- **Sequence** (see blue rectangle in Fig. 3.2): if the child behavior succeeds, the sequence continues and eventually the parent node succeeds if all child behaviors succeed; otherwise the sequence fails.
- **Selector** (see red rounded rectangle in Fig. 3.2): there are two main types of selector nodes: the *probability* and the *priority* selectors. When a probability selector is used, child behaviors are selected based on parent-child probabilities set by the

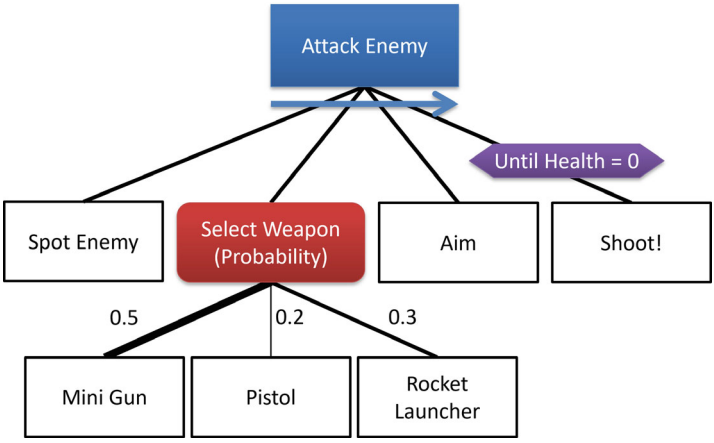


Fig. 3.2 A behavior tree example. The root of the BT is a sequence behavior (attack enemy) which executes the child behaviors *spot enemy*, *select weapon*, *aim* and *shoot* in sequence from left to right. The *select weapon* behavior is a probability selector giving higher probability—denoted by the thickness of the parent-child connecting lines—to the mini gun (0.5) compared to the rocket launcher (0.3) or the pistol (0.2). Once in the *shoot* behavior the decorator *until health = 0* requests the behavior to run until the enemy dies

BT designer. If priority selectors are used, child behaviors are ordered in a list and tried one after the other. Regardless of the selector type used, if the child behavior succeeds the selector succeeds. If the child behavior fails, the next child in the order is selected (in priority selectors) or the selector fails (in probability selectors).

- **Decorator** (see purple hexagon in Fig. 3.2): the decorator node adds complexity to and enhances the capacity of a single child behavior. Decorator examples include the number of times a child behavior runs or the time given to a child behavior to complete the task.

Compared to FSMs, BTs are more flexible to design and easier to test, but they suffer from similar drawbacks. In particular, their dynamicity is rather low given that they are static knowledge representations. The probability selector nodes may add to their unpredictability. (Dynamicity can be enhanced via evolutionary computation methods that may adapt the tree structures of BTs [648].) There is also a certain degree of similarity between BTs and ABL (A Behavior Language) [720] introduced by Mateas and Stern for story-based believable characters; their dissimilarities have also been reported [1244]. For the interested reader there are several extensions to the basic structure of BTs that help them improve on their modularity and their capacity to deal with more complex behavior designs [284, 1037].

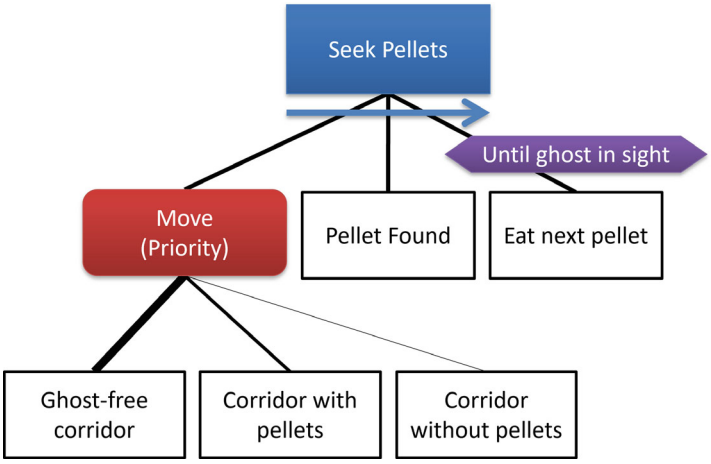


Fig. 3.3 A BT example for Ms Pac-Man. The root of the BT is a sequence behavior for *seek pellets* which executes the child behaviors *move*, *pellet found*, and *eat next pellet* in sequence from left to right. The *move* behavior is a priority selector giving higher priority—denoted by the thickness of the parent-child connecting lines—to ghost-fee corridors, compared to corridors with pellets, and corridors without pellets Once in the *eat next pellet* behavior the decorator *until ghost in sight* requests the behavior to run until a ghost is spotted

A BT for Ms Pac-Man

In Fig. 3.3 we illustrate a simple BT for the *seek pellets* behavior of Ms Pac-Man. While in the *seek pellets* sequence behavior Ms Pac-Man will first *move* (selector), it will then find a pellet and finally it will keep eating pellets until a ghost is found in sight (decorator). While in the *move* behavior—which is a priority selector—Ms Pac-Man will prioritize ghost-free corridors over corridors with pellets and over corridors without pellets.

3.1.3 Utility-Based AI

As has been pointed out by several game AI developers [285, 1008] the lack of behavioral modularity across games and in-game tasks is detrimental for the development of high quality AI. A popular method for behavior authoring that eliminates the modularity limitations of FSMs and BTs is utility-based AI—also known as **utility systems** [266]—which can be used for the design of control and decision making systems in games [705, 926]. Following this approach, instances in the game get assigned a particular utility function that gives a value for the importance of the particular instance [15, 282]. For example, the importance of an enemy being present

at a particular distance or the importance of an agent's health being low in this particular context. Given the set of all utilities available to an agent and all the options it has, utility-based AI decides which is the most important option it should consider at this moment [706]. The utility-based approach is grounded in the utility theory of economics and is based on utility function design. The approach is similar to the design of membership functions in a fuzzy set.

A utility can measure anything from observable data (e.g., enemy health) to subjective notions such as emotions, mood and threat. The various utilities about possible actions or decisions can be aggregated into linear or non-linear formulas and guide the agent to take decisions based on the aggregated utility. The utility values can be checked every n frames of the game. So while FSMs and BTs would examine one decision at a time, utility-based AI architectures examine all available options, assign a utility to them and select the option that is most appropriate (highest utility).

As an example of utility-based AI we will build on the one appearing in [706] for weapon selection. For selecting a weapon an agent needs to consider the following aspects: range, inertia, random noise, ammo and indoors. The *range* utility function adds value to the utility of a weapon depending on the distance to the enemy—for instance, if the distance is short, pistols are assigned higher utility. *Inertia* assigns higher utility value to the current weapon so that changes of weapons are not very frequent. *Random noise* adds non-determinism to the selection so that the agent does not always pick the same weapon given the same game situation. *Ammo* returns a utility about the current level of ammunition and *indoors* penalizes the indoor use of particular weapons such as grenades through a boolean utility function (e.g., 0 utility value if the grenade is used indoors; 1 otherwise). Our agent makes a regular check of the available weapons, assigns utility scores to all of them and selects the weapon with the highest total utility.

Utility-based AI has certain advantages compared to other ad-hoc authoring techniques. It is *modular* as the decision of the game agent is dependent on a number of different factors (or considerations); this list of factors can be dynamic. Utility-based AI is also *extensible* as we can easily author new types of considerations as we see them fit. Finally, the method is *reusable* as utility components can be transferred from one decision to another and from one game to another game. Utility-based AI is thus getting substantial traction in the game industry [283, 926] across game genres and has been featured, among others, in *Kohan 2: Kings of War* (Take Two Interactive and Global Star Software, 2004), in *Iron Man* (Sega, 2008) for controlling the boss, in *Red Dead Redemption* (Rockstar Games, 2010) for weapon and dialog selection, and in *Killzone 2* (Sony Computer Entertainment, 2009) and in *F.E.A.R.* (Sierra Entertainment, 2005) for dynamic tactical decision making [706].

Utility-based AI for Ms Pac-Man

Figure 3.4 illustrates an example of three simple utility functions that could be considered by Ms Pac-Man during play. Each function corresponds to a different behavior that is dependent on the current threat level of the game. The threat level for Ms Pac-Man can be measured as the inverse average Manhattan distance between Ms

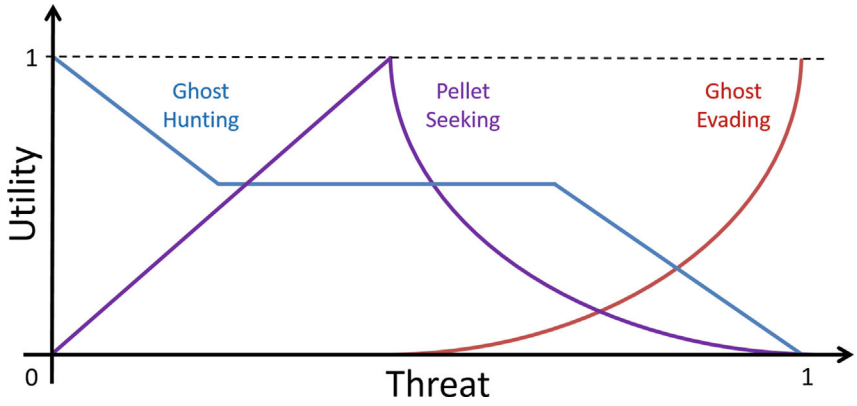


Fig. 3.4 A utility-based approach for controlling Ms Pac-Man behavior. The threat level (x-axis) is a function that lies between 0 and 1 which is based on the current position of ghosts. Ms Pac-Man considers the current level of threat, assigns utility values (through the three different curves) and decides to follow the behavior with the highest utility value. In this example the utility of *ghost evading* rises exponentially with the level of threat. The utility of *ghost hunting* decreases linearly with respect to threat up to a point where it stabilizes; it then decreases linearly as the threat level increases above a threshold value. Finally, the utility of *pellet seeking* increases linearly up to considerable threat level from which point it decreases exponentially

Pac-Man and the four ghosts. Ms Pac-Man selects and executes the behavior with the highest utility value at every tick of the game.

3.1.4 Further Reading

It is important to remember that all three methods covered in this section (and, in general, the methods covered in this chapter) represent the very *basic variants* of the algorithms. As a result, the algorithms we covered appear as static representations of states, behaviors or utility functions. It is possible, however, to create dynamic variants of those by adding non-deterministic or fuzzy elements; for instance, one may employ fuzzy transitions in an FSM [191] or evolve behaviors in a BT [648]. Further, it is important to note that these ad-hoc designed architectures can feature any of the methods this book covers in the remainder of this chapter. Basic processing elements such as an FSM state, a BT behavior or a utility function or even more complex hierarchies of nodes, trees or functions can be replaced by any other AI method yielding hybrid algorithms and agent architectures. Note that possible extensions of the algorithms can be found in the work we cite in the corresponding section of each algorithm.

Details on how to build and test FSMs and hierarchical FSMs can be found in [612]. For behavior trees we recommend the online tutorials and blogposts of Champandard [192, 193] and variations of the vanilla BT method as in [1037].

Finally, the *Behavioral Mathematics for game AI* book [705] is a good starting point for the study of utility-based AI and its application to control and decision making in games. The more recent *Game AI Pro 360* book series [911, 913, 914] detail several popular behavior selection methods such as **goal-oriented action planners** and **hierarchical task networks** [266] that we do not cover in this book. When it comes to software, a BT tool has been integrated within the Unreal Engine[1] while several other BT Unity tools[2] are available for the interested reader.

3.2 Tree Search Methods

As covered in Sect. 2.2.1 a **game tree** is a graph representation of all possible states a system can be in. The system might be a game, or (more commonly) a subset of the game, such as the position of the player character. Any computational method that attempts to solve game trees and play the represented game belongs to the tree search family of methods. The methods presented here can all be characterized as tree search algorithms as they build a **search tree** where the root is the node representing the state the search starts from. Edges in this tree represent actions the agent takes to get from one state to another, and nodes represent states. Because there are typically several different actions that can be taken in a given state, the tree branches. Tree search algorithms mainly differ in which branches are explored and in what order.

For any tree search method to be applicable in games (e.g., to control a game agent), it needs to quickly simulate what would happen if an action to be taken. This is because the tree search might need to explore hundreds, thousands or maybe even millions of nodes before an action is actually taken in the game. A common solution is to use tree search for path planning, only planning movements of an agent and not simulating the rest of the state of the world. It is also possible to plan in a space of logical predicates describing the game world in an abstract form. Or, if the game engine is very fast, it could be possible to plan by simulating the whole game.

3.2.1 Uninformed Search

Uninformed search algorithms are algorithms which search a state space without any further information about the goal. The basic uninformed search algorithms are fundamental computer science algorithms.

Depth-first search is a search algorithm which explores each branch as far as possible before backtracking and trying another branch. At every iteration of its main loop, depth-first search selects a branch and then moves on to explore the resulting node in the next iteration. When a terminal node is reached—one from

[1] https://docs.unrealengine.com/latest/INT/Engine/.

[2] For instance, see http://nodecanvas.paradoxnotion.com/ or http://www.opsive.com/.

which it is not possible to advance further—depth-first search advances up the list of visited nodes until it finds one which has unexplored actions. When used for playing a game, depth-first search explores the consequences of a single move until the game is won or lost, and then goes on to explore the consequences of taking a different move close to the end states.

Breadth-first search does the opposite of depth-first search. Instead of exploring all the consequences of a single action, breadth-first search explores all the actions from a single node before exploring any of the nodes resulting from taking those actions. So, all nodes at depth one are explored before all nodes at depth two, then all nodes at depth three, etc.

While the aforementioned are fundamental uninformed search algorithms, there are many variations and combinations of these algorithms, and new uninformed search algorithms are being developed. More information about uninformed search algorithms can be found in Chap. 4 of [975].

It is rare to see uninformed search algorithms used effectively in games, but there are exceptions such as iterative width search [104], which does surprisingly well in general video game playing, and the use of breadth-first search to evaluate exploration in strategy game maps of *Sentient Sketchbook* [640]. Moreover, it is often illuminating to compare the performance of state-of-the-art algorithms with a simple uninformed search algorithm.

Uninformed Search for Ms Pac-Man

A depth-first approach in Ms Pac-Man would normally consider the branches of the game tree until Ms Pac-Man either completes the level or dies. The outcome of this search for each possible action would determine which action to take at a given moment. Breadth-first instead would first explore all possible actions of Ms Pac-Man at the current state of the game (e.g., going left, up, down or right) and would then explore all their resulting nodes (children) and so on. The game tree of either method is too big and complex to visualize within a Ms Pac-Man example and is not practically applicable for Ms Pac-Man control.

3.2.2 Best-First Search

In best-first search, the expansion of nodes in the search tree is informed by some knowledge about the goal state. In general, the node that is closest to the goal state by some criterion is expanded first. The most well-known best-first search algorithm is A* (pronounced A star). The A* algorithm keeps a list of "open" nodes, which are next to an explored node but which have not themselves been explored. For each open node, an estimate of its distance from the goal is made. New nodes are chosen to explore based on a lowest cost basis, where the cost is the distance from the origin node plus the estimate of the distance to the goal.

A* can easily be understood as navigation in two- or three-dimensional space. Variants of this algorithm are therefore commonly used for **pathfinding** in games. In many games, the "AI" essentially amounts to non-player characters using A* pathfinding to traverse scripted points. Numerous modifications of this basic algorithm have been proposed for large and deceptive spaces, including hierarchical versions of A* [111, 1092], real-time heuristic search [148], jump point search for uniform-cost grids [428], navmesh-based pathfinding algorithms [123], planning algorithms for dynamic game worlds [818] that enable the animation of crowds in collision-free paths [1047] and approaches for pathfinding in navigation meshes [123, 1181]. The work of Rabin and Sturtevant on grid-based pathfinding [916, 1094] and pathfinding architectures [915] are notable examples. Sturtevant and colleagues have also been running a dedicated competition to **grid-based path-planning** [1097] since 2012.[3] For the interested reader Sturtevant [1095] has released a list of benchmarks for 2D, 3D, and multi-agent [1086] pathfinding in games[4] including *Dragon Age: Origins* (Electronic Arts, 2009), *StarCraft* (Blizzard Entertainment, 1998) and *Warcraft III: Reign of Chaos* (Blizzard Entertainment, 2002).

However, A* can also be used to search in the space of game states, as opposed to searching physical locations. This way, best-first search can be used for **planning** rather than just navigation. The difference is in taking the changing state of the world (rather than just the changing state of a single agent) into account. Planning with A* can be surprisingly effective, as evidenced by the winner of the 2009 **Mario AI Competition**—where competitors submitted agents playing *Super Mario Bros* (Nintendo, 1985)—being based on a simple A* planner that simply tried to get to the right end of the screen at all times [1159, 1171] (see also Fig. 3.5). Crucially, the state was the *whole state of the game*, including positions of enemies, rather than just the position of Mario. This makes the difference between planning and mere pathfinding.

Best-first Search for Ms Pac-Man

Best-first search can be applicable in Pac-Man in the form of A*. Following the paradigm of the 2009 Mario AI competition champion, Ms Pac-Man can be controlled by an A* algorithm that searches through possible game states within a short time frame and takes a decision on where to move next (up, down, left or right). The game state can be represented in various ways: from a very direct, yet costly, representation that takes ghost and pellet coordinates into account to an indirect representation that considers the distance to the closest ghost or pellet. Regardless of the representation chosen, A* requires the design of a cost function that will drive the search. Relevant cost functions for Ms Pac-Man would normally reward moves to areas containing pellets and penalizing areas containing ghosts.

[3] http://movingai.com/GPPC/.

[4] http://movingai.com/benchmarks/.

Fig. 3.5 The A* controller of the 2009 Mario AI Competition champion by Baumgarten [1159]. The red lines illustrate possible future trajectories considered by the A* controller of Mario, taking the dynamic nature of the game into account

3.2.3 Minimax

For single-player games, simple uninformed or informed search algorithms can be used to find a path to the optimal game state. However, for two-player adversarial games, there is another player that tries to win as well, and the actions of each player depend very much on the actions of the other player. For such games we need adversarial search, which includes the actions of two (or more) adversarial players. The basic adversarial search algorithm is called *Minimax*. This algorithm has been used very successfully for playing classic perfect-information two-player board games such as Checkers and Chess, and was in fact (re)invented specifically for the purpose of building a Chess-playing program [1191].

The core loop of the Minimax algorithm alternates between player 1 and player 2—such as the white and black player in Chess—named the *min* and the *max* player. For each player, all possible moves are explored. For each of the resulting states, all possible moves by the other player are also explored, and so on until all the possible combinations of moves have been explored to the point where the game ends (e.g., with a win, a loss or a draw). The result of this process is the generation of the whole game tree from the root node down to the leaves. The outcome of the game informs the utility function which is applied onto the leaf nodes. The utility

function estimates how good the current game configuration is for a player. Then, the algorithm traverses up the search tree to determine what action each player would have taken at any given state by backing-up values from leaves through the branch nodes. In doing so, it assumes that each player tries to play optimally. Thus, from the standpoint of the *max* player, it tries to maximize its score, whereas the *min* player tries to minimize the score of the *max* player; hence, the name *Minimax*. In other words, a *max* node of the tree computes the max of its child values whereas a *min* node computes the min of its child values. The optimal winning strategy is then obtained for *max* if, on *min*'s turn, a win is obtainable for *max* for *all moves* that *min* can make. The corresponding optimal strategy for *min* is when a win is possible independently of what move *max* will take. To obtain a winning strategy for max, for instance, we start at the root of the tree and we iteratively choose the moves leading to child nodes of highest value (on min's turn the child nodes with the lowest value are selected instead). Figure 3.6 illustrates the basic steps of Minimax through a pruned tree example of Tic-Tac-Toe.

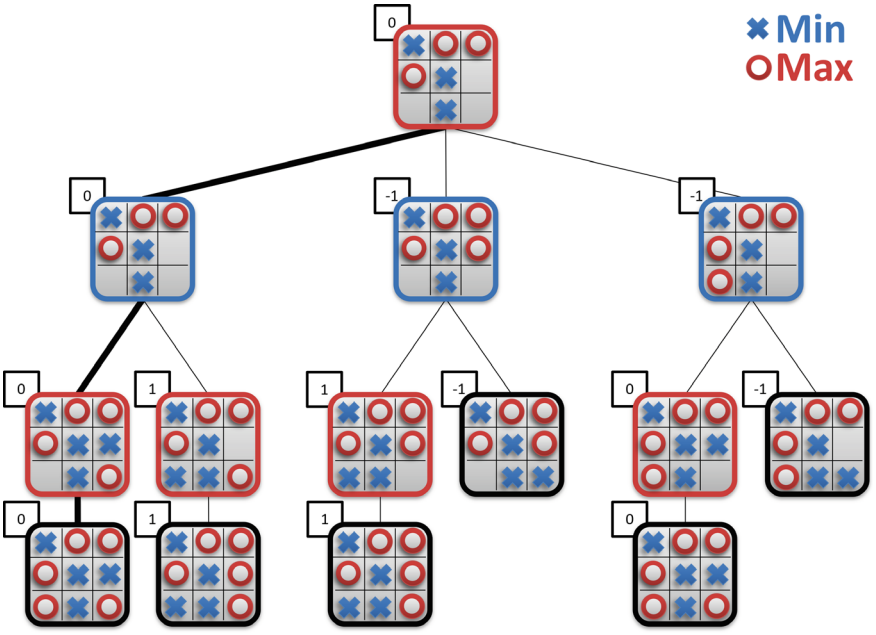


Fig. 3.6 Illustrating the Minimax algorithm via a branch of the Tic-Tac-Toe game tree. Note that we visualize only 14 out of the $9! = 362,880$ board configurations possible in this game. In this game scenario both players have each played three times already (see root of the tree). Player *max* (represented as red Os) plays first at the root of the tree, then player *min* (represented as blue Xs) plays second, and then max plays one last time if possible. Boards with black outline denote terminal nodes containing a winning (1), a losing (-1) or a draw (0) score for the max player. Following the Minimax strategy, the scores in black squares (utility) next to each node (Tic-Tac-Toe board) are traversed up to the root of the game tree. The optimal play for max and min is illustrated with bold lines. In this simple example if both players play optimally, the game is a draw

Of course, exploring all possible moves and countermoves is infeasible for any game of interesting complexity, as the size of the search tree increases exponentially with the depth of the game or the number of moves that are simulated. Indicatively, Tic-Tac-Toe has a game tree size of $9! = 362,880$ states which is feasible to traverse through but still not easy to visualize. The Chess game tree, however, has approximately 10^{154} nodes which is infeasible to search through with modern computers. Therefore, almost all actual applications of the Minimax algorithm cut off search at a given depth, and use a **state evaluation** function to evaluate the desirability of each game state at that depth. For example, in Chess a simple state evaluation function would be to merely sum the number of white pieces on the board and subtract the number of black pieces; the higher this number is, the better the situation is for the white player. Of course, much more sophisticated board evaluation functions are commonly used. Together with improvements to the basic Minimax algorithm such as α-β **pruning** [992] and the use of non-deterministic state evaluation functions, some very competent programs emerged for many classic games (e.g., IBM's Deep Blue). More information about Minimax and other adversarial search algorithms can be found in Chap. 6 of [975].

Minimax for Ms Pac-Man

Strictly speaking, Minimax is not applicable to Ms Pac-Man as the game is non-deterministic and, thus, the Minimax tree is formally unknown. (Of course Minimax variants with heuristic evaluation functions can be eventually applicable.) Minimax is however applicable to Ms Pac-Man's deterministic ancestor, *Pac-Man* (Namco, 1980). Again strictly speaking, Pac-Man is a single-player adversarial game. As such, Minimax is applicable only if we assume that Pac-Man plays against adversaries (ghosts) who make optimal decisions. It is important to note that ghosts' movements are not represented by tree nodes; instead, they are simulated based on their assumed optimal play. Game tree nodes in Pac-Man may represent the game state including the position of Pac-Man, the ghosts, and the current pellets and power pills available. The branches of the Minimax tree are the available moves of the Pac-Man in each game state. The terminal nodes can, for instance, feature either a binary utility (1 if Pac-Man completes the level; 0 if Pac-Man was killed by a ghost) or the final score of the game.

3.2.4 Monte Carlo Tree Search

There are many games which Minimax will not play well. Games with a high **branching factor** (where there are many potential actions to take at any given point in time) lead to Minimax that will only ever search a very shallow tree. Another aspect of games that contests Minimax is when it is hard to construct a good state evaluation function. The board game Go is a deterministic, perfect information game that is a

good example of both of these challenges. Go has a branching factor of approximately 300, whereas Chess typically has around 30 actions to choose from. The positional nature of Go, which is all about surrounding the adversary, makes it very hard to correctly estimate the value of a given board state. For a long time, the best Go-playing programs in the world, most of which were based on Minimax, could barely exceed the playing strength of a human beginner. In 2007, *Monte Carlo Tree Search* (MCTS) was invented and the playing strength of the best Go programs increased drastically. Beyond complex perfect information, deterministic games such as Go, Chess and Checkers, **imperfect information** games such a Battleship, Poker, Bridge and/or **non-deterministic** games such as backgammon and monopoly cannot be solved via Minimax due to the nature of the algorithm. In such games, MCTS not only overcomes the tree size limitation of Minimax but, given sufficient computation, it approximates the Minimax tree of the game.

So how does MCTS handle high branching factors, lack of good state evaluation functions, and lack of perfect information and determinism? To begin with, it does not search all branches of the search tree to an even depth, instead it concentrates on the more promising branches. This makes it possible to search certain branches to a considerable depth even though the branching factor is high. Further, to get around the lack of good evaluation functions, determinism and imperfect information, the standard formulation of MCTS uses **rollouts** to estimate the quality of the game state, randomly playing from a game state until the end of the game to see the expected outcome (e.g., win, loss or draw). The utility values obtained via the random simulations may be used efficiently to adjust the policy towards a best-first strategy (a Minimax tree approximation).

At the start of a run of the MCTS algorithm, the tree consists of a single node representing the current state of the game. The algorithm then iteratively builds a search tree by adding and evaluating new nodes representing game states. This process can be interrupted at any time, rendering MCTS an **anytime** algorithm. MCTS requires only two pieces of information to operate: the *game rules* that would, in turn, yield the available moves in the game and the *terminal state evaluation*—whether that is win, a loss, a draw, or a game score. The vanilla version of MCTS does not require a heuristic function, which is, in turn, a key advantage over Minimax.

The core loop of the MCTS algorithm can be divided into four steps: **Selection**, **Expansion** (the first two steps are also known as *tree policy*), **Simulation** and **Backpropagation**. The steps are also depicted in Fig. 3.7. We detail the basic steps of the algorithm within the text-box that follows; we will be using this presentation format for most methods we cover in the remainder of this chapter.

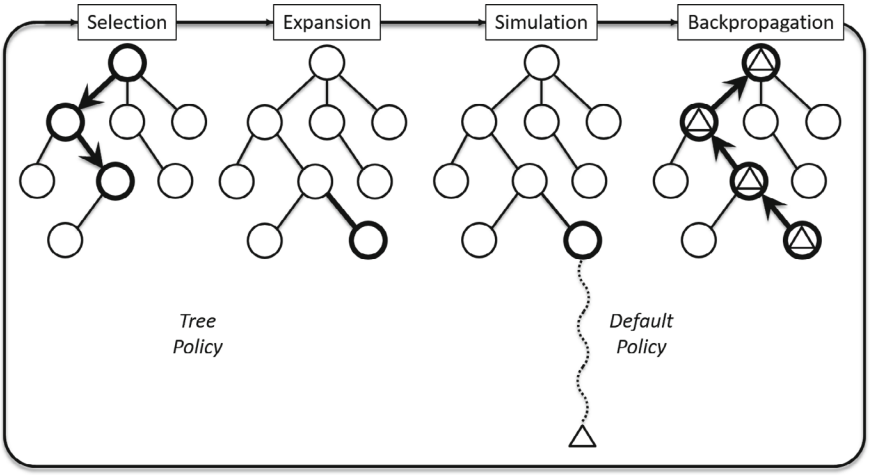


Fig. 3.7 The four basic steps of MCTS exemplified through one iteration of the algorithm. The figure is a recreation of the corresponding MCTS outline figure by Chaslot et al. [202]

Selection: In this phase, it is decided which node should be expanded. The process starts at the root of the tree, and continues until a node is selected which has unexpanded children. Every time a node (action) is to be selected within the existing tree a child node j is selected to maximise the UCB1 formula:

$$\text{UCB1} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{3.1}$$

where $\overline{X}_j$ is the average reward of all nodes beneath this node, C_p is an exploration constant (often set to $1/\sqrt{2}$), n is the number of times the parent node has been visited, and n_j is the number of times the child node j has been visited. While UCB1 is the most popular formula used for action selection it is certainly not the only one available. Beyond Eq. (3.1) other options include ϵ-greedy, Thompson sampling, and Bayesian bandits [137]. For instance, Thompson sampling selects actions stochastically based on their posterior probabilities of being optimal [1140].

Expansion: When a node is selected that has unexpanded children—i.e., that represents a state from which actions can be taken that have not been attempted yet—one of these children is chosen for *expansion*, meaning that a simulation is done starting in that state. Selecting which child to expand is often done at random.

Simulation (Default Policy): After a node is expanded, a simulation (or *roll-out*) is done starting from the non-terminal node that was just expanded until

the end of game to produce a value estimate. Usually, this is performed by taking random actions until a termination state is reached, i.e., until the game is either won or lost. The state at the end of the game (e.g., -1 if losing, $+1$ if winning) is used as the reward (Δ) for this simulation, and propagated up the search tree.

Backpropagation: The reward is added to the total reward of the new node, X so as $X \leftarrow X + \Delta$. It is also "backed up": Δ is added to the total reward of its parent node, its parent's parent and so on until the root of the tree.

The simulation step might appear counter-intuitive—taking random actions seems like no good way to play a game—but it provides a relatively unbiased estimate of the quality of a game state. Essentially, the better a game state is, the more simulations are likely to end up winning the game. At least, this is true for games like Go where a game will always reach a terminal state within a certain relatively small number of moves (400 for Go). For other games like Chess, it is theoretically possible to play an arbitrary number of moves without winning or losing the game. For many video games, it is *probable* that any random sequence of actions will not end the game unless some timer runs out, meaning that most simulations will be very long (tens or hundreds of thousands of steps) and not yield useful information. For example, in *Super Mario Bros* (Nintendo,1985), the application of random actions would most likely make Mario dance around his starting point until his time is up [502]. In many cases it is therefore useful to complement the simulation step with a state evaluation function (as commonly used in Minimax), so that a simulation is performed for a set number of steps and if a terminal state is not reached a state evaluation is performed in lieu of a win-lose evaluation. In some cases it might even be beneficial to replace the simulation step entirely with a state evaluation function. It is worth noting that there are many variations of the basic MCTS algorithm—it may in fact be more useful to see MCTS as an algorithm family or framework rather than a single algorithm.

MCTS for Ms Pac-Man

While MCTS can be employed for the real-time control of Ms Pac-Man a number of challenges become critical. First, the algorithm has limited rollout computational budget which increases the importance of heuristic knowledge. Second, the action space can be particularly fine-grained which suggests that macro-actions are a more powerful way to model the game tree; otherwise the agent's planning will be very short-term. Third, there might be no terminal node in sight which calls for good heuristics and possibly restricting the simulation depth.

The MCTS agent of Pepels et al. [863] managed to cope with all the aforementioned challenges of employing MCTS for real-time control by using a restricted game tree and a junction-based game state representation (see Fig. 3.8). Their agent, named Maastricht, managed to obtain over 87, 000 points and was ranked first (among

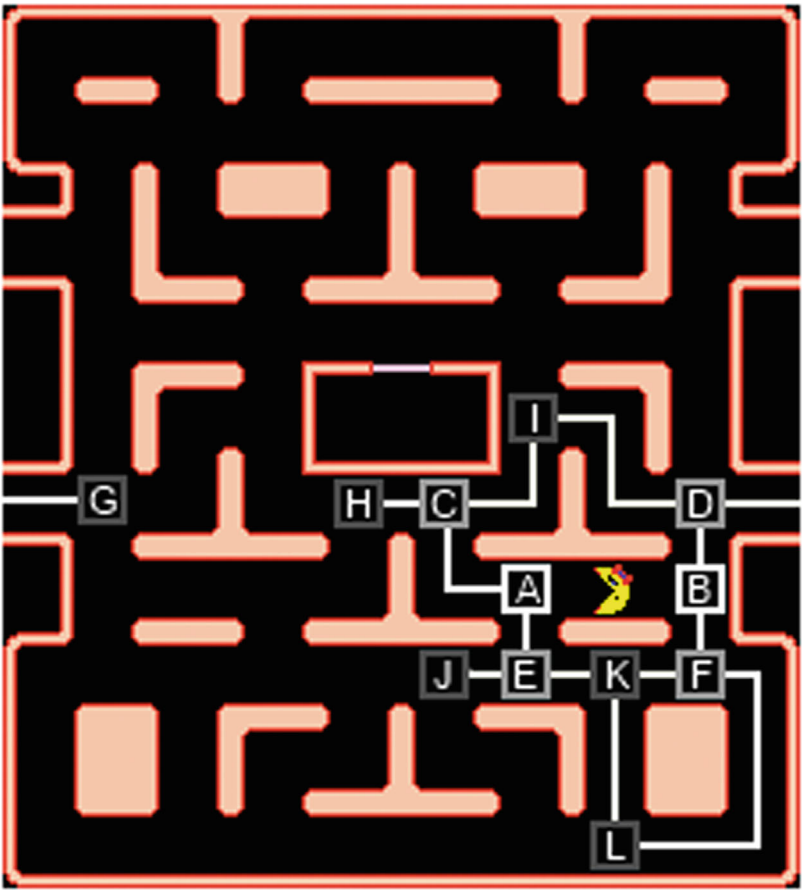


Fig. 3.8 The junction-based representation of a game state for the Maastricht MCTS controller [863]. All letter nodes refer to game tree nodes (decisions) for Ms Pac-Man. Imaged adapted from [863] with permission from authors

36 agents) in the Ms Pac-Man competition of the IEEE Computational Intelligence and Games conference in 2012. Following their approach we can represent the game's mazes directly as a graph where the junctions are nodes, and corridors between junctions are edges. The decisions Ms Pac-Man takes are the moves made at the nodes of the tree (i.e., maze junctions). As the tree is viewed from Ms Pac-Man's single-player perspective the movements of the ghosts are not represented but they are rather simulated while Ms Pac-Man traverses the tree. The reward obtained from simulations may entail aspects of ghost behavior, and the existent patterns of pellets and power pills as in [863].

3.2.5 Further Reading

The basic search algorithms are well covered in Russell and Norvig's classic AI textbook [975]. The A* algorithm was invented in 1972 for robot navigation [429]; a good description of the algorithm can be found in Chap. 4 of [975]. More advanced material on tailoring and optimizing this algorithm for specific game problems can be found in dedicated game AI books such as [907]. The different components of Monte Carlo tree search [241] in the context of playing Go [242] are also recommended readings. A good overview of and introduction to MCTS and some of its variants is given in a survey paper by Browne et al. [137].

3.3 Local and Global Search Methods

Optimization algorithms search the space of solutions to some problem directly. This could be the problem of playing a game best, where the solution space is a space of policies. It could also the problem of best challenging or entertaining the player with a game level, where the solution space is a space of levels, or the problem of retaining players in a casual game, where the optimization algorithms could search a space of strategies for rewarding or challenging the player.

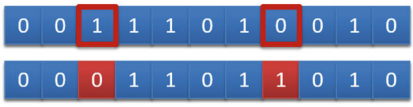
For an optimization algorithm to work, we need a good representation of the solutions. For example, policies could be represented as neural networks or behavior trees; levels could be represented as tile-maps or more abstract level-building scripts. We also need a way to traverse the solution space by changing the solutions slightly, for example adding random values to the neural network weights or adding or removing nodes to the behavior tree. But most importantly, we need a utility function, which tells us how good the solution is. The optimization algorithm searches the space to find solutions which maximize the utility (known as objective in search). For example, the objective for a gameplaying policy could be the score it achieves at the end of the game, whereas the objective for a level could be based on an agent playing the level to evaluate its quality. While their key concepts are outlined in Sect. 2.2.2, here we will discuss two families of optimization methods that are popular in AI and games: local search algorithms, and evolutionary algorithms.

3.3.1 Local Search

The simplest optimization algorithms are the local optimization algorithms. These are so called because they only search "locally", in a small part of the search space, at any given time. A local optimization algorithm generally just keeps a single solution candidate at any given time, and explores variations of that solution. Arguably the simplest possible optimization algorithm is the **hill climber**. In its most common formulation, which we can call the deterministic formulation, it works as follows:

1. *Initialization:* Create a solution s by choosing a random point in search space. Evaluate its utility (or fitness) function. (For example, evaluate the height of a location s of a game level.)
2. Generate all possible neighbors of s. A neighbor is any solution that differs from s by at most a certain given distance. (For example, generate all discrete locations of a level that are within a predetermined Euclidean distance from location s.)
3. Evaluate all neighbors with the fitness function.
4. If none of the neighbors has a better fitness score than s, exit the algorithm and return s.
5. Otherwise, replace s with the neighbor that has the highest fitness value and go to step 2.

The deterministic hill climber is only practical when the representation is such that each solution has a small number of neighbors. In many representations there is an astronomically high number of neighbors. It is therefore preferable to use variants of hill climbers that may guide the search effectively. One approach is the **gradient-based hill climber** that follows the gradient towards minimizing a cost function. Another approach that we cover here is the **randomized hill climber**. This instead relies on the concept of **mutation**: a small, random change to a solution. For example, a string of bits can be mutated by randomly flipping one or two bits (see Fig. 3.9), and a vector of real numbers can be mutated by adding another vector to it drawn from a random distribution around zero, and with a very small standard deviation. Other types of mutation such as gene **inversion** can also be applied as visualized in Fig. 3.9. Given a representation, fitness function and mutation operator, the randomized hill climber works as follows:



(a) **Mutation**: A number of genes is selected to be mutated with a small probability e.g., less than 1%. The selected genes are highlighted with a red outline at the top chromosome and are mutated by flipping their binary value (red genes) at the bottom chromosome.

(b) **Inversion**: Two positions in the offspring are randomly chosen and the positions between them—the gene sequence highlighted by a red outline at the top chromosome—are inversed (red genes) at the bottom chromosome.

Fig. 3.9 Two ways of mutating a binary chromosome. In this example we use a chromosome of eleven genes. A chromosome is selected (top bit-string) and mutated (bottom bit-string)

1. *Initialization*: Create a solution s by choosing a random point in the search space. Evaluate its fitness.
2. *Mutation*: Generate an offspring s' by mutating s.
3. *Evaluation*: Evaluate the fitness of s'.
4. *Replacement*: If s' has higher fitness than s, replace s with s'.
5. Go to step 2.

While very simple, the randomized hill climber can be surprisingly effective. Its main limitation is that it is liable to get stuck in local optima. A **local optimum** is sort of a "dead end" in search space from which there is "no way out"; a point from which there are no better points within the immediate vicinity. There are many ways of dealing with this problem. One is to simply restart the hill climber at a new randomly chosen point in the search space whenever it gets stuck. Another is **simulated annealing**, to accept moving to solutions with *lower* fitness with a given probability; this probability gradually diminishes during the search. A far more popular response to the problem of local optima is to keep not just a single solution at any time, but a *population* of solutions (see Sect. 3.3.2).

Local Search for Ms Pac-Man

There are many ways in which one could apply local search in Ms Pac-Man. Most directly, you could do local search in physical space, e.g. best-first search to find a path to the nearest pill. Each node expansion would correspond to moving a to a neighboring position (grid cell). This method can be effective, but suffers from the problem that it does not simulate the movement of the ghosts, so it could easily find paths that lead straight into the ghosts.

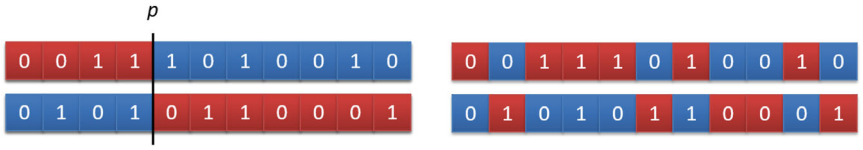
A more sophisticated approach would be to use local search in the space of short plans (action sequences), which are evaluated by simulation; this, obviously, requires a fast forward model. Here, a "point in space" would be a short sequence of actions, maybe 10 actions long. Node expansion would mean mutating this sequence by changing it randomly. To evaluate the quality (or fitness) of a plan, the plan is executed in simulation (where the simulation includes the movement of the ghosts) and the score difference between the start and final state of the plan would be used as a quality measure. If Ms Pac-Man dies, that results in a large negative value. Breadth-first search in this space could work well, but might be computationally expensive.

3.3.2 *Evolutionary Algorithms*

Evolutionary algorithms are randomized **global** optimization algorithms; they are called global rather than local because they search many points in the search space simultaneously, and these points can be far apart. They accomplish this by keeping a population of solutions in memory at any given time. The general idea of evolutionary computation is to optimize by generating many solutions, throw away the bad ones and keep the good (or at least less bad) ones, and create new solutions from the good ones.

The idea of keeping a population is taken from Darwinian evolution [265] by natural selection, from which evolutionary algorithms also get their name. The size of the population is one of the key parameters of an evolutionary algorithm; a population size of 1 yields something like a randomized hill climber, whereas populations of several thousand solutions are not unheard of.

Another idea which is taken from evolution in nature is **crossover**, also called *recombination*. This is the equivalent of sexual reproduction in the natural world; two or more solutions (called *parents*) produce an offspring by combining elements of themselves. The idea is that if we take two good solution s, a solution that is a combination of these two ought to be good as well, maybe even better than the parents. The crossover operator is highly dependent on the solution representation. Two types of crossover operators are popular when the solution is represented as a string or a vector. First, *uniform crossover* flips a fair coin and randomly picks values from each parent for each position in the offspring, Second, *one-point crossover* picks a random position p in the offspring and then assigns the values of positions before and after p, respectively, from parent 1 and parent 2. Crossover can be applied to any chromosome representation varying from a bit-string to a real-valued vector. Figure 3.10 illustrates these two crossover operators. It is in no way guaranteed, however, that the crossover operator generates an offspring that is anything as highly fit as the parents. In many cases, crossover can be highly destructive. If crossover is



(a) **1-point crossover**: The vertical line across the two parents denotes the crossover point at position p.

(b) **Uniform crossover**: To select genes from each parent to form offspring the operator flips a fair coin at each position of the chromosome.

Fig. 3.10 Two popular types of crossover used in evolutionary algorithms. In this example we use a *binary* representation and a chromosome size of eleven genes. The two bit-strings used in both crossover operators represent the two parents selected for recombination. Red and blue genes represent the two different offspring emerged from each crossover operator. Note that the operators are directly applicable to *real-valued* (floating point) representations too

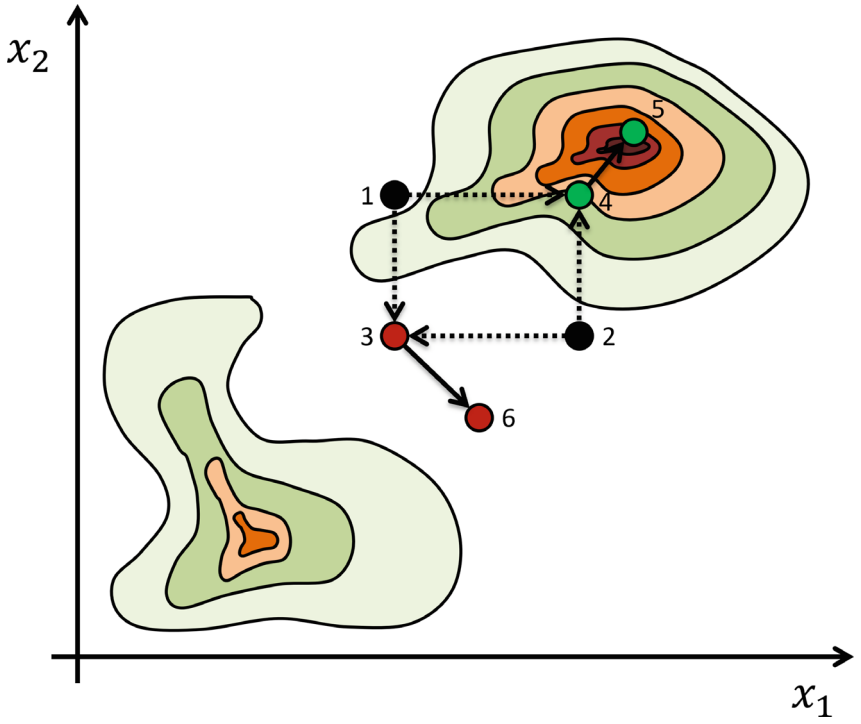


Fig. 3.11 An illustration of the mutation and crossover operators in a simplified two-dimensional fitness landscape. The problem is represented by two real-valued variables (x_1 and x_2) that define the two genes of the vector chromosome. The fitness landscape is represented by the contour lines on the 2D plane (the darker the color the higher the fitness). Chromosomes 1 and 2 are selected to be parents. They are recombined via 1-point crossover (dotted arrows) which yields offspring 3 and 4. Both offspring are mutated (solid arrows) to yield solutions 5 and 6. Operators that lead to poorer-fit or higher-fit solutions compared to their parents are, respectively, depicted with green and red color

used, it is therefore important that the specific operator is chosen with care for each problem. Figure 3.11 illustrates this possibility through a simple two-dimensional example.

The basic template for an evolutionary algorithm is as follows:

1. *Initialization:* The population is filled with P solutions created randomly, i.e., random points in the search space. Known highly-fit solutions can also be added to this initial population.
2. *Evaluation:* The fitness function evaluates all solutions in the population and assign fitness values to them.

3. *Parent selection:* Based on fitness and possibly other criteria, such as distance between solutions, those population members that will be used for reproduction are selected. Selection strategies include methods directly or indirectly dependent on the fitness of the solutions, including roulette-wheel (proportionally to fitness), ranking (proportionally to rank in population) and tournament.
4. *Reproduction:* Offspring are generated by crossover from parents, or by simply copying parent solutions.
5. *Variation:* Mutation is applied to some or all of the offspring.
6. *Replacement:* In this step, we select which of the parents and/or offspring will make it to the next generation. Popular replacement strategies of the current population include the *generational* (parents die; offspring replace them), *steady state* (offspring replaces worst parent if offspring is better) and *elitism* (generational, but best $x\%$ of parents survive) approaches.
7. *Termination:* Are we done yet? Decide to terminate the algorithm based on how many generations or evaluations have elapsed (*exhaustion*), the highest fitness attained by any solution (*success*), and/or some other termination condition.
8. Go to step 2.

Every iteration of the main loop (i.e., every time we reach step 2) is called a **generation**, keeping with the nature-inspired terminology. The total number of fitness evaluations performed is typically proportional to the size of the population times the number of generations.

This high-level template can be implemented and expanded in a myriad different ways; evolutionary or evolution-like algorithm variants rearrange the overall flow, add new steps or remove existing steps. To make this template a bit more concrete, next we will give a simple example of a working evolutionary algorithm. This is a form of **evolution strategy** [85], one of the main families of evolutionary algorithms. While the $\mu + \lambda$ evolution strategy is a simple algorithm that can be implemented in 10 to 20 lines of code, it is a fully functional global optimizer and quite useful. The two main parameters are μ, which signifies the "elite" or the size of the part of the population that is kept every generation, and λ, the size of the part of the population that is re-generated every generation.

1. Fill the population with $\mu + \lambda$ randomly generated solutions.
2. Evaluate the fitness of all solutions.
3. Sort the population by decreasing fitness.
4. Remove the least fit λ individuals.
5. Replace the removed individuals with offspring, copies of the μ best individuals.
6. Mutate the offspring.
7. Stop if success or exhaustion. Otherwise go to step 2.

Evolution strategies are characterized by a reliance on mutation rather than crossover to create variation, and by the use of self-adaptation to adjust mutation parameters (though that is not part of the simple algorithm). They are also generally well suited to optimize artifacts represented as vectors of real numbers, so-called continuous optimization. Some of the very best algorithms for continuous optimization, such as the covariance matrix adaptation evolution strategy (CMA-ES) [427] and the natural evolution strategy (NES) [1249], are conceptual descendants of this family of algorithms.

Another prominent family of evolutionary algorithms is **genetic algorithms** (GAs) [771]. These are characterized by a reliance on crossover rather than mutation for variation (some genetic algorithms have no mutation at all), fitness-proportional selection and solutions being often represented as bit-strings as opposed to strings of real values. It should be noted, however, that the distinctions between different types of evolutionary algorithms are mainly based on their historical origins. These days, there is such extensive hybridization that it often makes little sense to categorize a particular algorithm as belonging to one or the other family.

Constrained Optimization

A subfield of evolutionary computation emerges from the need of satisfying particular constraints within which a solution is not only fit but also *feasible*. For example, we may calculate the fitness of a trajectory for an NPC, but only if it takes feasible actions; if the trajectory takes actions that would be undefined at some point, it is not merely unfit but completely broken, i.e., *infeasible*. This is called constrained (or constraint) optimization. When evolutionary algorithms are used for constrained optimization we are faced with a number of challenges such as that mutation and crossover cannot in general preserve or guarantee the feasibility of a solution. It may very well be that a mutation or a recombination between two parents may yield an infeasible offspring. One approach to deal with constraint handling is **repair**, which could be any process that turns infeasible individuals into feasible ones. A second approach is to modify the genetic operators so that the probability of an infeasible individual to appear becomes smaller. A popular approach is to merely penalize the existence of infeasible solutions by assigning them low fitness values

or, alternatively, in proportion to the number of constraint violations [219]. This strategy however may over-penalize the actual fitness of a solution which in turn will result in its rapid elimination from the population. Such a property might be undesirable and is often accused for the weak performance of evolutionary algorithms on handling constraints [756]. As a response to this limitation the **feasible-infeasible 2-population** (FI-2pop) algorithm [565] evolves two populations, one with feasible and one with infeasible solutions. The infeasible population optimizes its members towards minimizing the distance from feasibility. As the infeasible population converges to the border of feasibility, the likelihood of discovering new feasible individuals increases. Feasible offspring of infeasible parents are transferred to the feasible population, boosting its diversity (and vice versa for infeasible offspring). FI-2pop has been used in games on instances where we require fit and feasible solutions such as playable game levels [634, 640, 1071].

Multiobjective Evolutionary Algorithms

Another subfield of evolutionary computation studies algorithms that consider more than one objective when attempting to find a solution to a problem. For many problems it is hard to combine all requirements and specifications into a single objective measure. It is also often the case that these objectives are conflicting; for instance, if our objectives are to buy the fastest and cheapest possible laptop we will soon realize the two objectives are partially conflicting.

The most intuitive solution is to merely add the different objective values—as a weighted sum—and use this as your fitness under optimization [642]. Doing so, however, has several drawbacks such as the non-trivial ad-hoc design of the weighting among the objectives, the lack of insight on the interactions between the objectives (e.g., what is the price threshold above which faster laptops are not more expensive?) and the fact that a weighted-sum single-objective approach cannot reach solutions that achieve an optimal compromise among their weighted objectives.

The response to these limitations is the family of algorithms known as **multiobjective evolutionary algorithms**. A multiobjective evolutionary algorithm considers at least two objective functions—that are partially conflicting—and searches for a **Pareto front** of these objectives. The Pareto front contains solutions that are non-dominated, meaning that no other solution is better ("dominates") in all respects. The true Pareto front would contain solely solutions that cannot be improved in one objective without worsening in another, but the actual Pareto front is in most cases merely an approximation of the true front. Various multiobjective evolutionary algorithms employ different strategies for retaining the population and selecting which solutions to use as the basis for reproduction. A widely used algorithm is the Nondominated Sorting Genetic Algorithm II, or NSGA II [270], which sorts and evaluates individuals depending on their relative position across objectives. In particular, NSGA-II uses a form of binary tournament selection strategy through which each individual of the population is first selected via rank-based methods on the entire population (non-dominated sorting). During the second round of sorting, individuals are selected via their crowding distance to other individuals. The core

advantage of evolutionary algorithms, when employed for multi-objective optimization is their global search capacity. By generating sets of solutions they allow a better approximation of the Pareto front as a whole. Evolution, however, normally comes with increased computational cost.

Unlike a standard single-objective evolutionary algorithm, where you simply use the solution with the highest fitness, it is not always clear which solution to use after a multiobjective evolutionary search has concluded. All of the non-dominated solutions are in their own sense "best". One might decide to use some formula that combines the multiple objectives into one to select the overall best solution (notice that there are still advantages to using multiobjective algorithms even if the objectives are eventually combined, as a multiobjective algorithm might search the space better), or perhaps use different solutions for different problem instances, or offer the player a choice.

Further details about multiobjective optimization by means of evolutionary algorithms can be found in [220, 270]. The approach is applicable in game AI on instances where more than one objective is relevant for the problem we attempt to solve: for instance, we might wish to optimize both the balance and the asymmetry of a strategy game map [1166, 1167], or design non-player characters that are interestingly diverse in their behavioral space [7].

Quality-diversity Algorithms

Another popular class of evolutionary algorithms for game AI applications are named *quality-diversity* algorithms, popularly shortened to QD algorithms [902]. These algorithms attempt to find a diverse set of fit solutions. Unlike multiobjective evolutionary algorithms, they only employ one objective of fitness function. However, they also use one or several distance metrics that measure how far solutions are from each other in some space. These distance metrics traditionally refer to the behavior of the solution, assuming the solution is a policy; however, they could really be any kind of distance metric you care about, including distances on the representation level. The output of a QD algorithm is a set of high-fitness solutions spread over a space defined by the distance function(s), where each solution is the highest-fit solution within some small region (niche) of that space.

The most popular QD algorithm is probably the Multi-dimensional Archive of Phenotypic Elites (**MAP-Elites**) algorithm [781]. MAP-Elites assumes one or more measures or descriptors, and a single fitness function. The central data structure of MAP-Elites, taking the place of the population in a regular evolutionary algorithm, is the **archive**. The archive is structured as an n-dimensional array known as the *feature map*, where n is the number of measures, also known as *behavior characterizations*. For two behavior characterizations, a common case, this is just a regular grid whose granularity is arbitrary. Each cell contains the best solution found so far within that particular region of the feature map. It is important to note that the number of behavior characterizations, and their granularity is based on the designer's preference and their available computational resources.

Given a particular fitness, n behavioral characterizations and their discretization, MAP-Elites searches for the highest performing solution of each cell of the grid within the n-dimensional feature map. For example, the algorithm will search to find the most efficient Ms Pac-Man player (fitness as score) in terms of game level exploration (first behavior characterization) and distance to the nearest ghost (second behavior characterization). MAP-Elites will likely find efficient Ms Pac-Man players that explore the maze most and keep the highest distance from ghosts, or alternatively, find efficient Ms Pac-Man players that explore the maze the least and keep the lowest distance from ghosts (among other combinations of behaviors). The selection process of parents occurs locally within each cell of the n-dimensional map but several other strategies have been presented in the literature such as the use of UCB1 formula (see Eq. (3.1)) for that matter [1015].

Further details about QD algorithms can be found in [902]. QD methods are directly applicable in games when one requires high performing solutions that are also diverse to the degree possible. Given their capacity to optimize for both quality and diversity, QD methods have been primarily employed for procedural content generation tasks as summarized in [405] and detailed in Part III.

Evolutionary Algorithms for Ms Pac-Man

A simple way to employ evolutionary algorithms (EAs) in Ms Pac-Man is as follows. You could design a utility function based on a number of important parameters Ms Pac-Man must consider for taking the right decision on where to move next. These parameters, for instance, could be the current placement of ghosts, the presence of power pills, the number of pellets available on the level and so on. The next step would be to design a utility function as the weighted sum of these parameters. At each junction, Ms Pac-Man would need to consult its utility function for all its possible moves and pick the move with the highest utility. The weights of the utility function are unknown of course and this is where an EA can be of help by evolving the weights of the utility so that they optimize the score for Ms Pac-Man. In other words, the fitness of each chromosome (weight vector of utility) is determined by the score obtained from Ms Pac-Man within a number of simulation steps, or game levels played. A similar approach was followed in [458] for the construction of agents that play dungeon crawler games in a particular playing style.

3.3.3 Further Reading

Beyond the specific papers that are tied to the algorithms introduced in this section, we recommend three books for further reading on evolutionary computation: Eiben's and Smith's *Introduction to Evolutionary Computing* [308], Ashlock's *Evolutionary Computation for Modeling and Optimization* [33] and finally, the genetic programming field guide by Poli et al. [894].

3.4 Unsupervised Learning Methods

In this section we outline some of the most popular unsupervised learning algorithms used broadly in the AI and games field. The majority of the unsupervised learning studies we discuss in this book focus on the paradigms of **clustering** and **frequent pattern mining**. For the former paradigm we cover the *k-means* and the *hierarchical clustering* methods; for the latter paradigm we cover the *Apriori* and the *generalized sequential patterns* methods. The interested reader is referred to Sect. 2.3.1 for the theoretical aspects of unsupervised learning.

3.4.1 K-Means Clustering

K-means [683] is considered the most popular clustering algorithm as it offers a good balance between simplicity and effectiveness. It follows a simple data partitioning approach according to which it partitions a database of objects into a set of k clusters, such that the sum of squared Euclidean distances between data points and their corresponding cluster center (centroid) is minimized—this distance is also known as the **quantization error**.

In k-means each cluster is defined by one point, that is the centroid of the cluster, and each data sample is assigned to the closest centroid. The centroid is the mean of the data samples in the cluster. The intra-cluster validity metric used by k-means is the average distance to the centroid (see Sect. 2.3.1). Initially, the data samples are randomly assigned to a cluster and then the algorithm proceeds by alternating between the re-assignment of data into clusters and the update of the resulting centroids. The basic steps of the algorithm are as follows:

Given k

1. Randomly partition the data points into k nonempty clusters.
2. Compute the position of the centroids of the clusters of the current partitioning. Centroids are the mean points of the cluster's members.
3. Assign each data point to the cluster with the nearest centroid.
4. Stop when the assignment does not change; otherwise go to step 2.

While k-means is very popular due to its simplicity it has a number of considerable weaknesses. First, it is applicable only to continuous data attributes. Second, one needs to specify the number of clusters, k, in advance. Third, it is not suitable to discover clusters with non-convex shapes; it can only find hyper-spherical clusters. Finally, k-means is sensitive to outliers as data points with extremely large (or small) values may substantially distort the distribution of the data and affect the performance

of the algorithm. As we will see in Sect. 3.4.2, hierarchical clustering manages to overcome some of these drawbacks, suggesting a useful alternative approach to data clustering.

3.4.2 *Hierarchical Clustering*

Clustering methods that attempt to build a hierarchy of clusters fall under the hierarchical clustering approach. Generally speaking there are two strategies available: the *agglomerative* and the *divisive*. The former constructs hierarchies in a bottom-up fashion by gradually merging data points, whereas the latter constructs hierarchies of clusters by gradually splitting the dataset in a top-down fashion. Both clustering strategies are greedy. Hierarchical clustering uses a distance matrix as the clustering strategy (whether agglomerative or divisive). This method does not require the number of clusters k as an input, but needs a termination condition.

We present the basic steps of the agglomerative clustering algorithm as follows:

Given k:

1. Create one cluster per data sample.
2. Find the two closest data samples (i.e., by shortest Euclidean distance between two points), which are not in the same cluster.
3. Merge the clusters containing these two samples.
4. Stop if there are k clusters; otherwise go to step 2.

In divisive hierarchical clustering instead, all data is initially in the same cluster. This cluster is then split until every data point is on its own cluster. This can be achieved via a predefined split strategy such as the DIvisive ANAlysis Clustering (DIANA) [550]. Alternatively, one may employ another clustering algorithm, such as 2-means, to recursively split the data in two clusters

Once clusters of data are iteratively merged (or split), one can visualize the clusters by decomposing the data into several levels of nested partitioning. In other words, one can observe a tree representation of clusters which is also known as a **dendrogram**. The clustering of data is obtained by cutting the dendrogram at the desired level of squared Euclidean distance. A dendrogram example is illustrated in Chap. 12.

Hierarchical clustering represents clusters as the set of data samples contained in them and, as a result, a data sample belongs to the same cluster as its closest sample. In k-means instead, each cluster is represented by a centroid and thus a data sample belongs to the cluster represented by the closest centroid. Further, when it comes to cluster validity metrics, agglomerative clustering uses the shortest distance between any sample in one cluster and a sample in another whereas k-means uses the average distance to the centroid. Due to these different algorithmic properties hierarchical

clustering has the capacity to cluster data that come in any form of a connected shape; k-means, on the other hand, only forms hyper-spherical clusters.

Clustering for Ms Pac-Man

One potential application of clustering for controlling Ms Pac-Man would be to model ghost behaviors and use that information as an input to the controller of Ms Pac-Man. Whether it is k-means or hierarchical clustering, the algorithm would consider different attributes of ghost behavior—such as level exploration, behavior divergence, and distance between ghosts—and cluster the ghosts into behavioral patterns or profiles. The controller of Ms Pac-Man would then consider the ghost profile met in a particular level as an additional input for guiding the agent better.

An alternative, and arguably more effective, application of clustering would be to use existing trajectories of Ms Pac-Man players (e.g., from the Atari-HEAD dataset [1313]) to cluster players based on their skill level as determined by their score. The derived clusters can then inform a supervised learning method (as its input) that will attempt to predict Ms Pac-Man actions as conditioned by the player skill profile. We further elaborate on the use of supervised learning for Ms Pac-Man control in Sect. 2.3.2.

3.4.3 Apriori

Apriori [9] is an algorithm for frequent itemset mining. The algorithm is appropriate for mining datasets that contain sets of instances that each feature a set of items, or an **itemset**. Examples of instances include books bought by an Amazon customer or apps bought by a smartphone user. The algorithm is very simple and can be described as follows: given a predetermined threshold named **support** (T), Apriori detects the itemsets which are subsets of at least T instances in the database. In other words, Apriori will attempt to identify all itemsets that have at least a minimum support which is the minimum number of times an itemset exists in the dataset.

To demonstrate Apriori in a game example, we indicatively list events from four players of an online role playing game. Note that an itemset containing A, B, and C is represented as <A; B; C >.

- <Completed more than 10 levels; Most achievements unlocked; Bought the shield of the magi>
- <Completed more than 10 levels; Bought the shield of the magi>
- <Most achievements unlocked; Bought the shield of the magi; Found the Wizard's purple hat>
- <Most achievements unlocked; Found the Wizard's purple hat; Completed more than 10 levels; Bought the shield of the magi>

If in the aforementioned example dataset we assume that the support is 3, the following 1-itemsets (sets of only one item) can be found: <Completed more than

10 levels>, <Most achievements unlocked> and <Bought the shield of the magi>. If instead, we seek 2-itemsets with a support threshold of 3 we can find <Completed more than 10 levels; Bought the shield of the magi>, as three of the instances above contain both of these items. Longer itemsets are not available (not frequent) for support count 3. The process can be repeated for any support threshold we wish to detect frequent itemsets for.

3.4.4 Generalized Sequential Patterns

Frequent itemset mining algorithms are not adequate if the *sequence* of events is the critical information we wish to mine from a dataset. The dataset may contain events in an ordered set of sequences such as time series. Instead, we need to opt for a frequent sequence mining approach. The sequence mining problem can be simply described as the process of finding frequently occurring subsequences given a sequence or a set of sequences.

More formally, given a dataset in which each sample is a sequence of events, namely a **data sequence**, a **frequent sequence** can be defined as a sequential pattern that is supported by a number of data sequences. This amount is determined by a threshold named minimum support value. A data sequence supports a sequential pattern if and only if it contains all the events present in the pattern in the same order. For example, the data-sequence $< x_0, x_1, x_2, x_3, x_4, x_5 >$ supports the pattern $< x_0, x_5 >$. As with frequent itemset mining, the amount of data sequences that support a sequential pattern is referred as the **support count**.

The Generalized Sequential Patterns (GSP) algorithm [1076] is a popular method for mining frequent sequences in data. GSP starts by extracting the frequent sequences with a single event, namely 1-sequences. That set of sequences is self-joined to generate all 2-sequence candidates for which we calculate their support count. Those sequences that are frequent (i.e., their support count is greater than a threshold value) are then self-joined to generate the set of 3-sequence candidates. The algorithm gradually increases the length of the sequences in each algorithmic step until the next set of candidates is empty. The basic principle of the algorithm is that if a sequential pattern is frequent, then its *contiguous* subsequences are also frequent.

Frequent Pattern Mining for Ms Pac-Man

Patterns of events of sequences can be extracted to assist the control of Ms Pac-Man. Itemsets may be identified across successful events of expert Ms Pac-Man players given a particular support count. For instance, an Apriori algorithm running on events across several different expert players might reveal that a frequent 2-itemset is the following: <player went for the upper left corner first, player ate the bottom right power pill first>. Such information can be useful for explicitly designing rules for controlling Ms Pac-Man.

Beyond itemsets, frequencies of ghost events can be considered for playing Ms Pac-Man. For example, by running GSP on extracted attributes of ghosts it might turn out that when Ms Pac-Man eats a power pill it is very likely that the Blinky ghost moves left (<power pill, Blinky left>). Such frequent sequences can form additional inputs of any Ms Pac-Man controller—e.g., an ANN. Chapter 12 details an example on this frequent sequence mining approach in a 3D prey-predator game.

3.4.5 Further Reading

A general introduction to frequent pattern mining is offered in [8]. The Apriori algorithm is detailed in [9] whereas GSP is covered thoroughly in [1076].

3.5 Supervised Learning Methods

There are many ways of doing supervised learning. In this book, we focus on some of the more popular and useful machine learning algorithms for game AI tasks such as game playing (see Part II), player behavior imitation or player preference prediction (see Part IV). The supervised learning algorithms outlined in the remainder of this section are support vector machines, decision tree induction and artificial neural networks trained via gradient descent. All algorithms covered here belong to the, so called, **shallow learning** paradigm [390] and can be used for any supervised learning downstream task as we have detailed in Chap. 2. **Deep learning** refers to training neural networks with more layers, typically using gradient descent. Those methods are discussed in Sect. 3.6 that follows, yet build on the methods described here.

3.5.1 Support Vector Machines

Support vector machines (SVMs) [239] are a very popular set of supervised learning algorithms that can be used for classification [479], regression [296] and preference learning [512] tasks. A support vector machine is a binary linear classifier that is trained to maximize the margin between the training examples of the separate classes in the data (e.g., apples and pears). As with every other supervised learning algorithm, the attributes of new and unseen examples are seeding the SVM which predicts the class they belong to. SVMs have been used widely for text categorization, speech recognition, image classification, and hand-written character recognition among many other areas.

SVMs construct a hyperplane that divides the input space and represents the function f that maps between the input and the target outputs. Instead of implicitly attempting to minimize the difference between the model's actual output and the

target output following the gradient of the error (as backpropagation in neural network training; see Sect. 3.5.3), SVMs construct a hyperplane that maintains the largest distance to the nearest training-data point of any other class. That distance is called a **maximum-margin** and its corresponding hyperplane divides the points ($\vec{x}_i$) of class with label (y_i) 1 from those with label -1 in a dataset of n samples in total. In other words, the distance between the derived hyperplane and the nearest point $\vec{x}_i$ from either class is maximized. Given the input attributes of a training dataset, $\vec{x}$, the general form of a hyperplane can be defined as: $\vec{w} \cdot \vec{x} - b = 0$ where, as in backpropagation training, $\vec{w}$ is the weight (normal) vector of the hyperplane and $\frac{b}{\|\vec{w}\|}$ determines the offset (or weight threshold/bias) of the hyperplane from the origin (see Fig. 3.12). Thus, formally put, an SVM is a function $f(\vec{x}; \vec{w}, b)$ that predicts target outputs ($\vec{y}$) and attempts to

$$\textbf{minimize} \quad \|\vec{w}\|^2, \tag{3.2}$$

$$\textbf{subject to} \quad y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \text{for } i = 1, \ldots, n \tag{3.3}$$

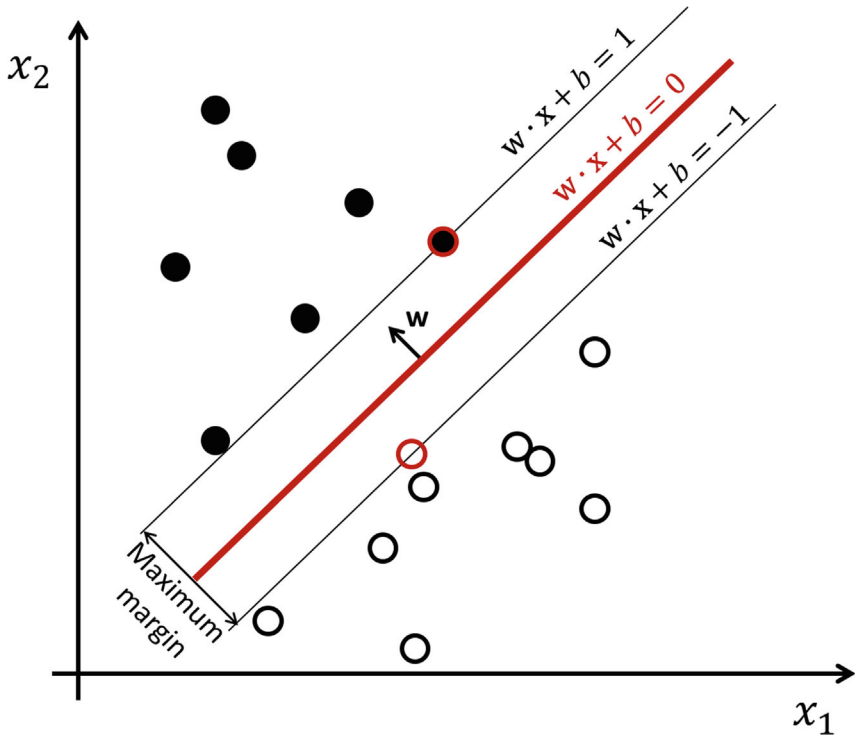


Fig. 3.12 An example of a maximum-margin hyperplane (red thick line) and margins (black lines) for an SVM which is trained on data samples from two classes. Solid and empty circles correspond to data with labels 1 and -1, respectively. The classification is mapped onto a two-dimensional input vector (x_1, x_2) in this example. The two data samples on the margin—the circles depicted with red outline—are the support vectors

The weights $\vec{w}$ and b determine the SVM classifier. The $\vec{x}_i$ vectors that lie nearest to the derived hyperplane are called **support vectors**. This problem is solvable if the training data is linearly separable (also known as a **hard-margin** classification task; see Fig. 3.12). If the data is not linearly separable (**soft-margin**) the SVM instead attempts to

$$\textbf{minimize} \quad \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right) \right] + \lambda ||\vec{w}||^2 \tag{3.4}$$

which equals $\lambda ||\vec{w}||^2$ if the hard constraints of Eq. (3.3) are satisfied—i.e., if all data points are correctly classified on the right side of the margin. The value of Eq. (3.4) is proportional to the distance from the margin for misclassified data and λ is designed so as to qualitatively determine the degree to which the margin-size should be increased versus ensuring that the $\vec{x}_i$ will lie on the correct side of the margin. Evidently, if we choose a small value for λ we approximate the hard-margin classifier for linearly separable data.

The standard approach for training soft-margin classifiers is to treat the learning task as a quadratic programming problem and search the space of $\vec{w}$ and b to find the widest possible margin that matches all data points [239]. Other approaches include sub-gradient descent [82] and coordinate descent [1256].

In addition to linear classification tasks, SVMs can support non-linear classification by employing a number of different non-linear **kernels** which map the input space onto higher-dimensional feature spaces. The SVM task remains similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. Popular kernels used in conjunction with SVMs include polynomial functions, Gaussian radial basis functions or hyperbolic tangent functions. While SVMs were originally designed to tackle *binary* classification problems there exist several SVM variants that can tackle multi-class classification [479], regression [296] and preference learning [512].

SVMs have a number of advantages compared to other supervised learning approaches. They are efficient in finding solutions when dealing with large, yet sparse, datasets as they only depend on support vectors to construct hyperplanes. They also handle large feature spaces well as the learning task complexity does not depend on the dimensionality of the feature space. SVMs feature a simple convex optimization problem which can be guaranteed to converge to a single global solution. Finally, overfitting can be controlled easily through the soft margin classification approach.

SVMs for Ms Pac-Man

SVMs can be used for imitating the behavior of Ms Pac-Man players, preferably good ones. There are several relevant considerations about the feature (input) space and the action (output) space. One can think of a corpus of expert gameplaying traces that we

can manually extract features from and then use as the input of the SVM. The SVM input may vary from simple statistics of the game (such as the closest distance to a ghost) to higher resolution pixel-to-pixel RGB values of the game screen. Outputs, on the other hand, may contain Ms Pac-Man actions across the frames of the game. In addition to the design of the input and output vectors, the size and quality of the data obtained from expert players will determine the performance of the SVM controlling Ms Pac-Man towards maximizing its score.

To take a concrete example, we could imagine a playtrace containing a few hours or gameplay from a medium-skilled Ms Pac-Man player such as yourself. The input here is $100 * 100$ pixel grid containing a downsampled monochrome view of the map centered on the player character, Ms Pac-Man. The target (output) is simply which of the four cardinal directions, or none at all, you are inputting via the keyboard at that particular time.

3.5.2 Decision Tree Learning

In decision tree learning [121, 904], the model is structured as a tree. Specifically, the function f we attempt to derive uses a decision tree representation which maps attributes of data observations to their target values. The former (inputs) are represented as the nodes and the latter (outputs) are represented as the leaves of the tree. The possible values of each node (input) are represented by the various branches of that node. As with any other supervised learning algorithm, decision trees can be classified depending on the output data type they attempt to learn. In particular, decision trees can be distinguished into classification, regression and rank trees if, respectively, the target output is a finite set of values, a set of continuous (interval) values, or a set of ordinal relations among observations.

An example of a decision tree is illustrated in Fig. 3.13. Tree nodes correspond to input attributes; there are branches to children for each of the possible values of each input attribute. Further leaves represent values of the output—Ms Pac-Man behavior in this example—given the values of the input attributes as determined by the path from the root to the leaf.

The goal of decision tree learning is to construct a mapping (a tree model) that predicts the value of target outputs based on a number of input attributes. The basic and most common approach for learning decision trees from data follows a top-down *recursive* tree induction strategy which has the characteristics of a greedy process. The algorithm assumes that both the input attributes and the target outputs have finite discrete domains and are of categorical nature. If inputs or outputs are continuous values, they can be discretized prior to constructing the tree. A tree is gradually constructed by splitting the available training dataset into subsets based on selections made for the attributes of the dataset. This process is repeated on an attribute-per-attribute basis in a recursive manner.

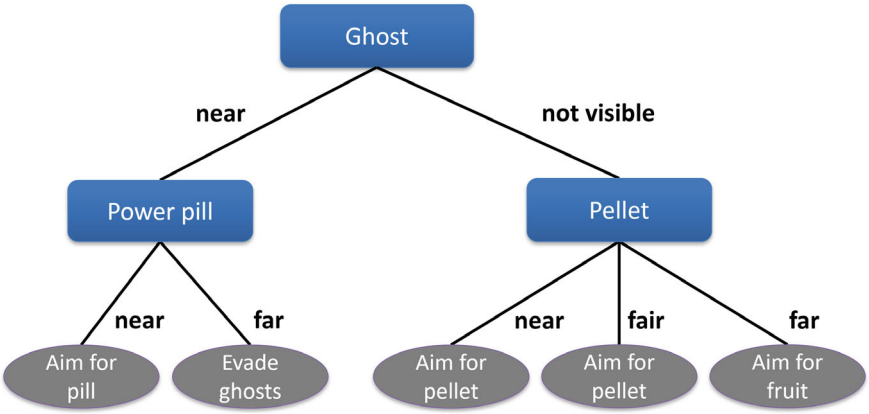


Fig. 3.13 A decision tree example for controlling Ms Pac-Man. The tree is trained on data from expert Ms Pac-Man players. Given the distance from the nearest *ghost*, *power pill* and *pellet* (data attributes) the tree predicts the *strategy* Ms Pac-Man needs to follow

There are several variants of this process that lead to dissimilar decision-tree algorithms. The two most notable variants of decision tree learning, however, are the **Iterative Dichotomiser 3 (ID3)** [904] and its successor **C4.5** [905]. The basic tree learning algorithm has the following general steps:

1. At start, all the training examples are at the root of the tree.
2. Select an attribute on the basis of a heuristic and pick the attribute with the maximum heuristic value. The two most popular heuristics are as follows:

   - **Information gain**: This heuristic is used by both the ID3 and the C4.5 tree-generation algorithms. Information gain $G(A)$ is based on the concept of entropy from information theory and measures the difference in entropy H from before to after the dataset D is split on an attribute A.

   $$G(A) = H(D) - H_A(D) \tag{3.5}$$

   where $H(D)$ is the entropy of D as $H(D) = -\sum_i^m p_i \log_2(p_i)$; p_i is the probability that an arbitrary sample in D belongs to class i; m is the total number of classes; $H_A(D)$ is the information needed (after using attribute A to split D into v partitions) to classify D and is calculated as $H_A(D) = -\sum_{j \in v}(|D_j|/|D|)H(D_j)$ with $|x|$ being the size of x.
   - **Gain ratio**: The C4.5 algorithm uses the gain ratio heuristic to reduce the bias of information gain towards attributes with a large number of values. The gain ratio normalizes information gain by taking into account the number of branches when choosing an attribute. The information gain

ratio is the ratio between the information gain and the intrinsic value IV_A of attribute A:

$$GR(A) = G(A)/IV_A(D) \tag{3.6}$$

where

$$IV_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log_2 \left(\frac{|D_j|}{|D|} \right) \tag{3.7}$$

3. Based on the selected attribute from step 2, construct a new node of the tree and split the dataset into subsets according to the possible values of the selected attribute. The possible values of the attribute become the branches of the node.
4. Repeat steps 2 and 3 until one of the following occurs:
   - All samples for a given node belong to the same class.
   - There are no remaining attributes for further partitioning.
   - There are no data samples left.

Decision Trees for Ms Pac-Man

As with SVMs, decision tree learning requires data to be trained on. Presuming that data from expert Ms Pac-Man players—e.g., obtained from the Atari-HEAD dataset [1313]—would be of good quality and quantity, decision trees can be constructed to predict the strategy of Ms Pac-Man based on a number of ad-hoc designed attributes of the game state. Figure 3.13 illustrates a simplified hypothetical decision tree for controlling Ms Pac-Man. According to that example, if a ghost is nearby then Ms Pac-Man checks if power pills are available in a close distance and aims for those; otherwise it takes actions so that it evades the ghost. Alternatively, if ghosts are not visible, Ms Pac-Man checks for pellets. If those are nearby or in a fair distance then it aims for them; otherwise it aims for the fruit, if that is available on the level. It is important to note that the leaves of the tree in our example represent control strategies (macro-actions) rather than actual actions (up, down, left, right) for Ms Pac-Man. Such macro actions (e.g., *aim for pill*) can be obtained though clustering expert player trajectories—from datasets such as Atari-HEAD [1313]—that demonstrate such strategies (e.g., a sequence of micro-actions that lead to a power pill).

3.5.3 *Artificial Neural Networks*

Artificial Neural Networks (ANNs) are a type of machine learning model originally inspired by neuroscience. An ANN is a set of interconnected processing units (named **neurons**) which was originally designed to model the way a biological brain—containing over 10^{11} neurons—processes information, operates, learns and performs in several tasks. Biological neurons have a cell body, a number of dendrites which bring information into the neuron and an axon which transmits electrochemical information outside the neuron. The artificial neuron (see Fig. 3.14) resembles the biological neuron as it has a number of *inputs* x_i (corresponding to the neuron dendrites) each with an associated *weight* parameter w_i (corresponding to the synaptic strength). It also has a processing unit that combines inputs with their corresponding weights via an inner product (weighted sum) and adds a *bias* (or threshold) weight b to the weighted sum as follows: $\mathbf{x} \cdot \mathbf{w} + b$. This value is then fed to an **activation function** g (cell body) that yields the *output* of the neuron (corresponding to an axon terminal). ANNs are essentially simple mathematical models defining a function $f : \mathbf{x} \rightarrow \mathbf{y}$.

Various forms of ANNs [91, 92] are applicable for regression analysis, classification, and preference learning, and even unsupervised learning (via e.g., Hebbian learning [439] and self-organizing maps [577]). Core application areas include pattern recognition, robot and agent control, gameplaying, decision making, gesture,

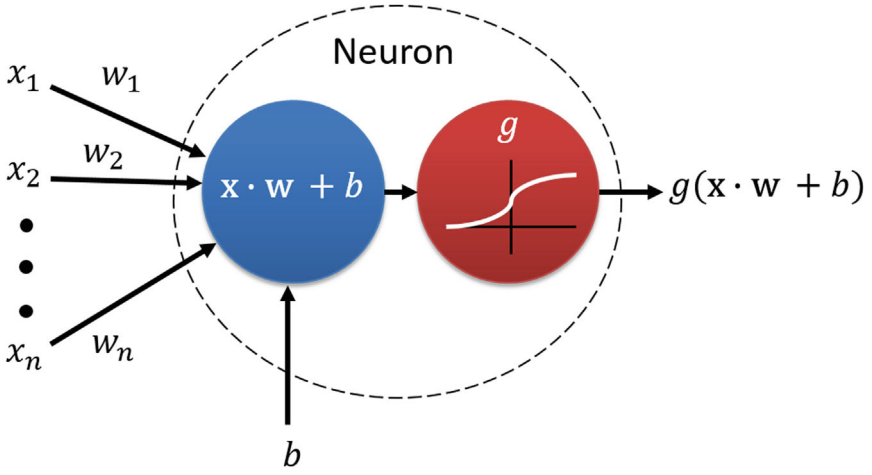


Fig. 3.14 An illustration of an artificial neuron. The neuron is fed with the input vector $\vec{x}$ through n connections with corresponding weight values $\vec{w}$. The neuron processes the input by calculating the weighted sum of inputs and corresponding connection weights and adding a bias weight (b): $\mathbf{x} \cdot \mathbf{w} + b$. The resulting formula feeds an activation function (g), the value of which defines the output of the neuron

speech and text recognition, medical and financial applications, affective modeling, and image recognition. The benefits of ANNs compared to other supervised learning approaches is their capacity to approximate any continuous real-valued function given sufficiently large ANN architectures and computational resources [254, 578]. This capacity characterizes ANNs as *universal approximators* [473].

Activation Functions

Which activation function should one use in an ANN? The original model of a neuron by McCulloch and Pitts [734] in 1943 featured a Heaviside step activation function which either allows the neuron to *fire* or not. When such neurons are employed and connected to a multi-layered ANN the resulting network can merely solve linearly separable problems. The algorithm that trains such ANNs was invented in 1958 [966] and is known as the Rosenblatt's perceptron algorithm. Non-linearly separable problems such as the exclusive-or gate could only be solved after the invention of the **backpropagation** algorithm in 1975 [1247]. Nowadays, there are several activation functions used in conjunction with ANNs and their training. The use of the activation function, in turn, yields different types of ANNs. Examples include Gaussian activation function that is used in radial basis function (RBF) networks [129] and the numerous types of activation functions that can be used in the compositional pattern producing networks (CPPNs) [1078]. The most common function used for ANN training is the sigmoid-shaped logistic function ($g(x) = 1/(1 + e^{-x})$) for the following properties: (1) it is bounded, monotonic and non-linear; (2) it is continuous and smooth and (3) its derivative is calculated trivially as $g'(x) = g(x)(1 - g(x))$. Given these properties, the logistic function can be used in conjunction with gradient-based optimization algorithms such as backpropagation which is described in Sect. 3.5.3.1. Beyond the sigmoidal-shaped logistic function the **softmax** function, $\sigma(x_i) = e_{x_i} / \sum_{x \in X} e^x$, is a very popular and generalized form of the logistic function to multiple dimensions, hence primarily used in multi-class classification problems. The softmax function is often employed in the final layer of a (shallow or deep) ANN classifier to normalize the outputs to a probability distribution over the outputs (i.e., target classes). Other popular activation functions for training *deep* architectures of neural networks include the **rectifier**—named **rectified linear unit** (ReLU) when employed to a neuron—and its smooth approximation, the **softplus** function [390]. Compared to sigmoid-shaped activation functions, ReLUs allow for faster and (empirically) more effective training of deep ANNs, which are generally trained on large datasets (see more in Sect. 3.6).

From a Neuron to a Network: Multi-Layer Perceptron

To form an ANN a number of neurons need to be structured and connected. The most common approach is to structure neurons into layers. In its simplest form, known as the multi-layer perceptron (MLP), neurons are layered across one or more layers but not connected to other neurons in the same layer (see Fig. 3.15 for a typical MLP structure). The output of each neuron in each layer is connected to all

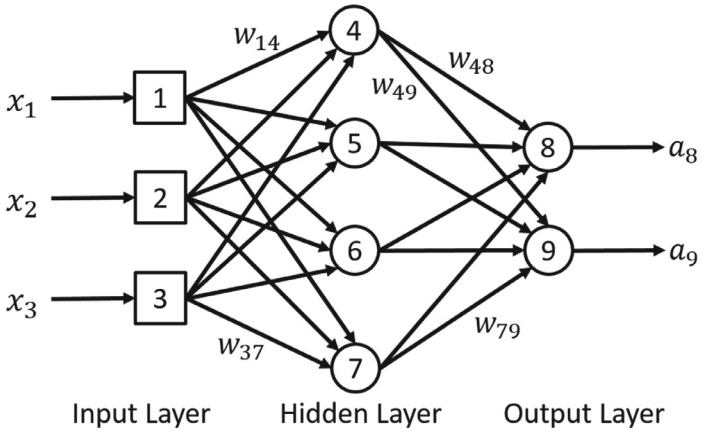


Fig. 3.15 An MLP example with three inputs, one hidden layer containing four hidden neurons and two outputs. The ANN has labeled and ordered neurons and example connection weight labels. Bias weights b_j are not illustrated in this example but are connected to each neuron j of the ANN

the neurons in the next layer. Note that a neuron's output value feeds merely the neurons of the next layer, becoming their input. Consequently, the outputs of the neurons in the last layer are the outputs of the ANN. The last layer of the ANN is also known as the **output layer**; all intermediate layers between the output and the input are the **hidden layers**. It is important to note that the inputs of the ANN, $\vec{x}$, are connected to all the neurons of the first hidden layer. We illustrate this with an additional layer we call the **input layer**. The input layer does not contain neurons as it only distributes the inputs to the first layer of neurons. In summary, MLPs are (1) *layered* because they are grouped in layers; (2) *feed-forward* because their connections are unidirectional and always forward (from a previous layer to the next); and (3) *fully connected* because every neuron is connected to all neurons of the next layer.

Forward Operation

In this section we will see how we compute the output of the ANN when an input pattern is presented. The process is called *forward operation* and propagates the inputs of the ANN throughout its consecutive layers to yield the outputs. The basic steps of the forward operation are as follows:

1. Label and order neurons. We typically start numbering at the input layer and increment the numbers towards the output layer (see Fig. 3.15). Note that the input layer does not contain neurons, nevertheless is treated as such for numbering purposes only.
2. Label connection weights assuming that w_{ij} is the connection weight from neuron i (pre-synaptic neuron) to neuron j (post-synaptic neuron). Label bias weights that connect to neuron j as b_j.
3. Present an input pattern $\vec{x}$.
4. For each neuron j compute its output as follows: $\alpha_j = g(\sum_i \{w_{ij}\alpha_i\} + b_j)$, where α_j and α_i are, respectively, the output of and the inputs to neuron j (n.b. $\alpha_i = x_i$ in the input layer); g is the activation function (usually the logistic sigmoid function).
5. The outputs of the neurons of the output layer are the outputs of the ANN.

3.5.3.1 Training an ANN

How do we approximate $f(\mathbf{x}; \mathbf{w}, \mathbf{b})$ so that the outputs of the ANN match the desired outputs (labels) of our dataset, $\mathbf{y}$? We will need a training algorithm that adjusts the weights ($\mathbf{w}$ and $\mathbf{b}$) so that $f : \mathbf{x} \rightarrow \mathbf{y}$. A training algorithm as such requires two components. First, it requires a cost function to evaluate the quality of any set of weights. Second, it requires a search strategy within the space of possible solutions (i.e., the weight space). We outline these aspects in what follows next.

Cost (Error) Function

Before we attempt to adjust the weights to approximate f, we need a utility for MLP training. The most common utility for training ANNs for a regression task is the squared Euclidean distance (error) between the vectors of the actual output of the ANN (α) and the desired labeled output y (see Eq. 3.8).

$$E = \frac{1}{2}\sum_j (y_j - \alpha_j)^2 \tag{3.8}$$

where the sum is taken over all the output neurons (the neurons in the final layer). Note that the y_j labels are constant values and E is a function of all the weights of the ANN since the actual outputs depend on them. As we will see next, ANN training algorithms build strongly upon this relationship between error and weights.

Backpropagation

The backpropagation (or backprop for short) [970] algorithm is a type of gradient descent optimization and is probably the most common algorithm for training shallow ANN architectures. Backpropagation stands for *backward propagation of errors* as it calculates weight updates that minimize the error function of Eq. (3.8) from the output to the input layer. In a nutshell, backpropagation computes the partial derivative (gradient) of the error function E with respect to each weight of the ANN and adjusts the weights of the ANN following the (opposite direction of the) gradient that minimizes E.

As mentioned earlier, E in Eq. (3.8) depends on the weights as the ANN output which is essentially the $f(\mathbf{x}; \mathbf{w}, \mathbf{b})$ function. As such we can calculate the gradient of E with respect to any weight ($\frac{\theta E}{\theta w_{ij}}$) and any bias weight ($\frac{\theta E}{\theta b_j}$) in the ANN, which in turn will determine the degree to which the error will change if we change the weight values. We can then determine how much of such change we desire through a parameter $\eta \in [0, 1]$ called **learning rate**. In the absence of any information about the general shape of the function between the error and the weights it appears that a **gradient descent** approach would seem to be a good fit for attempting to find the global minimum of the error function. Given the lack of information about E, the search can start from some random point in the weight space (i.e., random initial weight values) and follow the gradient towards lower E values. This process is repeated iteratively until we reach E values we are happy with or we run out of computational resources.

More formally, the basic steps of the backpropagation algorithm are as follows:

1. Initialize **w** and **b** to random (usually small) values.
2. For each training pattern (input-output pair):
   (a) Present input pattern $\vec{x}$, ideally normalized to a range (e.g., [0, 1]).
   (b) Compute ANN actual outputs α_j using the forward operation.
   (c) Compute E according to Eq. (3.8).
   (d) Compute error derivatives with respect to each weight $\frac{\theta E}{\theta w_{ij}}$ and bias weight $\frac{\theta E}{\theta b_j}$ of the ANN from the output all the way to the input layer.
   (e) Update weights and bias weights as $\Delta w_{ij} = -\eta \frac{\theta E}{\theta w_{ij}}$ and $\Delta b_j - \eta \frac{\theta E}{\theta b_j}$, respectively.
3. If E is *small* or you are out of *computational budget*, stop! Otherwise go to step 2.

Note that we do not wish to detail the derivative calculations of step 2(d) as doing so would be out of scope for this book. We instead refer the interested reader to the original backpropagation paper [970] for the exact formulas and to Sect. 3.5.4 for further reading.

Limitations and Solutions

It is worth noting that backpropagation is not guaranteed to find the global minimum of E given its hill-climbing strategy. Further the algorithm fails to overcome potential plateau areas in the error function landscape. As these are areas with near-zero gradient, crossing them results in near-zero weight updates and leads to **premature convergence** of the algorithm. Typical enhancements of the algorithm to overcome convergence to local minima include:

- **Random restarts**: One can rerun the algorithm with new random connection weight values in the hope that the ANN is not too dependent on luck. No ANN model is good if it depends too much on luck—for instance, if it performs well only in one or two out of ten runs.
- **Dynamic learning rate**: One can either modify the learning rate parameter and observe changes in the performance of the ANN or introduce a dynamic learning rate parameter that increases when convergence is slow and decreases when convergence is fast.
- **Momentum**: Alternatively, one may add a momentum to the weight update rule as follows:

$$\Delta w_{ij}^{(t)} = m\Delta w_{ij}^{(t-1)} - \eta\frac{\theta E}{\theta w_{ij}} \tag{3.9}$$

  where $m \in [0, 1]$ is the momentum parameter and t is the iteration step of the weight update. The addition of a momentum value of the previous weight update ($m\Delta w_{ij}^{(t-1)}$) attempts to help backpropagation overcome a potential local minimum.

While the aforementioned solutions are directly applicable to ANNs of small size, practical wisdom and empirical evidence with modern (deep) ANN architectures suggests that such drawbacks are largely eliminated [611].

Batch versus Non-batch Training

Backpropagation can be employed following a batch or a non-batch learning mode. In **non-batch** mode, weights are updated every time a training sample is presented to the ANN. In **batch mode**, weights are updated after all training samples are presented to the ANN. In that case, errors are accumulated over the samples of the batch prior to the weight update. The non-batch mode is more unstable as it iteratively relies on a single data point; however, this might be beneficial for avoiding a convergence to a local minimum. The batch mode, on the other hand, is naturally a more stable gradient descent approach as weight updates are driven by the average error of all training samples in the batch. To best utilize the advantages of both approaches it is common to apply batch learning of randomly selected samples in small batch sizes. The **stochastic gradient descent** algorithm that we cover in Sect. 3.6.1.2 is based on this principle.

Types of ANNs

Beyond MLPs, there are numerous other types of shallow ANN architectures used for classification, regression, preference learning, data processing and filtering, and clustering tasks. **Recurrent** neural networks (RNNs) such as Hopfield networks [472], Boltzmann machines [6] and Long Short-Term Memory [456] allow connections between neurons to form directed cycles. This enables an RNN to capture dynamic and temporal phenomena (e.g., time-series processing and prediction). Further, there are ANN types mostly used for clustering and data dimensionality reduction such as Kohonen self-organizing maps [577] and Autoencoders [76]. Section 3.6 takes us through a number of alternative ANN architectures and training methods including autoencoders but also convolutional neural networks, generative adversarial networks, and transformer architectures.

ANNs for Ms Pac-Man

As with every other method in this chapter we will attempt to employ ANNs in the Ms Pac-Man game. One straightforward way to use ANNs in Ms Pac-Man is to attempt to imitate expert players of the game. Thus, one can ask experts to play the game and record their playthroughs as in [1313], through which a number of features can be extracted and used as the input of the ANN. The resolution of the ANN input may vary from simple statistics of the game—such as the average distance between ghosts and Ms Pac-Man—to detailed pixel-to-pixel RGB values of the game level image. In this example we can employ an object detection model like YOLO [929] to identify the position of Ms Pac-Man, ghosts and pellets on screen by looking at the frames of the Atari-HEAD dataset [1313]. From these positions we can derive Euclidean distances of Ms Pac-Man to all ghosts, the nearest pellet and power pill which define the input vector of a 3-layer MLP. The output data, on the other hand, contain the actions selected by Ms Pac-Man (up, down, left, right) in each frame of the game. The ANN is trained via backpropagation to predict the action performed by expert players (ANN output) given the current game state (ANN input). The performance of the ANN can be tested across many hyperparamaeters and it largely depends on the size (width and depth) of the MLP architecture, the amount of data available from the expert Ms Pac-Man players, the size of the input vector considered, the learning rate and so on.

3.5.4 Further Reading

The core shallow machine learning algorithms are covered in detail in the Russell and Norvig classic AI textbook [975] including decision tree learning (Chap. 18) and artificial neural networks (Chap. 19). Detailed descriptions of artificial neural networks and backpropagation can also be found in the book of Haykin [436]. Deep architectures of ANNs are covered in great detail in the deep learning book by

Goodfellow et al. [390] and the more recent book by Bishop and Bishop [92]. Finally, support vector machines are covered in the tutorial paper of Burges [152].

3.6 Deep Learning Methods

A critical parameter for ANN training is the size of the ANN. So, how *wide* and *deep* should my ANN architecture be to perform well on this particular task? While there is no formal and definite answer to this question, there is a generally accepted rule-of-thumb suggesting that the size of the network should match the complexity of the problem. Too small networks may not be able to represent the function that accurately models the data, and too large networks may be wasteful in terms of compute and memory requirements, and may also overfit easier. According to Goodfellow et al. [390] in their deep learning book an MLP containing at least 1 hidden layer is essentially a deep (feedforward) neural network, although many common deep learning architectures have many more layers. Its *depth* is determined by the number of hidden layers it contains. Goodfellow et al. state that "it is from this terminology that the name *deep learning* arises". The interested reader is referred to Sect. 3.5.3 for a more detailed description of **shallow** architectures. Deep learning architectures, instead, may be of dissimilar types including deep neural networks, convolutional neural networks, recurrent neural networks, and transformers, as covered in this section.

The remainder of this section takes us through the most popular methods under the *deep learning* definition. That includes deep neural networks (Sect. 3.6.1), autoencoders (Sect. 3.6.2), convolutional neural networks (Sect. 3.6.3), generative adversarial networks (Sect. 3.6.4), diffusion models (Sect. 3.6.5), the transformer (Sect. 3.6.6) and large language models (Sect. 3.6.7).

3.6.1 Deep Neural Networks

We first take the interested reader through basic concepts of *deep* neural networks including their core elements and the ways they learn. Most of the concepts we outline here are applicable to any neural network model and architecture we cover in the remainder of the section.

A deep neural network (DNN) is an ANN featuring multiple layers stacked between their input and their output layers. While there are numerous types of DNN, they all share the same core components we already described in the shallow ANN section earlier: neurons, weights, biases, and activation functions. DNNs can model complex non-linear relationships by gradually creating increasingly complex compositional models throughout their layers. Let us pick an image (e.g., apple vs. orange) recognition task as an example. A DNN will first observe the presented

object (input)—e.g., represented as RGB pixels—and will then try to extract a layered composition of primitive features through its first layers. The additional layers of the DNN will then enable the composition of features from the earlier layers. This process allows the DNN to model complex high-resolution data, such as image RGB pixels, with fewer units than a similarly performing shallow ANN [76].

Deep learning architectures often consist of layers of different types stacked on top of each other, meaning that the output from a layer of type A becomes the input to another layer of type B. In particular, it is very common to have several convolutional layers (see Sect. 3.6.3) for handling high-dimensional input (e.g., vision) and stack a few fully-connected (MLP) layers on top of them. This way, the convolutional layers can find regularities which can then be exploited by the fully-connected layers.

Similarly to ANNs, DNNs are typically feedforward networks that go through an analogous forward operation. Compared to shallow ANNs, however, there are a number of differences in terms of training algorithms, loss functions and activation functions used. We discuss popular activation functions employed in DNN architectures in Sect. 3.6.1.1 and then in Sect. 3.6.1.2 we look at ways DNNs can be trained to perform a downstream task.

3.6.1.1 Activation Functions

Beyond the sigmoidal-shaped logistic function covered in Sect. 3.5.3 for shallow ANN training, the literature has proposed numerous other activation functions that are more suitable for deep architecture training. The **softmax** function introduced in Sect. 3.5.3 is primarily used in multi-class classification problems and employed in final layer of a DNN classifier to normalize the outputs to a probability distribution over the outputs (i.e., target classes). This function is commonly combined with a cross-entropy loss for DNN training.

Another popular activation function used in conjunction with DNN architectures is the rectified linear unit (**ReLU**), which is computed as follows $R(x) = \max(0, x)$. Compared to sigmoid-shaped activation functions, ReLUs enable faster and better training of deep networks for a number of reasons. First, a DNN with ReLUs yields sparse neuron activations; in a randomly initialized DNN network, for a example, only approximately 50% of ReLU neurons are activated (i.e., having a non-zero output). Second, compared to sigmoidal activation functions, ReLUs mitigate challenges associated with vanishing gradients across the several layers of a DNN. Finally, ReLUs are scale invariant and their gradient is faster to compute. In addition to the vanilla ReLU, a number of ReLU variants have been proposed in the literature that offer various advantages in certain domains and tasks. Most notably the **leaky ReLU** is designed so as to yield a small positive gradient even when when the unit is not active [681], thereby further mitigating the vanishing gradient problem.

3.6.1.2 Training a Deep Neural Network

Various methods have been introduced in recent years to enable training of deep architectures containing several layers. The methods largely rely on stochastic gradient search and are covered in detail in [390]. Here we outline the stochastic gradient descent algorithm and we put an emphasis on popular loss functions and optimizers used in conjunction with this algorithm. We end the section discussing the challenges and limitations of DNN architectures and training.

Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) algorithm is an iterative method for optimizing an objective function such as a loss function of an ANN. SGD can be viewed as a stochastic approximation gradient descent, as it replaces the gradient with a gradient estimate. The estimate of the gradient is not derived from loss function calculations on the entire dataset, but on a randomly sampled subset of the data. This property makes it a preferred optimization method in problems of high-dimensionality—such as those tackled by DNNs—as it reduces the computational effort through faster iterations [112].

To train the weights of the DNN, stochastic gradient descent will attempt to adjust the weights ($\mathbf{w}$ and $\mathbf{b}$) so that $f : \mathbf{x} \rightarrow \mathbf{y}$. SGD approximates the true gradient of $L(\mathbf{w})$ by a gradient calculated on a single sample of the data as follows:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_i(\mathbf{w}) \tag{3.10}$$

where $\leftarrow$ denotes the update of a variable, $\mathbf{w}$ are the network weights that minimize the loss function L and $L_i(\mathbf{w})$ is typically associated with the i-th observation in the data set. The SGD algorithm observes the training set in an iterative fashion and hence it essentially performs the update of Eq. (3.10) for each training sample. More formally the basic steps of the SGD algorithm are as follows:

> Given a vector of initial weights $\mathbf{w}$, a learning rate η, and a number of samples (or batches) per iteration n, repeat steps 1-2 until a termination condition is met:
>
> 1. Randomly shuffle samples in the training set.
> 2. For $i = 1, \ldots, n$, compute:
> $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_i(\mathbf{w})$

A popular compromise between computing the true gradient of the entire dataset versus the gradient of a single data point is to compute the gradient against a **batch** of samples (or mini-batch) at each iterative step of the algorithm. As with any other gradient descent method, SGD requires a) a cost (error or loss) function to evaluate the quality of weights and b) an optimization strategy within the space of possible solutions (i.e., the weight space). We outline these two aspects in the following paragraphs.

Loss Functions

The utility of DNN training is typically referred to as *loss function*. Normally any loss function minimizes the distance (loss) between the actual output vector of the DNN (α) and the desired labeled output vector $\mathbf{y}$. Several loss functions have been proposed in the literature depending on the downstream task (e.g., image recognition vs. behavior imitation) and the learning paradigm adopted (e.g., regression vs. classification). The most common loss functions for training DNNs in a supervised manner are the **mean squared error** and the **cross entropy** loss, respectively, for regression and classification downstream tasks. The mean squared error, or else L2 loss, measures the magnitude of the error between a DNN output and an actual output by computing the average of the squared difference between them as follows:

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \alpha_i)^2 \tag{3.11}$$

where N is the number of samples considered. As with the MSE error covered earlier (see Eq. (3.8)) the sum of losses is taken over all the output neurons (the entire vector of neurons in the final layer).

As noted earlier, when the DNN is applied on a classification task the loss function used most commonly is the cross-entropy loss, L_{CE}. L_{CE} measures the performance of a classification model whose predicted output is a probability value between 0 and 1. If the number of classes is 2 (i.e., binary classification) then the binary cross entropy L_{BCE} is computed as follows for each of the considered samples, N:

$$L_{BCE} = -(y\log(\alpha) + (1-y)\log(1-\alpha)) \tag{3.12}$$

where y in this case is the target label and α is the predicted ANN output (probability). L_{BCE} is typically averaged across all data samples N.

When instead, the classification task considers more than two classes C the following L_{CE} loss (or categorical cross-entropy loss as it is often referred to) is computed across all classes i and then averaged over N:

$$L_{CE} = -\sum_{i}^{C}(y_i\log(\alpha_i)) \tag{3.13}$$

Optimizers

Beyond the vanilla SGD optimization process the literature is rich with SGD variants that enhance the capabilities of stochastic gradient search. A conceptually simple and popular extension to SGD employs *learning rate schedules* according to which the learning rate decreases over algorithmic iterations according to a pre-determined schedule. Doing so results in more emphatic parameter changes during the first iterations and fine tuning adjustments in later iterations. Another popular SGD variant

is **Adam** [568] (short for *Adaptive Moment Estimation*). Adam adopts the principles of the *momentum* method we covered earlier in Sect. 3.5.3.1 but it introduces exponential forgetting parameters of both the gradients and the second moments of the gradients.

Challenges and Limitations

The core challenges of training deep ANNs are overfitting and computational effort. As with shallow ANN architectures, DNNs are prone to **overfitting** since their depth allows them to model complex phenomena but often unnecessarily capture rare dependencies in the data. The most popular ways of tackling this issue in DNNs is through **dropout** and data **augmentation**. The dropout mechanism [453] omits a percentage of neurons from the DNN during training by setting their corresponding weights to zero. The unit selection can occur entirely at random or alternatively, rely on a specific search strategy. Beyond dropout, data augmentation methods [1036]—commonly used in computer vision tasks—apply a number of transformation methods on the DNN observations (e.g., images or frames of games). Transformations may be of *geometric* nature such as image cropping, rotation, translation and flipping, related to *color* such as brightness, contrast and saturation alterations and even *noise* such as various types of blurring and distortion. This process ends up augmenting small training sets to larger datasets, thereby, reducing the chances of DNN model overfitting.

Due to the large size of their trainable parameters (often in the billions or even more), DNN training is often costly and in some cases entirely infeasible for AI researchers [1173]. Despite the challenges associated with the **computational cost** of training large models there are certain tricks one might try out to alleviate some of this burden. Training in batches—computing the gradient across several examples at once—is a popular method to reduce compute. Fine-tuning existing pre-trained large models is another popular alternative. Finally, one may adopt methods from **knowledge distillation** [394] so that experiments are performed with smaller models that are trained to behave like much larger models.

Deep Neural Networks for Ms Pac-Man

DNNs may find multiple use cases for playing games, generating content and modeling players of games. In the context of Ms Pac-Man, any DNN architecture can be used to observe the game state—e.g., as a set of game tile vectors or as a set of pixels—and learn to take decisions in a sequential manner. Such policy networks can be trained in an unsupervised, supervised, self-supervised or even reinforcement learning manner. Clearly, DNNs can also act as evaluation or even attention networks that may learn, respectively, to infer the quality of a certain game state or observe particular aspects of the game screen during play. Throughout the methods we cover in this section we list a number of dissimilar ways deep architectures can be used for playing Ms Pac-Man.

3.6.2 Autoencoders

An autoencoder is normally a deep neural network architecture that learns to compress and encode unlabeled data following the self-supervised learning paradigm [390]. In general terms an autoencoder learns to encode input x into a latent vector z and recreate (decode) the input x from the encoded representation z. Autoencoder architectures are typically used for dimensionality reduction where the model is trained to learn a lower-dimensional representation z that characterizes and captures higher-dimensional data such as images, text, physiological signals or behavioral data.

Typically a **reconstruction loss** function is employed to train the model that measures the similarity between the input and the reconstructed input x' (i.e., autoencoder output). The most popular loss functions for reconstruction include the least absolute deviations (or L1; see Eq. (3.14)) or the mean squared error (MSE) loss.

$$L = |x - x'| \tag{3.14}$$

Since the inception of the algorithm a few decades ago [970], the literature is rich with both algorithmic variants and uses of the algorithm in conjunction with other learning paradigms. Popular examples include the **variational autoencoder** [569] that generates a prior and noise distribution and is trained using variants of the expectation-maximization algorithm, and the **contrastive autoencoder** that operates on the basis that inputs that are similar should also yield similar latent space representations.

The non-linear dimensionality reduction capacity of autoencoders makes them directly applicable to any domain including image, text and audio processing. Autoencoders are also broadly used as automatic feature extractors of various data modalities including facial expressions [1307] and speech cues [497]. The trained decoder can also be used as a generative AI model that creates new data outputs (e.g., images) by varying its trained latent vector z. Beyond any of their aforementioned uses, autoencoders are nowadays largely considered versatile **representation learning** methods. This capacity makes the method particularly important for video game applications since a pre-trained encoder can serve several downstream tasks including modeling players [715], generating content [633] or designing gameplaying agents.

Autoencoders for Ms Pac-Man

Figure 3.16 offers an illustrative example of an autoencoder used to compress pixel information obtained from Ms Pac-Man to a latent vector z. The latent vector of the trained encoder is a compressed representation (e.g., up to a few hundred values) of the RGB pixels values of Ms Pac-Man frames. The autoencoder might, for instance, observe a low-resolution RGB image of the game (32×32) and gradually compress it over a number of encoding layers to a latent vector of 32 parameters. This relatively

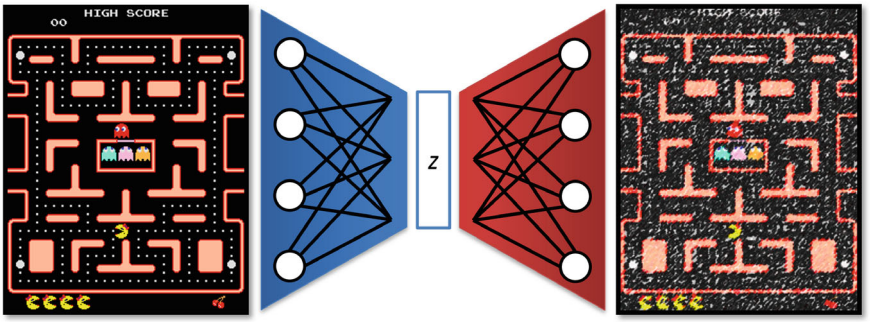


Fig. 3.16 Using an autoencoder in Ms Pac-Man. The autoencoder attempts to reconstruct the pixels of the game frames. The reconstruction loss is used to train the algorithm that learns to transform the input image to a compressed latent vector z. The encoder and decoder are depicted in blue and red trapezoids, respectivelyUsing an autoencoder in Ms Pac-Man. The autoencoder attempts to reconstruct the pixels of the game frames. The reconstruction loss is used to train the algorithm that learns to transform the input image to a compressed latent vector z. The encoder and decoder are depicted in blue and red trapezoids, respectively

small vector can feed an MLP that would, in turn, perform action classification of 4 possible outputs (up, down, left, right) on a dataset of Ms Pac-Man human demonstrations, as in the example we outlined in Sect. 3.5.3. The same low-dimensional representation (z) can be used for any downstream task beyond gameplaying, including modeling player behavior or even generating (reconstructing) new aspects of the game level via algorithms that search within a latent vector (as in [654, 1038]).

3.6.3 Convolutional Networks

Convolutional neural networks (CNNs) [610] are bio-inspired neural architectures that are designed to resemble the underlying perception processes of the visual cortex. Each cortical neuron of animals is responsible for, and is triggered by, a restricted area within the visual field of the animal, known as the receptive field. The receptive fields of the different cortical neurons overlap to a certain degree and in turn manage to cover the visual field as a whole. The CNN emulates this process and can be viewed as a variant of a multi-layer perceptron that exploits the spatial local correlations of an observation, such as an image. CNNs can be used as feature extractors in an unsupervised learning manner; the pre-trained CNNs can feed any relevant downstream task such as player action prediction [856], player affect modeling [692] or even content generation [654]. CNNs can also be trained *end-to-end* (i.e., all trainable parameters are trained at once) to predict a desired output from observations, as any other supervised learning method. While similar to an MLP a CNN has a number of distinct features that gives it superior performance, particularly on visual perception tasks. First, each neuron of a convolutional layer is connected to a restricted

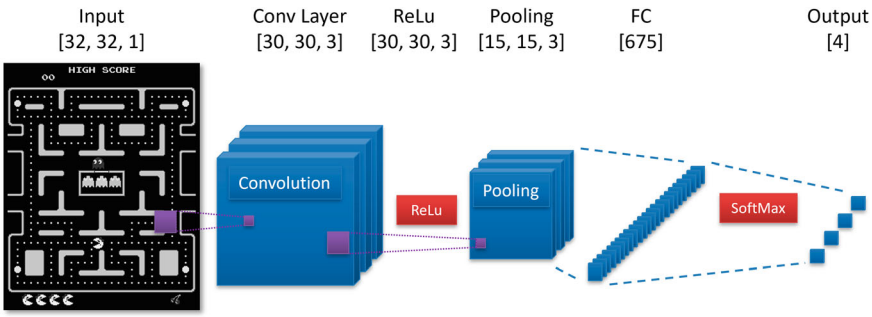


Fig. 3.17 Playing Ms Pac-Man with a CNN. In this simplified example the CNN perceives a low-resolution grayscale image of the game (32×32) that feeds just one convolutional layer of 3 neurons (resulting in 3 feature maps). The resulting maps pass through a ReLu activation function and their size is reduced through a max pooling layer. The output of the pooling layer is flattened and connected to the four output values (up, down, left, right) via a fully connected (FC) layer that employs the softmax function to the output layer

region of the layer before it (namely the receptive field). Typically, each layer of a CNN architecture arranges its neurons across 3 dimensions: width, height and depth; see Fig 3.17 for an example convolutional layer. Multiple such convolutional layers are connected (fully or partially) to form a CNN architecture. Second, CNNs build on the principle of receptive fields and their spatial locality. While spatial locality is maintained between adjacent layers of a CNN when one stacks several convolutional layers together the CNN gradually learns to yield global relationships out of the local spatial relationships existent in between layers. At its early layers the CNN creates representations of small regions of its observations. Those local representations are gradually assembled into representations of larger regions and eventually end up representing the entire image. Third, each layer of the CNN employs a number of filters (kernels) with trainable weights and predetermined size across the entire visual field. Each filter has the same trainable weights that multiply the input (via dot product) to produce a feature map (see Fig. 3.17). Finally, a CNN contains pooling layers that first divide each feature map into rectangle-shared areas and then each of these areas is compressed into a single value that represents the average value (average pooling) or the maximum value (max pooling) of the area considered.

The weight sharing property of filters reduces the number of trainable parameters, thereby lowering the computational effort of training the CNN and furthermore allowing training to scale to larger and more efficient networks. CNNs are normally trained though efficient variants of stochastic gradient descent such as Adam [568].

CNNs for Ms Pac-Man

Figure 3.17 offers an illustrative example of a CNN employed to play Ms Pac-Man. To train the CNN to play Ms. Pac-Man we can rely on a dataset of expert play in the game. Such a dataset may include in-game footage data such as grayscaled values of

each pixel on the screen, on the one hand, and corresponding player actions on the other. The Atari-HEAD dataset [1313] for instance can serve such a propose. The simplified CNN shown in Fig. 3.17 observes a grayscale and low resolution version of the game's image (i.e., 32×32) and employs a single convolutional layer and a max pooling layer to extract visual features from the in-game screen. A ReLu activation ($ReLU(z) = \max(0, z)$) is employed in between the two layers. The extracted visual features of the game screens are represented as the fully connected layer of size 675. This feature vector can be used, in turn, to perform action classification in the last layer of the CNN.

The CNN can be pretrained as a visual encoder that compresses images to a latent vector of 675 values for any downstream task such as gameplaying. Alternatively, a CNN can be trained end-to-end to directly predict actions from images. Normally a CNN stacks more convolutional and pooling layers than the simplified example illustrated here. It is also expected that the layers have more neurons and that the CNN input is an RGB image of higher resolution.

3.6.4 Generative Adversarial Networks

Generative Adversarial Networks (GAN) are, as the name suggests, networks capable of generating artifacts (such as images, sound, or 3D models) which are trained using adversarial learning methods. GANs became popular after they were invented by a team led by Goodfellow in 2014 [391], but the basic idea has been around for a long time; for example, GANs have much in common with competitive coevolution. GANs are generally seen as a self-supervised learning method, meaning that they are trained on a set of artifacts (typically images) without targets, and learn to produce new artifacts from the same distribution as the original artifacts (i.e., new artifacts similar to the artifacts they were trained on). In the remainder of this section we discuss image generation, as this is the modality GANs have been applied most often to and have yielded the most impressive results with.

A GAN consists of two different neural networks: a **generator**, G, and a **discriminator**, D; refer to Fig. 3.18. The generator is trained to generate images that are similar to those in the training set. The discriminator, instead, is trained to tell the real images in the training set apart from the generated images. At the end of training both networks we normally disregard the discriminator and we use the generator for creating new samples. The high-level steps of the algorithm are as follows:

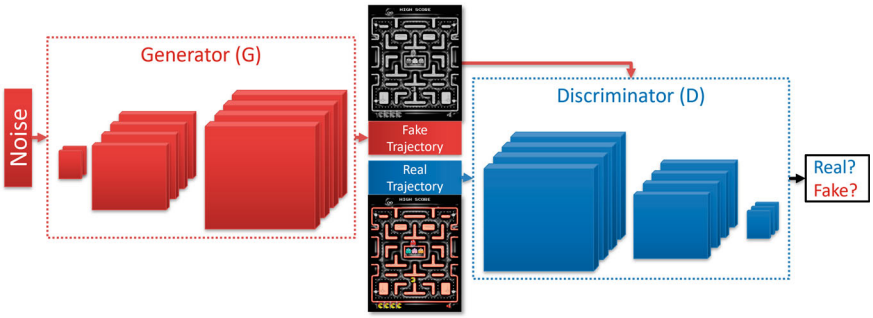


Fig. 3.18 Learning to play Ms Pac-Man in a human-like fashion with a GAN

1. The generator, G, produces a set of fake generated samples such as images.
2. Real samples (images) are obtained from a training corpus. Both sample types (real and fake) define the training corpus of the GAN.
3. The discriminator, D, attempts to predict which of the samples are real and which are fake. Its loss is backpropagated and D is updated.
4. After a predefined number of iterations of steps 1–3 is completed, the cumulative loss is backpropagated to the generator. Note that G wishes to fool D so that the latter incorrectly identifies the output of the former as real.
5. Steps 1–4 are repeated until convergence.

The input to the generator is some fixed-length vector from a latent space that one can alter and observe variations in the generated output. We meet GANs in AI and games research primarily as a content generation method often coupled with search methods such as *latent variable evolution* [1223] or reinforcement learning methods [1038, 1237, 1239]. While Part III covers a number of GAN uses cases for content generation, in this section we remain consistent with our running example and outline the use of a GAN for playing Ms Pac-Man.

GANs for Ms Pac-Man

To exemplify the use of GANs for playing Ms Pac-Man we adopt the principles of the approach introduced in [535] for imitating human mouse trajectories in first person shooters. Here instead we will employ a GAN as a gameplaying trajectory generator in Ms Pac-Man. Figure 3.18 illustrates a high level overview of the method. Supposedly we have a corpus of human gameplaying trajectories in Ms Pac-Man such as those available in the Atari-HEAD dataset [1313]. We train the generator (G) to generate fake gameplayng trajectories and we train the discriminator to distinguish between the generated fake trajectories and the real ones available in our corpus. The resulting trained generator is a Ms Pac-Man player that is able to imitate human play.

3.6.5 *Diffusion Models*

Diffusion probabilistic models [455], or *diffusion models* for brevity, are generative methods that learn a diffusion process over a dataset that can, in turn, generate a probability distribution and sample new data from. The diffusion model is the underlying architecture of state of the art high resolution image generation methods such as DALLE-3 [84] and Stable Diffusion [1077]. Diffusion models can be employed for a variety of tasks in games, including visuals and image denoising and generation. Beyond image-based downstream tasks, diffusion models can be adapted and used for any content generation task and even for controlling the behavior of game agents [856]. Recent work has also shown that diffusion-based architectures can function as trained world models, meaning that they can learn dynamics from existing game engines, creating what has tentatively become known as *neural game engines* [1199].

A diffusion model differs from other generative methods in both training and inference. Its core innovation is the *iterative* nature of the training (diffusion) process, which is a Markov chain that gradually adds noise to the data until the signal is destroyed [455] (see Fig. 3.19). While diffusion models can potentially generate various modalities of output we will focus on image generation as our illustrative example in this section. Instead of constructing the image in a forward pass, the model decomposes the image generation process (sampling) in many small "denoising" steps. The generation process essentially starts with random noise which is gradually refined to an image over a number of steps. At each step, the diffusion model estimates how to reach a completely denoised image from the current noised image. In practice, a diffusion model trains a neural network to learn to sequentially remove noise from images which are blurred with Gaussian noise. To train such a model we iteratively: (i) load images from a dataset, (ii) add Gaussian noise to those images, (iii) present all created images as inputs to the model, (iv) evaluate the capacity of the model on denoising these images, (v) update the weights of the model based on its ability to denoise images.

Diffusion models are commonly formulated as Markov chains and trained using variational inference methods such as Denoising Diffusion Probabilistic Models (DDPMs) [455]. Formally, a diffusion model is a latent variable model that generates images through a two step process as illustrated in Fig. 3.19. In the first (**forward**) process, the diffusion model takes the input image $\mathbf{x}_0$ and gradually adds Gaussian noise to it through a series of steps T according to a variance schedule. The approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0) \leftarrow \Pi_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$, namely the forward process, is fixed to a Markov chain as depicted in Fig. 3.19. The **reverse diffusion (sampling)** process is also defined as a Markov chain with learned Gaussian transitions $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. A neural network is trained to recover the original data by reversing the noising process. By modeling the reverse process (i.e., predicting the noise), the diffusion model can be used for image generation. Specifically, the trained diffusion model can be prompted with any input image which is viewed as random noise. The model will then attempt to iteratively denoise the input image to a new image.

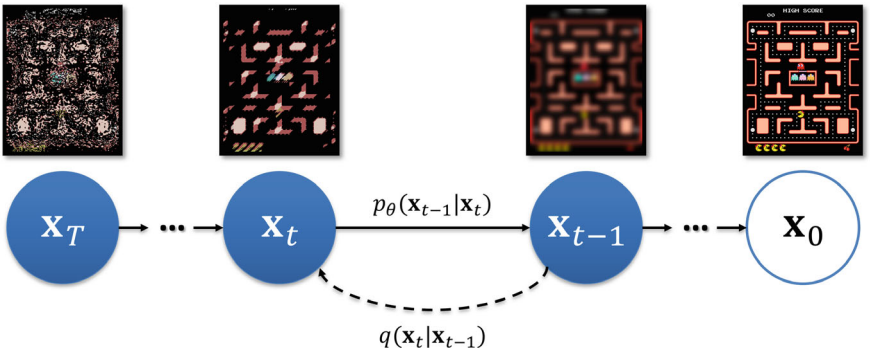


Fig. 3.19 The directed graphical diffusion model featuring its two core operational steps: **forward** and **reverse** diffusion. Image inspired by [455]

The core idea of diffusion models, to iteratively destroy an artifact (such as an image) in order to learn to regenerate it from a destroyed state, is simple enough to have been independently invented, probably multiple times. For example, the *Path of Destruction* algorithm is akin to a discrete, sequential diffusion model; it operates by changing tiles one at a time rather than modifying a whole image simultaneously. As a result, it is slower but more data-efficient [1048, 1049].

Diffusion Models for Ms Pac-Man

How can an image generation method be used to guide the decision making process of Ms Pac-Man? What if, instead of images, a diffusion model could be trained to predict the noise of (and denoise) an action vector? In principle, such a model could be utilized to generate sequences of gameplaying actions (see Fig. 3.20). Our proposal for a Ms Pac-Man controller is inspired by the *observation-to-action* diffusion model,

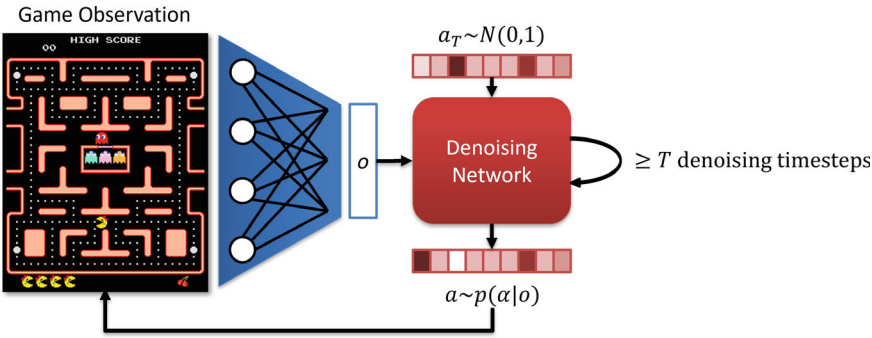


Fig. 3.20 Illustrating the use of a (observation-to-action) diffusion model in Ms Pac-Man. Image inspired by [856]

introduced in [856], and depicted in Fig. 3.20. An encoder (e.g., a CNN) observes the pixels of the in-game screen and is trained to compress that information to a latent observation vector o. The denoising network considers the current observation and a noisy action α_T (similarly to a noisy image) and performs T steps to yield a denoised action α that controls Ms Pac-Man.

3.6.6 Transformers

The transformer architecture was introduced in 2017 by Vaswani et al. [1211]. While the original application of the transformer was on language translation, the method has been extremely influential across several domains and downstream tasks including computer vision, and multi-modal processing. Transformers are the key elements in the architecture of most large language and large multimodal models such as the Bidirectional Encoder Representations from Transformers (BERT) [278] and the Generative Pre-trained Transformer (GPT) models. The ability of the transformer to perform parallelized processing of input sequences makes it a faster autoregressive method compared to earlier recurrent neural architectures (e.g., LSTMs [456]).

The basic steps of the algorithm are as follows. The input (text in the original version of the algorithm) is split into a number of tokens for a given context length. Each token is transformed to a (word) embedding vector and then it is aligned with other tokens of the context. This is achieved through multiple multi-head attention layers which end up emphasizing key tokens of the input and ignoring less important tokens. Performing a forward operation throughout the several alternating multi-head attention and feed-forward layers (the basic elements of each transformer layer) the output of the transformer predicts the next token (e.g., the next most probable word of the input sentence).

The transformer architecture relies on a number of core building blocks. First, the transformer processes tokens; each **token** is a unit of information that can be literally anything including text, images and audio.[5] Second, **embeddings** are normally used to transform tokens to continuous-valued vectors that can then be processed by the transformer; an embedding is a low-dimensional representation of a symbol (such as a word, a character, a sentence, a part of an image, or a few notes from a song). Embeddings can also be used to position tokens in the continuous-valued representation. Third, transformers are trained through the process of **autoregression** which considers previously generated tokens as input when generating the next token(s). Finally, transformer architectures rely on the principles of attention and self-attention. **Attention** is a measure that describes the relationship between any two vectors and captures how "aligned" they are; normally attention is measured through variants of *cosine similarity* [1211]. When instead we attempt to measure

[5] For context, the tokenizer for OpenAI's GPT-4o model has around 200, 000 unique tokens, where each token correspond to, on average, 4 characters. In English, a token equals approximately 0.75 words.

the similarity within the same sequence, then we are faced with the **self-attention** mechanism that is core to a transformer. As a result, self-attention enables us to find associations between different words of an input sentence or patches of an input image. Think of the sentence "I love Artificial Intelligence and Games", for example. Many self-attention heads in a trained transformer will likely associate the word "Artificial" with "Intelligence" with a higher weight value than the words "I" or "love". This is because "Artificial" and "Intelligence" co-occur more frequently in existing text corpora that any other word combination within this sentence. Likewise a patch of an image showing a car will be more tightly associated with patches showing parts of a road instead of patches depicting a blue sky. In both examples the available training data (text or images) offer the necessary linguistic or visual information for deriving such associations.

The transformer layer (consisting of multi-head attention and feed-forward layers) can be part of either an encoder or a decoder. While the original transformer architecture uses both [1211], several following variants of the architecture solely consist of an encoder (e.g., BERT) or decoders (e.g., GPT). Out of the myriad transformer-based methods that are available in the literature today we will focus on the **vision transformer** and the **decision transformer** as the most representative and relevant to AI and games downstream tasks. Other types of transformer have also been tested and found to be particularly well-suited for processing board game states—often consisting of several dissimilar discrete components—[850]. Importantly, note that the *AlphaStar* agent discussed in various chapters of this book also uses a transformer core architecture that processes input data from the raw game interface of *StarCraft II* (Blizzard Entertainment, 2010) [1219].

Vision Transformer

The Vision Transformer (ViT) [290] relies on aspects of the original transformer architecture but it is used for processing images instead of text. The core idea of a ViT is to treat images as a sequence of patches—analogously to sequences of text tokens—and perform autoregressive modeling onto these patch sequences using transformer layers. The basic steps of the algorithm are as follows (see also Fig. 3.21).

1. An input image is split into a number of patches following the tokenization principle. Images are thus reshaped into a sequence of 2D square patches of arbitrary size p.
2. The 2D patches are flattened to a vector of size cp^2 where c is the number of channels of the image.
3. The flattened patches are treated as tokens which are converted to lower-dimensional linear embeddings.
4. Positional embeddings are added to indicate where each patch is situated within an image.

5. The sequence is fed as an input to a transformer encoder.
6. The ViT is trained on labeled images of a dataset such as annotated game frames.
7. The pretrained ViT model can be further trained (i.e., finetuned) to perform any downstream task such as detecting game objects, playing a game or recognizing human affect from pixels. A new downstream dataset is required.

The analogies between a text transformer and an image transformer are many. Instead of text sequences, the attention mechanism of a ViT transforms vectors that represent image patches, autoregressively yielding semantic relations between patches in an image. Compared to CNNs covered earlier, ViTs lack inductive biases. In particular ViT are *translation invariant* methods capable of recognizing an object that appears on an image independently of its position. Their receptive field is also not locally restricted; CNNs instead only perceive the neighbor values determined by a kernel. However, a ViT cannot process (by autoregressive design) grid-based representations of images and instead requires to process sequences. This is the reason why one needs to convert a spatial, yet non-temporal and non-sequential, input such as an image to a sequence of patches. While the patch-based perception of an image allows a ViT to be able to capture global relationships in the image, this benefit normally comes at the cost of heavier training processes and larger datasets.

Vision Transformer for Ms Pac-Man

Figure 3.21 depicts an example of a Vision Transformer used to process pixel information obtained from Ms Pac-Man as 9 patches. The ViT processes input patch and position embeddings and autoregressively builds a game frame representation via the transformer encoder. As with the autoencoder model covered earlier the low-dimensional representation obtained from the transformer can be used for any AI downstream task within games. That includes Ms Pac-Man gameplaying agents that are trained to behave in a human like-fashion (conditioned on a set of human gameplay demonstrations) or playing to maximize performance through reinforcement learning (given an appropriately designed reward function such as the game score). Alternatively, the ViT latent can, for instance, feed downstream models that are able to predict player skill of in-game engagement, given a set of human demonstrations.

Decision Transformer

Decision Transformers (DTs) were introduced in 2021 [203] for re-purposing reinforcement learning as a sequence modeling task. A DT relies on a transformer architecture that trains an autoregressive model on a reward function, earlier observations (s), and actions (a) to predict future actions aiming for a desired return ($\hat{R}$). DTs use a GPT architecture equipped with a causal self-attention mask. The causal mask

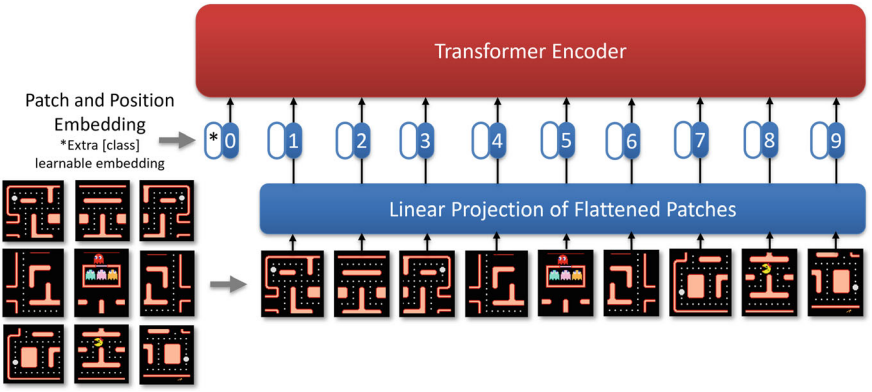


Fig. 3.21 Illustrating the use of a Vision Transformer in Ms Pac-Man. The game's frames are observed as 9 patches that are flattened and positioned. The patch and position embeddings feed a transformer encoder. Image inspired by [290]

forces DT predictions to only attend to the tokens at previous positions. The DT will generate optimal trajectories by training a GPT model to predict the next token in a sequence of returns-to-go (i.e., desired returns), states, and actions. GPTs are powerful at predicting the next token in a sequence and are thus used by DTs as efficient surrogate models of good sequences. The core algorithmic steps of a DT are as follows (the reader is also referred to Fig. 3.22):

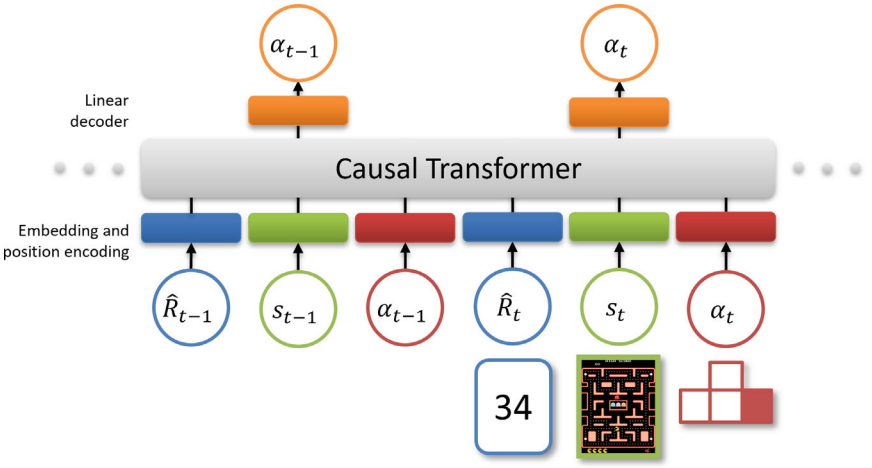


Fig. 3.22 Illustrating the use of a Decision Transformer for playing Ms Pac-Man. The causal transformer of the DT considers K timesteps of the following tokens: state (e.g., game frames), action (e.g., Ms Pac-Man action) and return-to-go (e.g., desired score in a number of timesteps). Each token type is embedded and positioned via an appropriate encoding scheme. The causal mask of the transformer forces Ms Pac-Man action predictions to only attend to the tokens at previous positions (i.e., $t-1$). Image inspired by [203]

1. We feed K timesteps into the DT. Each timestep t considers 3 token types: (i) Return-to-go, $\hat{R}_t$ (ii) State, S_t and (iii) Action, a_t.
2. All tokens are embedded and positioned via an appropriate encoding scheme. One for instance may use a linear layer if the state is a vector or, alternatively, a CNN encoder if the state is a frame.
3. The inputs are processed by a GPT model which learns to the predict the future action, a_{t+1}, via autoregressive modeling.

A DT can serve any RL problem but it is particularly powerful when applied to offline RL tasks. Instead of obtaining data via environment interactions, offline RL only has access to a labeled dataset (e.g., human demonstrations) consisting of trajectory rollouts of arbitrary policies. Imagine for instance human gameplaying demonstrations (i.e., state–action pairs) and corresponding scores (i.e., returns-to-go) of an Atari game. One might claim that offline RL is harder than online RL as it restricts the ability of game agents to explore the environment and collect additional experience and rewards. In the remainder of this section we employ a DT for the downstream task of playing Ms Pac-Man.

Decision Transformer for Ms Pac-Man

Let us consider that you have access to many human gameplaying demonstrations for Ms Pac-Man as obtained, for instance, from the Atari-HEAD dataset [1313]. Demonstrations include trajectories of state action pairs combined with aggregated in-game scores over time. To train a DT to learn to play Ms Pac-Man like the best of human players in the dataset you will need to condition the autoregressive model to high returns-to-go $\hat{R}$ such as the highest possible scores obtained within K timesteps of the game. In addition to the $\hat{R}$, the K actions of the players and corresponding states of the game—represented as game frames—are embedded and feed the causal transformer of the DT. The DT is trained to predict the next action for Ms Pac-Man based on the provided trajectories available in the dataset as in [203].

3.6.7 Large Language Models

Not so long time ago, research on models that are trained on text to predict text—i.e., autoregressive language modeling—was a niche area within natural language processing. The release of the GPT-2 model in 2019 [920] initially turned the niche topic to a mainstream area within NLP and gradually dominated AI research at large. LLMs (of the GPT family, the Llama family [299, 1180], the Claude family, the Gemini family [1129], and beyond) showcased that transformer architectures trained on

large text corpora could generate coherent text of impressively high-quality. Moreover, it was shown that text generation could be controlled within bounds through text prompts given by the user.

Even more recently the large *scale* in terms of data, compute and model size led to breakthrough models such as those of the Llama [1180] and the Mistral [509] family. Scaling was supported by algorithmic innovations such as instruction fine-tuning and reinforcement learning from human feedback [551] that collectively led to LLMs like ChatGPT. During the last few years it feels like—almost out of nowhere—LLMs can answer any question we have for them and can do almost anything they are tasked with. It is no surprise that, at the time of writing, LLMs are a very active research field with a particular focus on ways algorithmic innovations may improve the quality of LLM outcomes and artifacts (text, images, sounds and even videos), on ways methods may reduce their compute and memory requirements, on ways we can better analyze them and understand their current capabilities, and on ways we can use LLMs as building blocks for various agents and other systems. Related to the latter research focus, opinions vary widely, from those who view LLMs as "*sparks* of artificial general intelligence" [144] to those who view them as information retrieval approximators from a lossy compression of the internet [732].

How can a large model that is trained on text be used in video games, however? The answer might be surprising as LLMs can be useful under any of the 3 main roles of AI in games: to play, to create and to model players. LLMs for instance can be used to equip non-player characters with high standard conversational and reasoning abilities which can, in turn, act as friends, enemies, assistants or even game masters [365]. Importantly, an LLM can operate within the game as a player that plays the game with super-human performance [331, 930], imitates human play, or tests the game for developers. Beyond gameplaying (as a human player or an NPC), LLMs can act as a (co-)designer of parts of the player experience. Finally, LLMs can potentially help us identify critical aspects of player behavior and experience that can be used to model players. The book covers use cases of LLMs for gameplaying, content creation and player modeling in its corresponding parts.

LLMs for Ms Pac-Man

Employing LLMs for playing the game of Ms Pac-Man can be achieved in many ways. The most straightforward way we can think of is to use a large multimodal model—such as GPT-4 [5]—that observes frames (or even short videos) of Ms Pac-Man and is then trained to perform the next best action to take in the game. An alternative way is to request an LLM to generate code that plays the game well and keep asking the model to refine and improve on its performance, similarly to the Voyager GPT-4 based agent that was trained to play *Minecraft* (Mojang, 2011) [1232].

3.6.8 Further Reading

The ideal place to start investigating deep learning principles and architectures in detail in the deep learning book by Goodfellow et al. [390] and the more recent book by Bishop and Bishop [92]. The interested reader may also refer to the various papers that we cite independently throughout the deep learning methods of this section. In addition, we recommend a relatively recent survey paper that covers the use of various DNN methods (including autoencoders and CNNs) for gameplaying [520] and a recent application of diffusion models for gameplaying agents [856]. GANs are detailed in [392] whereas vision transformers and decision transformers are detailed in [290] and [203], respectively, for the interested reader. When it comes LLMs, the reader is referred to a recent survey on their various applications to games [365].

3.7 Reinforcement Learning Methods

Reinforcement learning can be applied to any of the three use cases we examine in this book. RL is particularly useful for generating gameplaying agents that learn to win a game or learn to do anything else but winning (e.g., play in a human-like fashion or test a game) as we discuss in Part II. RL is also useful for the generation of content through the PCGRL framework [559] that we detail in Part III. Finally, RL methods can be used to model aspects of playing behavior and experience as covered in Part IV. While we detailed the core principles of RL in the previous chapter, in this chapter we start with **Q-learning**, the most popular algorithm in the RL literature with the widest use in game AI research. The reminder of the section is dedicated to methods that train and use neural networks as function approximators for RL. Beyond the underlying algorithm used in TD-Gammon [1134] we discuss the use of **deep Q network**, and **proximal policy optimization** in games as more recent methods that rely upon the principles of TD-Gammon. We end this chapter with a brief presentation of **neuroevolution** in games.

3.7.1 Q-Learning

Q-learning [1243] is a model-free, off-policy, TD learning algorithm that relies on a tabular representation of $Q(s, a)$ values (hence its name). Informally, $Q(s, a)$ represents how good it is to pick action a in state s. Formally, $Q(s, a)$ is the expected discounted reinforcement of taking action a in state s. The Q-learning agent learns from experience by picking actions and receiving rewards via bootstrapping. The goal of the Q-learning agent is to maximize its utility (i.e., expected reward) by picking the right action at each state. The reward is a weighted sum of the expected values of the discounted future rewards. The Q-learning algorithm is a simple update

on the Q values in an iterative fashion. Initially, the Q table has arbitrary values as set by the designer. Then each time the agent selects an action a from state s, it visits state s', it receives an immediate reward r, and updates its $Q(s, a)$ value according to the following *learning rule*:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\{r + \gamma \max_{a'} Q(s', a') - Q(s, a)\} \tag{3.15}$$

where $\alpha \in [0, 1]$ is the **learning rate** and $\gamma \in [0, 1]$ is the **discount factor**. The learning rate determines the extent to which the new estimate for Q will override the old estimate. The discount factor weighs the importance of earlier versus later rewards; the closer γ is to 1, the greater the impact of future reinforcements. As seen from Eq. (3.15), the algorithm uses bootstrapping since it maintains estimates of how good a state-action pair is (i.e., $Q(s, a)$) based on how good it thinks the next state is (i.e., $Q(s', a')$). It also uses a one-step-depth, full-breadth backup to estimate Q by taking into consideration all Q values of all possible actions a' of the newly visited state s'. It is proven that by using the learning rule of Eq. (3.15) the $Q(s, a)$ values converge to the expected future discounted reward [1243]. The optimal policy can then be calculated based on the Q-values; the agent in state s selects the action a with the highest $Q(s, a)$ value. In summary, the basic steps of the algorithm are as follows:

Given an immediate reward function r and a table of $Q(s, a)$ values for all possible actions in each state:

1. Initialize the table with arbitrary Q values; e.g., $Q(s, a) = 0$.
2. $s \leftarrow$ Start state.
3. While not finished* do:
   (a) Choose an action a based on policy derived from Q (e.g., ϵ-greedy).
   (b) Apply the action, transit to state s', and receive an immediate reward r.
   (c) Update the value of $Q(s, a)$ as per Eq. (3.15).
   (d) $s \leftarrow s'$.

*The most commonly used termination conditions are the algorithm's *speed*—i.e., stop within a number of iterations—or the *quality* of convergence—i.e., stop if you are satisfied with the obtained policy.

Limitations of Q-learning

Q-learning has a number of limitations associated primarily with its tabular representation. First of all, the size of the state-action space might be computationally very expensive to handle. As the Q table size grows, our computational needs for memory allocation and information retrieval increase. Further, we may experience very long convergence times since learning time is exponential to the size of the state-action

space. To overcome these obstacles and get decent performance from RL learners we need to devise a way of reducing the state-action space. Section 3.7.2 outlines the approach of using artificial neural networks as Q-value function approximators, directly bypassing the Q-table limitation and yielding compressed representations for our RL learner.

Q-learning for Ms Pac-Man

Q-learning is applicable for controlling Ms Pac-Man as long as we define a suitable state-action space and we design an appropriate reward function. A state in Ms Pac-Man could be represented directly as the current snapshot of the game—i.e., where Ms Pac-Man and ghosts are and which pellets and power pills are still available. That representation, however, yields a prohibitive number of game states for a Q-table to be constructed and processed. Instead, a more indirect representation such as whether ghosts and pellets are nearby or not. Possible actions for Ms Pac-Man could be that it either keeps its current direction, it turns backward, it turns left, or it turns right. Finally, the reward function can reward Ms Pac-Man positively when it eats a pellet ($+1$), a ghost ($+50$) or a power pill ($+100$) and penalize Ms Pac-Man when it dies ($-1,000$). Dissimilar reward schemes, tabular representations, and hyperparameters such as the learning rate, are expected to influence the performance of Q-learning in playing Ms Pac-Man. As with every other method presented the reader is expected to evaluate and compare various experimental settings in terms of in-game score.

It is important to note that both Pac-Man and Ms Pac-Man follow the Markov property in the sense that any future game states may depend only upon the present game state. There is one core difference, however: while the transition probability in Pac-Man is known given its deterministic nature, it is largely unknown in Ms Pac-Man given the stochastic behavior of the ghosts in that game. Thereby, Pac-Man can theoretically be solved via model-based approaches (e.g., dynamic programming) whereas the world model of Ms Pac-Man can only be approximated via model-free methods such as temporal difference learning.

3.7.2 TD Learning with Artificial Neural Networks

AI methods can be interwoven in numerous ways to yield new sophisticated algorithms that aggregate the strengths of their combined parts. You can, for instance, let GAs evolve your behavior trees or FSMs; you can instead empower MCTS with ANN estimators for tree pruning; or you can add a component of local search in every search algorithm covered earlier. We call the resulting combinations of AI methods **hybrid** algorithms and in the remainder of this section we cover a number of influential hybrid game AI algorithms: variants of **TD learning with ANN function approximators**.

Historically, reinforcement learning used tabular representations to store knowledge. Representing knowledge this way may is extremely inefficient in terms of

memory and compute since the size of the look-up table increases exponentially with respect to the action-state space. Given the sparsity of rewards, for any high-dimensional problem these tables would populate so slowly as to be useless. The most popular way of addressing this challenge is to use an ANN as a value (or Q value) *approximator*, thereby replacing the table. The algorithm can then be applied to larger spaces of action-state representations. Further, an ANN as a function approximator of Q can handle problems with continuous state spaces which are infinitely large.

In this section, we outline two milestone examples of hybridizing the ANN universal approximation capacity with temporal difference learning. While the core algorithm is the same in both cases, there are many differences in the details. **TD-Gammon** is the system that first managed to effectively combine TD-learning with self-play and (relatively shallow) neural networks, achieving very strong performance of the board game Backgammon. For more than a decade, TD-Gammon was the go-to example of reinforcement learning that actually worked. **Deep Q-networks** were first applied to playing Atari 2600 arcade games, and managed to reach super-human proficiency. The underlying ideas behind these examples extend to many other problems, but here we will discuss these variations of TD+ANN along with the games they were first applied to.

TD-Gammon

Arguably one of the most popular success stories of AI in games is that of Tesauro's TD-Gammon software that plays backgammon on the grandmaster-level [1134]. The learning algorithm was a hybrid combination of an MLP and a temporal difference variant named TD(λ); see Chap. 7 of [1109] for further details on the TD(λ) algorithm.

TD-Gammon used a standard multi-layer neural network to approximate the value function. The input of the MLP was a representation of the current state of the board (Tesauro used 192 inputs) whereas the output of the MLP was the predicted probability of winning given the current state. Rewards were defined as zero for all board states except those on which the game was won. The MLP was then trained iteratively by playing the game against itself and selecting actions based on the estimated probability of winning. Each game was treated as a training episode containing a sequence of positions which were used to train the weights of the MLP by backpropagating temporal difference errors of its output.

TD-Gammon 0.0 played about $300,000$ games against itself and managed to play as well as the best backgammon computer of its time. While TD-Gammon 0.0 did not win the performance horse race, it gave us a first indication of what is achievable with RL even without any backgammon expert knowledge integrated in the AI algorithm. The next iteration of the algorithm (TD-Gammon 1.0) naturally incorporated expert knowledge through specialized backgammon features that altered the input of the MLP and achieved substantially higher performance. From that point onwards the number of hidden neurons and the number of self-played games determined the capacity of the algorithm. From TD-Gammon 2.0 (40 hidden neurons) to TD-Gammon 2.1 (80 hidden neurons) the performance of TD-Gammon

gradually increased and, with TD Gammon 3.0 (160 hidden neurons), it reached the playing strength of the best human player in backgammon [1134].

Deep Q Network

While the combination of RL and ANNs results in very powerful hybrid algorithms, the performance of the algorithm traditionally depended on the design of the input space for the ANN. As we saw earlier, even the most successful applications of RL such as the TD-Gammon agent managed to reach human-level playing performance by integrating game specific features in the input space, thereby adding expert knowledge about the game. It was only recently that the combination of RL and ANNs managed to reach human-level performance in a game without considering ad-hoc designed features, instead discovering them merely through learning. A team from Google's DeepMind [773] developed a reinforcement learning agent called *deep Q network* (DQN) that trains a deep convolutional ANN via Q-learning. DQN managed to reach or exceed human-level playing performance in 29 out of 46 arcade (Atari 2600) games of the Arcade Learning Environment [75] it was trained on [773].

DQN is inspired by and based upon TD-Gammon since it uses an ANN as the function approximator for TD learning via gradient descent. As in TD-Gammon, the gradient is calculated by backpropagating the temporal difference errors. However, instead of using TD(λ) as the underlying RL algorithm, DQN uses Q-learning. Further, the ANN is not a simple MLP but rather a deep convolutional neural network. DQN played each game of ALE for a large number of frames (50 million frames). This amounts to about 38 days of playing time for each game [773].

The DQN analyses a sequence of four game screens simultaneously and approximates the future game score per each possible action given its current state. In particular, the DQN uses the pixels from the four most recent game screens as its inputs, resulting in ANN input size of 84×84 (screen size in pixels) $\times 4$. No other game-specific knowledge was given to the DQN beyond the screen pixel information. The architecture used for the convolutional ANN has three hidden layers. The first (low-level) layers of the DQN process the pixels of the game screen and extract specialized visual features. The convolutional layers are followed by a fully connected hidden layer and an output layer. Given a game state represented by the network's input, the outputs of the DQN are the estimated optimal action values (optimal Q-values) of the corresponding state-action pairs. The DQN is trained to approximate the Q-values (the actual score of the game) by receiving immediate rewards from the game environment. DQN uses an ϵ-greedy policy for its action-selection strategy. It is worth mentioning that, at the time of writing, there are newer and more efficient implementations of the deep reinforcement learning concept such as the Asynchronous Advantage Actor-Critic (A3C) algorithm [772] or the proximal policy optimization method [1006] we cover in Sect. 3.7.3. Variants of deep RL have been successful at playing games with human-level performance across a number of games including *StarCraft II* (Blizzard Entertainment, 2010) [1219] and *Dota 2* (Valve, 2013) [77].

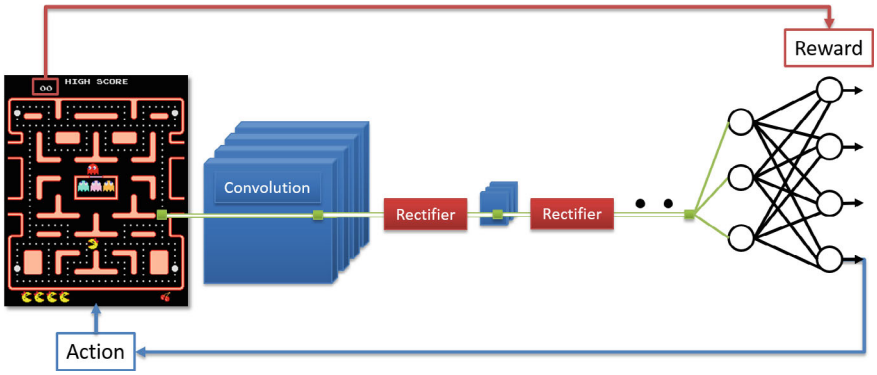


Fig. 3.23 A deep Q-learning approach for Ms Pac-Man. Following [773], the network's first part contains a set of convolution layers which are followed by rectifier nonlinearities. The final layers of the DQN we present in this example are fully connected employing ReLUs, as in [773]

TD Learning with ANN Function Appoximator for Ms Pac-Man

Our proposed DQN approach for controlling Ms Pac-Man is similar to the DQN agents trained at the ALE framework [773]. A deep convolutional neural network scans the level image on a pixel-to-pixel basis (see Fig. 3.23). The image goes through a number of convolution and fully connected layers which eventually feed the input of an MLP that outputs the four possible actions for Ms Pac-Man (keep direction, move backwards, turn left, turn right). Once an action is applied, the score of the game is used as the immediate reward for updating the weights of the deep network (the convolutional ANN and the MLP). In particular, the reward is $+1$ if the score increases in between two successive time steps (frames), it is -1 if the score decreases, and 0 otherwise. Playing for a sufficient time period allows the controller to gather experience (image snapshots, actions, and corresponding rewards) that would train the deep ANN to approximate a policy that maximizes the score for Ms Pac-Man.

3.7.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [1006] is an important **policy gradient** algorithm, that trains an agent's *policy network*. The policy network is normally a neural network function that the agent uses to make decisions that maximize expected rewards. PPO has received wide attention after its successes across a number of domains including beating professional players in *Dota 2* (Valve, 2013) [77]. One could argue that the PPO family of algorithms define the current state of the art as they seem to offer competitive advantages in terms of simplicity, robustness, and sample efficiency.

To train the policy network to perform well under a predetermined reward function, the PPO algorithm iteratively requests policy updates (e.g., ANN weight adjustments). Similarly to a learning rate, the magnitude of such updates is a critical parameter for training. Large steps may direct policy in the areas of the search space with little potential in terms of optimization (e.g., local minima). Unnecessarily small steps may increase the computational effort and the algorithm's efficiency. PPO features a function $\hat{A}_t$ (i.e., the *advantage estimate*) that constrains the policy update from being way too large or way too small.

More formally, the vanilla form of a policy gradient method [1006] uses the following policy gradient loss to update a set of θ policy parameters (i.e., typically ANN weights):

$$L^{PG}(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{3.16}$$

where $\hat{E}_t$ is the empirical expectation over timestep t, π_θ is the current policy of the policy network (i.e., represented by θ); hence $\log \pi_\theta(a_t|s_t)$ represents the log probabilities derived from the output of the policy network that takes action a_t considering state s_t as its input. $\hat{A}_t$ is the relative value of the selected action (i.e., the estimated advantage) in the current state at timestep t. To compute $\hat{A}_t$ one requires two components: the discounted sum of rewards (or return R_t) and a baseline estimate $b(s_t)$. The former is the total discounted reward from timestep t, $\sum_{k=1}^{\infty} \gamma^k R_{t+k}$, where $\gamma \in [0, 1]$ is the discount factor that, as in Q-learning, the higher it gets, the more it promotes far-sighted evaluation of the reward. The latter component, $b(s_t)$, is the value function of a neural network which estimates the discounted return from timestep t onwards. The advantage estimate is then computed as: $\hat{A}_t = R_t - b(s_t)$ and, in principle, it informs the policy gradient method about how much better or worse the action it ended up taking is compared to its own estimate. Updating the policy network via the L_{PG} loss, however, often leads to destructively large policy updates [1006].

PPO is a method that builds on *trust region policy optimization* (TRPO) [1005] but instead proposes a simpler and more efficient policy gradient objective. For that purpose, PPO defines the ratio of the probability under the new and old policies $r_t(\theta)$ as $\pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ where θ_{old} is the vector of policy parameters before the update. The PPO objective under optimization is as follows:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)] \tag{3.17}$$

where ε is hyperparameter for clipping in the policy objective, usually set to 0.1 or 0.2. This new learning objective—compatible with stochastic gradient descent—yields better sample efficiency (e.g., compared to Q-learning), taking far less steps to learn a decent policy. Results reported across several use cases of PPO showcase superior performance of the algorithm on continuous control tasks despite being far simpler that other policy gradient RL methods.

PPO for Ms Pac-Man

Similarly to the application of deep Q-learning for controlling Ms Pac-Man, employing PPO for the same downstream task adopts a reward function that is related to the game's score. The policy network may observe the game s_t via its pixels or alternatively via specific engineered features that are important for controlling the agent such as the distance to the nearest pellet or ghost. The weights of the policy network θ are updated via the $L^{CLIP}(\theta)$ objective to learn to take the action (up, down, left, right) that maximizes the expected return.

3.7.4 Neuroevolution

The evolution of artificial neural networks, or else *neuroevolution*, refers to the design of artificial neural networks—their connection weights, their topology, or both—using evolutionary algorithms [1293]. Neuroevolution is a method grounded in biological metaphors and evolutionary theory and inspired by the way brains evolve [946].

This evolutionary (reinforcement) learning approach is applicable in a wide range of AI problems since many of those can be viewed as global optimization problems whose underlying general function can be approximated via an ANN. The method has been successfully applied in the domains of artificial life, robot control, generative systems and computer games. However, it is particularly suitable either when the error function available is not differentiable or when target outputs are not available. The former may occur, for instance, when the activation functions employed in the ANN are not continuous and, thus, not differentiable. This is a prominent condition, for instance, in the compositional pattern producing networks [1078]. The latter may occur in a domain for which we have no samples of good (or bad) behavior or it is impossible to define objectively what a good behavior might be. Instead of backpropagating the error and adjusting the ANN based on gradient search, neuroevolution designs ANNs via metaheuristic (evolutionary) search. In contrast to supervised learning, neuroevolution does not require a dataset of input-output pairs to train ANNs. Rather, it requires only a utility of an ANN's performance on the problem under investigation, for instance, the score of a game playing agent that is controlled by an ANN.

The core algorithmic steps of neuroevolution are as follows:

1. A population of chromosomes that represent ANNs is evolved to optimize a fitness function that characterizes the utility (quality) of the ANN representation. The population of chromosomes (ANNs) is typically initialized randomly.
2. Each chromosome is encoded into an ANN which is, in turn, tested on the task under optimization.
3. The testing procedure assigns a fitness value for each ANN of the population. The fitness of an ANN defines its measure of performance on the task, such as the score of a gameplaying session.
4. Once the fitness values for all genotypes in the current population are determined, a selection strategy (e.g., roulette-wheel, tournament) is applied to pick the parents for the next generation.
5. A new population of offspring is generated by applying genetic operators on the selected ANN-encoded chromosomes. Mutation and/or crossover are applied on the chromosomes in the same way as in any evolutionary algorithm.
6. A replacement strategy (e.g., steady-state, elitism, generational) is applied to determine the final members of the new population.
7. Similarly to a typical evolutionary algorithm, the generational loop (steps 2 to 6) is repeated until we exhaust our computational budget or we are happy with the obtained fitness of the current population.

There are two main types of neuroevolution approaches: those that consider the evolution of a network's connection weights only and those that evolve both the connection weights and the topology of the network (including connection types and activation functions). In the former type of neuroevolution, the weight vector is encoded and represented genetically as a chromosome; in the latter type, the genetic representation includes an encoding of the ANN topology. Beyond simple MLPs, the ANN types that have been considered for evolution include the NeuroEvolution of Augmenting Topologies (NEAT) [1081] and the compositional pattern producing networks [1078].

Neuroevolution has found extensive use in the games domain [946] in roles such as evaluating the state-action space of a game, selecting an appropriate action, selecting among possible strategies, modeling opponent strategies, generating content, and modeling player experience. The algorithm's efficiency, scalability, broad applicability, and open-ended learning are a few of the reasons that make neuroevolution a good general method for many game AI tasks [946]. On the other hand, many evolutionary algorithms struggle with high-dimensional problems, such as deep neural networks which might have millions of parameters; in those situations, and when a differentiable error is available, a method built on gradient descent is often superior.

Neuroevolution for Ms Pac-Man

One simple way to implement neuroevolution in Ms Pac-Man is to first design an ANN that considers the game state as input and output actions for Ms Pac-Man. The weights of the ANN can be evolved using a typical evolutionary algorithm and following the steps of neuroevolution described earlier. The fitness of each ANN in the population is obtained by equipping Ms Pac-Man with each ANN in the population and letting her play the game for a while. The performance of the agent within that simulation time (e.g., the score) can determine the fitness value of the ANN. Figure 3.24 illustrates the steps of ANN encoding and fitness assignment in this hypothetical implementation of neuroevolution in Ms Pac-Man.

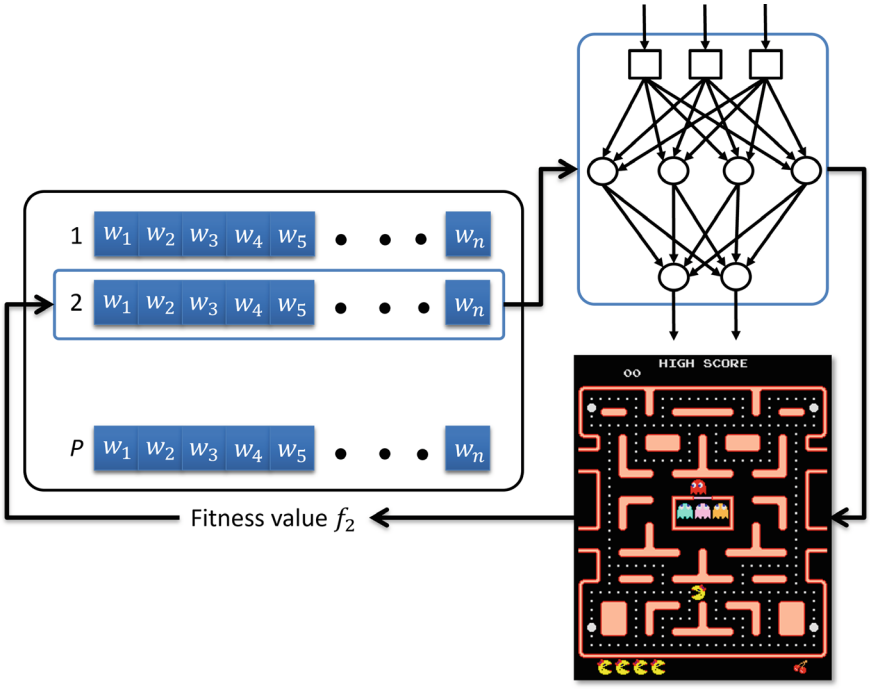


Fig. 3.24 Neuroevolution in Ms Pac-Man. The figure visualizes step 2 (ANN encoding) and step 3 (fitness assignment) of the algorithm for assigning a fitness value to chromosome 2 in the population (of size P). In this example, only the weights of the ANN are evolved. The n weights of the chromosome are first encoded in the ANN and then the ANN is tested in Ms Pac-Man for a number of simulation steps (or game levels). The result of the game simulation determines the fitness value (f_2) of the ANN

3.7.5 *Further Reading*

The RL book of Sutton and Barto [1109] is highly recommended for a thorough presentation of RL including Q-learning (Chap. 6). The book is freely available online.[6] The latest (2018) version of the book is also available and it includes material on policy gradient methods and PPO (Chap. 13).[7] For an in-depth analysis of model-based RL approaches you are referred to [83]. TD-Gammon and DQN are covered in detail in [1134] and [773], respectively. Both are also placed within the greater RL field in the second edition of [1109].

For a relatively recent and thorough survey on the application of neuroevolution in games the reader may refer to [946]. For a complete review of neuroevolution please refer to Floreano et al. [340]. CPPNs and NEAT are covered in detail in [1078] and [1081], respectively.

3.8 Summary

This chapter covered the AI methods we feel the reader of this book needs to be familiar with. We expect, however, that our readers have a basic background in AI or have completed a course in fundamentals of AI prior to reading this book. Hence, the algorithms were not covered in detail since the emphasis of this book is on the application of AI within the domain of games and not on AI per se. On that basis, we used the game of Ms Pac-Man as the overarching application testbed of all algorithms throughout this chapter.

The families of algorithms we discussed include traditional behavior authoring methods (such as finite state machines and behavior trees), tree search (such as best-first, Minimax and Monte Carlo tree search), local search and global search methods (such as evolutionary algorithms), unsupervised learning (such as clustering and frequent pattern mining), supervised learning (e.g., support vector machines, decision trees and neural networks), deep learning (e.g., CNNs, GANs, diffusion models, transformer architectures and LLMs) and reinforcement learning (e.g., Q-learning, PPO, evolving artificial neural networks, and artificial neural networks as approximators of expected rewards).

With this chapter we reached the end of the first, introductory, part of the book. The next part begins with a chapter on the most traditional and widely explored task of AI in games: playing!

[6] http://incompleteideas.net/sutton/book/ebook/the-book.html.

[7] http://incompleteideas.net/sutton/book/the-book-2nd.html.

Part II
Play

Chapter 4
Playing Games

Abstract This chapter lays the foundation for our study of game-playing AI, providing a taxonomy of roles and uses. Two key distinctions are whether the AI plays in the role of a player, or in the role of the non-player character, and whether AI is playing to win or to elicit an experience. Further dimensions of the taxonomy focus on characteristics of the game, and of the interface between the algorithm and the game. The taxonomy given here helps make sense of the very different use cases for AI game playing, and explains why different AI game-playing problems require very different methods.

When most people think of AI in games, they think of an AI playing the game or controlling the non-player characters you meet in the game. This might be because of the association between AI and the idea of autonomous action, or the association between game characters and robots. It could also be because of the colloquial use of "game AI" for whatever controls the NPCs, and for the media coverage of game-playing systems like AlphaGo. While playing games is far from being the only interesting application for AI in games, it is a very important one and the one with the longest history. Many methods for content generation (Part III) and player modeling (Part IV) are also dependent on methods for playing games, and therefore it makes sense to discuss playing games before content generation and player modeling.

This chapter is devoted to the ways AI can play games, including approaches for creating interesting non-player characters in games. While winning a game, appearing human-like, and providing entertainment are very different objectives, they face many of the same challenges. In fact, there are many different reasons why one might want to use AI methods to play a game. We start the chapter with discussing these various motivations (Sect. 4.1). Regardless of why you want to use AI to play a game, which methods you can effectively use to play the game is determined by the various characteristics of the game that, in turn, affect the choice, design and deployment of the AI method. So the next section in this chapter (Sect. 4.2) is devoted to characterizing games and AI algorithms according to several criteria and placing them as dimensions to an overall taxonomy of gameplaying AI. Once you have understood your game sufficiently, you can make an informed choice on which algorithm to

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_4

play it. The taxonomy also includes a detailed discussion on various important use cases of AI playing agents in games. Throughout the chapter we will mostly discuss the use of AI methods that play to **win**, but also make numerous references to the **experience**-creation aspect of gameplaying.

4.1 Why Use AI to Play Games?

The question of why you might want to deploy some kind or artificial intelligence to play a game can be reduced to two more specific questions:

Is the AI Playing to Win?

The question here is whether achieving as high a performance as possible in the game is the overarching goal of the AI method. High performance here means getting a high score, winning over the opponent, surviving for a long time or similar. It is not always possible to define what high performance and "playing to win" means—for example, *The Sims* (Electronic Arts, 2000) has no clear winning state and the winning condition in *Minecraft* (Mojang, 2011) is not strongly related to playing the game well—but in a very large number of games, from *Tetris* (Alexey Pajitnov and Vladimir Pokhilko, 1984) to Go to the *Forza* series (Xbox Game Studios, 2005–2023), it is straightforward to define what playing better means. However, not all players play to win, and few players play to win in every game all the time. Players play to pass time, relax, test new strategies, explore the game, role-play, keep their friends company and so on (see a more detailed discussion on this topic in Chap. 10). An AI algorithm might likewise be used in a number of roles beyond simply playing as well as possible. For example, the agent might play in a human-like manner, play in an entertaining manner, or play predictably so that the player can learn its patterns. It is important to note that optimizing an agent for playing a game to win might be at odds with some of the other ways of playing: many high-performing AI agents play in distinctly non-human, boring and/or unpredictable ways, as we will see in some case studies.

Is the AI Taking the Role of a Human Player?

Some games are single-player, and some games are multi-player where all players are human. This is particularly true for classic board games. But many, probably most, video games include various non-player characters. These are controlled by the computer software in some way—in fact, for many game developers "game AI" refers to the program code that controls the NPCs, regardless of how simple or sophisticated that code is. In the discussion of this chapter we refer to non-player roles as those that a human could not take, or would not want to take. Thus, all roles in an exclusively multi-player first-person shooter (FPS) such as *Counter-Strike* (Valve Corporation, 2000) are player roles, whereas a typical single-player role-playing game (RPG) such as *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011) has

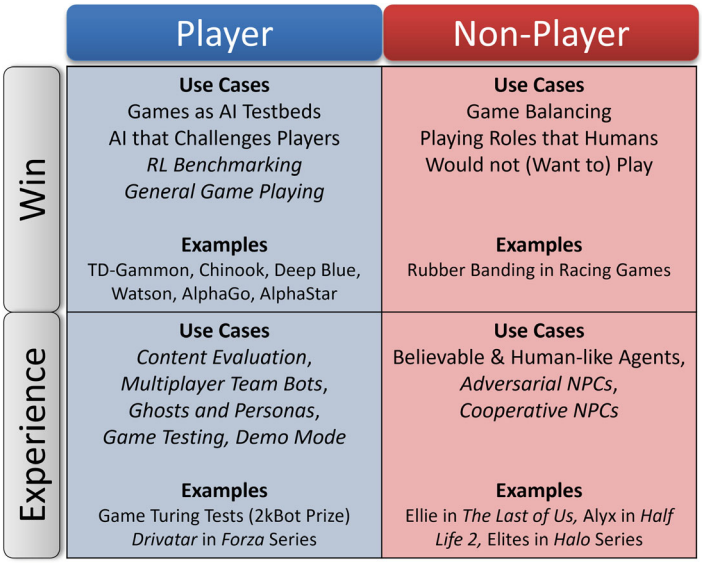


Fig. 4.1 Why use AI to play games? The two possible *goals* (win, experience) AI can aim for and the two roles (player, non-player) AI can take in a gameplaying setting. We provide indicative use cases and examples for each of the four options. Use cases that appear in italics are detailed in Sect. 4.2.1

only one player role, the rest are non-player characters. In general, non-player roles have more limited possibilities than player roles.

In summary, AI could be playing a game to **win** or for the **experience** of play either by taking the role of the **player** or the role of a **non-player** character. This yields *four core uses of AI for playing games* as illustrated in Fig. 4.1. With these distinctions in mind, we will now look at these four key motivations for building gameplaying AI in further detail through a general taxonomy.

4.2 Taxonomy: Playing Games

Given the vast variety of gameplaying tasks, AI methods, and games that are available out there we feel it helps the reader (and us) to have a structure that can highlight the differences and emphasize the similarities among them. In the remainder of this section, we introduce a taxonomy of gameplaying AI across a number of dimensions. While some dimensions are binary most represent individual roles or characteristics that usually lie somewhere on a continuum between the ends of that dimension. Beyond the taxonomy for the four possible **uses** we introduced earlier (Sect. 4.2.1)—when we asked *why we should use AI to play games*—the taxonomy

includes important **game characteristics** (Sect. 4.2.2) and **deployment** aspects that affect AI gameplaying (Sect. 4.2.3). Figure 4.2 depicts an illustration of the taxonomy we discuss in this section.

4.2.1 Uses

We have already identified the two possible *goals* for an AI gameplaying agent and the two *roles* it can take as the two core dimensions that define the four possible AI *uses* in this taxonomy (see the first two rows of Fig. 4.2). In this section we detail each of the four uses of AI for playing games. We also detail a number of common use

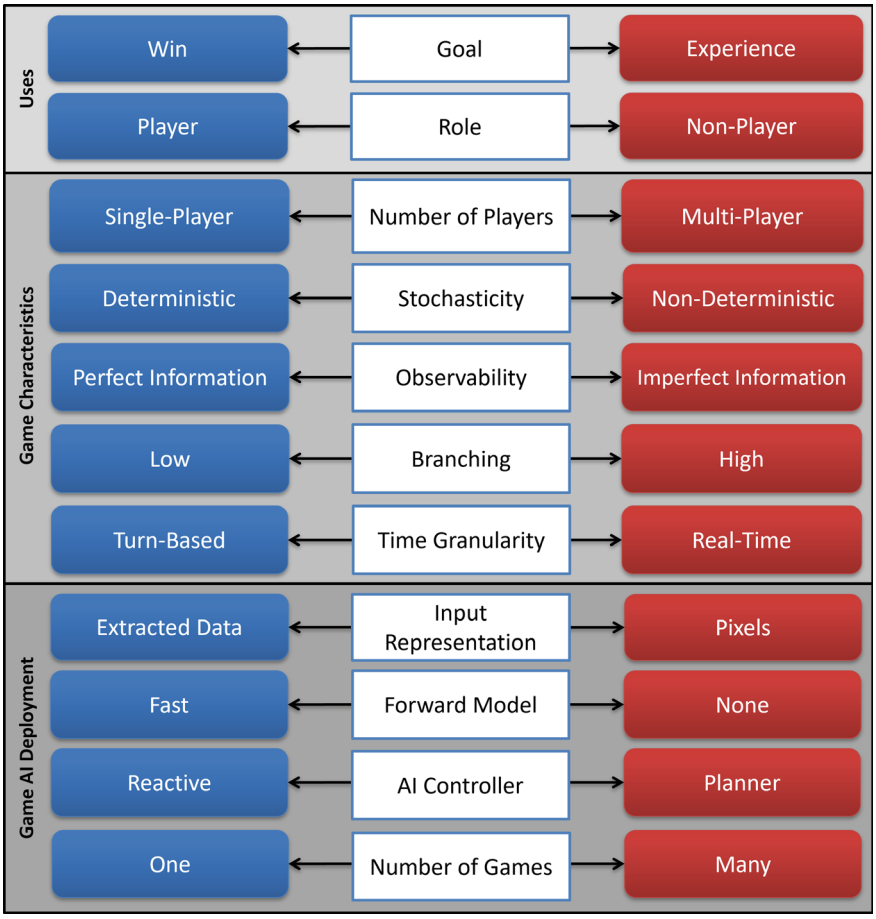


Fig. 4.2 AI playing games: an illustration of the taxonomy discussed in Sect. 4.2

cases of AI in games under each category. For each of those use cases, we will discuss the reason for using AI to play the game, the typical game characteristics, algorithm characteristics, and give examples of methods used. This is not an exhaustive list, but merely a way of shining a light on some of the more common or salient use cases.

4.2.1.1 Playing to Win in the Player Role

Perhaps the most common use of AI together with games in academic settings is to play to win, while taking the role of a human player. This is especially common when using games as an AI testbed. Games have been used to test the capabilities and performance of AI algorithms for a very long time, as we discussed in Sect. 1.2. Many of the milestones in AI and games research have taken the form of some sort of AI program beating the best human player in the world at some games. See, for example, IBM's Deep Blue winning over Garry Kasparov in Chess, Google DeepMind's *AlphaGo* winning over Lee Sedol [1040] and Ke Jie in Go, IBM's Watson winning Jeopardy! [335], and more recently DeepMind's *AlphaStar* defeating TLO and MaNa, professional players of *StarCraft II* (Blizzard Entertainment, 2010) [1219]. All of these were highly publicized events widely seen as confirmations of the increasing capabilities of AI methods.

Games are excellent testbeds for artificial intelligence for a number of reasons, as elaborated on in Sect. 1.3. An important reason is that games are made to test human intelligence. Well-designed games exercise many of our cognitive abilities. Much of the fun we have in playing games comes from learning the games through playing them [581], meaning that well-designed games are also great teachers. This, in turn, means that they offer the kind of gradual skill progression that allows for testing of AI at different capability levels.

There are some reasons besides AI benchmarking for why you might want to use an AI instead of a human to play games to win. For example, there are some games where you need strong AI to provide a challenge to players. This includes many strategic games of perfect information, such as classic board games, including Chess, Checkers and Go. However, for games with hidden information, it is often easier to provide challenge by simply "cheating". This can be done by giving the AI player access to the hidden state of the game or even by modifying the hidden state so as to make it harder to play for the human. For example, in the epic strategy game *Civilization* (MicroProse, 1991), all civilizations can be played by human players. However, playing any Civilization game well under the same conditions as a human is very challenging, and there is, to our knowledge, no AI capable of playing these games as well as a good human player. Therefore, when playing against several computer-controlled civilizations, the game typically cheats by providing these with preferential conditions in various ways.

Another use case for AI that plays to win in a player role is to test games. When designing a new game, or a new game level, you can use a gameplaying agent to test whether the game or level is playable, so called **simulation-based testing**. However,

in many cases you want the agent to also play the game in a human-like manner to make the testing more relevant; see Sects. 4.2.1.3 and 4.2.1.4 dedicated to playing for experience.

Historically, the use of AI to play to win in a player role has been so dominant in academic work that some researchers have not even considered other roles for AI in playing games. In game development, on the other hand, this particular motivation for gameplaying AI is much more rare; most gameplaying AI in existing games is focused on non-player roles and/or playing for experience. This mismatch has historically contributed to the lack of understanding between academia and industry on game AI. In recent years however, there has been a growing understanding of the multitude of motivations for gameplaying AI.

In summary, we can think of several popular examples of using AI to win a game while sitting at the player's seat. In the remainder of this section we detail **RL benchmarking** and **general game playing** as the two most representative examples. While game testing is a popular use case when AI aims to win a game, we feel it is somewhat more relevant as a use case when AI plays for other purposes beyond winning; hence we detail this use case in Sect. 4.2.1.3.

Reinforcement Learning Benchmarking

Within AI research, the most common use case for gameplaying agents is for reinforcement learning benchmarking. Here, reinforcement learning algorithms are tested by learning to play games (in the player role). Success on the game is used as a benchmark for the algorithms. In the deep learning era, the *Arcade Learning Environment* [75] and its Atari games in particular has provided a very common set of benchmarks for RL methods, following the success of DeepMind's 2015 paper establishing deep reinforcement learning as a field, where a selection of Atari games from ALE was used as the benchmark set [773]. Since then, an ever growing number of reinforcement learning papers have used these games as benchmarks. Before the deep learning age, some board games, such as Backgammon, were regularly used as RL benchmarks [1134].

Given the need for most RL methods to play the game thousands or even millions of times, it is highly desirable that the game can be run faster than realtime. In practice, most RL benchmarks achieve this by relying on very old games, made for very old hardware. As modern computers are so much faster than a game console from the late 1970s, Atari games can be run nowadays thousands of times faster than realtime.

In most benchmark situations, the interface is such that the algorithm only receives pixel inputs and outputs simulated controller outputs. This is partly for simplicity, and partly for perceived fairness; if the algorithms are only fed pixels, you cannot tailor your solution to a particular form of game-internal representation. Instead, the idea is that the neural network sees (more or less) what a human observer would see, and has more or less the same possibilities to affect the game. But achieving a situation where humans and computer programs compete fairly is a surprisingly difficult endeavor; there are many issues here, including potential input lag [173].

General Game Playing

Arguably the most ambitious use case for AI gameplaying is *general game playing*. Here, the idea is to develop AI agents that can play not just one game well, but any game within some category or genre. Attempts at general game playing have had some success within more limited domains, such as the *General Game Playing Competition* [374], the *General Video Game AI Competition* [869, 870] and even the *Arcade Learning Environment* [75], that focus on specific genres, have very well-defined interfaces, and where the agents are given time to plan each move and access to a fast game engine. The really general case, where games could be of any genre, and the agents are only provided pixel inputs and no formalized description of the game, is of course considerably harder. One could make the argument that a fully general game player would be as much of a test of general intelligence as anything.

4.2.1.2 Playing to Win in a Non-player Role

Non-player characters are very often designed to *not* offer maximum challenge or be as effective as possible, but instead to be entertaining or human-like; see Sect. 4.2.1.4 for non-player characters playing for experience. However, there are instances when you want a non-player character to play as well as possible. As mentioned earlier, strategy games such as *Civilization* (MicroProse, 1991) have an (unanswered) need for high-performing non-cheating opponents, though here we are talking about playing roles that other human players could in principle have taken. Other strategy games, such as *XCOM: Enemy Unknown* (2K Games, 2012), have adversary roles that humans would not play (the aliens), creating a need for NPC AI playing to win.

Other times, creating an NPC playing to win is a necessary precursor to creating an NPC playing for experience. For example, in a racing game, you might want to implement *rubber band AI* where the NPC cars adapt their speed to the human player, so that they are never too far behind or ahead. Doing this is easy, but only if you already have an AI controller—such as *Sophy AI* [1258] in *Gran Turismo Sport* (Sony Interactive Entertainment, 2017)—that can play the game well, either through actually playing the game well or through cheating in a way that cannot easily be detected. The performance of the controller can then be reduced when necessary so as to match the player's performance.

4.2.1.3 Playing for Experience in the Player Role

Why would you want an agent that takes the role of a human player, but that does not focus on winning? For example, when you want a **human-like agent**. Perhaps the most important reason for such agents is alluded to above: simulation-based testing. This is important both when designing games and game content manually, and when generating content procedurally. In the latter case, the quality of the game content is often evaluated automatically with the help of an agent playing the game, as discussed

in Chap. 7. The AI agent in this role plays in a human-like manner, meaning that it has performance comparable to a human, has similar reaction speed, makes the same sort of mistakes that a human would do, is curious about and explores the same areas as a human would, etc. If the AI agent plays significantly differently from how a human would play, it might give the wrong information about e.g., whether a game is winnable (it might be winnable but only if you have superhuman reflexes) or whether a game mechanic is used (maybe a human would use it, but not an AI that tries to play optimally).

Another situation where human-like play is necessary is when you want to demonstrate how to play a level to a human player. A common feature of games is some kind of **demo mode**, which shows the game in action. Some games even have a demonstration feature built into the core gameplay mode. For example, *New Super Mario Bros* (Nintendo, 2006) for the Nintendo Wii will show you how to play a particular part of a level if you fail it repeatedly. The game simply takes over the controls and plays for you for about 10–20 s, and lets you continue afterwards. If all the level content is known beforehand, and there are no other players, such demonstrations can be hardcoded. If some parts of the game are user-designed, or procedurally generated, the game needs to generate these demonstrations itself.

Playing in a "human-like" fashion may seem a rather fuzzy and subjective aim, and it is. There are many ways in which a typical AI agent plays differently from a typical human player. How humans and AIs differ depends on the algorithm used to play the game, the nature of the game itself, and a multitude of other factors. To investigate these differences further, and spur the development of agents that can play in a human-like manner, two different Turing test-like competitions have been held. The 2K BotPrize was held from 2008 to 2013, and challenged competitors to develop agents that could play the FPS *Unreal Tournament 2004* (Epic Games, 2004) in such a way that human participants thought that the bots were human [449, 450, 1069]. Similarly, the Turing test track of the Mario AI Competition let people submit playing agents of *Super Mario Bros* (Nintendo, 1985), who were judged by human onlookers as to whether they were human or not [1026, 1171]. While it takes us too far to go through all of the results of these competitions here, there are some very obvious signs of non-humanness that recur across games for many types of AI agents. These include having extremely fast reactions, switching between actions faster than a human could, not attempting actions which fail (because of having a too good model of the outcome of actions), not doing unnecessary actions (such as jumping when one could just be running) and not hesitating or stopping to think.

Multiplayer Team Bots

Many games are at their core multiplayer games. Examples include racing games, first-person shooters, and real-time strategy games. These games can be fully cooperative or fully competitive, but there is a large fraction of them that contain a mix of the two. In particular, there are the team-based games, where you collaborate with other members of your team while competing with one or more other teams such as most MOBA games, and tactical FPS games such as *Counter-Strike* (Valve Corporation, 2000), *Team Fortress* (Valve, 2007) or *Valorant* (Riot Games, 2020).

Generally, AI bots are in very limited use in such games. The issue here is that the gameplaying AI would be used in a player role rather than a non-player role. Much of the enjoyment for multiplayer team games is derived from the fact that you play with and against other humans. This means that players expect human-level intelligence, or at least human-like behaviors from the characters they are playing against. But it also means that players have an expectation that the other characters are played by actual humans, lending much higher weight to the decisions taken during gameplay. Game developers may therefore try to create bots that are somewhat human-like but clearly inform players that these are not actual humans. Alternatively, a developer could attempt the riskier gambit of creating more human-like bots and simply lie to their players, saying that these bots are humans (we are not aware of any reputably developer doing this, but if they did we would either not be aware of it or the developer would not be reputable).

Traditional game AI methods, such as behavior trees combined with pathfinding, are generally not deployed as multiplayer team bots, given that it is very hard to create sufficiently human-like behavior using these methods. The same properties of these methods that make them useful for authoring adversarial NPC behavior make them generally unsuitable for creating human-like behavior. This means that good, generally applicable methods for multiplayer team bots is very much an open research issue. Recent approaches on creating bots that imitate human play in games such as *Counter-Strike* (Valve Corporation, 2000) have instead employed CNNs, transformer architectures, and even diffusion models [856] for that purpose. We cover more examples of human-like AI team bots in Chap. 6.

Ghosts and Personas

There are many cases where you would want an agent to play a game (in the player role) in the style of a particular player, or a type of player. One such use case is **ghosts**. This means simply replaying a player's actions precisely on the levels or environments that the player was playing. This can be done in parallel with another player playing those levels. Arguably, the most popular example of a ghost agent is the *Drivatar* system that was introduced in *Forza Motorsport* (MS Game Studios, 2005) and is shipped with all games of the *Forza* series (Xbox Game Studios, 2005–2023) since then.

Of course, not all players play in the same way. In fact, as discussed further in Chap. 10, if one analyzes a set of play traces of any game one can often find a number of player "archetypes" or **personas**, clusters of players who play the game in markedly different ways in terms of e.g., aggression, speed, curiosity and skill. Within work on AI that plays games in human-like styles, there has been work both on learning and mirroring the playstyle of individual players [547, 701, 702, 837, 856, 1004] and on learning to play games in the style of different personas [62, 459, 461].

Procedural personas are generative models of player behavior, meaning that they can replicate in-game behavior and be used for playing games in the same role as players. Additionally, procedural personas are meant to represent archetypical players rather than individual players [60, 459–461]. A procedural persona can be

defined as the parameters of a utility vector that describe the preferences of a player. For example, a player might allocate different weight to finishing a game fast, exploring dialog options, getting a high score, etc.; these preferences can be numerically encoded in a utility vector where each parameter corresponds to the persona's interest in a particular activity or outcome. Once these utilities are defined, reinforcement learning methods can be used to find a policy that reflects these utilities, or a tree search algorithm can be used with these utilities as evaluation functions. Approaches similar to procedural personas have also been used for modeling the learning process of the player in educational games via reinforcement learning [809].

Game Testing

A sizable part of a typical game development budget goes to game testing. Like any software, games need to be tested before being released. Unlike many other types of software, the type of testing that needs to be carried out on games is unusually multifaceted and sometimes hard to define. Of course, the game code must be tested for crashes, freezing, and so on. Game design must also be tested, such as the end of the level not being reachable from the beginning, certain objects not always being available at certain locations, textures loading too slowly, frame rates dropping, and also for the *game feel* being off [879, 1117]. Often, it is hard to specify the concrete steps that lead to a particular bug or other errors and this is what makes game testing a particularly challenging task.

Manual game testing consists in humans playing particular parts of games again and again, systematically testing alternative actions and paths to try to find bugs and other unwanted game states. This is generally considered the most boring task in game development. It is sometimes a way into the games industry for junior employees, who are tempted by the offer to play games and get paid for it, until they realize how repetitive it is. This repetitiveness is a key reason why there seems to be many opportunities to enhance, or perhaps even replace, human testers with AI-driven bots. And, in striking difference to the reactions among many game artists to the looming opportunity (or even threat) of generative AI, few people would protest if we could automate at least some tasks in game testing.

As testing is multifaceted, different types of testing pose different AI challenges. One of the more elementary forms of testing is **regression testing**, where you simply want to know that the same sequence of actions will result in the expected results after the game is changed in some way. In many cases, changes to one part of the game may introduce errors elsewhere. For example, if you open this chest, retrieve this key, and then talk to this person, will you still be able to open that door? One way you can implement this is by simply replaying the same actions, which is not even an AI task, just a matter of recording a sequence of actions. A more complex problem is **approximate replay**, where you see if the same set of actions would work even if you had moved the chest slightly. This might be approached by using pathfinding algorithms or reinforcement learning algorithms to move between waypoints.

Another testing task is to try to find out if some game state, can cause a particular dysfunction, such as a crash or a drop in frame rate. The problem here is to explore as many game states as possible to see if that state is triggered. This can be done with

exploration algorithms. Relatively simple solutions here include *rapidly-exploring random trees* [606] (RRTs) and more sophisticated solutions include the Go-Explore algorithm [304].

Another abstraction level up, you may want to test whether a particular level or level segment can be completed. A number of different methods, including tree search and reinforcement learning, might be useful here, depending on the game and the access to the game engine. It might also be desirable to test whether a level or level segment can be completed by a player with a particular playstyle or particular limitations (for example in terms of reaction speed or the ability to perform maneuvers such as double jump). Here, imitation learning can be useful to learn the playstyle of the human player; see the section covering ghosts and personas.

At the top of the hierarchy of game testing use cases is actual **general game playing**. You may want to know if a complete game is playable from beginning to end, and if playing through it will cause any performance issues, crashes, service term violations, etc. There are many reasons for wanting to do this: you may be a game publisher, a distribution platform (such as an app store), a hardware manufacturer or operating system developer wanting to test whether a game works with your hardware or software, and so on. While very useful, this form of game testing is also the hardest, as it essentially implies general game playing capacity.

In almost all conceivable cases, game testing agents would take the role of the player. However, one might imagine game testing through exploring the space of states the non-player characters could be in. Access to game state will largely depend on who develops and uses the game testing solution. A game studio that uses a game testing tool as part of their normal development pipeline will be able to give it full access to the game engine, so all aspects of the game state can be used. A publisher or distributor, however, will likely not have access to the source code of the game and will be forced to interface through processing pixels from the screen and inputting simulated keyboard or controller inputs, as in [925].

Content Evaluation

Many of the procedural content generation algorithms discussed in this book rely on some form of content evaluation. Search-based algorithms [1176] need to evaluate the quality of every generated artifact (such as a level). A similar situation holds in PCG through reinforcement learning [559], where content evaluation is part of the reward function. While there are several different types of content evaluation, a very common type is simulation-based evaluation. This means using an agent (or several agents) to play through the content and evaluate it. For example, you may want a level to be playable by a strong agent and not by a weak agent.

Methods for simulation-based content evaluation will likely have to be run thousands or even millions of times in the course of content production. Therefore, speed is of the essence. This means that successful methods for simulation-based content evaluation typically rely on means of speeding up or simulating the game engine. Section 9.7 covers this use of AI for playing games in higher detail.

4.2.1.4 Playing for Experience in a Non-player Role

The most common goal for gameplaying AI in the game industry is to make non-player characters act, almost always in ways which are not primarily meant to beat the player or otherwise "win" the game (for many NPCs it may not even be defined what winning the game means). NPCs may exist in games for many, sometimes overlapping, purposes: to act as adversaries, to provide assistance and guidance, to form part of a puzzle, to tell a story, to provide a backdrop to the action of the game, to be emotively expressive and so on [1184]. The sophistication and behavioral complexity of NPCs likewise vary widely, from the regular left-right movements of the aliens in *Space Invaders* (Midway, 1978) and Koopas in the *Super Mario Bros* series (Nintendo, 1985–2016) to the nuanced and varied behavior of non-player characters in *Bioshock Infinite* (2K Games, 2013) and the alien in *Alien: Isolation* (Sega, 2014).

Depending on the role of the NPC, very different tasks can be asked of the AI algorithms that control it. In many cases what the game designers look for is *the illusion of intelligence*: for the player to believe that the NPC in some sense is intelligent even though the code controlling it is very simple. Human-likeness in the sense discussed in the previous section might or might not be the objective here, depending on what kind of NPC it is: a robot or a dragon should perhaps not behave in a too human-like manner.

In other cases, the most important feature of an NPC is its *predictability*. In a typical stealth game, a large part of the challenge is for the player to memorize and predict the regularities of guards and other characters that should be avoided. In such cases, it makes sense that the patrols are entirely regular, so that their schedule can be gleaned by the player. Similarly, the *boss monsters* in many games are designed to repeat certain movements in a sequence, and are only vulnerable to the player's attack when in certain phases of the animation cycle. In such cases, too "intelligent" and adaptive behavior would be incompatible with the game design.

It should be noted that even in cases where you would expect to need supple, complex behavior from NPCs, an agent that plays to win might be very problematic. Many high-performing strategies are seen as very *boring* by the player, and prime examples of "unsportsmanlike" behavior. For example, in an experiment with building high-performing AI for a turn-based strategy game, it was found that one of the solutions (based on neuroevolution) was extremely boring to play against, as it simply took a defensive position and attacked any incoming units with long-distance attacks [812]. Similarly, *camping* (staying stationary in a protected position and waiting for enemies to expose themselves to fire) is a behavior that is generally frowned upon and often banned in FPS games, but it is often highly effective and easy to learn for an AI. Another interesting example is the work by Denzinger et al. [275] in which an evolutionary algorithm found that the best way to score a goal in *FIFA 99* (Electronic Arts, 1999) was by forcing a penalty kick. The evolutionary process found a local optimum in the fitness landscape corresponding to a *sweet spot* or *exploit* of the game's mechanics which yielded highly efficient, yet predictable

and boring gameplay. Exploiting the game's bugs for winning is a *creative* strategy that is not only followed by AIs but also by human players [643].

Adversarial NPCs in Single-player Games

Adversarial NPCs in single-player games, constitute the typical use case for game AI in most game designers' and game players' minds. Prominent examples of this use case are the ghosts in *Pac-Man* (Namco, 1980), aliens in *Space Invaders* (Atari, 1978) or in the *Halo* series (Microsoft Studios, 2011–2017), and the Medusa in *Assassin's Creed Odyssey* (Ubisoft, 2018).

These types of NPCs are quintessentially playing for your experience. They are not there to defeat you, just to give you a good challenge. They are also not playing in a player role, as the player can typically not take any of the roles filled by NPCs, and they tend to have rather different capabilities. There are exceptions to this: the other cars in a racing game from the *Forza* series (Xbox Game Studios, 2005–2023) or the aliens in *StarCraft II* (Blizzard Entertainment, 2010) can be seen as either player or non-player roles. But when they are controlled by AI systems, these systems are playing for experience.

The way the NPCs bring about the sought-after experience varies, but often through a combination of predictability and lifelikeness. Predictability of NPCs is an important design feature of many games, as it allows the player to deepen their skill of the game by learning to predict the movements of the enemies. This goes for both simpler NPCs with easily predictably behaviors, such as the Goombas in *Super Mario Bros* (Nintendo, 1985) that simply move forward until they perceive an obstacle and then turn back, and more complicated ones such as the various boss fights in *Elden Ring* (Bandai Namco Entertainment, 2022), where success is dependent on learning to predict the next move the boss will make out of a long sequence of moves that branch at particular points depending on, among other things, player actions.

In most cases, such NPC behaviors are implemented through a combination of either finite state machines or behavior trees with pathfinding using some version of A* on either a mesh or a grid. Behavior trees in particular are well-suited for authoring predictable yet flexible behaviors. Importantly, these structures are easy to interpret for the behavior designer, meaning that they can be designed and debugged iteratively. These classic methods typically require a deep integration with the game engine and are sometimes interleaved with the animation routines. On the other hand, these methods do not require any training time or forward model, as no simulation is performed.

A recurring theme throughout the history of AI and games is that researchers or others from outside of the game industry suggest that machine learning or optimization methods can be used to develop adversarial NPCs. While this is technically true, it is generally a use case for which these methods are not well suited. This is partly because it is hard to express the requirements of these NPCs as optimization objectives or rewards and partly because many machine-learned models are so opaque. But more importantly, the standard tools, such as behavior trees and pathfinding, are very well suited to the design needs of adversarial NPCs in most video games.

Cooperative NPCs in Single-player Games

Cooperative NPCs is an interesting use case that has much in common with adversarial NPCs, but also crucial differences. Think team-mates, sidekicks, troops under your command, support personnel, etc. Much like adversarial NPCs are not actually optimized to defeat you, just to give you a good challenge, cooperative NPCs are not optimized to play the game for you, just to give you the experience of being supported. Famous examples include Ellie in *The Last of Us* (Sony Computer Entertainment, 2013), Alyx in *Half Life 2* (Valve, 2004), and Cortana in the *Halo* series (Microsoft Studios, 2011–2017).

Generally, cooperative NPCs are implemented in similar ways to adversarial NPCs, as described earlier. However, these methods are not necessarily always up to the task; gripes about the behavior of such NPCs abound from players. See, for example, the squad based RPG-shooter *Mass Effect* series (Electronic Arts, 2007–2021), which received universal praise for many aspects of its design, but was widely panned for the team mates' behavior in combat, where they often were simply in the way. (The interested reader may browse the several *Mass Effect* player discussions and complaints over Reddit for example). Cooperative NPCs are generally harder to get right than adversarial NPCs. Perhaps the chief reason for this is that in order to effectively cooperate with a human player, these NPCs often need to be able to understand, or at least predict, the human player's actions. This is a hard task. Another reason is that these NPCs often persist for much longer than adversarial NPCs.

Summary of AI Gameplaying Uses

We argued earlier that playing to win in the player role has been overemphasized, to the point of neglecting other perspectives, in much of academic research. In the same way, work on AI in the game industry has generally overemphasized playing for experience in a non-player role, to the point of neglecting other perspectives. This has led to an emphasis in the industry on behavior authoring methods such as finite state machines and behavior trees, as AI methods based on search, optimization and learning have been seen as not conducive to playing for experience; a common gripe has been a perceived lack of predictability and authorial control with such methods. However, given a better understanding of what roles AI can be used to play in games, this neglect of methods and perspectives is gradually coming to an end in both academia and industry.

4.2.2 Game Characteristics

When choosing an AI method for playing a particular game (in any of the roles discussed in Sects. 4.1 and 4.2.1) it is crucial to know the *characteristics* of the game you are playing and the characteristics of the algorithms you are about to design. These collectively determine what type of algorithms can be effective. In this section

we first discuss the challenges we face due to the characteristics of the game per se and then we discuss aspects of AI algorithmic deployment (Sect. 4.2.3) that need to be considered independently of the game we examine.

Our taxonomy lists five core characteristics of games (see rows 2–6 of Fig. 4.2) and the impact they have on the potential use of AI methods. All characteristics covered are tied to the design of the game. Much of our discussion is inspired by the book *Characteristics of Games* by Elias et al. [320], which discusses many of these factors from a board game design perspective. For illustrative purposes, Fig. 4.3 places a number of core game examples onto the three-dimensional space of **observability**, **stochasticity** and **time granularity**.

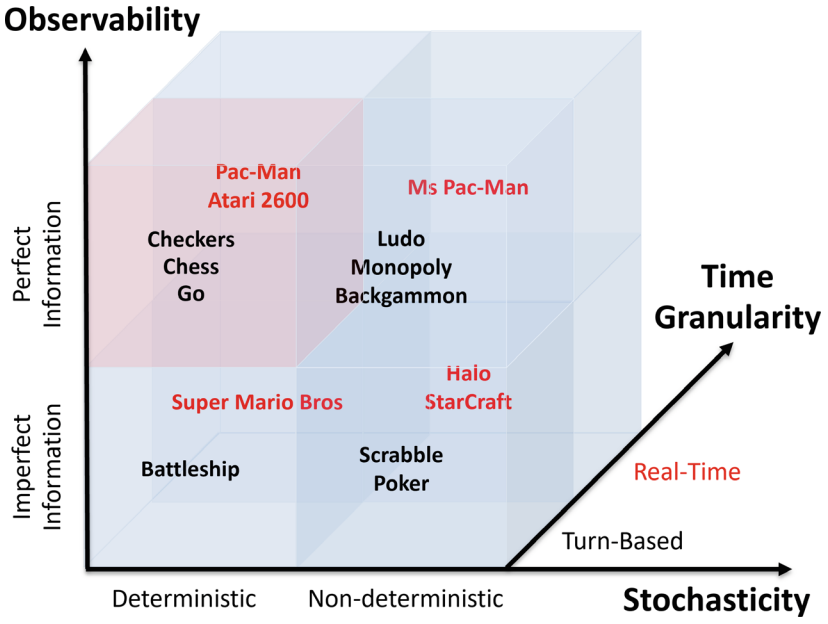


Fig. 4.3 Characteristics of games: game examples across the dimensions of stochasticity, observability and time granularity. Note that the game examples presented are sorted by complexity (action space and branching factor) within each cube. Game titles in red are real-time. Minimax can *theoretically* solve any deterministic, turn-based game of perfect information such as Checkers, Chess and Go (red cube in the figure)—in practice, it is still impossible to solve games with substantially large branching factors and action spaces such as Chess and Go via Minimax. Any AI method that eventually approximates the Minimax tree (e.g., MCTS) can be used to tackle imperfect information, non-determinism and real-time decision making (see blue cubes in figure). Strictly speaking, *Super Mario Bros* (Nintendo, 1985) involves a small degree of non-determinism only when a player helps creating a particular scene; we can, thus, safely classify the game as deterministic [273]

Number of Players: Single-player Versus Multi-player

A good place to start is the number of players a game has. Elias et al. [320] distinguish between:

- **single-player** games, such as puzzles and time-trial racing;
- **one-and-a-half-player** games, such as the campaign mode of an FPS with non-trivial NPCs;
- **two-player** games, such as Chess, Checkers and *Spacewar!* (Russell, 1962); and
- **multi-player** games, such as *League of Legends* (Riot Games, 2009), the *Mario Kart* series (Nintendo, 1992–2014) and the online modes of most FPS games.

The distinction between single-player and one-and-a-half-player games is not a sharp one—there is no clear boundary for how advanced NPCs should be to count as a "half player". In the case of multi-player games, many can be played with only two players, at which point they are effectively two-player games. It is not always the case that other players (or NPCs) are adversarial and try to stop the player—there are many collaborative games, or games where relations between players are complex and have elements of both competition and cooperation. Still, keeping the number of players in mind is very useful when thinking about algorithms for playing games.

When using tree search algorithms to play games, some algorithms are a good match to the numbers of players involved in the game. Standard single-agent tree search algorithms such as breadth-first, depth-first and A* fit the **single-player** case particularly well. This includes games which have NPCs, but where those NPCs are so simple and predictable as to be treated as part of the environment. In such games, what happens in the game is determined entirely by the actions the player takes and any potential random effects; there are no other "intentional" players. This fits very well with single-agent tree search algorithms, which are based on the **Markov property**, that the probabilities of visiting the next state are entirely known and determined by the previous state and the action taken at that point.

A particular case is the **two-player** zero-sum adversarial games, i.e., there are exactly two players; one player will win, the other will lose (or perhaps there will be a draw). We do not know what the other player will do, but we can safely assume that they will do everything they can to win, and thereby deny you the victory. The Minimax algorithm (with or without α-β pruning) is perfectly suited to this case, and will lead to optimal play given sufficient computation time.

But how do we cope with the challenge when we have **many players**, or perhaps a single player surrounded by very complicated non-player agents? While it is theoretically possible to expand Minimax to multiple players, this only works if there can be no collusion (or alliances of any kind) between players and the zero-sum nature of the game still remains. Further, the computational complexity of Minimax quickly gets unmanageable with more than two players, as for every move you take you do not just need to consider the countermove of one player, but of all players. So this approach is rarely workable. It is more common in the multi-player case to treat the game as a single-player game, but use some kind of model of what the other players do. This could be an assumption that the other players will oppose the player, a learned

model based on observed behavior or even a random model. With an appropriate model of what the other players will do, many standard single-player gameplaying methods can be used in multi-player settings. It is also common nowadays to employ a multi-agent (or population-based) variant of popular methods for playing games such as the multi-agent deep reinforcement learning approach adopted by AlphaStar [1219] to play *StarCraft II* (Blizzard Entertainment, 2015).

Stochasticity: Deterministic Versus Non-deterministic

A common way in which many games violate the Markov property is by being **stochastic** (or **non-deterministic**)[1] whereby the transition probabilities between states are unknown. In many games, some of what happens is random. As standard digital computer architectures do not allow for "true" randomness, effective randomness is provided by pseudo-random number generators. The word "stochastic" is used to denote processes which cannot be practically predicted, whether they result from true randomness or complex calculations. Games can have varying amounts of stochasticity, from completely **deterministic** games like Chess to games dominated by stochastic outcomes like Roulette, Ludo, Yahtzee or even Monopoly. It is common for games to have mostly or fully deterministic game mechanics combined with some stochastic element through card-drawing, dice-throwing or some similar mechanism to reduce the possibility of planning. However, stochasticity can occur in essentially any part of a game.

In a game with stochasticity, the outcome of the game is not entirely determined by the actions the players take. In other words, if you play several playthroughs of the same game, taking the same actions at the same points in time, you are not guaranteed the same outcome. This has consequences for AI algorithms. For **tree search** algorithms, it means that we cannot be sure about the state which a sequence of actions will lead to, and therefore about the results of the algorithm. This leads to problems with using many tree search algorithms in their canonical forms, and requires that we add some modifications to address the non-deterministic uncertainty in the forward model. For example, modifications of Monte Carlo tree search such as *determinization* allow the algorithm to explore different possible outcomes of each action separately [137]. While these algorithm variations can be effective, they generally increase the computational complexity of the base algorithm.

For **reinforcement learning** approaches it means that we have reduced certainty in exactly how good a given strategy/policy is. A good policy may achieve bad outcomes, or a bad policy good outcomes, because of random events in the game. Such outcome uncertainty can be mitigated by evaluating every policy multiple times, though this has significant computational cost. On the other hand, stochasticity can sometimes actually be an advantage when learning policies: a policy which is learned for a stochastic game may be more robust than one learned for a deterministic game, as in the latter case it is possible to learn a very brittle policy that only works for

[1] Strictly speaking there is a distinction between a *stochastic* and a *non-deterministic* process in that the former has a defined random distribution whereas the latter does not. For the purposes of this book, however, we use the two terms interchangeably.

a specific configuration of the game. For example, learning a policy which attacks enemies in specific places at specific times, rather than being able to handle enemies that might arrive from any direction at any time.

While it is very common for digital games to include some form of stochasticity, an interesting case is very early games hardware such as the 1977-vintage Atari 2600, which does not have the facilities for implementing pseudo-random number generators (mainly because it lacks a system clock). If a player takes exactly the same actions at exactly the same times (including the key press that starts the game), exactly the same outcome will be achieved. The Arcade Learning Environment is a widely used game-based AI-benchmark built around an emulator of the Atari 2600 [75]. When training AI agents to play games with no stochasticity, it is entirely possible to learn brittle policies that effectively bypass the complexities of the full game (whether this actually happens, or whether most agents learn more general strategies, is an open question). As we already saw across the various examples of Chap. 3, *Ms Pac-Man* (Namco, 1982) is arguably the most popular non-deterministic arcade game of that era.

Observability: Perfect Versus Imperfect Information

Observability is strongly related to stochasticity. It refers to how much information about the game state is available to the player(s). At one extreme we have classic board games such as Chess, Go and Checkers, where the full board state is always available to the players, and puzzles such as Sudoku and *SpellTower* (Noodlecake Studios, 2011). These games have **perfect information**. At the other extreme we can think of classic text adventures such as *Zork* (Personal Software, 1980) or *Colossal Cave Adventure*, where initially very little of the world and its state is revealed to the player and much of the game is about exploring what the world is like. Those games have **hidden information** and therefore only *partial observability*. Many, if not most, computer games have significant hidden information: think of a typical platform game such as *Super Mario Bros* (Nintendo, 1985), or FPS such as the *Halo* series (Microsoft Studios, 2001–2015), where at any point you can only perceive a small part of the game world. Within computer strategy games such as *StarCraft II* (Blizzard Entertainment, 2015) or *Civilization* (MicroProse, 1991) the common term for hidden information is *fog of war*. Even many classic non-digital games have hidden information; some of them mostly revolve around figuring out the hidden information, including card games such as Poker and board games such as *Battleship* (Starex, 1931).

When developing an AI agent for a game with hidden information, the simplest approach you can follow is to merely ignore the hidden information. At each point in time, just feed the available information to the agent and use that to decide the next action. Doing so actually works quite well in some games, notably action-focused games with linear levels; for example, simple *Super Mario Bros* (Nintendo, 1985) levels can be played well based on only instantaneously available information [1160]. However, if you play a strategy game such as *StarCraft* (Blizzard Entertainment, 1998) based on only the available information, you are not even going to see the enemy until it is too late—good play involves active information gathering. Even

in *Super Mario Bros* (Nintendo, 1985), complicated levels that feature backtracking require remembering off-screen parts of the level [539]. In a card game such as Poker the available information (your own hand) is actually of comparatively less relevance; the core of the game is modeling the hidden information (i.e., your adversaries' hands and minds).

Therefore, effective AI for games with partial observability often requires some kind of *modeling* of the hidden information. For some games, notably variants of Poker such as Heads-up limit hold'em, considerable research has been done on game-specific methods for modeling hidden information that includes opponents' play [115]. There are also more generic methods of adding some form of hidden state modeling to existing algorithms, such as Information Set Monte Carlo tree search [246]. Just like when dealing with stochasticity, these methods typically add considerable computational complexity compared to the base algorithm.

Action Space and Branching Factor: Low Versus High

When you play the minimalist mobile game *Flappy Bird* (dotGEARS, 2013), you have a single choice at any point in time: to flap or not to flap. *Flappy Bird* and similar one-button games, such as *Canabalt* (Beatshapers, 2009), probably have the lowest possible **branching factor**. The branching factor is the number of different actions you can take at any decision point. The branching factor of *Flappy Bird* is 2: flap or no flap.

For comparison, *Pac-Man* (Namco, 1980) has a branching factor of 4: up, down, left and right. *Super Mario Bros* (Nintendo,1985) has a branching factor of up to 32: eight D-pad directions times two buttons (though you may argue that some of these combinations are nonsensical and should not actually be considered). Chess has an average branching factor of 35, whereas Checkers has a somewhat lower branching factor. Go has a whooping 400 for the very first move—as the board is populated, the branching factor decreases, but there are typically a few hundred potential positions to put any given stone.

While 400 is a very high branching factor compared to 35 or 2, it is nothing compared to the branching factor of many computer strategy games where multiple units can be moved every turn. Considering each combination of movements of individual units as an action, this means that the branching factor of the game is the product of the branching factor of the individual units. If you have 6 different units that can each take 10 different actions at a given time—a rather conservative estimate compared to typical games of, say, *StarCraft* (Blizzard Entertainment, 1998) or *Civilization* (MicroProse, 1991)—then your branching factor is a million!

But wait, it gets worse. For many games, it is not even possible to enumerate all the actions, as the input space is *continuous*. Think of any modern first-person game played on a computer or console. While it is true that computers do not really capture infinities well, and that continuous inputs such as computer mice, touchscreens and thumbsticks (on e.g., Xbox and Playstation controllers) actually return a number with such fine resolution that it is for all practical purposes continuous. The only way to create a practically enumerable set of actions is to discretize the continuous input

space somehow, and strike a compromise between overwhelming branching factors and reducing the input space so much as to not be able to play the game effectively.

The branching factor is a key determinant of the effectiveness of **tree search** algorithms. The complexity of the breadth-first algorithm (for single-player games) and the Minimax algorithm (for adversarial two-player games) for searching to depth d is b^d, where b is the branching factor. In other words, a high branching factor makes it almost impossible to search more than a few steps ahead. This fact dictates which games can be played with tree search methods; for example, Go has an order of magnitude higher branching factor than Chess, and this was arguably the main reason for the very poor performance of all kinds of AI methods on Go for decades (during which the same methods performed well on Chess). Monte Carlo tree search handles high branching factors better because it builds imbalanced trees, but it is by no means immune to the problem. Once the branching factor gets high enough (say, a million), it becomes impractical to enumerate even the actions at depth 1 and therefore to use tree search at all.

High branching factors are an issue for **reinforcement learning** algorithms as well. Mostly this has to do with the controller/policy representation. If you are using a neural network (or some other function approximator) for representing your policy, you may need to have outputs for each action; alternatively, if you are using the network to assign values to all actions, you need to iterate over them. In both cases, a large number of possible actions carries a cost. Another problem is the *exploration-exploitation* dilemma: the higher the number of possible actions, the longer it will take to explore them all while learning.

A final comment on branching factors is that for many games, they are not constant. In Chess, you have fewer moves available at the beginning of the game when most of your pieces are blocked, more towards the midgame, and fewer again in the endgame when most pieces may be blocked. In a typical RPG such as those in the *Final Fantasy* series (Square Enix, 1987–2016) the number of available actions increases as the player character accrues items, spells and other possibilities. As mentioned earlier, the number of available actions in Go decreases as you play.

Time Granularity: Turn-based Versus Real-time

When discussing branching factors, we talked about the number of possible actions to take at any "point in time". But how many points in time are there? How often can the player take an action? A fundamental distinction is between **turn-based** and **real-time** games. Most classic board games are turn-based games. In such games, players take turns, and at each turn a player can take an action, or a specified number of actions. The amount of real time that passes between turns is generally not of any importance inside the game (though tournaments and professional play often incorporate some form of time limit). Real-time games include many popular genres of computer games, such as FPS, racing games and platformers. Even within real-time games, there is considerable variation in how often an in-game action can be taken in practice. At the extreme there is the screen update frequency; the current generation of video games typically strives to have an update frequency of 60 frames per second to ensure a perceived smooth movement, but many games update the screen half as

often (or even less) because of the complexity of rendering complicated scenes. In practice, the number of actions a player character (or any other in-game character) could take per second is usually more limited than that.

To take two examples far apart on the time granularity scale, let us consider two adversarial games: Chess and *StarCraft* (Blizzard Entertainment, 1998). A game of Chess between skilled players on average lasts about 40 turns.[2] In *StarCraft*, a highly competitive real-time strategy (RTS) game, professional players often take two to four actions per second (each action is typically executed with a mouse click or a shortcut key). With a typical game lasting 10–20 min, this means that thousands of actions are taken in a game. But there are not that many more significant events in a game of *StarCraft* than in a game of Chess—the lead does not change much more often, and grand strategic decisions are not made that much more often. This means that the number of actions between significant game events is much higher in *StarCraft* (Blizzard Entertainment, 1998) than in Chess.

Time granularity limits how far ahead AI game-playing methods can look. A given depth of search means very different things depending on the time granularity of the game. Ten turns in Chess is enough to execute a whole strategy; ten actions ahead in *StarCraft* might just be a few seconds, during which the game might not have changed in any significant way. To play *StarCraft* well using tree search, one would need an exceptional search depth (thousands of actions), which would clearly be computationally infeasible. One way to address this challenge is to consider *macro-actions* (e.g., as in [863, 864]), which are sets or sequences of smaller, fine-grained, actions.

4.2.3 Game AI Deployment

Next we discuss some important issues in applying AI algorithms to games as dimensions of our taxonomy (see the bottom four rows of Fig. 4.2). These are design choices relating not so much to game design (covered in the previous section), as to AI algorithm design and the constraints under which the algorithm is used. We refer to these dimensions of our taxonomy as *game AI deployment*. This section expands the discussion about representation and utility covered in Sect. 2.1 with a focus on gameplaying AI.

Input Representation: Extracted Data Versus Pixels

Games differ in what information they present to the player, and how. Text adventures output text, the state of a classic board game can be described by the positions of all board pieces, and graphical video games serve moving graphics together with sound and occasionally outputs such as controller rumble. For digital games, the technical limitations of the hardware on which the game is implemented influences how it is

[2] http://chess.stackexchange.com/questions/2506/.

presented; as processor speed and memory capacity increases, the pixel resolution and scene complexity of video games has increased commensurably.

Importantly, the same game can be represented in different ways, and which way it is represented matters greatly to an algorithm playing the game. We view the input representation dimension as a continuum of many possibilities between **extracted data** and **pixels**. To take a racing game as an example, the algorithm could receive a first-person view out of the windscreen of the car rendered in 3D (*egocentric*), or an overhead view of the track rendering the track and various cars in 2D (*exocentric*). It could also simply receive a list of positions and velocities of all cars on the track in the frame of reference of the track (along with a model of the track), or a set of angles and distances to other cars (and track edges) in the frame of reference of the track.

The choices regarding input representation matter a lot when designing an AI solution for playing a game. If you want to learn a policy for driving a car around a track, and the inputs to the policy are the three continuous variables associated with speed and distance to the left and right edge of the track, learning a competent driving policy is comparatively simple. If your input is instead an unprocessed visual feed—i.e., tens of thousands of pixel values—finding a good policy is likely to be much harder. Not only is the policy search space in the latter case vastly greater, the proportion of the search space that corresponds to decently-performing policies is likely to be much smaller, as many more nonsensical policies are possible (e.g., turn left if even-numbered pixels are lighter than odd-numbered pixels; this policy does not map to the game state in any sensible way, and if it works it is a fluke). In order to learn to drive well based on visual input—at least in cases where illumination, roadside scenery, etc. vary significantly—you likely need to learn a visual system of some kind. In light of this, most naive policies applied to full visual input would likely not fare very well. Continuing the car racing example, even in cases where you have very few inputs, how these are represented matters; for example, it is much easier to learn a good driving policy if the inputs are represented in the frame of reference of the car rather than that of the track [1161, 1168]. A somewhat more comprehensive discussion on ways of representing low-dimensional inputs to neural networks can be found in [946].

Within game AI research focused on RL benchmarking, it has become common to learn policies that use full visual feeds as inputs. An early example of this is when Koutnik et al. [583] evolved neural networks to play *The Open Racing Car Simulator* (TORCS) from high-resolution video, Similarly, Kempka et al. [554] used the pixels of the screen as input to a deep Q network that was trained to play a version of *Doom* (id Software, 1993), Mnih et al. [773] trained deep networks to play Atari 2600 games using Q-learning and Pearce et al. [856] trained diffusion models to learn to play *Counter-Strike: Global Offensive* (Valve, 2014) in a human-like fashion. Using the raw pixel inputs is often motivated by the AI having the same conditions as a human would have, and thus achieving a level playing field between human and AI. Another motivation is that if you want to use your algorithm to play a game "out of the box", without any API or additional engineering to expose the internal state of the game, you will likely have to resort to the raw visual feed. However, in cases

where you have access to the source code of the game or a useful API—as you would always have when developing AI for a new game—there is no reason to not utilize the "digested" game state in whatever form makes the task of the AI algorithm easiest. Whether or not to give the AI information that a human player does not have access to, i.e., "cheating", is a separate question.

Forward Model: Fast Versus None

A very important factor when designing an AI to play a game is whether there is a simulator of the game, a so-called **forward model**, available. A forward model is a model which, given a state s and an action a, reaches the same state s' as the real game would reach if it it was given a at s. In other words, it is a way of playing the game in simulation, so that consequences of multiple actions can be explored before actually taking some of those actions in the real game. Having a forward model of the game is necessary in order to be able to use any tree search-based approaches to playing a game, as those approaches depend on simulating the outcome of multiple actions.

A very desirable property of a forward model is that it is *fast*. In order to be able to use a tree search algorithm effectively for control in a real-time game, one would generally need to be able to simulate gameplay at least a thousand times faster than real-time, preferably tens or hundreds of thousands of times faster.

It is very easy to construct a forward model for classic board games such as Chess and Go, as the game state is simply the board state and the rules are very easy to encode. For many video games, constructing a forward model can be done by simply copying (or otherwise reusing) the same code that controls the game itself, but without waiting for user input or displaying graphics, and without performing all the calculations involved with graphics rendering. For some video games—notably games that were originally implemented for much older hardware, such as classic arcade games that were implemented on 8-bit or 16-bit processors—forward models can be made much faster than real-time, as the core game loop is not that computationally complex.

For many games, however, it is impossible or at least very hard to obtain a fast forward model. For most commercial games, the source code is not available, unless you are working at the company that develops the game. Even if the source code is available, current software engineering practices in the game industry make it very hard to extract forward models from game code, as the core control loops are often closely tied up with user interface management, rendering, animation and sometimes network code. This is reinforced by the design of game engines such as Unity and Unreal. A change in software engineering practices to separate the core game loop more cleanly from various input/output functions so that forward models could more easily be built would be one of the most important enablers of advanced AI methods in video games. We touch upon this aspect in our research frontiers chapter; see Chap. 14. However, in some cases the computational complexity of the core game loop might still be so high that any forward models built on the core game code would be too slow to be usable. In some of such cases, it might be practical to build and/or learn a simplified or **approximate forward model** [866], where the state resulting

from a series of actions taken in the forward model is not guaranteed to be identical to the state resulting from the same series of actions in the actual game. Whether an approximate forward model is acceptable or not depends on the particular use case and motivation for the AI implementation.

Note that a somewhat less accurate forward model might still be desirable. For example, when there is significant hidden information or stochasticity the AI designer might not want to provide the AI agent with an *oracle* that makes the hidden information observable and tells the agent which random events will occur; such an approach was proposed in [1014] for the design of gameplaying AI for *Pandemic* (Z-Man Games, 2008). Taking such a design decision might lead to unreasonably good performance and the appearance of cheating.

Research in world models [420, 422] and neural game engines [52, 1199] suggest that learning approximate forward models might be practical even when trained from pixels. There is a considerable opportunity in working out a way of integrating such models with legacy game engines to support fast planning and RL within the current game development pipeline.

When a forward model cannot be produced, **tree search** algorithms cannot be applied. It is still possible to manually construct agents, and also to learn agents through **supervised learning** or some form of **reinforcement learning**,. However, note that while the reinforcement learning approaches in general do not need a complete forward model in the sense that the results of taking any action in any state can be predicted, they still need a way to run the game faster than real-time. If the game cannot be sped up significantly beyond the pace at which it is naturally played, it is going to take the algorithm a very long time to learn to play.

AI Controller: Reactive Versus Planner

A crude but useful distinction in artificial intelligence is between algorithms that try to decide *what to do in a given situation* by examining possible actions and future states—roughly, tree search algorithms of various kinds—and algorithms that *learn a model* (such as a policy) over time—i.e., machine learning. The same distinction exists within AI for playing games. There are **reactive** algorithms developed that do not need to learn anything about the game, but do need a forward model (tree search); there are **planning** algorithms that do not need a forward model, but instead learn a policy as a mapping from state(s) to action (model-free reinforcement learning); and there are algorithms that require both a forward model and training time (model-based reinforcement learning and tree search with adaptive hyperparameters).

What type of algorithm you will want to use depends largely on your motivation for using AI to play games. If you are using the game as a testbed for your AI algorithm, your choice will be dictated by the type of algorithm you are testing. If you are using the AI to enable player experience in a game that you develop—for example, in a non-player role—then you will probably not want the AI to perform any learning while the game is being played, as this risks interfering with the gameplay as designed by the designer. In other cases you are looking for an algorithm that can play some range of games well, and do not have time to retrain the agent for each game.

Number of Games: One Versus Many

An aim the AI designer might wish to achieve is that of **general game playing**. Here, we are not looking for a policy for a single game, we are looking for a more generic agent that can play any game that it is presented with—or at least any game from within a particular distribution or genre, and which adheres to a given interface. General game playing is typically motivated by a desire to use games to progress towards artificial *general* intelligence, i.e., developing AI that is not only good at one thing but at many different things [998, 1116, 1229]. The idea is to avoid overfitting (manually or automatically) a given game and come up with agents that generalize well to many different games. It is a common phenomenon that when developing agents for a particular game, for example, for a game-based AI competition, many special-purpose solutions are devised that do not transfer well to other games [393, 1152].

For this reason, it is common to evaluate general game playing agents on unseen games, i.e., games on which they have not been trained and which the designers of the agent were not aware of when developing the agent. There are several frameworks for general game playing, including the General Game Playing Competition [374], the General Video Game AI Competition [869, 870], the Arcade Learning Environment [75] and the XLand framework [1130] that can generate simple 3D multiplayer games for the purpose of training generalist gameplaying agents [930].

General video game playing, where AI is developed to play to win in the player role across many games (ideally all games), can be seen as diametrically opposed to the typical use of AI for playing games in commercial game development, where the AI is playing for experience, and is carefully tuned to a particular game. However, the development of AI methods for general game playing certainly benefits commercial game AI in the end. And even when developing game AI as part of game development, it is good engineering practice to develop methods that are reusable to some extent.

4.3 Further Reading

As noted earlier, much of our discussion in this chapter and our taxonomy per se are inspired by the book *Characteristics of Games* by Elias et al. [320]. The interested reader is also referred to the various studies and tools we cite throughout the use cases presented in this chapter. Indicatively, for RL benchmarking and general game playing the reader is referred to [75] and [645], respectively. For multiplayer team bots we recommend the recent study of Pearce et al. [856]. Finally, for procedural personas and game testing we recommend the work of Holmgård et al. [458] and Gudmundsson et al. [410], respectively.

4.4 Summary

In this chapter we discussed the different *roles* AI can take and the different *goals* AI methods can have, which collectively define the four *uses* of AI for playing games. In particular, AI can either play to win or play in order to create a particular experience for a human player or observer. The former goal involves maximizing a utility that maps to the game performance whereas the latter goal involves objectives beyond merely winning such as engagement, believability, balance and interestingness. AI as an actor can take the role of either a player character or a non-player character that exists in a game. Moreover, the *characteristics of games* an AI method needs to consider when playing include: number of players, stochasticity, observability, action space, branching factor, and time granularity. Further, when we design and *deploy* an AI gameplaying algorithm we also need to consider aspects such as the algorithm's input representation, the existence and speed of a forward model, the training time available, and the number of games AI can play.

All aforementioned aspects were put in a taxonomy of gameplaying AI and were detailed in this first chapter of Part II. We argue that these dimensions are important and relevant regardless of the AI method used. The next chapter moves on to the various AI methods an agent can employ to play a game.

Chapter 5
Methods for Playing Games

Abstract This chapter presents an overview of AI methods for playing games. The main approaches discussed are planning, reinforcement learning, and supervised learning. There is also a brief discussion of behavior authoring, hybrid methods, and machine learning methods that learn game representations and forward models rather than policies. Throughout the chapter, there are multiple examples of these approaches from the scientific literature and published games.

This chapter is devoted to discussing the various methods that can be used to play games, and how the right choice of method depends on the characteristics of the game. The methods discussed here have been briefly discussed in Part I. Most of those methods can be used to play games and this chapter unfolds the multiple ways this may happen, following on from the discussion in Chap. 3 about the characteristics of games and roles AI could take in gameplaying.

We group the AI methods by following, in part, the structure of Chap. 3. We thus start with behavior authoring, the most popular family of game AI algorithms with the widest reach in the game industry (Sect. 5.1). We move on to planning based methods (Sect. 5.2), to RL based algorithms (Sect. 5.3), supervised learning (Sect. 5.4), large pretrained models (Sect. 5.5) and ultimately chimeric methods (Sect. 5.6). The chapter concludes with a section dedicated to methods that learn aspects beyond a gameplaying policy such as representations and forward (or world) models (Sect. 5.7).

5.1 Behavior Authoring

Most methods for controlling NPCs in games do not rely on either machine learning or planning (search in state space). Rather, they rely on hand-authored behaviors combined with navigation (search in the game's physical space). This usually takes the form of behavior structured as a behavior tree, where physical movement is executed using path planning. Before the recent popularity of behavior trees, finite state machines were very widely used in the role of high-level behavior structuring.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_5

For certain kinds of behavior there are other control structures. For example, dialogue is often structured as a dialogue tree, where the various lines of dialogue that the play can choose from and the responses from the NPC can be seen as a tree. Sometimes, the tree of dialogue options loops back, making it a graph; this makes it a type of finite state machine. The intricacies of behavior authoring fall somewhat outside the scope of this book. While Chap. 3 outlines such methods more in-depth information can be found in specialized textbooks [706, 767].

5.2 Planning-Based Approaches

Algorithms that select actions through planning a set of future actions in a state space are generally applicable to games, and do not in general require any training time. They do require a fast forward model if searching in the game's state space, but not if simply using them for searching in the physical space (path-planning). Tree search algorithms are widely used to play games, either on their own or in supporting roles in gameplaying agent architectures.

5.2.1 Classic Tree Search

Classic tree search methods, which feature little or no randomness, have been used in gameplaying roles since the very beginning of research on AI and games. As mentioned in Chap. 1, the Minimax algorithm and α-β pruning were originally invented in order to play classic board games such as Chess and Checkers [1191]. While the basic concepts of adversarial tree search have not really changed since then, there have been numerous tweaks to existing algorithms and some new algorithms. In general, classic tree search methods can easily be applied in games that feature full observability, a low branching factor and a fast forward model. Theoretically they can solve any deterministic game that features full observability for the player (see the red cube in Fig. 4.3); in practice, they still fail in games containing large state spaces.

Best-first search, in particular a myriad variations of the A* algorithm, is very commonly used for **path-planning** in modern video games. When an NPC in a modern 3D FPS or RPG decides how to get from point A to point B, this is typically done using some version of A*. In such instances, search is usually done in (in-game) physical space rather than state space, so no forward model is necessary. As the space is pseudo-continuous, search is usually done on the nodes of a mesh or lattice overlaid on the area to be traversed. Note that best-first search is only used for navigation and not for the full decision-making of the agent; methods such as behavior trees or finite-state machines (which are usually hand-authored) are used to determine *where* to go, whereas A* (or some variation of it) is used to determine *how* to get there.

Indeed, in games where player input is clicking at positions to go—think of an overhead brawler like *Diablo* (Blizzard Entertainment, 1996) or an RTS like *StarCraft* (Blizzard Entertainment, 1998)—the execution of the player's order usually involves a path-planning algorithm as well. Recent additions to the family of best-first algorithms include jump point search (JPS), which can improve performance by orders of magnitude compared to standard A* under the right circumstances [1094].

Hierarchical pathfinding is its own research area based on the idea of dividing up an area into subareas and using separate algorithms for deciding how to go between and within the areas. Choosing a path-planning algorithm for a modern video game is usually a matter of choosing the algorithm that works best given the shape of the environments the agents are traversing, the particular way a grid or navmesh is overlaid on top of this space, and the demands of the animation algorithm. Generally, one size does not fit all; some textbooks devoted to industry-oriented game AI discuss this in more depth [767].

Beyond path-planning, best-first algorithms such as A* can be used for controlling all aspects of NPC behavior. Searching in the state space of the game, not just the physical space, requires a (fast) forward model. The difference between path-planning, i.e. searching in physical space, and planning, i.e. searching in state space, is very important and sometimes overlooked, so it is worth dwelling on it. When searching in physical space, the search process only simulates moving the agent itself; the rest of the game environment is not updated. This means that if actions that the agent takes have effects in the world (beyond moving the agent itself), those are not modeled. Also, actions and movements of NPCs are not modeled. Searching in physical space therefore has significant limitations in what it can be used for. Searching in state space, on the other hand, steps the game engine forward between each simulated action, computing the effects of player and NPC actions. While this places much higher demands on access to the game engine and much higher computation, it is vastly more useful as it can be used for problem-solving beyond mere navigation.

To take an example, the winner of the 2009 Mario AI Competition was entirely based on A* search in state space [1159]. This competition tasked competitors with developing agents that could play a Java-based clone of the classic platform game *Super Mario Bros* (Nintendo, 1985)—it later evolved into a multi-track competition [539]. While a forward model was not supplied with the original competition software, the winner of the competition, Robin Baumgarten, created one by adapting parts of the core game code. He then built an A* agent which at any point simply tried to get to the right edge of the screen. An illustration of the agent can be seen in Figs. 3.5 and 5.1. This worked extremely well: the resulting agent played seemingly optimally, and managed to get to the end of all levels included in the competition software. A video showing the agent navigating one of the levels has gathered more than a million views on YouTube.[1] The appeal of seeing the agent playing the game is partly the extreme skill of the agent in navigating among multiple enemies.

[1] https://www.youtube.com/watch?v=DlkMs4ZHHr8.

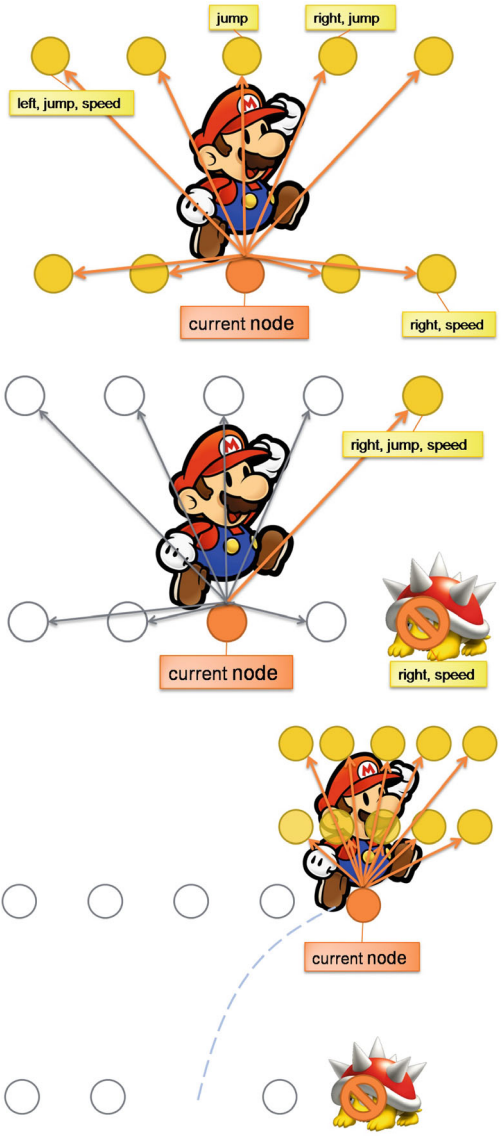


Fig. 5.1 An illustration of the key steps of A* search for playing *Super Mario Bros* (Nintendo, 1985). The agent considers a maximum of nine possible actions at each frame of the game, as a result of combining the jump and speed buttons with moving right or left (top figure). Then the agent picks the action with the highest heuristic value (middle figure). Finally, the Mario agent takes the action (i.e., right, jump, speed in this example), moves to a new state and evaluates the new action space in this new state (bottom figure). More details about the A* agent that won the Mario AI competition in 2009 can be found in [1159]

It is important to note that the success of this agent is due to several factors. One is that the levels are fairly linear. In a later edition of the competition, levels with dead ends which required back-tracking were introduced, which defeated the pure A* agent [539]. Three other factors are that *Super Mario Bros* (Nintendo, 1985) is deterministic, it has locally perfect information (at any instance the information in the current screen is completely known) and of course that a good forward model is available. If the A* would not have used a complete model of the game including the movement of enemies, it would have been impossible to plan paths around these enemies.

5.2.2 Stochastic Tree Search

The MCTS algorithm burst onto the scene of Go research in 2006 [137, 241], and heralded a quantum leap in performance of Go-playing AI. Classic adversarial search had performed poorly on Go, partly because the branching factor is too high (about an order of magnitude higher than Chess) and partly because the nature of Go makes it very hard to algorithmically judge the value of a board state. MCTS partly overcomes these challenges by building imbalanced trees where not all moves need to be explored to the same depth (reduces effective branching factor) and by doing random rollouts until the end of the game (reduces the need for a state evaluation function). The AlphaGo [1040] software which beat two of the best human Go players in the world in 2016 and 2017, is built around the MCTS algorithm.

The success of MCTS on Go has led researchers and practitioners to explore its use for playing a wide variety of other games, including trading card games [1240], platform games [502], real-time strategy games [524, 1063], racing games [338] and so on. Of course, these games differ in many ways from Go. While Go is a deterministic perfect information game, a real-time strategy game such as *StarCraft* (Blizzard Entertainment, 1998), a trading card game such as *Magic: The Gathering* (Wizards of the Coast, 1993), or any Poker variant feature both hidden information and stochasticity. Methods such as Information Set Monte Carlo tree search are one way of dealing with these issues, but impose computational costs of their own [246].

Another problem is that in games with fine time granularity, it might take a prohibitively long time for a rollout to reach a terminal state (a loss or a win); in many video games it is possible to take an arbitrary number of actions without winning or losing the game, or even doing something that materially affects the outcome of the game. For example, in *Super Mario Bros* (Nintendo, 1985), most randomly generated action sequences would not see Mario escaping the original screen, but basically pacing back and forth until time runs out, thousands of time steps later. One response to this problem is to only roll out a certain number of actions, and if a terminal state is not encountered use a state evaluation function [137]. Other ideas include pruning the action selection so as to make the algorithm search deeper [502]. Given the large number of modifications to all components of the MCTS algorithm,

it makes more sense to think of MCTS as a general algorithmic framework rather than as a single algorithm.

Many games could be played either through MCTS, uninformed search (such as breadth-first search) or informed search (such as A*). Deciding which method to use is not always straightforward, but luckily these methods are relatively simple to implement and test. Generally speaking, Minimax can only be used for two-player adversarial games whereas other forms of uninformed search are best used for single-player games. Best-first search requires some kind of estimate of a distance to a goal state, but this does not need to be a physical position or the end goal of the game. Varieties of MCTS can be used for both single-player and two-player games, and often outperform uninformed search when branching factors are high.

5.2.3 Evolutionary Planning

Interestingly, decision making through planning does not need to be built on tree search. Alternatively, one can use optimization algorithms for planning. The basic idea is that instead of searching for a sequence of actions starting from an initial point, you can optimize the whole action sequence. In other words, you are searching the space of complete action sequences for those that have maximum utility. Evaluating the utility of a given action sequence is done by simply taking all the actions in the sequence in simulation, and observing the value of the state reached after taking all those actions.

The appeal of this idea is that an optimization algorithm might search the plan space in a very different manner compared to a tree search algorithm: all tree search algorithms start from the root of the tree (the origin state) and build a tree from that point. Evolutionary algorithms instead regard the plan as simply a sequence, and can perform mutations or crossover at any point in the string. This could help in guiding the search at different areas of the plan space that a tree search algorithm would explore for the same problem.

While many different optimization algorithms could be used, the few studies on optimization-based planning in games that can be found in the literature use evolutionary algorithms. Perez et al. proposed using evolutionary planning for single-player action games, calling this approach **rolling horizon evolution** [865]. In the particular implementation for the Physical Traveling Salesman Problem—a hybrid between the classic Traveling Salesman Problem (TSP) problem and a racing game—an evolutionary algorithm was used to generate a plan every time step. The plan was represented as a sequence of 10–20 actions, and a standard evolutionary algorithm was used to search for plans. After a plan was found, the first step of the plan was executed, just as with a tree search algorithm. Agents based on evolutionary planning generally perform competitively in the General Video Game AI Competition [361, 870]. The basic algorithm can be improved by importing various tools from the evolutionary computation toolbox, such as self-adaptation [362].

Evolutionary planning is particularly promising as a technique for handling very large branching factors, as we have seen that games with multiple independent units (such as strategy games) can have. Justesen et al. [522] applied evolutionary computation to select actions in the turn-based strategy game *Hero Academy* (Robot Entertainment, 2012), calling this approach "online evolution". Given the number of units the player controls and the number of actions available per unit, the branching factor is about one million; therefore, only a single turn ahead was planned. Evolutionary planning was shown to outperform Monte Carlo tree search by a wide margin in that game. Wang et al. [1231], and Justesen and Risi [523] later applied variants of this technique to *StarCraft* (Blizzard Entertainment, 1998) tactics. Given the continuous-space nature of the game, the branching factor would be extreme if every possible movement direction for every unit was considered as a separate action. What was evolved was therefore not a sequence of actions, but rather which of several simple scripts (tactics) each unit would use in a given time step. This idea was borrowed from Churchill and Buro, who combined a "portfolio" of scripts with simple tree search [213]. Wang et al. [1231] showed that evolutionary planning performed better than several varieties of tree search algorithms in this simple *StarCraft* (Blizzard Entertainment, 1998) scenario. Rolling horizon evolution has also been employed to play cooperative board games such as *Pandemic* (Z-Man Games, 2008) [1014].

Evolutionary planning in games is a relatively recent invention, and there are only a few studies on this technique so far. It is not well understood under what conditions this technique performs well, or even really why it performs so well when it does. A major unsolved problem is how to perform evolutionary adversarial planning [979]; whereas planning based on tree search works in the presence of an adversary (for example, see the minimax algorithm), it is not clear how to integrate this into a genotype. Perhaps through competitive coevolution of actions taken by different players? There is, in other words, plenty of scope for further research in this area. Note that the previous paragraphs are essentially unchanged from the first edition of the book, meaning that opportunities are wide open here.

5.2.4 Planning with Symbolic Representations

While planning on the level of in-game actions requires a fast forward model, there are other ways of using planning in games. In particular, one can plan in an abstract representation of the game's state space. The field of automated planning has studied planning on the level of symbolic representations for decades [378]. Typically, a language based on first-order logic is used to represent events, states and actions, and tree search methods are applied to find paths from the current state to an end state. While technically any planning algorithm, including evolutionary planning, could be applied to symbolic representations, most existing technique are based on best-first search. This style of planning originated with the STRIPS representation

used in Shakey, the world's first digital mobile robot [817]; symbolic planning has since been used extensively in numerous domains.

The horror-themed first-person shooter *F.E.A.R.* (Sierra Entertainment, 2005) became famous within the AI community for its use of planning to coordinate NPC behavior. The game's NPC behavior also received nice reviews in the gaming press, partly because the player is able to hear the NPCs communicate with each other about their plan of attack, heightening immersion. In *F.E.A.R.*, a STRIPS-like representation is used to plan which NPCs perform which actions (flank, take cover, suppress, fire, etc.) in order to defeat the player character. The representation is on the level of individual rooms, where movement between one room and the next is usually a single action [833]. Using this high-level representation, it is possible to plan much further ahead than would be possible when planning on the scale of individual game actions. Such a representation, however, requires manually defining states and actions.

5.3 Reinforcement Learning

As discussed in Chap. 3, the reinforcement learning paradigm includes algorithms from the temporal difference or approximate dynamic programming family, or those based on policy gradients; for simplicity, we will refer to such algorithms as *classic* reinforcement learning methods. But the RL methods also include applications of evolutionary algorithms to reinforcement learning such as neuroevolution and genetic programming, and other more exotic things. In this section, we will discuss both classic methods (including those that involve deep neural networks) and evolutionary methods as they are applied to playing games. Another way of describing the difference between these methods is the difference between *ontogenetic* (which learns during "lifetimes") and *phylogenetic* (which learns between "lifetimes") methods [1169].

Reinforcement learning algorithms are applicable to games when there is learning time available. Usually this means plenty of training time: most reinforcement learning methods will need to play a game thousands, or perhaps even millions, of times in order to play it well. Therefore, it is very useful to have a way of playing the game much faster than real-time (or a very large server farm). Some reinforcement learning algorithms, but not all, also require a forward model. Once it has been trained, a reinforcement-learned policy can usually be executed very fast.

It is important to note that the planning-based methods described in Sect. 5.2 for playing games cannot be directly compared with the reinforcement learning methods described in this section. They solve different problems: planning requires a forward model and significant time at each time step; reinforcement learning instead needs learning time and may or may not need a forward model.

5.3.1 Classic and Deep Reinforcement Learning

As already mentioned in Chap. 1, classic reinforcement learning methods were used with games early on, in some cases with considerable success. Arthur Samuel devised an algorithm—which can be said to be the first classic reinforcement learning algorithm—in 1959 to create a self-learning Checkers player. Despite the very limited computational resources of the day, the algorithm learned to play well enough to beat its creator [987]. Another success for classic reinforcement learning in game-playing came a few decades later, when Gerald Tesauro used the modern formulation of temporal difference learning to teach a simple neural network to play Backgammon, named TD-gammon; it learned to play surprisingly well, after starting with no information and simply playing against itself [1134] (see Chap. 3). This success motivated much interest in reinforcement learning during the 1990s and early 2000s.

However, progress was limited by the lack of good function approximators for the value function (e.g., the Q function). While algorithms such as Q-learning will provably converge to the optimal policy under the right conditions, the right conditions are in fact very restrictive. In particular, they include all state values or {state, action} values that are stored separately, for example, in a table. However, for most interesting games there are far too many possible states for this to be feasible—almost any video game has at least billions of states. This means that the table would be too big to fit in memory, and that most states would never be visited during learning. It is necessary to use a compressed representation of the value function that occupies less memory and also does not require every state to be visited in order to calculate its value. It can, instead, calculate it based on neighboring states that have been visited. In other words, what is needed is a **function approximator**, such as a neural network.

However, using neural networks together with temporal difference learning turns out to be non-trivial. It is very easy to encounter **catastrophic forgetting** [353], where sophisticated strategies are unlearned in favor of degenerate strategies (such as always taking the same action). The reasons for this are complex and go beyond the discussion in this chapter. However, to intuitively understand one of the mechanisms involved, consider what would usually happen for a reinforcement learning agent playing a game. Rewards are very sparse, and the agent will typically see long stretches of no reward, or negative reward. When the same reward is encountered for a long time, the backpropagation algorithm will be trained only with the target value of that reward. In terms of supervised learning, this is akin to training for a long term on a single training example. The likely outcome is that the network learns to only output that target value, regardless of the input. More details on the method of approximating a value function using an ANN can be found in Sect. 3.7.2.

A major success in the use of reinforcement learning of the temporal difference variety together with function approximators came in 2015, when Google DeepMind published a paper where they managed to train deep neural networks to play a number of different games from the classic Atari 2600 games console [773]. Each network was trained to play a single game, with the inputs being the raw pixels of the game's

visuals, together with the score, and the output being the controller's directions and fire button. The method used to train the deep networks is **deep Q networks**, (see Chap. 3. Crucially, they managed to overcome the problems associated with using temporal difference techniques together with neural networks by a method called *experience replay*. Here, short sequences of gameplay are stored, and replayed to the network in varying order, in order to break up the long chains of similar states and reward. This can be seen as akin to batch-based training in supervised learning, using small batches.

Another major step forward for reinforcement learning in games was the development of the **Proximal Policy Optimization** algorithm [1006] that we covered in Chap. 3. PPO is a policy gradient method, meaning that it directly optimized the policy in the direction of higher rewards. Crucially, PPO limits the changes to the policy that can be made at any one step by clipping the advantage function, leading to an algorithm that is more stable, meaning that it has a much reduced risk of catastrophic forgetting. PPO was initially tested on the classic Atari games, but became more widely known after it was used to train agents to play the MOBA *Dota 2* (Valve, 2013) [77]. Because of its simplicity, stability, and versatility, PPO is often used as a default RL algorithm in many applications, including gameplaying.

One issue with much, or perhaps all, of reinforcement learning is **overfitting**. When training a network to play a game, you would expect to get a policy that is specific for that particular game; it is limited to playing that game, and cannot play games it has not been trained on, but it plays that game well. However, in practice the policy tends to be much more limited than that. Typically, the trained policy will work only for the particular level(s) of the game used in training, with those particular starting positions, and will cease to function if even minor changes are applied to the visual presentation, such as changing some textures or adjusting the screen size [525, 847]. In short, reinforcement learning tends to overfit.

While the tendency of RL to overfit has been known for a while, it became more acute with the advent of deep RL, as RL was to learn more complex games with more complex state representations (e.g., pixels). Around 2018 and 2019, a number of studies sought to characterize the overfitting, and conversely the lack of generalization, in deep RL [218, 332, 525, 847]. One method for combating overfitting in RL is to ensure that the training scenario varies as much as possible, while maintaining the key characteristics of the game that you want the policy to exploit. This can be done through procedural content generation [930, 947], where a new level is generated for each episode. If the PCG method includes a way of controlling the difficulty of the generated levels, you can use this to create a form of curriculum, where harder levels are generated as the policy gets better at solving existing levels; this can further increase generalization [525].

5.3.2 *Evolutionary Reinforcement Learning*

The other main family of reinforcement learning methods is evolutionary methods (see Part I for more details). In particular, using evolutionary algorithms to evolve the weights and/or topology of neural networks (**neuroevolution**) or programs, typically structured as expression trees (**genetic programming**). The fitness evaluation consists in using the neural network or program to play the game, and using the result (e.g., score) as a fitness function.

This basic idea has been around for a long time, but was surprisingly underexplored for a long time. John Koza, a prominent researcher within genetic programming, used an example of evolving programs for playing Pac-Man in his 1992 book [588]. A couple of years later, Pollack and Blair showed that evolutionary computation can be used to train backgammon players, with similar results to TD-Gammon [895]. Outside of games, a community of researchers was forming in the 1990s exploring the idea of using evolutionary computation to learn control strategies for small robots; this field came to be called **evolutionary robotics** [820]. Training robots to solve simple tasks (e.g., navigation, obstacle-avoidance and situational learning) has very much in common with training NPCs to play games, in particular two-dimensional arcade-like games [946, 1271, 1272].

Starting around 2005, a number of advances were made in applying neuroevolution to play different types of video games. This includes applications to car racing [583, 658, 1161, 1163], first-person shooters [852, 1204], strategy games [141], real-time strategy games [1079] and classic arcade games such as Pac-Man [671, 1271]. Perhaps the main takeaway from this work is that neuroevolution is extremely versatile, and can be applied to a wide range of games, usually in several different ways for each game. For example, for a simple car racing game it was shown that evolving neural networks that acted as state evaluators, even in combination with a simple one-step lookahead search, substantially outperformed evolving neural networks working as action evaluators [676]. Input representation matters too; as discussed in Sect. 4.2.3, *egocentric* inputs are generally strongly preferred, and there are additional considerations for individual game types, such as how to represent multiple adversaries [1079].

Neuroevolution has seen great success in learning policies in cases where the state can be represented using relatively few dimensions (say, fewer than 50 units in the neural network's input layer), and is often easier to tune and get working than classic reinforcement learning algorithms of the temporal difference variety. However, neuroevolution seems to have problems scaling up to problems with very large input spaces that require large and deep neural networks, such as those using high-dimensional pixel inputs. The likely reason for this is that stochastic search in weight space suffers from the *curse of dimensionality* in a way that gradient descent search (such as backpropagation) does not. Currently, most successful examples of learning directly from high-dimensional pixel inputs use deep Q-learning or similar methods, though there are approaches that combine neuroevolution with unsupervised learning, so that controllers are learned that use a compressed representation of

the visual feed as input [20, 252, 583]. It is important to note that "raw" neuroevolution (without a dimensionality-reducing preprocessing layer) is far from useless at playing games from high-dimensional pixel representations; there are a number of examples of well-performing policies with pixel inputs and millions of parameters trained with neuroevolution [209, 983, 1098]. However, gradient-based methods tend to do *better* at these scales.

For more details on the general method of neuroevolution and pointers to the literature the reader is referred to Sect. 3.7.4, and to a survey paper on neuroevolution in games [946].

5.4 Supervised Learning

Games can also be played using supervised learning. Specifically, policies or controllers for playing games can be learned through supervised learning. This is also called **behavior cloning**. The basic idea here is to record traces of human players playing a game and train some function approximator to behave like the human player. The traces are stored as lists of input-output pairs <features, target> where the features represent the game state (or an observation of it that would be available to the agent) and the target is the action the human took in that state. Once the function approximator is adequately trained, the game can be played—in the style of the human(s) it was trained on—by simply taking whatever action the trained function approximator returns when presented with the current game state. Alternatively, instead of learning to predict what action to take, one can also learn to predict the value of states, and use the trained function approximator in conjunction with a search algorithm to play the game. Further details about supervised algorithms that can be used can be found in Chap. 3.

Beyond behavior cloning, there are many ways of learning behavior from game traces. **Generative Adversarial Imitation Learning** (GAIL) takes inspiration from GANs in image synthesis, but synthesizes behavior rather than images [454]. In GAIL, just like in GANs, there is a generator and a discriminator; the generator generates behavior based on observations, so is in other words the policy. The discriminator tries to tell generated behavior from real behavior from the dataset. This is an elegant idea with good results. While we are on the topic of using popular image generation methods to imitate behavior, it should be pointed out that diffusion models can also do this. For example, Pearce et al. [856] used diffusion models to imitate human playing styles in *Counter-Strike: Global Offensive* (Valve, 2014).

Another improvement on behavior cloning is the Dagger algorithm, which addresses issues with behavior cloning potentially learning brittle strategies that do not work outside the training distribution. It does this by learning a policy, running the policy and observing when it falls outside of distribution, and then querying a human or algorithm expert for what to do in those situations. It can be seen as a form of interactive imitation learning [967].

5.5 Large Pretrained Models

The huge capability advances in large language models in recent years poses the question whether they could be used to play games. Either simply as a pre-trained model with an appropriate prompt, or a model fine-tuned on information relevant to a game or a type of games.

Obviously, the input and output representation of the game is of crucial importance here. If the game state and action set are naturally represented as text, then it is much easier to use an LLM to play the game. In particular, this is true for various language-based games and puzzles that are popular pastimes, for example the very popular games hosted by *The New York Times*. One of these is the *Connections* game, where the player is asked to sort 16 words into four categories. What constitutes a category varies widely; it could be words that are connected to the same phenomenon, that are semantically related somehow, or maybe all are viable words if you remove the letter "t". Merino et al. [1147] studied playing the *Connections* game as a benchmark for various NLP methods, including various types of LLMs, and found that *Connections*-playing ability scales with the general competence of the model and its ability to represent complex concepts.

Another genre of games that seems naturally suited to LLM playing are text adventure games. This type of games is discussed in the next chapter, in Sect. 6.8. When the input is not naturally text-based, there are various strategies one can adopt in order to deliver a state description in a format an LLM can parse. One is to create an API for the game, where the LLM can query the game and receive text-formatted responses. Queries could be things such as the state game variables such as health or ammunition, how many enemies are in front of the player character, or a list of the items that are in the inventory. The *Voyager* is an interesting example of this; it is an open-ended *Minecraft*-playing agent that plays the game by accessing information and issuing commands through function calling [1232]. This requires not only an extensive API for the game, but also that the LLM is highly familiar with the game. This is true for *Minecraft* but not for most other games, especially those still in development. Voyager is discussed further in the next chapter, in Sect. 6.4.3.

If the game is laid out in a discrete 2D grid, you could feed the game state to the LLM as a string, with each tile type represented by a different character, similar to ASCII graphics in old games. This can be done for example with classic 2D top-down RPGs. GameTraversalBenchmark is an LLM benchmark that generates maps with various navigation challenges (e.g., get from house to the forest) and asks the LLM to plan a path. While larger and more modern LLMs generally perform better than older ones, even the best LLMs cannot reliable navigate these relatively simple game environments [798].

With vision-language models (VLMs), the possibility arises of using vision input in addition to, or instead of, text-based input. Vision-based foundation models for game playing are very under-explored at this point. The SIMA agent from DeepMind [906] is an early attempt on developing an agent that can play a large variety of games based on visual input only. SIMA can take instructions expressed as text and follow

them with reasonable accuracy in a variety of games, though it performs better on games that it has been trained on.

5.6 Chimeric Game Players

While planning, reinforcement learning and supervised learning are fundamentally different approaches to playing games, solving the gameplaying problem under different constraints, which does not mean that they cannot be combined. In fact, there are many examples of successful **hybrid** approaches or **chimeras** of approaches from these three broad classes. One example is **dynamic scripting** [1074] which can be viewed as a form of a **learning classifier system** [605] in that it involves a rule-based representation coupled with reinforcement learning. Dynamic scripting adjusts the importance of scripts via reinforcement learning at runtime and is based on the current game state and immediate rewards obtained. Dynamic scripting has seen several applications in games including fighting games [690] and real-time strategy games [257, 678]. The approach has been used mainly for AI that adapts to the skills of the player, thereby aiming at the experience of the player and not necessarily at winning the game.

Another good example is AlphaGo. This extremely high-performing Go-playing agent actually combines planning through search, reinforcement learning and supervised learning [1040]. At the core of the agent is a Monte Carlo tree search algorithm which searches in the state space (planning). Rollouts, however, are combined with evaluations from a neural network which estimates the value of states, and node selection is informed by a position estimation network. Both the state network and position network are initially trained on databases of games between Go grandmasters (supervised learning), and later on further trained by self-play (reinforcement learning).

The basic AlphaGo method has been further developed into AlphaGo Zero and, eventually, AlphaZero. AlphaGo Zero simplifies the AlphaGo method in several ways [1042]. It gets rid of the supervised pretraining, and simply uses reinforcement learning to train the networks through self-play only, and also uses a single network in the role of both policy and value estimator. A key advance to make this possible is to use MCTS search in the rollout phase, and update the network with the values found from MCTS. The search procedure in MCTS essentially acts as a policy improvement operator.

AlphaZero takes this fusion of RL and tree search further. AlphaZero removes various optimizations specific to the game of Go, such as using symmetries, making the algorithm applicable to a number of other games and game-like problems [1041]. To showcase this more general ability, the team behind AlphaZero applied the algorithm to Chess and Shogi as well as Go, achieving extremely strong play for each game. A fresh network needed to be trained for each game, and the system had to be reconfigured slightly as the games have different board sizes.

The ideas of AlphaGo and its derivatives can also be applied to video games. MuZero, from more or less the same team at DeepMind, combines the algorithm from AlphaZero with learning a forward model and applies this algorithm to Atari games as well as Chess, Shogi, and Go [1003]. One perspective on the resounding success of this family of algorithms is that tree search and reinforcement learning (and sometimes supervised learning) are highly complementary, and if you find a way to combine them effectively for your gameplaying problem you will probably do very well.

5.7 Learning Beyond Policies

There are various machine learning methods that are not directly learning policies, but learning models that are useful for creating and learning policies. Such models can also be used for other downstream tasks beyond gameplaying such as content generation and player modeling. Here, we will specifically look at learning representations of the state space, and learning forward models or world models.

5.7.1 Learning Representations

Pixel inputs are in some sense the "native" representation for most video games, but it is also a very high-dimensional representation, meaning that large neural networks need to be trained. This is an important challenge because of the high computational cost and memory requirements for such networks, which make them unsuitable for many game applications. For example, to deliver real-time performance in a game that operates at 60 frames per second, using the same CPU as is executing the game loop, a neural network controlling player character actions will often have only a few milliseconds for inference. (On CPU, as the GPU is typically busy with rendering graphics, or not present if this is done on a server.) In practice, this typically means a network of a few million parameters at most. Another challenge is that not all methods work well for training such large networks; in particular, evolutionary methods and related stochastic search methods perform less well in large parameter spaces.

One way of dealing with such challenges is to use some form of dimensionality reduction on the pixel input data. For example, one could simply use an autoencoder. The derived latent vector z of the autoencoder (see Sect. 3.6.2) would constitute a compressed representation of the visual information, which can be used as input for a smaller neural network that implements the actual policy. The autoencoder would learn to compress the visuals of the scenes the agent encounters into an encoding that captures the relevant differences between those visuals. The problem then is which data to train on. If you train only on the first level of a game, the autoencoder will learn an encoding that represents the various scenes from that level well. However, scenes from the next level are out-of-distribution and might not be represented well.

One way of dealing with this is to periodically retrain the autoencoder as new scenes or areas in the game are explored. Alvernaz and Togelius used this technique to evolve small networks to play the FPS game *Doom* (id Software, 1993) [20].

But dimensionality reduction techniques do not need to be based on autoencoders, or even on neural networks. Cuccu et al. [252, 253] used an increasing dictionary vector quantizer to learn a small, discrete representation of game states. When a sufficiently novel game state is observed, it is added to a dictionary, and the input to the neural network is a discrete encoding of which game states are observed. Using this encoding, they managed to evolve surprisingly small networks—on the order of a few thousand parameters—that play Atari games well. Such findings raise interesting questions about how large a model needs to be to learn a policy. State-of-the-art Atari-playing networks have millions of parameters, but the vast majority of these might be devoted to visual processing. And if a tiny network can learn to play an Atari game well, maybe the policy does not need to be that complex?

As noted earlier, the availability of diverse game content may enhance the efficacy of AI methods (particularly RL) and improve their generalizability [930, 947]. When it comes to learning representations, however, building such representations often requires access to the internal state of several games, which is not really an option. As a response to this challenge both game prototypes and commercial-standard games have been used to improve the generalizability of the derived representations.

For instance, *Grand Theft Auto 5* (Rockstar, 2013) has been utilized for the collection of annotated data which is achieved by capturing the rendering communication between the graphics hardware and the screen buffer [1123] or via in-game modifications allowing for infrared vision [409]. An approximation of the internal game state information was also made possible in *Counter-Strike: Global Offensive* (Valve, 2014) [857] by probing the machine's memory.

Instead of reverse-engineering a game engine, an alternative approach is to capture high-level game state information—such as player actions—through raw inputs obtained from the machine's I/O devices, as done recently in [925]. The BehAVE framework introduced in that study allows the transferability of high-level gameplaying behaviors (i.e., navigation, panning, weapon usage) across 25 commercial FPS games, but also *surprisingly* in between *Minecraft* (Mojang, 2011) and FPS games. BehAVE does not require access to the game engine and attempts to align video encodings of similar player behavior. Player actions are provided as textual descriptions (e.g., move forward or aim gun) which feed a pretrained text encoder (e.g., from the GPT family). The derived representations of BehAVE showcase very good zero-shot gameplaying performance across games; we thus expect similar representation learning methods to shine when offered more computational resources in the future.

5.7.2 *Learning Forward Models*

As discussed in this chapter and the previous, the availability of a fast and accurate forward model is of crucial importance for gameplaying AI. Unfortunately, most video games do not feature a fast forward model and the architecture of standard game engines is anything but friendly to forward models. This raises the questions whether we could simply learn a forward model from the game, and use that model for training and/or planning.

The last few years have seen a number of interesting advances in learning forward models, or also known as **world models**. A key architecture was proposed by Ha and Schmidhuber [420]. This architecture uses variational autoencoders to learn to encode pixel information, similar to Alvernaz' and Togelius' work [20] on *Doom* (id Software, 1993) discussed above. It then uses a recurrent network architecture, in particular an LSTM, to learn to predict the next state in encoding space based on the previous state and the action taken at that state. While the LSTM could technically have been applied to predict the full pixel space, that would have been far too high-dimensional, and probably individual pixels are not the most meaningful level at which to predict. However, predicting in encoding space works relatively well, and the authors apply this technique both to a 2D racing game and to the 3D FPS *Doom*, showing its versatility. Crucially, they also show that they can use reinforcement learning to train a controller using only the learned model for training.

Learning forward models is challenging because the learned simulator only works well for a limited number of timesteps. As the next timestep is fed with the prediction from the previous timestep and so on, small prediction errors accumulate and the simulation eventually breaks down. This can take the form of improbable predictions of impossible game states or simply just visual noise, and renders the forward model useless after some point when it departs from reality too sharply.

The Dreamer family of methods represent advances on the basic world models idea, with each iteration providing better performance. At of the time of writing, the state of the art is *DreamerV3* [422]. Unlike the world modeling approach of Ha and Schmidhuber, *DreamerV3* encodes the observations into a discrete format. It also includes an integrated actor-critic reinforcement learning algorithm, which learns policies in the discrete embedding space. The result is a very capable combination of forward model and reinforcement learning algorithm that has been shown to learn good policies in a short time on a range of benchmark games.

Ultimately, learning a forward model means learning a game. A couple of recent papers have made great advances in learning the dynamics of the game based on pixel inputs and actions. These papers do not directly address the use of reinforcement learning, but simply try to simulate the game engine, and are often referred to as *neural game engines* [52, 1199]. The most effective approach is based on diffusion models. Such neural game engines might prove to be very impactful, especially if authoring can be done by searching spaces of learned games, but they currently suffer from issues such as tracking hidden states, among other things.

Finally, one of the most ambitious attempts at learning forward models is the *Genie* system [138]. *Genie* attempts to learn complete games from videos only, without associated action inputs. While this may initially seem impossible, the system is able to learn latent actions from videos that make it possible to associate certain inputs with certain transformations. This is done with a latent diffusion model, which uses diffusion in the latent space of an autoencoder that encodes and produces raw pixels. The trained system can take a short video as input and provide a playable "game", or at least an interactive experience where you can control a character for a while. In its current state, the system produces experiences that cohere for just a few seconds, is extremely computationally expensive, and is limited to 2D platform-like games. At the time of writing, *Genie 2* was just announced with improved visual quality, coherence and frame rate, but no details are available on that system yet. While *Genie* is still some way from being able to generate complete games or even coherent worlds in real-time, it suggests a future with game engines that could be programmed by demonstration or prompt.

5.8 Further Reading

We refer interested readers in aspects of behavior authoring to the several papers, resources and textbooks such as [706, 767, 950, 951]. Similarly the sections dedicated to search, supervised and reinforcement learning (and their corresponding sections in Part I) provide numerous reading resources. Chimeric methods such as dynamic scripting, AlphaGo, AlphaZero and so on, have their own dedicated articles that we cite in Sect. 5.6. If you are interested in learning aspects of the game state that are not linked to a policy we recommend a few indicative articles on learning representations [20, 857, 925] and learning forward models [52, 420, 422]; more papers are discussed in the corresponding sections. The interested reader in applications of neuroevolution to games is referred to a rather recent survey paper [946].

5.9 Summary

This chapter detailed the family of **methods** used for building AI agents that play games. The methods covered in this chapter focused on behavior authoring, planning-based methods (such as tree-search and evolutionary planning), classic and deep reinforcement learning, supervised learning, and chimeric approaches for playing games. In a sense, we tailored most of the methods outlined in Chap. 3 to the task of gameplaying and we also included a few methods that are are tied to gameplaying and were not covered in the background part of the book. Those include methods that learn beyond policies such as representations and forward models. The next, and final, chapter of this part focuses on a number of AI gameplaying examples across game genres.

Chapter 6
Gameplaying AI by Game Genre

Abstract Different game genres differ in what challenges they pose to a human or machine game player. Some games require long-term planning, others complex visual processing, fast reactions, social and emotional modeling, judicious resource balancing, or the ability to predict complex environment dynamics. This chapter inventories a number of popular game genres, discusses their particular challenges, and gives examples of deployed AI as well as research projects on gameplaying in each genre. The genres are board games, card games, real-time 2D and 3D video games, dungeon crawlers, puzzle games, strategy games, text games, and social simulation games.

Different games pose different challenges for AI playing, in the same way they pose different challenges for human playing. Not only are there differences in what kind of access the AI player has to the games, but there are also different between game types. A policy for playing Chess is unlikely to be proficient at playing the games in the *Grand Theft Auto* series (Rockstar Games, 1997–2021), even if it could somehow be made to use the same inputs and outputs.

The last chapter of the gameplaying part of this book divides up the space of games by game genre, and discusses how AI methods can be applied in various types of games. For each game genre we discuss which cognitive, perceptual, behavioral and kinesthetic challenges games of that genre pose. We then give an overview of how AI methods have been used to play that particular game genre. The chapter contains examples from the literature, from published games and extended examples from academic research, giving some detail about particular implementations. We also introduce several commonly used game-based frameworks and competitions for testing AI gameplaying algorithms. Once again it is important to note that the list is not inclusive of all possible game genres AI can play as a player or non-player character; the selection is made on the basis of popularity of game genres and the available published work on AI for playing games in each genre. The division into genres might also strike some as somewhat arbitrary, as it must be given that there does not exist a generally agreed-upon taxonomy of games into genres.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_6

6.1 Board Games

As discussed in Chap. 1, the earliest work on AI for playing games was done in classic board games. For a long time that was the only way in which AI was applied to playing games. In particular, Chess was so commonly used for AI research that it was called the "drosophila of artificial intelligence" [324], alluding to the use of the common fruit fly as a model organism in genetics research. The reasons for this seem to have been that board games were simple to implement on the limited computer hardware available in the early days of AI research, and that these games were seen to require something akin to "pure thought". What a game such as Chess or Go does require is **adversarial planning**. Classic board games typically place no demand at all on perception, reactions, motor skills or estimation of continuous movements, meaning that their skill demands are particularly narrow, especially compared to most video games.

6.1.1 Classic Board Games

Most board games have very simple discrete state representations and deterministic forward models—the full state of the game can often be represented in less than 100 bytes, and calculating the next state of the game is as simple as applying a small set of rules—and reasonably small branching factors. This makes it very easy to apply **tree search**, and almost all successful board game-playing agents use some kind of tree search algorithm. As discussed in the sections on tree search in Part I, the Minimax algorithm was originally invented in the context of playing Chess. Decades of research concentrated on playing Chess with a lesser amount of research on Checkers and Go, led to a number of algorithmic advances that improved the performance of the Minimax algorithm on some particular board game. Many of these have limited applicability outside of the particular game they were developed on, and it would take us too far to go into these algorithmic variations here. A good overview of advances in Chess playing is given in the main paper about *Deep Blue* [168].

In Checkers the reigning human champion was beaten by the Chinook software in 1994 [994] and the game was *solved* in 2007, meaning that the optimal set of moves for both players was found (it is a draw if you play optimally) [993]; in Chess, Garry Kasparov was famously beaten by *Deep Blue* in 1997 [168]. It took until 2016 for Google DeepMind to beat a human Go champion with their AlphaGo software [1040], mainly. While Chess and Checkers can be played effectively with some variation of the Minimax algorithm combined with relatively shallow state evaluations, the larger branching factor of Go necessitated and spurred the development of MCTS [137], among other innovations such as the coupling of MCTS with DNNs as state evaluation functions.

While MCTS can be utilized to play board games without a state evaluation function, supplementing that algorithm with state and action evaluation functions can massively enhance the performance. This is evidenced in AlphaGo, which uses deep neural networks for state and action evaluation. On the other hand, when using some version of Minimax it is necessary to use state evaluation functions as all interesting board games (i.e., more complex than Tic-Tac-Toe) have too large state spaces to be searched until the end of the game in acceptable time. These evaluation functions can be manually constructed, but in general it is a very good idea to use some form of learning algorithm to learn their parameters (even though the structure of the function is specified by the algorithm designer). As discussed in earlier chapters, Samuel was the first to use a form of reinforcement learning to learn a state evaluation function in a board game [987], and Tesauro later used TD learning to very good effect in Backgammon [1134]. Evolutionary computation can also be used to learn evaluation functions, for example, Pollack and Blair showed that co-evolution could perform well on Backgammon using a very similar setup to Tesauro [895]. Noteworthy examples of strong board game players based on evolved evaluation functions are *Blondie24* [344] and *Blondie25* [345], a Checkers- and a Chess-playing program respectively. The evaluation functions were based on five-layered deep convolutional networks, and *Blondie25* in particular performed well against very strong Chess players.

Given the simplicity of using tree search for board game playing, it is not surprising that every approach we have discussed so far builds on one tree search algorithm or another. However, it is possible to play board games without forward models—usually with results that are "interesting" rather than good. For example, Stanley and Miikkulainen developed a "roving eye" approach to playing Go, where an evolved neural network self-directedly scans the Go board and decides where to place the next piece [1082]. Relatedly, it is reportedly possible for the position evaluation network of AlphaGo to play a high-quality game of Go on its own, though it naturally plays stronger if combined with search. Given enough data and compute, transformer-based architectures can also be trained to play a good game of Chess—though, again, not as good as if it also had access to search [776, 819].

In addition to work on playing specific classical board games, classic board games have been used for research into general game playing. This line of work seemingly originated with Pell's work on Chess-like games, where he developed a domain-specific language that could represent variations on Chess, and used tree search to play these games [860, 861]. This work informed the **General Game Playing system** and competition, where competitors submitted agents that could play games specified in a game description language capable of representing many classic board games [374, 667]. The language used for that competition is relatively low-level and game descriptions tend to be very long, which limits the set of games available.

The **Ludii** General Game System provides a more modern language and environment for general game playing, generation, and design research [881]. Ludii was originally developed as part of the Digital Ludeme Project, which aimed to model the world's traditional strategy games in a single, playable digital database [135].

Games within Ludii are defined as structures of ludemes—high-level, easily understandable game related information—allowing for concise, easily modifiable and human-interpretable game descriptions [854]. The ease with which new games can be written for Ludii has led to a significant community of board game designers, researchers and historians who have contributed over 1,300 unique board game descriptions,[1] encompassing traditional board games from as early as 3,500 BCE all the way to modern games of the 21st century. Ludii supports a wide range of board game properties, including between 1 and 16 players, hidden information, stochastic elements, alternating and simultaneous moves, along with a near infinite variety of board shapes and rule combinations [883, 1065]. Ludii's game library includes everything from well-known games such as Checkers, Go, and Hex to lesser-known titles such as Hnefatafl, Gurgaldaj, and Ngulungu. The Ludii library and environment has recently been utilized to investigate novel game generation through large language models and evolutionary computation, with the goal of developing a model capable of intelligently mutating and recombining game mechanics to produce innovative designs [1148].

This extensive collection of game types and development options makes Ludii one of the largest and most flexible general board game environments available for AI research. Ludii offers a fully functional API that provides external agents with access to any game's complete state and action space, enabling tree search algorithms like MCTS to operate effectively. In addition, Ludii includes a range of built-in general game-playing agents utilizing techniques such as UCT, AlphaBeta, MAST [336] and GRAVE [186]. More recent research has explored automated recognition of spatial-state features to facilitate knowledge transfer across similar games, enabling applications in transfer learning [1064, 1066]. Further studies have examined the development of hyper-agents or portfolio agents that identify the most effective heuristics of agents for a given game by analyzing the ludemes in its description [1085]. While these methods have yielded promising results for specific game types, achieving human-level performance across Ludii's diverse array of game mechanics and strategies remains a significant challenge for general game AI research [882].

6.1.2 Modern Board Games

While classic board games such as Go and Chess have existed for hundreds or even thousands of years, the past few decades have seen a rejuvenation of board game design. Many of the more recently designed board games mix up the formula of classic board games with design thinking from other game genres. A good example is *Ticket to Ride* (Days of Wonder, 2004), which is a board game that includes elements of card games, such as variable numbers of players, hidden information and stochasticity (in the draw of the cards). For these reasons, it is hard to construct well-performing AI players based on standard tree-search methods; the best known

[1] https://github.com/Ludeme/Ludii.

agents include substantial domain knowledge yet perform poorly compared to human players [268]. Other notable examples include the work of Sfikas and Liapis [1014] on the use of rolling horizon evolution for playing *Pandemic* (Z-Man Games, 2008). Bravi et al. [120] have developed a series of evolutionary and tree search methods for playing the popular board game *Splendor* (Space Cowboys, 2014). Modern board gameplaying AI research has also focused on variants of *7 Wonders* (Repos Production, 2010), a perfect information, non-deterministic board game with no social player social mechanics such as bluffing. In a recent paper [850], an agent learned to play the game without human supervision using a combination of MCTS and a transformer. The resulting agent plays the *7 Wonders Duel* variant at expert level and won 2 out of 3 live matches against top players of the game selected from the *Board Game Arena*.[2]

A modern board game that has been relatively well explored is *Settlers of Catan* (Kosmos, 1995), a popular multi-player game with a relatively complex set of interlocking scoring and progress mechanics, and both stochasticity and hidden information. Early work explored using MCTS with appropriate heuristics for such challenging conditions [1121], and there has been more recent work applying methods such as Partially Observable Monte-Carlo Planning [287] and heuristically derived strategies [412].

Creating generic well-performing agents for modern board games pose many interesting research challenges. Luckily, there is suitable research infrastructure. Most notably, the Tabletop Games Framework (TAG) provides implementations of a dozen or so modern board games (some are modern classics) along with a domain-specific language for implementing others, and a set of powerful gameplaying algorithms [50, 360].

6.2 Card Games

Card games are games centered on one or several decks of cards; these might or might not be the standard 52-card French deck which is commonly used in classic card games. Most card games involve players possessing different cards that change ownership between players, or between players, the deck, and various positions on the table such as Poker, *Dixit* (Libellud, 2008) and *Ticket to Ride* (Days of Wonder, 2004). Almost all card games are turn-based, though some games feature real-time elements such as *Set* (Set Enterprises, 1991) and even the single-player *Solitaire* card games. There is always a stochastic element due to the order cards are drawn; sometimes stochasticity entails the use of dice or other explicit randomizers. Another important element of most card games is that some cards are visible to the player who possesses them but not to other players. Therefore, almost all card games feature a large degree of **hidden information**. In fact, card games are perhaps the type of

[2] https://boardgamearena.com/.

games where hidden information most defines gameplay. High-level card game play almost always requires some means of modeling this hidden information.

6.2.1 Classic Card Games

As with board games, early work focused on card games that have been around for a long time, in some cases hundreds of years in different variations. Such games typically use the standard 52-card French deck.

Perhaps the most attention from within AI research has been given to the classic card game **Poker**, which is currently very popular in its *Texas hold 'em* variety. The rules are relatively simple: the player with the best hand of cards wins. Between the rounds, the player can exchange a number of cards for fresh cards drawn from the deck. If there was perfect information, i.e., all players could see each others' hands, this would be an uninteresting game that could be played according to a lookup table. What makes Texas hold 'em—and other Poker variants—challenging and interesting is that each player does not know what cards the other players have. The cognitive challenges of playing these games involve acting in the absence of information, which implies inferring the true game state from incomplete evidence, and potentially affecting other players' perception of the true game state. In other words, a game of Poker is largely about guessing and bluffing.

A key advance in playing Poker and similar games is the Counterfactual Regret Minimization (CFR) algorithm [1318]. In CFR, algorithms learn by self-play in a similar fashion to how temporal difference learning and other reinforcement learning algorithms have been used in perfect information games like Backgammon and Checkers. The basic principle is that after every action, when some hidden state has been revealed, it computes the alternative reward of all other actions that could have been taken, given the newly revealed information. The difference between the reward attained from the action that was actually taken and the best action that could have been taken is called the *regret*. The policy is then adjusted so as to minimize the regret. This is done iteratively, slowly converging on a policy that is optimal in the sense that it loses as little as possible over a large number of games. However, for games as complex as *Texas hold 'em*, simplifications have to be applied in order to use the CFR algorithm in practice.

DeepStack is a relatively recent agent and algorithm that has reached world-class performances in *Texas hold 'em* [779]. Like CFR, DeepStack uses self-play and recursive reasoning to learn a policy. However, it does not compute an explicit strategy before play. Instead, it uses tree search in combination with a state value approximation to select actions at each turn. In this sense, it is more like the heuristic search of AlphaGo than like the reinforcement-learned policy of TD-gammon. Similarly, the *Libratus* program by Brown and Sandholm [132] uses a version of Counterfactual Regret Minimization to play superhuman poker in two-player limited games.

Another classic card game that has attracted attention from the AI research community is Contract Bridge, often simply called **Bridge**. In its full form, Bridge is

an extremely complex game, posing the player a number of quite disparate challenges [603]. At its core, Bridge is a trick-taking game, where the player who plays the highest card wins the "trick", or set of cards that was played recently. But Bridge is a collaborative-competitive game played between two pairs of players; each player only observes their own hand and needs to collaborate with their partner to compete with another pair during each trick. Overlaid on top of the trick-taking game is the auction. By transmitting messages under a publicly known set of conventions, each player communicates with their partner—with the opponents listening—about the content of their hand in order to reach a suitable contract which dictates the number of tricks to be made and the trump suit.

Given the complexity of Bridge—the description above barely scratches the surface—it is not surprising that attempts at playing Bridge with AI methods cannot yet competently tackle all aspects of the game. However, there has been much progress. In particular, a team from the AI startup NukkAI entered their NooK system into a Bridge competition in 2022, and won over eight world champions [1073]. Nook is based on the $\alpha\mu$ algorithm, a version of MCTS optimized for imperfect information games [187]. Additionally, Nook includes a model of opponent's cards based on observations of which cards they have played so far; in other words it is a hybrid agent. While exhibiting very strong gameplaying performance, the 2022 competition did not feature bidding, a key component of Bridge. The NukkAI team have recently proposed a way of learning decision models for bidding strategy, based on supervised relational learning [1213].

6.2.2 Collectible Card Games

New card games keep being developed and played, and many video games include novel card games as part of their mechanics, such as the *Gwent* card game that first appeared as part of *The Witcher III* (CD Projekt, 2015) and later released as a game in its own right, namely, *Gwent: The Witcher Card Game* (CD Projekt, 2018). There are also games that incorporate elements of card games into a wider set of mechanics, such as *Balatro* (Playstack, 2024), a game which has been characterized as a mix of a roguelike and a card game.

One particular type of card game that has become enormously popular is the collectible card game, abbreviated CCG. CCGs typically feature a very large number of cards. *Magic: the Gathering* (MtG) (Wizards of the Coast, 1993), the game that pioneered the genre in 1993, boasts around 27,000 unique cards. Most players only own a small fraction of these. Most CCGs are two-player competitive zero-sum games, where players use the cards to attack and defend; each card features its own statistics, and often unique abilities to boot. Crucially, each game features a much smaller deck of cards for each player, usually around a few dozen cards, of which only a few cards are in play (on hand or on table) at any given tame. There are thus at least two distinct playing tasks in a CCG: form a deck from all available cards, and actually play the game. Both tasks have considerable skill depth, partly based on the

Fig. 6.1 A screenshot from *Hearthstone* (Blizzard Entertainment, 2014) displaying a number of different creature or spell cards available in the game. Screenshot taken from authors' own gameplay

sheer number of different cards available and their vastly different effects, requiring encyclopedic knowledge for high-level CCG playing.

The collectible card game which has been used most in AI research, partly because of the availability of an efficient and accessible simulator, is probably *Hearthstone* (Blizzard Entertainment, 2014) [470]; see Fig. 6.1. *Hearthstone* is a CCG in the tradition of *Magic: The Gathering*, but with somewhat simpler rules and which can only be played on computers. A game of *Hearthstone* takes place between two players, with each player having a deck of 30 cards. Each player has a handful of cards in hand hidden from the other player; at each turn draws a new card and has the option of playing one or more cards. The hundreds of different cards in the game, the possibility of choosing to take multiple actions each turn, the long time taken to play a game (i.e., around 20–30 turns), the presence of stochasticity and of course the hidden information (mainly what cards are in the opponent's hand) conspire to make *Hearthstone* a hard game to play for both humans and machines. In the next few paragraphs we will discuss work on both *Hearthstone* and *Magic: the Gathering*, given their extensive similarities, and that playing them can be broken down into roughly the same sub-tasks.

Perhaps the easiest way to play a CCG like *Hearthstone* or *MtG* is to simply ignore the hidden information and play each turn in a greedy fashion, i.e., search the space of possible actions within a single turn and choose the one that optimizes some criterion such as health point advantage at the end of that turn, given the available information only. Agents that implement such greedy policies are included

with some open source *Hearthstone* simulators, such as *Metastone*.[3] Standard tree search algorithms such as Minimax or MCTS are generally ineffective here (as in many other card games) because of the very high degree of hidden information. Despite the challenges, one might construct high-performing agents if hand-coded domain knowledge is integrated to the agent's gameplaying algorithm. For example, expert knowledge considered during the selection phase of MCTS can boost the performance of the algorithm in this game [988]. Another approach that exploits domain knowledge is to build an **ontology** of cards and search in an abstract symbolic space [1089].

Unlike in Poker, where the player has no control over what cards they are dealt, in *Hearthstone* and *MtG* the player can also construct a deck with which to play the game. This adds another level of challenge to playing the game: in addition to choosing what action(s) to take at each turn, the successful player must also construct what allows them to implement their strategy. The composition of the deck effectively constrains what strategy can be chosen, and then implemented tactically through action selection. While these two levels interplay—a strong player takes the composition of the deck and the strategy it affords into account when choosing a move, and vice versa—it is also true that the problems of deck building and action selection can to some extent be treated separately, and implemented in different agents.

While fewer papers focus on AI performing action selection, there are quite a few papers on deck-building in both *Hearthstone* and *MtG*. One approach to deck building is to use evolutionary computation. The deck is seen as the genome, and the fitness function involves simple heuristic agents using the deck for playing [94, 368, 470, 586, 759, 1214, 1264]. Using a fixed set of starter decks, new decks can be evolved to beat those decks, but also show some generality by playing well against other decks. New decks can also be evolved that beat the previously evolved decks, and these also beat the original decks, showing some transitivity of deck strength [86]. Still, the deck space is vast, and there are many possible good decks. To find a larger set of decks, one can use a quality-diversity algorithm such as MAP-Elites to find decks that play well while having significantly different composition [346]. Given the high computational cost of evaluating a deck—i.e., playing multiple simulated matches against opponents with other decks—one approach is to train a surrogate model that learns the mapping from a deck to a fitness value [1315].

One can also learn deck building in a *supervised learning* manner. While building a complete deck at once is difficult due to the vast amount of cards to choose from, sequential deck building settings, such as *drafting*, break the task into smaller parts. When drafting, decks are built by iteratively selecting a single card out of a limited set of options, thus deconstructing the selection of a set of cards into a series of discrete decisions. Datasets of human drafting selections are publicly available, enabling the use of supervised learning for modeling the selections [1241]. Alternatively, *contrastive learning* approaches can be employed to compare cards rather than directly predicting them [80, 81]. Another recent approach is to view

[3] https://github.com/demilich1/metastone.

deck building and playing as a unified task and train RL agents to both draft and play using the game's outcome as the reward. RL approaches as those have already been tested in games such as *Legends of Code and Magic* (3DO, 2001) [1259] and *Hearthstone* [1260].

The enormously popular *Pokemon* media franchise includes a collectible card game, along with multiple video games, movies, toys etc. Interestingly, most of the video games also feature prominent elements of CCGs. *Pokemon* battles, where two players fight in a turn-based manner by fielding a number of *Pokemon* creatures and attacking the other player's *Pokemons*, have strong similarities to games such as *Hearthstone*. Other parts of the game heavily revolve around collecting *Pokemons*, of which a small subset must be selected for each battle, akin to deck-building in a CCG. Several simulators of *Pokemon* battles have been built with the express purpose of allowing AI versus AI battles, starting with the *Showdown AI Competition* and its associated simulator [616]. Such simulators have been used to compare various algorithms, including planning algorithms and reinforcement learning [487, 1043].

At the time of writing this book a new CCG AI competition has emerged, showcasing the ongoing academic interest in this space. The *Tales of Tribute AI Competition* [585] is based on the two-player deck-building card game *Tales of Tribute* released as an expansion of *The Elder Scrolls Online* (Bethesda Softworks, 2014). Any AI method employed to play *Tales of Tribute* is tasked to handle the characteristics of CCG games (non-determinism, hidden information, large branching factor), but it is also required to handle long-term planning. The winning agent of the most recent version of the competition, in 2024, employed a chimeric approach that optimizes the hyperparamaters of MCTS via evolutionary computation.

6.2.3 Other Card Games

Deck-building games are games which borrow the deck-building aspect from CCGs, but differ in other aspects. *Dominion* (Rio Grande Games, 2008) is a well-known deck-building game, where a key mechanic is building your deck through buying cards from a common supply. Just like evolutionary computation was found to be useful for deck-building in CCGs, it was found that you could evolve the types of cards that were available for play so as to balance the game between different playing styles [688].

While hidden information is important to almost every card game, *what hidden* information is hidden, and *from whom* it is hidden, varies. *Hanabi* (R&R Games, 2010) is a cooperative card game which upends several card game conventions. In *Hanabi*, the hands of cards are held with faces towards the other players, so that each player does not know their own cards, but everyone else's card. It is also played collaboratively, where players work together to achieve a joint high score. This means that it becomes crucial to build a model of which cards you have yourself.

Communication is also crucial; there are limited possibilities for communication, and players must use this limited repertoire to effectively let each other know which cards they have. *Hanabi* has been extensively used as an AI research problem after the publication of an influential article arguing that the game has many interesting challenges due to its unusual design [55]. RL methods are able to play the game extremely well, but only if the policy is playing with itself (i.e., against another players with the same policy) [343, 484]. As assuming an identical policy trivializes the communication problem, this is not very applicable to playing *Hanabi* with others [171]. Separately, a competition was held focusing on playing *Hanabi* with unseen players, requiring a degree of generality, adaptability, or at least robustness from policies [1230]. Canaan et al. [174] devised a way of finding such policies through a form of genetic programming. Interestingly, this can be also combined with quality-diversity methods such as MAP-Elites. One can first evolve a large number policies (that differ amongst relevant playstyle dimensions) and then use a learned function to rapidly adapt to a new collaborator by selecting an existing policy that works well with the identified playstyle of the collaborator [172, 175]. Using a repository of pre-trained policies to enable fast adaptation is a general method that is likely to be useful in a variety of game genres, and for both competitive and cooperative settings.

6.3 Real-time 2D Video Games

Video games come in many shapes, presenting a wide variety of challenges for humans and computers playing them. With the remainder of this chapter focused on video games, we will try to organize them roughly by the types of challenges they present to us and to AI algorithms. This section will focus on 2D real-time video games. These are games where real-time responses are typically important. Some of them have full information (the whole game state is available to the player), while most of them do not. But in either case, there is typically some navigation challenge in the game. As the perspective of the game is not from the player character, but typically from the side or from above, this navigation challenge requires a different kind of mapping than if it were from a first-person view.

In the section after this, we will discuss 3D video games, which generally have different characteristics. It is important to note that this is far from a perfect distinction; there any many 2D video games that do not have some or all of the characteristics mentioned above, and 2D and 3D video games overlap and blend into each other. This distinction is simply our attempt at imposing some kind of structure to the discussion, and the other structures we could think of were worse.

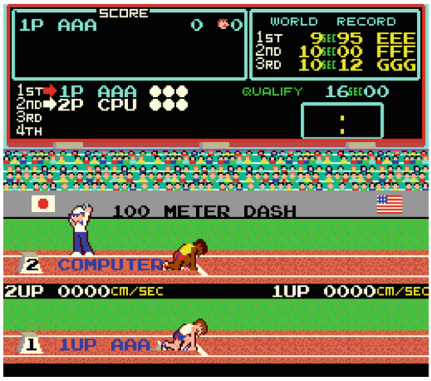
6.3.1 Classic Arcade Games

Classic arcade games, of the type found in late 1970s and early 1980s arcade cabinets, home video game consoles and home computers, have been widely used as AI benchmarks within the last decade. Representative platforms for this type of games are the Atari 2600, Nintendo NES, Commodore 64 and ZX Spectrum. Most classic arcade games are characterized by movement in a two-dimensional space (sometimes represented isometrically to provide the illusion of three-dimensional movement), heavy use of graphical logics (where game rules are triggered by intersection of sprites or images), continuous-time progression, and either continuous-space or discrete-space movement.

The cognitive challenges of playing such games vary. Most games require fast reactions and precise timing. A few games, in particular early sports games such as *Track & Field* (Konami, 1983) and *Decathlon* (Activision, 1983), rely almost exclusively on speed and reactions (see Fig. 6.2a). Very many games require prioritization of several co-occurring events, testing the player's ability to predict the behavior or trajectory of other entities in the game. This challenge is explicit in e.g., *Tapper* (Bally Midway, 1983)—see Fig. 6.2b—but also in platform games such as *Super Mario Bros* (Nintendo, 1985), shooting galleries such as *Duck Hunt* (Nintendo, 1984) or *Missile Command* (Atari Inc., 1980) and scrolling shooters such as *Defender* (Williams Electronics–Taito, 1981) or *R-type* (Irem, 1987). Another common requirement is navigating mazes or other complex environments, as exemplified most clearly by games such as *Pac-Man* (Namco, 1980), *Ms Pac-Man* (Namco, 1982), *Frogger* (Sega, 1981) and *Boulder Dash* (First Star Software, 1984), but also in many platform games. Some games, such as *Montezuma's Revenge* (Parker Brothers, 1984), require long-term planning involving the memorization of temporarily unobservable game states. Some games feature incomplete information and stochasticity, others are completely deterministic and fully observable.

6.3.2 Pac-Man and Ms Pac-Man

Various versions and clones of the classic *Pac-Man* (Namco, 1980) game have been frequently used for both research and teaching, due to the depth of challenge coupled with conceptual simplicity and ease of implementation. As seen in Chap. 3, the differences between the original *Pac-Man* and its successor *Ms Pac-Man* (Namco, 1982) may seem minor but are actually fundamental; the most important is that one of the ghosts in *Ms Pac-Man* has non-deterministic behavior, making it impossible to learn a fixed sequence of actions as a solution to the game. The appeal of this game to the research community is evidenced by a relatively recent survey covering over 20 years of active AI research using these two games as testbeds [958]. There is also



(a) *Track & Field* (Konami, 1983) is a game about athletics. The game screenshot depicts the start of the 100 m dash. Image obtained from Wikipedia.

(b) In *Tapper* (Bally Midway, 1983) the player controls a bartender who serves drinks to customers. Image obtained from Wikipedia.

Fig. 6.2 *Track & Field* (Konami, 1983), *Tapper* (Bally Midway, 1983), and most classic arcade games require rapid reactions and precision

an informative video on YouTube video[4] about the importance of Pac-Man in game AI research.

Koza, in his 1992 book [588], was among the first researchers to use Pac-Man as an AI test-bed, specifically assessing the capacity of *genetic programming* to evolve agents that play this game well. Since then, several frameworks have been developed for Pac-Man-based experimentation, some tied to competitions. The Pac-Man screen capture competition was based around the Microsoft Revenge of Arcade version of the original game, and does not provide a forward model nor facilities for speeding up the game [672]. The Ms Pac-Man vs. Ghost Team competition framework is written in Java and includes both a forward model and ability to speed up the game significantly; it also includes an interface for controlling the ghost team rather than Ms Pac-Man, the player character [959]. The Atari 2600 version of *Ms Pac-Man* is available as part of the ALE [75] framework.

As expected, the performance of AI players varies depending on the availability of a forward model, which allows the simulation of ghost behavior. The screen capture-based competition, which does not offer a forward model, was dominated by heuristic approaches. Some of them involved pathfinding in the maze without taking ghost movement into account and perform at the level of beginner human players [672]. Interestingly, even searching just one step ahead, and using a state evaluator based on an evolved neural network, can be an effective method for playing the game [671]. Of course, searching deeper than a single ply yields additional benefits; however, the stochasticity introduced in *Ms Pac-Man* poses challenges even in the presence of a forward model. Nevertheless, MCTS has been shown to work

[4] https://www.youtube.com/watch?t=49&v=w5kFmdkrIuY.

well in this case [863, 986]. Model-free approaches to reinforcement learning have also been used for playing the game with some success [102]. In general, the best competitors in the Ms Pac-Man vs. Ghost Team Competition play at the level of intermediate-skill human players [959].

Eventually, *Ms Pac-Man* was practically solved (reaching the maximum possible score of 999, 990 points) by the Microsoft Maluuba team in 2017 [1208]. The team used an RL technique called *hybrid reward architecture* which decomposes the reward function of the environment into different RL problems (a set of reward functions) that a corresponding number of agents need to solve. Each agent selects its actions by considering the aggregated Q values for each action across all agents.

Pac-Man can also be played for experience rather than performance. In a series of early experiments with a variant of this game, neural networks that controlled the ghosts in a clone of the game were evolved to make the game more entertaining for human players [1271]. The experiment was conceptually based on Malone's definition of fun in games as challenge, curiosity and fantasy dimensions [698] and sought to find ghost behavior that maximized these traits. In particular, the fitness was composed of three factors: (1) the appropriate level of challenge (i.e., when the game is neither too hard nor too easy), (2) the diversity of ghost behavior, and (3) the ghosts' spatial diversity (i.e., when ghosts' behavior is explorative rather than static). The fitness function used to evolve interesting ghost behaviors was cross-validated via user studies [1275].

6.3.3 *Super Mario Bros*

Versions and clones of Nintendo's landmark platformer *Super Mario Bros* (Nintendo, 1985) have been extremely popular for AI research, including research on game playing, content generation and player modeling. A major reason for this research activity is the *Mario AI Competition*, which in various incarnations ran from 2009 to 2012 and included several different tracks focused on playing for performance, playing in a human-like manner and generating levels [539, 1171]. The software framework for that competition, which is still available,[5] is based on *Infinite Mario Bros* (Notch, 2008), a Java-based clone of *Super Mario Bros* (Nintendo, 1985) featuring simple level generation [1159, 1160]. Different versions of the competition software, generally referred to as the Mario AI Framework or Mario AI Benchmark [539, 1171], have since been used in many dozens of research projects. A 10th anniversary edition of the software, which attempts to unify the various branches of the every-popular software, was released in 2019.[6] In the following, we will for simplicity refer to methods for playing various versions of *Super Mario Bros*, *Infinite Mario Bros* or the Mario AI Framework/Benchmark simply as playing "Mario".

[5] http://julian.togelius.com/mariocompetition2009/.

[6] https://github.com/amidos2006/Mario-AI-Framework.

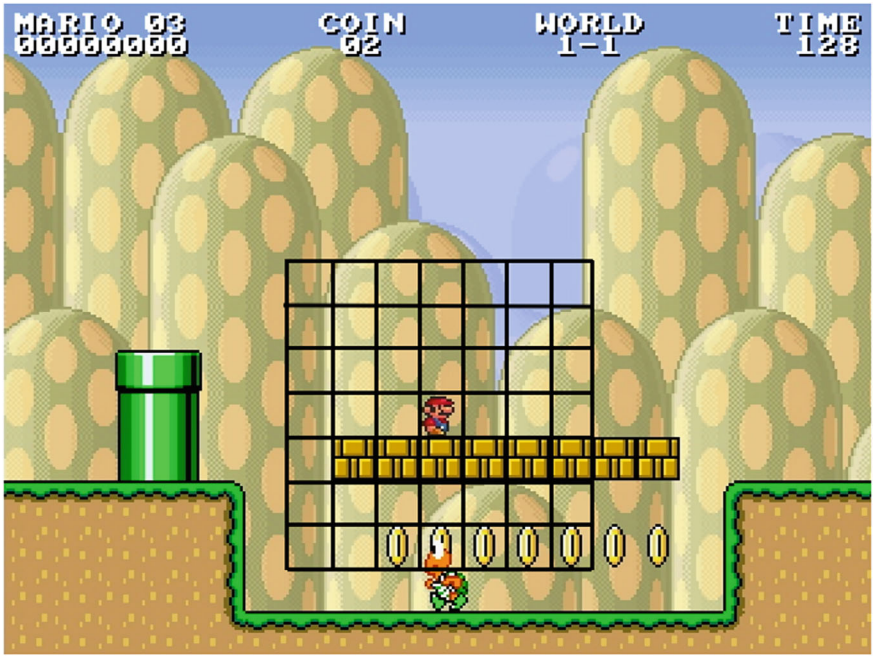


Fig. 6.3 The inputs to the Mario-playing neural network are structured as a Moore neighborhood centered on Mario. Each input is 1 if the corresponding tile is occupied by a tile (such as ground) that Mario cannot pass through, and 0 otherwise. In another version of the experiment, a second set of inputs was added where an input was 1 if there was an enemy at the corresponding tile. Image adopted from [1160]

The first version of the Mario AI framework could be simulated thousands of times faster than real-time, but did not include a forward model. Therefore the first attempts to learn a Mario-playing agent was through learning a function from a state observation directly to Mario actions [1160]. In that project, neural networks were evolved to guide Mario through simple procedurally generated levels. The inputs were the presence or absence of environment features or enemies in a coarse grid centered on Mario, and the outputs were interpreted as the button presses on the Nintendo controller (up, down, left, right). See Fig. 6.3 for an illustration of the state representation. A standard feedforward MLP architecture was used for the neural network, and the fitness function was simply how far the controller was able to progress on each level. Using this setup and a standard evolution strategy, neural networks were evolved that could win some levels but not all, and generally played at the strength of a human beginner.

As with almost every game, having a forward model makes a big difference. The first Mario AI Competition in 2009 was won by Baumgarten, who constructed a forward model for the game by reusing some of the open-source game engine code [1159]. Using this model, Baumgarten constructed an agent based on A* search

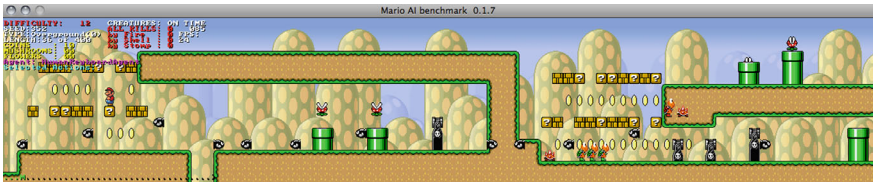


Fig. 6.4 An example level generated for the 2010 Mario AI Competition. Note the overhanging structure in the middle of the screenshot, creating a dead end for Mario; if he chooses to go beneath the overhanging platform, he will need to backtrack to the start of the platform and take the upper route instead after discovering the wall at the end of the structure. Agents based on simple A* search are unable to do this

in state space. At each time frame, the agent searches for the shortest path towards the right edge of the screen, and executes the first action in the resulting plan. As the search utilizes the forward model and therefore takes place in state space rather than just physical space, it can incorporate the predicted movements of the (deterministic) enemies in its planning. This agent was able to finish all the levels used in the 2009 Mario AI Competition, and produces behavior that appears optimal in terms of time to complete levels (it does not focus on collecting coins or killing enemies). The interested reader is referred to Sect. 3.2.2 for an explanation of the algorithm and a figure illustrating its use in Mario.

Given the success of the A*-based agent in the 2009 competition, the next year's edition of the competition updated the level generator so that it generated more challenging levels. Importantly, the new level generator created levels that included "dead ends", structures where Mario can take the wrong path and if so must backtrack to take the other path [539]. See Fig. 6.4 for an example of such a dead end. These structures effectively "trap" agents that rely on simple best-first search, as they end up searching a very large number of paths close to the current position, and time out before finding a path that backtracks all the way to beginning of the structure. The winner of the 2010 Mario AI Competition was instead the REALM agent [101]. This agent uses an evolved rule-based system to decide sub-goals within the current segment of the level, and then navigates to these sub-goals using A*. REALM successfully handled the dead ends that were part of the levels in the 2010 competition.

Other search algorithms beyond A* have been tried for playing Mario, including Monte Carlo tree search [502]. It was found that the standard formulation of MCTS did not perform very well, because the algorithm did not search deep enough and because the way the average reward of a branch is calculated results in risk-averse behavior. However, with certain modifications to remedy these problems, an MCTS variant was found that could play Mario as well as a pure A* algorithm [502]. In a follow-up experiment, noise was added to the Mario AI Benchmark and it was found that MCTS handled this added noise much better than A*, probably because MCTS relies on statistical averaging of the reward whereas A* assumes a deterministic world.

All of the above work has been focused on playing for performance. Work on playing Mario for experience has mostly focused on imitating human playing styles, or otherwise creating agents that play Mario similarly to a human. To further this research, a Turing test track of the Mario AI Competition was created [1026]. In this track, competitors submitted agents, and their performance on various levels was recorded. Videos of the agents playing different levels where shown to human spectators along with videos of other humans playing the same levels, and the spectators were asked to indicate which of the videos were of a human player. Agents were scored based on how often they managed to fool humans, similarly to the setup of the original **Turing test**. The results indicated simple heuristic solutions could be very effective in giving the appearance of human playing style when they included hand-coded routines for such things as sometimes standing still (giving the impression of "thinking about the next action") or occasionally misjudging a jump. Another way of providing human-like behavior is to explicitly **mimic** humans by learning from play traces. Ortega et al. [837] evolved neural networks where the fitness function is based on whether the agent would perform the same action as the human, when faced with the same situation. In a similar effort to create human-like Mario AI players, Munoz et al. [784] used both play traces and information about the position of the player's gaze on the screen (via gaze tracking) as inputs of an ANN, which was trained to approximate which keyboard action is performed at each time step. Their results show promise towards the development of more human-like Mario controllers based on information beyond gameplay data.

6.3.4 The ALE Framework

The Arcade Learning Environment is an environment for general gameplaying research based on an emulation of the classic video game console *Atari 2600* [75].[7] The Atari 2600, introduced in 1977, was the first commercially sucessful home video game console with removable game cartridges, and is enormously important in video game history. By modern standards, it is very simple. It has 128 bytes of RAM, maximum 32 kilobytes of ROM per game, and no screen buffer, posing severe limitations on the type of games that could be implemented on the system [777]. ALE provides an interface for agents to control games via the standard joystick input, but does not provide any processed version of the internal state; instead, it provides the 160×210 pixel screen output to the agent, which will need to parse this visual information somehow. There is a forward model, but it is relatively slow and generally not used.

Some of the early work on ALE used neuroevolution; in particular a study compared several neuroevolution algorithms on 61 Atari games [434]. They found that

[7] While the environment can technically accommodate other emulators as well, the Atari 2600 emulator is the one that has been used in practice, to the point that the ALE framework is sometimes simply referred to as "Atari".

they could use the **NEAT** algorithm [1081] to evolve well performing players for individual games, provided that these networks were given positions of in-game objects as recognized by a computer vision algorithm. The **HyperNEAT** algorithm [1080], an indirect encoding neuroevolution algorithm that can create arbitrarily large networks, was able to learn agents that could play based on the raw pixel inputs, even surpassing human performance in three of the tested games. In that paper, the neuroevolution approaches generally performed much better than the classic reinforcement learning methods tested.

Later, ALE was used in Google DeepMind's research on deep Q-learning, which was reported in a *Nature* paper in 2015 [773]. As detailed in Chap. 3, the study showed that by training a deep neural network (five layers, where the first two are convolutional) with Q-learning augmented with experience replay, human-level playing performance could be reached in 29 out of 49 tested Atari games. Research presented in [773] spurred a flurry of experiments in trying to improve the core deep reinforcement formula. A partial history of those developments is given by Justesen et al. [520]. ALE became a standard benchmark for deep reinforcement learning and it might very well be the case that there are more papers on deep reinforcement learning that use the ALE benchmark than those that don't. Intense competition meant that many research teams proposed solutions specifically tailored to improving performance on ALE. One noteworthy contribution is *Agent 57* [45], which combines meta-learning and exploration based on intrinsic curiosity with deep Q-learning of a recurrent network. Agent 57 was the first algorithm to achieve superhuman gameplay performance on all 57 games in the standard ALE benchmark set.

It is worth noting that almost all of the ALE work focuses on learning neural networks (or occasionally other agent representations) for individual games. That is, the network architecture and input representation is the same across all games, but the parameters (network weights) are learned for a single game and can only play that game. This seems to be at odds with the idea of **general game playing**. One could argue that ALE itself is better suited for research into playing individual games than for research on general game playing, as there is only a limited number of Atari 2600 games, and it is highly non-trivial to create new games for this platform. This makes it possible to tune architectures and even agents to individual games.

However, ALE and its Atari games have been the environment of choice for some recent advances in training single policies that play multiple games proficiently. One of these advances is the **decision transformer**, which is a method for doing offline RL by training a transformer to predict rewards [203]. In brief, a decision transformer takes a multi-step sequence of actions, observations, and rewards as input, and is then predicts the resulting reward in the following step. Once a capable predictor has been learned, it can then be queried for appropriate actions in new situations by simply inputting the desired reward. A team at Google DeepMind showed that a decision transformer could be trained on a number of different Atari games, yielding a single model that performs well on a set of 41 training games. The same transformer model could then be finetuned using very little data to play unseen games well. In concurrent work by another DeepMind team, a decision transformer

was trained to play a number of Atari games and also perform a number of other tasks, including chatting and simulated robot control [930]. Section 3.6.6 details the decision transformer method.

Another efficient method for training a single policy to play multiple Atari games is Scaled Q-learning [593]. The method is based on a refinement of conservative Q-learning [594], a version of deep Q-learning which limits overestimation of rewards. This is combined with large and deep networks, including ResNet blocks. While most of the network is shared between all games, the *head* (the last layer) is different for each game. This enables for the common part of the network to learn a joint policy representation, which is then specialized for each game in the final layer. When pretrained using offline reinforcement learning on traces from a large number of games, it achieved near human performance on 40 Atari games; by later fine-tuning on individual games, performance could be increased further.

While there are many attempts to train ever larger networks to achieve higher score or wider generality on Atari games, there have also been attempts to see what can be done with smaller networks. For example, Cuccu et al. [252] devised a way of playing Atari games with extremely small networks, with as few as six neurons. The trick is using a separate algorithm for processing the input data into a low-dimensional space, so that a very small network can process it. This approach makes it possible to study policy somewhat divorced from visual processing, and raises some questions about the depth of strategy needed to play many of the Atari games in the ALE benchmark.

6.3.5 General Video Game AI

The General Video Game AI (GVGAI) competition was a game-based AI competition that ran, in various incarnations, from 2014 to 2021 [870]. It was designed partly as a response to a trend seen in many game-based AI competitions, e.g., those organized at the CIG and AIIDE conferences, that submissions were getting increasingly game-specific by incorporating more and more domain knowledge. A central idea of GVGAI is that submissions to the competition are tested on *unseen* games, i.e., games that have not been released to the competitors before and which are therefore impossible to tailor the submissions to. At the time of writing, the GVGAI repository includes around 100 games.

To simplify the development of games in GVGAI, a language called the Video Game Description Language (VGDL) was developed [303, 997]. This language allows for concise specification of games in a Python-like syntax; a typical game description is 20–30 lines, with levels specified in separate files. Given the underlying assumptions of 2D movement and graphical logics, many of the games in the GVGAI corpus are remakes of (or inspired by) classic arcade games such as *Space Invaders* (Taito, 1978) (see Fig. 6.5a), *Frogger* (Sega, 1981) (see Fig. 6.5b), or *Boulder Dash* (First Star Software, 1984), but some are versions of modern indie games such as *A Good Snow Man Is Hard to Build* (Hazelden and Davis, 2015). Yet others are entirely

(a) Missile Command (b) Freeway

Fig. 6.5 Two example games that have been used in the GVGAI competition

original, and sometimes designed to exploit the weaknesses of various popular game-playing algorithms. For example, the GVGAI game *Wait For Breakfast* requires the player agent to move a table, and then do nothing for a certain number of timesteps; if the agent moves before the end of the sequence, it loses. This game has proven very hard for most agents.

The original track of the GVGAI competition was the single-player planning track. Here, agents were given a fast forward model of the game, and 40 ms to plan for the next action. Given these conditions, it stands to reason that planning algorithms of various kinds would rule the day. Most of the top performers in this track have been based on variations on MCTS or MCTS-like algorithms, such as the Open Loop Expectimax Tree Search algorithm [870] or MCTS with options search [269]. One high-performing agent used Iterative Width search, where the core idea is to build a propositional database of all the facts and then use this to prune a breadth-first search algorithm to only explore those branches which provide a specified amount of novelty, as measured by the size of the smallest set of facts seen for the first time [652]. Agents based on evolutionary planning also performed well, but not as well as those based on stochastic tree search or tree search with novelty pruning.

While some agents are better than others overall, there is clear non-transitivity in the rankings, in the sense that the best algorithms for one particular game may not be the best for another game. In fact, there seem to be patterns where families of algorithms perform better on families of games [105, 357]. Given these patterns, a natural idea for achieving higher playing performance is to use hyper-heuristics or algorithm selection to select at runtime which algorithm to use for which game, an approach which has seen some success so far [747].

The GVGAI benchmark has been used in various experiments to examine the generality and limits of deep RL gameplaying policies. It has been observed that standard deep RL methods tend to overfit to not only a single game, but even a single level of a single game. Justesen et al. [525] investigated the nature and extent of this overfitting. They then trained models with RL on procedurally generated levels, so that a new level was generated for every training episode and the difficulty of the generated levels increased in tandem with the strength of the policy. This significantly increased policy generalization. However, the models never learned to be fully general even across the different levels of the same game. One possible reason is that the networks did not have the capacity to learn sufficiently high-level

representations to develop full generality. As a takeaway, it is important to remember that RL tends to overfit unless it is explicitly incentivized (or forced) not to, and given a sufficiently high-capacity model for learning general policies.

6.3.6 Fighting Games

The genre of fighting games has received much interest from both academic and industrial stakeholders. Fighting games require cognitive skills mostly related to kinesthetic control and spatial navigation but also fast reaction times and rapid decision making, both of which need to be fast [579]. Early approaches for fighting games include classic reinforcement learning—in particular, the **SARSA** [971] algorithm for on-policy learning of Q values which are represented by linear and ANN function approximators—as applied by the Microsoft Research team in *Tao Feng: Fist of the Lotus* (Microsoft Game Studios, 2003) [397]. Reinforcement learning has also been applied with varying degrees of success for adaptive difficulty adjustment in such games [44, 263, 936]; i.e., AI that plays for the experience of the player. Evolutionary reinforcement learning variants have also been investigated for the task [717].

A notable effort in the fighting games AI research scene has been on the Java-based fighting game *FightingICE* provided by the *Fighting Game AI Competition*[8] [670] (see Fig. 6.6). The competition is organized by the Ritsumeikan University in Japan and has run since 2013 with the aim to derive the best possible fighting bot using various input sources. The most recent variant of the competition, namely *DareFightingICE* invites the submission of AI controllers that rely only on in-game audio. Approaches using *FightingICE* vary from dynamic scripting [690], to k-nearest neighbor [1263], to Monte Carlo tree search [1298], to neuroevolution [589], among others.

The *Street Fighter* series (Capcom, 1987–2023) is one of the most well-known and influential series of fighting games ever, with the various versions of *Street Fighter II* (Capcom, 1991) being particularly influential and still played widely, with an active competition scene. It is therefore no wonder that games from the *Street Fighter* series have been used as a testbed for AI. While there are multiple studies on reinforcement learning across different versions of *Street Fighter*, there is no commonly used benchmark. Research papers use different experimental conditions, so it is very hard to compare the performance of different approaches. As an example, strong performance has been reported for double deep Q-networks in *Street Fighter V* (Capcom, 2016) [628], though it is unclear exactly how the game state was represented. A recent, admirable attempt to create a benchmark based on popular fighting games, including versions of *Street Fighter*, is called *FightLadder*; it is not clear yet whether it will catch on [626].

A rather surprising way to play *Street Fighter*–or any real-time game–is to use large language models. A team called OpenGenerativeAI created a system to do

[8] https://github.com/TeamFightingICE/FightingICE.

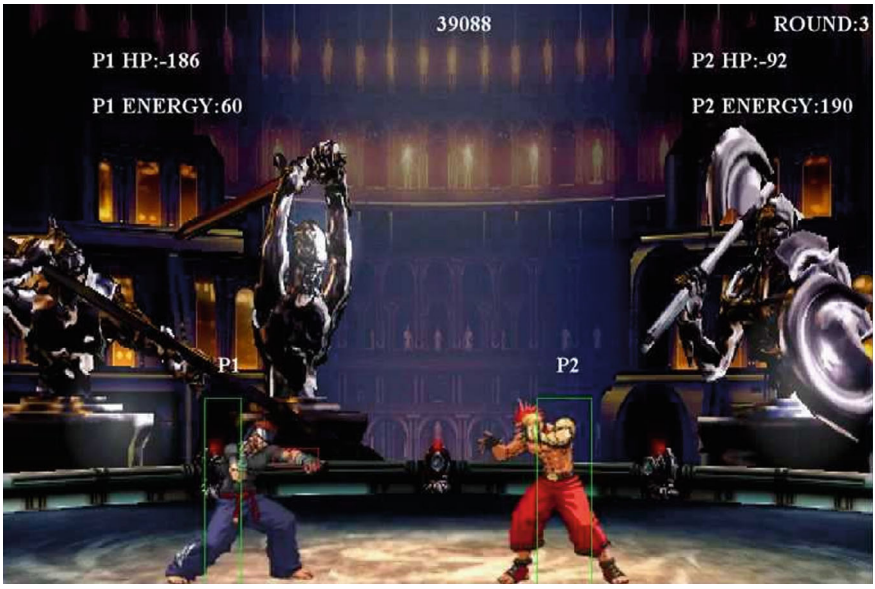


Fig. 6.6 A screenshot from the Java-based *FightingICE* framework for fighting games

exactly this, using *Street Fighter III* (Capcom, 1997) as a testbed. In their implementation, the LLM gets a prompt informing it that it is the best and most aggressive *Street Fighter III* (Capcom, 1997) player, followed by a description of the game state, and is then asked to return which action to take [843]. As rapid responses are of the essence in a fighting game, the inference time of an LLM is a major issue. Interestingly, this might mean that larger and more capable LLMs are at a disadvantage. At the time of writing, the leaderboards are very inconclusive regarding the relative strength of various LLMs that are tested in this game.

In a *Street Fighter* game or similar fighting games it is important to note that learning to beat the built-in NPC opponents is often not particularly hard, as they typically have known vulnerabilities that can be exploited even by quite simple policies. Sometimes, this is at least partly by design, to allow the player to achieve mastery. Simple policies that exploit particular weaknesses in the built-in opponents will typically not generalize well to other opponents, and even less well against other trained agents.

6.3.7 Other Environments

In addition to ALE and GVGAI and all other frameworks discussed earlier, there are several other environments that can be used for AI experimentation with arcade-style games. The *Retro Learning Environment* is a learning environment similar in concept

to ALE, but instead based on an emulation of the Super Nintendo console [87]. A more general, and less focused, system is *OpenAI Universe*,[9] which acts as a unified interface to many different games, from simple arcade games to complex modern adventure games.

While it is definitely interesting to build benchmarks of environments based on increasingly complex games, it is also interesting to explore the other direction: what can be done with games that are as simple as possible? An example of this is the work of Isaksen et al. [493, 494] where an AI agent is built to test the difficulty of *Flappy Bird* (dotGEARS, 2013) levels. *Flappy Bird* is a minimalist 2D video game where only a single action is available (flap) and the objective is to avoid crashing into a series of pipes. The game was briefly a global sensation in 2013. The baseline AI player follows a simple pathfinding algorithm that performs well in completing the *Flappy Bird* levels. To imitate human play, however, the AI player can be curtailed by elements of human motor skills such as precision, reaction time, and actions per second.

6.4 Real-Time 3D Video Games

In this section, we will discuss real-time video games where the player interacts with a three-dimensional game world. In particular, we will discuss games which are displayed from a first-person perspective, or something close to a first-person perspective. In other words, games which you see from the perspective of the player character (first-person) or where the player character is part of the view and the perspective changes with its movement (e.g., over the shoulder perspective). Being real-time video games, these games generally require fast reactions and often also fine-tuned control of movement in a continuous world. They generally also offer a perceptual and cognitive challenge in decoding the world from a first-person point of view, localizing the player character and other objects and characters, and navigating within a complex three-dimensional environment.

The same caveat applies here as for the discussion of 2D games: not all 3D games etc. You get the gist.

6.4.1 Racing Games

Racing games are games where the player is tasked with controlling some kind of vehicle or character to reach a goal in the shortest possible time, or to traverse as far as possible along a track in a given time. Usually the game employs a first-person perspective, or a vantage point from just behind the player-controlled vehicle. The vast majority of racing games take a continuous input signal as a steering input,

[9] https://openai.com/index/universe/.

similar to a steering wheel. Some games, such as those in the *Forza* (Microsoft Studios, 2005–2023) or *Real Racing* series (Firemint and EA Games, 2009–2013), allow for complex input including gear stick, clutch and handbrake, whereas more arcade-focused games such as those in the *Need for Speed* series (Electronic Arts, 1994–2022) typically have a simpler set of inputs and thus lower branching factor. Racing games such as those in the *WipeOut* (Sony Interactive Entertainment, 1995–2022) and *Mario Kart* series (Nintendo, 1992–2020) introduce additional elements, such as weapons that can be used to temporarily incapacitate competitors' vehicles.

While the cognitive challenges in playing racing games may at first appear simple, most racing games actually require multiple simultaneous tasks to be executed and have significant skill depth. At the most basic level, the agent needs to control for the position of the vehicle and adjust the acceleration or braking, using fine-tuned continuous input, so as to traverse the track as fast as possible. Doing this optimally requires at least short-term planning, one or two turns (of the track) forward. If there are resources to be managed in the game, such as fuel, damage or speed boosts, this requires longer-term planning. When other vehicles are present on the track, there is an **adversarial planning** aspect added, in trying to manage or block overtaking; this planning is often done in the presence of hidden information (position and resources of other vehicles on different parts of the track) and under time pressure, and benefits from models of the adversarial drivers.

One relatively early commercial game application that stands out is the AI for *Forza Motorsport* (Microsoft Studios, 2005), which was marketed under the name *Drivatar* [445] (see Fig. 6.7a). The early *Drivatar* agents are built on a form of supervised **lazy learning** [10]. To train the agents, humans drive a number of racing tracks, which are composed of a number of segments; all tracks in the game need to be composed of segments drawn from the same "alphabet". During driving, the agent selects the driving actions that most closely approximate the racing line taken by the players on the relevant segment. This approach was successful in realizing

(a) The *Drivatars* of *Forza Motorsport* (Xbox Game Studios, 2023). Screenshot taken by the authors.

(b) A screenshot from TORCS; the racing game used in the Simulated Car Racing Championship. Image obtained from `https://sourceforge.net/projects/torcs/`.

Fig. 6.7 Two popular examples of gameplaying AI in racing games detailed in Sect. 6.4.1. The *Drivatar* (**a**) is the longest-standing AI system in a video game series whereas the open-source TORCS (**b**) has been used as a test-bed for extensive research on racing game AI

personalized driving agents, i.e., agents that could drive new tracks in the style of human players they had been trained on, but posed restrictions on the design of the tracks.

There are various approaches to training agents to drive without supervision, through reinforcement learning. A sequence of papers has shown how neuroevolution can be used to train agents that drive a single track in the absence of other cars as well as a good human driver [1161], how incremental evolution can be used to train agents with sufficiently general driving skills to drive unseen tracks [1163], and how competitive co-evolution can be used to adversarially train agents to drive more or less aggressively in the presence of other cars [1162]. In all these experiments, the weights of a relatively small fixed-topology network were trained with an evolution strategy. The inputs to the network were the speed of the car and a handful of rangefinder sensors that returned the distance to the edges of the track, or other cars. The low dimensionality of the resulting network enabled high-performing networks to be found relatively easily. Recently, a series of papers demonstrated how a modified version of the Go-Explore algorithm can yield competitive results to PPO [60, 62] on building generative agents that play racing games like humans.

The *Simulated Car Racing Championship*, which run for nearly a decade (2007–2015), is partly based on the aforementioned work, and uses a similar sensor model. The first year, the competition was based on simple 2D racing game, and the winner of the competition was a controller based on fuzzy logic [1164]. In 2008, the competition software was rebuilt around TORCS, a 3D racing game with a reasonably sophisticated physics model [659] (see Fig. 6.7b). In the following years, a large number of competitors submitted agents based on various different architectures, including evolutionary computation, temporal difference learning, supervised learning and simple hand-coded rule-based systems [659]. A general trend observed over the course of the competition was that the winning agents incorporate more and more domain knowledge as hand-coded mechanisms, with learning algorithms generally only used for tuning parameters of these mechanisms. The best agents, such as *COBOSTAR* [157] or *Mr. Racer* [903] generally drive as well as or better than a good human driver when driving alone on a track, but still struggle with overtaking and other forms of adversarial driving.

As discussed above, the Simulated Car Racing Championship provides information in a form that is relatively easy to map to driving commands, making the learning of at least basic driving strategy (but not fine-tuning) relatively easy. However, some authors have attempted learning to drive from raw pixel data. Early on, Floreano et al. [341] evolved a neural network with a movable "retina" to drive in a simple simulated environment. The output of the neural network included both driving commands and commands for how to move the retina, and only the relatively few pixels in the retina were used as inputs to the network. Later, Koutnik et al. [583] evolved controllers that used higher-dimensional input by evolving the networks in compressed weight space; essentially, the parameters of a JPEG encoding of the network connections was evolved, allowing evolutionary search to work effectively

in the space of large neural networks. Supervised learning of deep networks has also been applied to visual driving, yielding high-performing TORCS drivers that learn from examples [1310].

The examples above do not use of a forward model of any kind. However, car dynamics are relatively simple to model, and it is easy to create a fast approximate model for racing games. Given such a model, standard tree search algorithms can easily be applied to car control. For example, Fischer et al. [338] showed that MCTS coupled with a simple forward model can produce decent performance in TORCS.

A recent high-profile achievement in AI for racing games is the GT **Sophy** agent by the Sony AI team, a rare case of a paper about game AI making the cover of *Nature* [1258]. GT Sophy is a driving agent for *Gran Turismo Sport* (Sony Interactive Entertainment, 2017) on the Playstation 4, a highly naturalistic racing game that accurately simulates various aspects of real car racing. The agent was trained with reinforcement learning, and achieves better lap times than top human players while adhering to a realistic driving style. The reinforcement learning algorithm is a variation of Q-learning, and the input feature set is relatively clean and low-dimensional: position on the track, distance to track edges, relative distance and position to other cars, and so on. The reward function is quite complex, consisting of a general reward for progress on the track combined with a handful of penalties for collisions, going off track, tire slipping etc. While GT Sophy may not be an algorithmic milestone for RL, it is notable not only for the excellent performance of the final policy, but also for that it resulted in an agent that can be played by millions of people on their own home consoles.

6.4.2 First-Person Shooters

First-person shooters constitute an important genre of video games ever since the success of *Wolfenstein 3D* (Apogee Software and FormGen, 1992) and *Doom* (id Software, 1993) in the early 1990s. While a basic tenet of an FPS would seem to be that the world is observed through a first-person point of view, there are games that are generally recognized as FPSes which have the camera positioned slightly behind and/or above the player, such as the *Gears of War* (Microsoft Studios, 2006–2019) and the *Tom Clancy's The Division* series (Ubisoft, 2016–2025). Similarly, the word "shooter" signifies that the games revolve around shooting projectiles with some kind of weapon. On that basis, a game such as *Portal* (Electronic Arts, 2007) can be seen as an FPS though it is debatable whether the player implement is actually a weapon.

Shooters are often seen as fast-paced games where speed of perception and reaction is crucial, and this is true to an extent, although the speed of gameplay varies between different shooters. Obviously, quick reactions are in general not an issue for a computer program, meaning that an AI player has a certain advantage over a human by default. But there are other cognitive challenges as well, including orientation and movement in a complex three-dimensional environment, predicting actions

and locations of multiple adversaries, and in some game modes also team-based collaboration. If visual inputs are used, there is the added challenge of extracting relevant information from pixels.

Some early work on playing FPS games focused on optimizing parameters for existing agents in order to improve their efficiency [221]. Even before the AI gameplaying advancements via deep RL, some early studies in this game genre evolved FPS gameplaying controllers acting on visual input of *Quake* (GT Interactive, 1996); the evolved controllers however reached only limited success [853]. More ambitious work on AI for FPS games was spurred by two competitions that we detail in the remainder of this section: first, the **2K BotPrize** and more recently **VizDoom**.

Unreal Tournament 2004 and the 2K BotPrize

Unreal Tournament 2004 (UT2k4) (Epic Games, 2004) is an FPS which was released in 2004, with what was at the time state-of-the-art graphics and gameplay. While the game itself has not been open sourced, a team based at the Charles University in Prague created *Pogamut*, a Java-based API that allows for simple control of the game [373]. Pogamut supplies the agent with an object-based information interface, which the agent can query about the locations of objects and characters, and also provides convenience functions for executing actions such as firing a projectile towards a specific point.

Some early work using UT2k4 attempted to build high-performing agents for one or several in-game tasks, using techniques such as neuroevolution. For example, van Hoorn et al. [1204] subdivided the task of playing UT2k4 into three sub-tasks: shooting, exploring and path-following. Using a layered neuroevolution approach [1149], which combines neuroevolution with the subsumption architecture of Rodney Brooks [128], they then evolved neural networks for each of these tasks in succession. The resulting agent was able to play some game scenarios relatively effectively.

However, the main use of the UT2k4 benchmark has been in the 2K BotPrize (see Fig. 6.8). This competition, which ran from 2008 to 2014, stands out among game-based AI competitions for not focusing on playing for performance, but for experience. Specifically, it was a form of **Turing test**, where submitted agents were judged not by how well they survived firefights with other agents, but by whether they could fool human judges (who in later configurations of the competition also participated in the game) that they were humans [449–451].

The winners of the final 2K BotPrize in 2014 were two teams whose bots managed to convince more than half of the human judges that the bots were human. The first winning team, UT^2, from the University of Texas at Austin, is primarily based on neuroevolution through multiobjective evolution [1004]. It consists of a number of separate controllers, where most of these are based on neural networks; at each frame, it cycles through all of these controllers, and uses a set of priorities to decide the outputs of which controller will command various aspects of the agent. In addition to neural networks, some controllers are built on different principles, in particular

(a) The judges' room. (b) The players' room.

Fig. 6.8 The first 2K BotPrize competition held in Perth, Australia, on 17 December 2008, as part of the 2008 IEEE Symposium on Computational Intelligence and Games. Photos taken by the authors

the Human Retrace Controller, which uses traces of human players to help the agent navigate out of stuck positions. The second winner, *MirrorBot* by Mihai Polceanu, is built around the idea of observing other players in the game and mirroring their behavior [893].

Raw Screen Inputs and the VizDoom AI Challenge

The VizDoom framework [554] is build around a version of the classic *Doom* (id Software, 1993) FPS game that allows researchers to develop AI bots that play the game using only the screen buffer. VizDoom was developed as an AI testbed by a team of researchers at the Institute of Computing Science, Poznan University of Technology (see Fig. 6.9). The framework includes several tasks of varying complexity, from health pack collection and maze navigation to all-out deathmatch. An annual competition based on VizDoom[10] was held at the IEEE CIG conference in between 2016 and 2018, and the framework is also included in *OpenAI Gym* [126],[11] a collection of games which can be used for AI research.

Most of the published work on VizDoom has been based on deep reinforcement learning with convolutional neural networks, given that method's proven strength in learning to act based on raw pixel inputs. For example, *Arnold*, a well-performing agent in the first VizDoom competition, is based on deep reinforcement learning of two different networks, one for exploration and one for fighting [197].

But it is also possible to use evolutionary computation to train neural network controllers. As the very large input size requires huge networks, which do not work well with evolutionary optimization in general, it is necessary to compress the information somehow. This can be done by using an autoencoder trained on the visual

[10] https://vizdoom.cs.put.edu.pl/.

[11] https://gym.openai.com/.

Fig. 6.9 A screenshot from the VizDoom framework which—at the moment of writing—is used in the Visual Doom AI Challenge. The framework is giving access to the depth of the level (enabling 3D vision). Image obtained from http://vizdoom.cs.put.edu.pl/ with permission

stream as the game is played; the activations of the bottleneck layer of the autoencoder can then be used as inputs to a neural network that decides about the actions, and the weights of the neural network can be evolved [20].

Imitation Learning

Despite its success, reinforcement learning is very computationally expensive, as running the actual game is part of the inner loop, and it often results in non-human-like behavior. Just like in other games, you can learn a good policy from behavior cloning if you have access to playtraces. In FPS games, we have seen some convincing examples of this. Pearce and Zhu [857] collected 95 h of gameplay from the deathmatch mode of *Counter-Strike: Global Offensive* (CS:GO) (Valve, 2014) and trained networks consisting on a few convolutional layers and an LSTM layer on this data. The input simply consisted of a cropped and downscaled version of the player view, yielding 280×150 pixels, and the output from the network corresponds to key presses and mouse movements. Standard supervised learning is used to train the network. The trained network outperforms the game's built-in bots and matches both the skills and playstyle of a casual player. The same dataset was used for several

follow-up studies. In particular, one paper demonstrated the use of diffusion models for behavior cloning in *CS:GO* (Valve, 2014). This is an unusual way of using diffusion models—the input is the game state observation (rather than a prompt), and the output is an action vector (rather than an image). The method, though computationally expensive, works very well [856]. In a more recent study, Rašajski et al. [925] attempted to transfer playing behaviors learned in *CS:GO* (Valve, 2014) to 25 unseen FPS games with decent degrees of success. They did so through aligning the visual encoding of videos of FPS games to the player behavioral pattern displayed in the video, such as navigating, shooting with a weapon and panning. Player behavior was not typically represented as a series of states but rather described semantically and encoded via LLMs.

Integrating a model into a real game requires the model to be computationally lightweight, especially if a decision needs to be made every frame and the frame rate is high. Given the competition with other computational tasks in running and rendering the game, inference time for the model might need to be cut down to as little as one or a few milliseconds. This makes it very hard to use pixel inputs given current capabilities of consumer hardware (in some cases, such models may also need to be run on servers without GPU capacity, rendering the question of rendering moot). One solution to this problem is to use simulated sensors as inputs, such as those in the work of van Hoorn et al. discussed above. A team from modl.ai and Riot Games built a game that is a simplified version of *Valorant* (Riot Games, 2020), a popular tactical shooter (see Fig. 6.10a). *Valorant* (Riot Games, 2020) has tactical similarities with the core bomb planting/defusal mode of *Counter-Strike* (Valve, 2000), as opposed to the deathmatch mode used in [857]. In the study by modl.ai and Riot Games, a relatively complex sensor architecture was devised totaling a few hundred inputs, which is still orders of magnitude less than the number of inputs needed when using pixel data (see Fig. 6.10b). Standard supervised learning methods were able to learn strong gameplay performance and, more relevantly, very human-like behavior from only around 50 h of gameplay demonstration. Several different network sizes were tested, and while there is a relation between network capacity and top performance, good performance can be had from only one or two million parameters [521].

6.4.3 Minecraft

The last game we will cover in this section is *Minecraft* (Mojang, 2011), which has become a major platform for game AI research. *Minecraft* (Mojang, 2011) is a **sandbox** game played in a 3D procedurally generated world that players can navigate through. The game features game mechanics that enable players to build structures out of cubes (see Fig. 6.11) but it does not require players to accomplish any specific goals. Beyond exploration and building, players can also gather resources, craft objects and combat opponents. At the time of writing, the game has sold more than 300 million copies across all platforms [73, 95]. The 3D open-world nature of *Minecraft* and the lack of required goals provides players a great degree freedom

(a) The *Val-Lite* AI research testbed modeled on *Valorant* (Riot Games, 2020).

(b) Some of the sensory inputs used by the *Val-Lite* bots. The use of sensors rather than pixel input lead to large computational efficiency gains, allowing real-time decision making with a fraction of a GPU.

Fig. 6.10 Gameplaying bots in *Val-Lite*, an AI research testbed modeled on the team shooter *Valorant* (Riot Games, 2020), discussed in Sect. 6.4.2. Screenshots adopted from [521] with permission

to play and explore the world in their own ways. The benefits of playing *Minecraft* appear to be many and some of them have already been reported by the educational research community [799]. For example, the blocks available in the game can be arranged to produce any object a player might think of. The blocks' functionalities can be combined and extended, resulting in new knowledge for the players that is acquired gradually. Moreover, the simple stylized voxel-based graphics of the game allows the player to concentrate on the gameplaying and exploration tasks within a simple, yet aesthetically pleasing, environment. The multitude of reasons that make *Minecraft* so appealing to millions of players are also the reasons that make the game a great testbed for AI and game AI research. In particular, the game offers

Fig. 6.11 A screenshot from *Minecraft* (Mojang, 2011) showcasing a city hall structure made by the MCFRArchitect Build Team. Image obtained from Wikipedia

an AI player an open world ready to be explored with numerous possibilities for open-ended play. Further, in-game tasks for an AI agent vary from exploration and seeking treasure to making objects and building structures, alone or as a team of agents.

Much AI work using *Minecraft* uses infrastructure from **Project Malmo** [516], which is supported by Microsoft Research. Project Malmo is a Java-based AI experimentation platform built as a game mod of the original game and designed to support research within the areas of robotics, computer vision, machine learning, planning, multi-agent systems and general game AI [516]. The open-source platform is accessible via GitHub.[12] Early experiments with AI in *Project Malmo* include the application of deep neural networks for navigating the 3D mazes [774] and for fighting the opponents of the game [1193]. Beyond *Project Malmo*, it is also worth noting that various other mods of the game have been used directly for teaching robotics [16]—including algorithms for maze navigation and planning—and teaching general AI methods [72].

A very interesting approach to playing *Minecraft*, with potential applications to other interactive environments, is the *Voyager* agent by a team from Nvidia and several US universities [1232]. Voyager is an LLM-powered "lifelong learning" agent that aims to continually increase its capabilities. The agent consists of an automated curriculum which contains a growing list of tasks to complete, an iterative prompting mechanism which uses chain-of-thought prompting to try to derive solutions to these tasks, and a skill library. Interestingly, everything that goes on inside *Voyager* is

[12] https://github.com/microsoft/malmo.

passing strings of text around. At no point are any pixels or button presses processed. Descriptions of the environment are textual, including a list of items in the player's inventory, list of entities nearby, variables such as hunger and health, and so on. Outputs from the agent to the game are function calls with parameters, such as to mine a particular block or craft an item. Skills are stored as JavaScript code. It is worth noting that these conditions are very different from most gameplaying agents discussed in this chapter, which receive either pixels or a structured game state representation as input, and outputs actions as button presses on a controller.

Voyager's performance is indeed impressive, gradually bootstrapping itself to learn to perform complex tasks that require many prerequisites, such as mining diamond and crafting tools from that material. The open-ended skill learning approach might be transferable to other games. However, we should not forget the very specific circumstances that makes *Voyager* work, including a set of methods for describing the environment in textual form and an API for automatically taking actions such as crafting items. *Minecraft* is also one of the most famous games in the world, so an LLM like GPT-4 could be expected to know a lot about it, including what materials are necessary to craft various items. This is a fact that *Voyager* exploits, as it acquires skills by repeatedly asking GPT-4 how to do things.

Another important contribution to learning game agent policies that was tested in *Minecraft* (Mojang, 2011) is the **Video Pretraining** (VPT) method by Baker et al. [46]. VPT addresses the problem of learning a policy from playtraces when all you have is video, and not the actions taken. In VPT, an inverse dynamics model is first trained based on a small dataset that is labeled, i.e. where both actions and pixels are included. The inverse dynamics model associates a video with the actions that were likely taken to produce those movements. This model is then used to label a huge number (i.e., 70,000 h) of videos from YouTube with actions, and this video is then used to train a much larger model that is used to play *Minecraft*. This model was then further trained with reinforcement learning to achieve specific goals. VPT is a clever solution to learning behavior from unlabeled video, but presupposes that such extensive video corpus is available to begin with. Just like *Voyager*, it only works for games that are already famous.

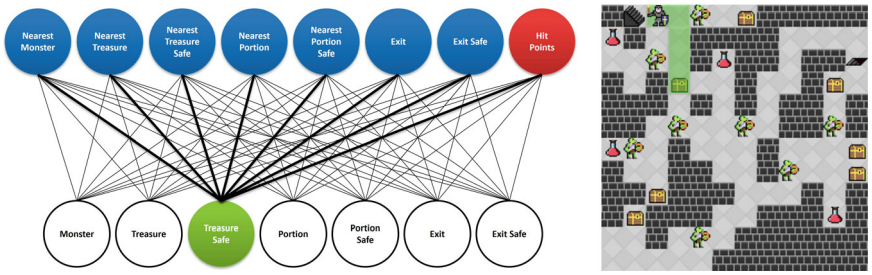
6.5 Dungeon Crawlers

In this section we explore the ways AI algorithms can play dungeon crawlers. This genre considers games typically taking place within fantasy role-playing worlds in which a player navigates through some sort of labyrinth environment (dungeon). The player is typically required to battle enemies, collect treasures and solve puzzles as they descend through the dungeon. Popular games of this genre include the game *Rogue* (Toy and Wichmann, 1980), and the many roguelikes that emerged from this influential game, such as *Nethack* (Nethack DevTeam, 1987), *Unexplored* (Ludomotion, 2017), *Diablo* (Blizzard Entertainment, 1997), *The Binding of Isaac* (Nicalis, 2014), *Hades* (Supergiant, 2020) and *Returnal* (Housemarque, 2023). But

many games that are not roguelikes per se as they are not based on procedural content generation still feature heavy elements of dungeon crawling, such as those in the *Legend of Zelda* (Nintendo, 1986–present) and *The Elder Scrolls* series (Bethesda Softworks, 1994–present). Depending on the game mechanics they feature, dungeon crawlers require long-term planning and exploration. Most such games feature hidden information—e.g., hidden areas behind locked doors—that are revealed during play. The research interest on gameplaying agents for dungeon crawlers has been expanding over the years due to the complex long-term planning nature of such games [34, 424, 459]. In the remainder of this section we put our focus on research performed on two open-source and representative games of the roguelike genre: *Nethack* and *MiniDungeons* [459].

Nethack is an open-source classic roguelike game which has been in constant development since 1987, but its history is actually longer than that, as it builds on *Hack*, which was first released in 1982. As such, it is a highly complex game with hundreds of item and monster types, many with unique behaviors. As with other roguelikes, levels are generated when you enter them; there is also a high degree of hidden information as each level must be explored to be visible. The **Nethack Challenge** is a competition to play *Nethack* with AI agents. So far, the game remains unsolved, and standard Deep RL agents are very far from solving the game, or even playing the game well [424]. The difficulty and low performance of standard methods makes *Nethack* an exciting challenge, and likely to spur more innovation. One interesting approach that was developed for the Nethack Challenge is *Motif* [574], which aims to solve the problem of sparse and potentially misleading rewards in Nethack by constructing intrinsic rewards. These rewards are generated by using LLMs for preference elicitation. Captions of two different game states are shown to an LLM, and a reward model is trained based on the output of the LLM. Using the intrinsic reward in combination with the reward from the environment leads to superior performance compared to the environment reward alone [574].

An early series of studies of gameplaying AI agents within the dungeon crawler genre was made in the *MiniDungeons* game [460, 461]. The focus was on building agents that can play this game emulating the different ways humans would play it. *MiniDungeons* is a simple roguelike game which features turn-based discrete movement, largely deterministic mechanics (i.e., the only stochastic element is the random amount of damage inflicted to the player when hitting an enemy), and full information. The player avatar must reach the exit of each level to win it. Monsters block the way and can be destroyed, at the cost of decreasing the player character's health; health can be restored by collecting potions. Additionally, treasures are distributed throughout levels. Like many games, it is possible to play *MiniDungeons* with different goals, such as reaching the exit in the shortest possible time, collecting as much treasure as possible or killing all the monsters. These different playing styles can be formalized as **procedural personas** by attaching differing utilities to measures such as the number of treasures collected, the number of monsters killed or the number of turns taken to reach the exit. Q-learning was used to learn policies that implement the appropriate persona in single levels [460]. Alternatively, evolutionary algorithms



(a) An artificial neural network mapping between a game state (input) and plans (output). The input contains blue circles representing distances to various important elements of the game and a red circle representing hit points of the player character.

(b) A level of *MiniDungeons 2* depicting the current state of the game.

Fig. 6.12 A example of a *procedural persona*: In this example we evolve the weights of artificial neural networks—an ANN per persona. The ANN takes observations of the player character and its environment and uses these to choose from a selection of possible plans. During evolution the utility function of each persona is used as the fitness function for adjusting its network weights. Each individual of each generation is evaluated by simulating a full game. A utility function allows us to evolve a network to pursue multiple goals across a range of different situations. The method depends on the designer providing carefully chosen observations, appropriate planning algorithms, and well-constructed utility functions. In this example the player opts to move towards a safe treasure. This is illustrated with a green output neuron and highlighted corresponding weights (**a**) and a green path on the game level (**b**)

were used to train neural networks that implement a procedural persona across multiple levels [459]. It is also possible to learn the utility values for a procedural persona clone of a particular human player by evolutionary search for those values that make the persona best match a particular play trace [461] (see Fig. 6.12).

6.6 Puzzle Games

While many games feature puzzles, some video games revolve around puzzle-solving. Among these, we can distinguish between physics-based and physics-free puzzle games, and between real-time and turn-based puzzle games. Being puzzles, these games generally test your ability to foresee the consequences of your actions several steps ahead. The dynamics of the game are largely deterministic, to make it possible for the player to predict what will happen, but there may be other sources of stochasticity, such as shuffling of elements at the start of levels or as new elements are introduced. Typically, puzzle games feature complete or near-complete observability, as too much hidden information would impair the player's ability to predict the consequences of their action.

One example of using AI to play a real-time physics-based puzzle game is a variant of *Cut the Rope* (Chillingo, 2010). The AI agents featured in the *Ropossum*

authoring tool both perform automatic playtesting and also optimize the playability of a level using a combination of tree search, first-order logic, and grammatical genetic programming [1016, 1021, 1022]. The level generation elements of *Ropossum* are further discussed in Part III.

A physics-based puzzle game that has attracted the interest of game AI researchers is *Angry Birds* (Chillingo, 2009). The game has a limited move repertoire: each turn, the player chooses which angle and with which force to throw a bird at a rickety construction made by pigs, and the goal is to raze the ramshackle pigsty and kill all the pigs in *n* bird-launches. *Angry Birds* has an established AI competition [934], named the **Angry Birds AI Competition**, that runs since 2012 mainly in conjunction with the International Joint Conference on Artificial Intelligence. Early AI approaches in *Angry Birds* focused mainly on **planning** and reasoning techniques. Examples include a qualitative spatial reasoning approach which evaluates level structural properties and game rules, and infers which of these are satisfied for each building block of the level [1312]. The usefulness of each level building block (i.e., how good it is to hit it) is then computed based on these requirements. Other approaches model discrete knowledge about the current game state of *Angry Birds* and then attempt to satisfy the constraints of the modeled world [160] based on extensions of answer set programming [124]. More recent work in this game includes approaches on deep RL [816] and deceptive level design [1084] for the improved generalizability of AI agents. A very popular type of turn-based physics-free puzzle games is matching tile games, also often known **match-3 games**. These include the landmark *Bejeweled* (PopCap Games, 2001), where the game screen is a grid of tiles that can be swapped, and if you make a row of three or more tiles they disappear. The mechanics of *Bejeweled* form the basis for the phenomenally successful *Candy Crush Saga* (King, 2012), which adds a number of additional mechanics and tile types, and thousands of clones and descendants, many of them financially successful. Juul [528] provides an excellent analytic history of early developments in matching tile games, but much has happened since.

Given that matching tile games are not bound by complex physics or animations, they can in principle be cheaply simulated, making them amenable to planning methods. There is however a certain degree of stochasticity brought to the game by any new tiles appearing to the level. Despite the challenges, planning-based agents have been developed to play *Bejeweled* (Electronic Arts, 2000) variants in a proficient manner using different playstyles [782]. Imitation learning can also be used with great success to play such games, as evidenced by work from King, the developer of *Candy Crush*, on learning to play that very game [410]. The usual trade-off applies here: planning requires a fast forward model, whereas imitation learning requires plenty of playtraces from human players. There has also been more recent work on applying RL to *Candy Crush* with some success [543].

6.7 Strategy Games

Strategy games, particularly computer strategy games, are games where the player controls multiple characters or units, and the objective of the game is to prevail in some sort of conquest or conflict. Usually, but not always, the narrative and graphics reflect a military conflict, where units may be e.g., knights, tanks or battleships. The perhaps most important distinction within strategy games is between **turn-based** and **real-time** strategy games, where the former leave plenty of time for the player to decide which actions to take each time, and the latter impose a time pressure. Well-known turn-based strategy games include epic games such as the *Civilization* (MicroProse, 1991) and the *XCOM* series (MicroProse, Hasbro Interactive, Infogrames and 2K Games, 1994–2020), as well as shorter games such as *Hero Academy* (Robot Entertainment, 2012). Prominent real-time strategy games include *StarCraft* I (Blizzard Entertainment, 1998) and II (Blizzard Entertainment, 2010), the *Age of Empires* series (Microsoft Studios, 1997–2021) and the *Command and Conquer* series (Electronic Arts, 1995–2020). Another distinction is between single-player games that focus on exploration, such as the *Civilization* games,[13] and multi-player competitive games such as *StarCraft* (Blizzard Entertainment, 1998–2017). Most strategy games feature hidden information.

The cognitive challenge in strategy games is to lay and execute complex plans involving multiple units. This challenge is in general significantly harder than the planning challenge in classical board games such as Chess mainly because multiple units must be moved at every turn; the number of units a player controls can easily exceed the limits of short-term memory. The planning horizon can be extremely long, where for example in *Civilization V* (2K Games, 2010) decisions you make regarding the building of individual cities will affect gameplay for several hundred turns. The order in which units are moved can significantly affect the outcome of a move, in particular because a single action might reveal new information during a move, leading to a prioritization challenge. In addition, there is the challenge of predicting the moves of adversaries, who frequently have multiple units. For real-time strategy games, there are additional perceptual and motoric challenges related to the speed of the game. This cognitive complexity is mirrored in the computational complexity for agents playing these games. As discussed in Sect. 4.2.2, the branching factor for a strategy game can easily reach millions or more.

The massive search spaces and large branching factors of strategy games pose serious problems for most search algorithms, as just searching a single turn forward might already be infeasible. One way of handling this is to decompose the problem, so unit acts on its own; this creates one search problem for each unit, with the branching factor equivalent to the branching factor of the individual unit. This has the advantage of being tractable and the disadvantage of preventing coordinated action among units. Nevertheless, heuristic approaches where units are treated separately

[13] Some of the Civilization games have multiplayer modes, but these are decidedly not the most popular way to play the games.

are used in the built-in AI in many strategy games. Not coincidentally the built-in AI of many strategy games is generally considered inadequate.

In research on playing strategy games, some solutions to this involve cleverly sub-sampling the space of turns so that standard search algorithms can be used. An example of such an approach is an MCTS variant based on decomposition through *Naive Sampling* [827], which relies on a collection of ϵ-greedy sampling policies on the tree. Another approach is non-linear Monte Carlo, which was applied to *Civilization II* (MicroProse, 1996) with very promising results [117]. The basic idea here is to sample the space of turns (where each turn consists of actions for all units) randomly, and get an estimate for the value of each turn by performing a rollout (take random actions) until a specified point. Based on these estimates, a neural network was trained to predict the value of turns; regression can then be used to search for the turn with the highest predicted value.

But planning does not need to be based on tree search. Justesen et al. [522] applied online evolutionary planning to *Hero Academy* (Robot Entertainment, 2012), a two-player competitive strategy game with perfect information and a relatively low number of moves per turn (five in the standard setting). Each chromosome consisted of the actions to take during a single turn, with the fitness function being the material difference at the end of the turn. It was found that this approach vastly outperformed MCTS (and other tree search algorithms) despite the shallow search depth of a single turn, likely because the branching factor made it impossible for the tree search algorithms to sufficiently explore even the space of this single turn.

Evolution has also been used to create agents that play strategy games. NEAT-based [1081] macro-management controllers were trained for the strategy game *Globulation 2* (2009) [823]. In that study, however, the NEAT controller does not aim to win but playing for the experience; in particular, it is evolved to take macro-actions (e.g., build planning, battle planning) in order to provide a balanced game for all players. More recently, AI approaches based on artificial evolution and deep learning have also been used as **playtesting** [982] and **balancing** [505, 875, 876] mechanisms for a number of other strategy games: from prototypical strategy games [505] to combat aspects of Massively-multiplayer Online Role-playing Games (MMORPGs) like *Aion* (NCSoft, 2008) [875, 876].

A popular and very accessible testbed for strategy game research is *microRTS* (or μRTS), originally developed by Santiago Ontañón. The *microRTS* environment implements a simplified strategy game with RTS-like characteristics, which can be simulated simply and rapidly.[14] For several years, there has been a *microRTS* competition running, which has helped draw a number of diverse entries [828]. Among the algorithmic approaches that have been applied successfully to *microRTS* are numerous types of search algorithms [841, 842], as well as deep reinforcement learning [488]; a deep RL algorithm won the microRTS competition in 2023 [393].

[14] https://sites.google.com/site/micrortsaicompetition/microrts.

6.7.1 Diplomacy

Diplomacy (Avalon Hill Games, 1976) is a strategic board game which has existed longer than video games—it was introduced in 1959—but given that its challenges have much in common with computerized turn-based strategy games, we will discuss it in this section. *Diplomacy* is, however, unusual in that it is designed so that dealmaking between the players, in particular secret dealmaking, is key to success. In *Diplomacy*, all players make their moves in parallel every turn; moves are written down on pieces of paper which are overturned at the end of the turn. Conflict resolution is deterministic, and a key mechanic is for one unit to support another unit (which may belong to another player) in a battle against a third unit. Much of the play time is spent in secret negotiations between pairs of players. This means that a winning strategy is to secure support from other players for your own attacks, but it is also often advantageous, at least in the short term, to renege on promises you have given other players. *Diplomacy* is said to have been the favorite game of John F. Kennedy, and as many players can attest to, the game can also be a good way to destroy friendships.

Early research on *Diplomacy*-playing yielded agents of moderate performance, partly because of the challenge of modeling the belief spaces of other players, but most of all because of the complexity of unbounded communication between players [267, 1137]. This all changed in 2022, when a team from Meta presented ***Cicero***, a system that can play *Diplomacy* at a strong human level [331]. *Cicero* is a quite complicated hybrid system, where the main parts are a model-based RL component and a language model component, which inform each other. The RL model outputs its intents to the dialogue model, which converts these to fluent and highly naturalistic English that works in a conversational context; similarly, the dialogue model infers intents of other players and feeds them into the RL model. Interestingly, whereas the system had not been explicitly trained to deceive other players, it would sometimes try to do so anyway. Like most systems incorporating LLMs, it would also say things that fit into the conversational format but were not grounded in facts, such as lying about moves it was going to take, or apologizing for taking a long time making a move because it was on the phone with its girlfriend [277].

6.7.2 StarCraft

The original *StarCraft*, released in 1998 by Blizzard Entertainment, is still widely played competitively, a testament to its strong game design and in particular its unusual depth of challenge. It is usually played with the *Brood War* expansion, and referred to as *SC:BW*. The existence of *Brood War API (BWAPI)*,[15] an interface for playing the game with artificial agents, has enabled a thriving community of researchers and hobbyists working on AI for *StarCraft*. Several competitions have

[15] https://github.com/bwapi/bwapi.

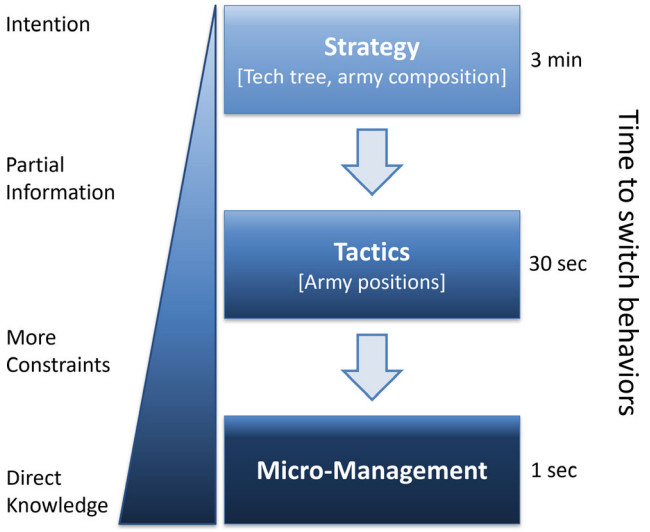


Fig. 6.13 Three different levels of decision making in *StarCraft* (Blizzard Entertainment, 1998). The width of the triangle represents amount of information whereas its colored gradient illustrates the degree of partial observability. For instance, at the highest strategic level both the level of observability and available information to the player are relatively low. On the other end of the triangle, at the lowest level of micro-management the player must consider the type, position, and other dynamic properties of each of the units she controls; that information is mostly observable. For reference, a full game of *StarCraft* often takes around 20 minutes, though there is considerable variation. The image is reproduced with permission from Gabriel Synnaeve

been held annually based on *SC:BW* and BWAPI,[16] including competitions at the IEEE CIG and AIIDE conferences [153]. *TorchCraft* was also an early environment built on top of SC:BW and BWAPI to facilitate machine learning research using *StarCraft* [1119].

As *SC:BW* is such a complex game most early research work on this game focused on only part of the problem, commonly playing it at some level of abstraction. It was for instance common to divide the different levels of decision-making in *SC:BW* (and similar real-time strategy games) into three levels, depending on the time scale: Strategy, Tactics and Micro-Management (see Fig. 6.13). Several early attempts yielded agents that can play a complete game at decent level (e.g., an intermediate human player); however, significant progress was made on several sub-problems of the very formidable problem of playing this game. For a fuller overview of research on AI for playing *SC:BW*, the reader is referred to an early survey focusing on that game [829]. Next, we will exemplify some of the research done in this space, with no pretense of complete coverage.

On the *micro* level AI plays out over the timescale of usually less than a minute, where time between taking actions is typically on the order of a second. When

[16] http://www.starcraftai.com/.

focusing on this most low-level form of *SC:BW* battle, one need not consider base building, research, fog of war, exploration and many other aspects of the full *SC:BW* game. Usually, two factions face off, each with a set of units. The goal is to destroy the opponent's units. This game mode can be played in the actual game, which does not allow for significant speed-up and does not provide a forward model, or in the simulators *SparCraft* [213] and *JarCraft* [524], which do provide a forward model.

In the model-free scenario, agents must be based on either hand-crafted policies or policies learned through reinforcement learning or supervised learning, without the luxury of a forward model. Hand-crafted policies can be implemented e.g., based on **artificial potential fields**, where different units are attracted to or repelled by other units in order to create effective combat patterns [423], or on fuzzy logic [899]. When it comes to machine learning methods for model-free scenarios, standard [1028] and deep reinforcement learning [1197] have been used with some effect to learn policies. Typically, the problem is decomposed so that a single Q-function is learned that is then applied to each unit separately [1197].

Using the *SparCraft* simulator, we saw some early advancements in gameplaying performance because of the availability of a forward model. For instance, Churchill and Buro developed a simple approach to dealing with the excessive branching factor called Portfolio Greedy Search [213]. The core idea is that instead of selecting between actions for each unit, a small number (a portfolio) of simple scripts are used, and a greedy search algorithm is used to assign one of these scripts to each unit. This approach prunes the branching drastically, but limits the space of discoverable policies to those that can be described as combinations of the scripts. It was shown that the portfolio of scripts idea can be combined with MCTS with good results [524], and even better results were obtained by doing portfolio selection through evolutionary planning [1231].

Moving to the other end of the micro-management-tactics-strategy continuum, large-scale *strategy* adaptation remained a very hard problem for decades of research in *SC:BW*. For a long time, *SC:BW* bots were rarely able to implement multiple strategies, let alone adapt their strategy based on how the game progresses. In order to do this, it is necessary to create a model of what the opponent is trying to do based on limited evidence. Here, early pioneering work by Weber and Mateas focused on mining logs of *SC:BW* matches to predict what strategy will be taken by a player from game actions [1245]. Early on, ambitious attempts were made to create complete agents that can handle strategy, tactics and micro-management in a principled fashion within *SC:BW*. For example, Synnaeve and Bessière built an agent based on Bayesian programming that was able to perform reasonably well [1118].

Much of the early work in *StarCraft* (Blizzard Entertainment, 1998) AI feels like a prelude to the most capable and impressive StarCraft-playing system of all, namely **AlphaStar**. AlphaStar was a multi-year project from DeepMind with the help of Blizzard to create a bot to play *StarCraft II* (Blizzard Entertainment, 2010) the mechanically similar successor to *StarCraft* (see Fig. 6.14). The first AlphaStar paper has 42 authors and an (undisclosed but unarguably high) amount of money

Fig. 6.14 A screenshot from *StarCraft II: Wings of Liberty* (Blizzard Entertainment, 2010) displaying a number of different units available in the game attacking a base. Image obtained from the 10th anniversary press kit of the game (fair use)

that was spent on compute resources, making this the kind of project that could not be attempted by a mere academic team [1219]. The results are similarly impressive and complex.

The first thing to note is that AlphaStar plays the full version of *StarCraft II*, which means that no forward simulation is practically possible. Therefore, the AlphaStar system as a whole is a model-free reinforcement learning system trained via self-play, but it is first pre-trained using supervised learning on human playtraces. The model receives two main types of input at the same time: an overview map of the game, and a list of units and their locations. As these are very differently structured views of the game state, they pass through their own input networks before converging on shared network layers; the overhead map input passes through a convolutional layer structure, and variable-length list of units through a transformer. The outputs from the model are primarily a list of commands for unit movements, build orders, etc.

Like all RL systems, AlphaStar is susceptible to overfitting. Therefore, the team designed an elaborate league system, where "exploiter" agents are trained to exploit particular weaknesses of other agents, and the exploiter agents then become part of the training for other agents. This league system is similar to competitive coevolution, or a quality-diversity search mechanism where performance against particular other agents are seen as behavioral dimensions. The result was an agent that could perform at grandmaster level, which is a very high performance, but not competitive to highest-level human play.

This description of AlphaStar barely scratches the surface; it is a very complex system, and a landmark in AI for gameplaying. Still, it is worth pointing out that this very impressive system is oddly brittle in various ways. For example, it only worked

for a single map on one particular sub-version of *StarCraft II*; when the game was updated, it stopped working. It was also possible for strong human players who played it several time to "cheese" AlphaStar by identifying particular weaknesses in its playstyle and exploit them. It is likely that such exploits are possible for any gameplaying agent that does not adapt its playstyle during inference time.

6.7.3 Team Sports Games

Team sports, like Association football (a.k.a. soccer, or simply football in most of the world) and American football, can be seen as strategy games in that they involve the coordination of multiple units that work together towards the goal of beating the other team. Astute observers might remark that football is more a real-time strategy game, whereas American football with its constant pauses is more of a turn-based strategy game. Basketball, with its smaller teams, has a stronger tactics element. The authors do not understand cricket well enough to comment on what kind of strategy game it is.

Given the popularity of team sports games, one would expect there to be ample AI research on playing such games. This is partially true, yet much of the interesting research comes from the robotics community. The *RoboCup* series of competitions which has been organized annually since 1996 and has the ambitious goal of, by 2050, fielding a team of autonomous humanoid football players and win against the winners of the most recent World Cup [571]. At the time of writing, there's 26 years to go, so there's a chance. Not all of *RoboCup* takes place with physical robots, however; there are several simulation leagues, both in 2D and 3D. The literature on *RoboCup* simulation is substantial, but it is unfortunately hard to disentangle the gameplaying aspects from problems derived from robotics [366]. Moving away from robots but sticking to the game of football, the simulator named *Google Research Football Environment* (GRFE)[17] is a relatively recent yet increasingly popular framework for studying aspects of RL-based gameplaying agents [595]. The interested reader is referred to a recent survey paper [1068] that covers the various attempts made to train AI to play football in this environment.

While football (American or not) is an almost fully observable game when you play it, it turns into a hidden information game when it is viewed on screen (TV watching or video game playing). Actually, the same holds for several sport games out there including basketball, volleyball, and hockey among many. In football only some of the players are visible at all times, usually the ones closer to the ball. Any information not visible has to be inferred by e.g., a computer vision method which is tasked to estimate the game's state in order to play the game well.

A recent sequence of studies [1186–1188] attempted to investigate ways to represent team sport games through their pixels for the purpose of building general-purpose and versatile gameplaying agents. As games are not merely images, any computer

[17] https://github.com/google-research/football.

vision model trained on game pixels will likely end up capturing differences of the image's visual style rather than the actual game state. The resulting models are thereby not able to generalize well and play dissimilar games even within the same genre. The Sports10 dataset was created to enable such studies on style-invariant representation learning for gameplaying and other downstream tasks. The dataset contains 100,000 gameplay images of 175 different sport games across 10 genres—American football, basketball, bike racing, car racing, fighting, hockey, football, table tennis, tennis and volleyball—extracted from publicly available sports video game titles. Each genre contains exactly 10,000 hand-curated in-game images ensuring that all menu frames, transition frames, or cut-scenes are removed. Recent studies on that dataset showcased the benefits of contrastive learning for representation learning in games [1186, 1187]. It turns out that if you train a visual encoder to contrast images of sports games results to models that ignore the visual style and focus, instead, on more meaningful aspects of the game state. Such models can predict where the ball and football players are on the screen with higher degrees of accuracy compared to methods from conventional supervised learning.

6.8 Text-Based Games

Text-based games are ones in which players must use natural language commands in order to play, typically by interacting with objects and characters in the environment and solving puzzles. Fantasy is a popular genre for text-based games (also often referred to as **interactive fiction**), with games like *Zork* (Infocom 1979–1982) tasking players with retrieving treasure from a monster-filled and sprawling underground dungeon. However, more recent games like *Façade* (Procedural Arts, 2005) [721]. and *Mystery House Possessed* (Emily Short, 2005) adapt the classic interactive fiction approach to new styles and forms. Early examples of development tools for text-based games include the *Inform* series of design systems and programming languages, which are inspired by the Z-machine and led to the development of several text-based games and interactive fiction based on natural language. *Inform 7*[18] was also used for the design of the *Mystery House Possessed* (Emily Short, 2005) game.

In this game genre, AI can play the role of understanding player's text in a natural language format. In other words, the game AI can feature **natural language processing** (NLP) for playing the game as the player's companion or as an opponent. Further NLP can be used as an input for the generation of a dialog, a text or a story in an interactive fashion with the player. It is normally the case that text-based input is used to drive a story (interactive narrative) which is often communicated via embodied conversational agents and is represented through the lens of a virtual camera. Similarly to traditional cinematography, both the camera position and the communicated narrative contribute to the experience of the viewer. Opposed to traditional cinematography, however (but similarly to interactive drama), the story

[18] http://inform7.com/.

in games can be influenced by the player themselves. AI has been used primarily to play these games with approaches varying from interactive planners [1311], RL [262], LLM-based agents [181, 1190], and self-supervised models [1235]. It goes without saying that research in text-based games is also naturally interwoven with simulators [1236] and benchmarks [189] of such environments, **believable agents** [1195, 1316] (as covered in Sect. 6.9) and computational and **interactive narrative** [24, 721, 939, 1141, 1300] (as primarily covered in Part III of this book).

Work on text-based AI starts from the early language-based interaction with the **Eliza** program [1246] and the Z-Machine (Infocom, 1979) used by text adventure games such as *Zork I* (Infocom, 1980), to *Façade* (Procedural Arts, 2005) [718, 721], to word2vec [763] approaches (e.g., its TensorFlow implementation [2]) for playing Q&A games [564] and text-based adventure games [582] and more recently to LLM-based approaches [181, 365, 1190, 1235]. Beyond the use of AI to understand natural language we can use AI to play text-based games. A notable early example of an agent that manages to handle both tasks is the one developed by Kostka et al. [582], named *Golovin*. *Golovin* used related corpora such as fantasy books to create language models (via word2vec [763]) that are appropriate to this game domain. To play fiction games *Golovin* used five types of command generators: battle mode, gathering items, inventory commands, general actions and movement. *Golovin* was validated on 50 interactive fiction games demonstrating comparable performance to state of the art of the time. Another example is the agent of Narasimhan et al. [793] that plays *multi-user dungeon* games, a form of multi-player or collaborative interactive fiction. Their agent converts text representations to state representations using Long Short-Term Memory (LSTM) networks. These representations feed a deep Q network which learns approximate evaluations for each action in a given game state [793]. Their approach significantly outperformed other baselines in terms of number of completed quests.

Text-based interaction with games has, naturally, been dominated by LLMs in recent years [365]. LLMs are to natural role-players, and so should be very well suited to both playing text-based games and providing NPC interaction. But the devil is in the details. AI agents [181, 1190, 1235] sitting at the player's or the non-player's role are empowered by various types of language models. Agents that play to win such games usually combine LLMs with RL [181] or self-supervised learning methods [1235] for that purpose. LLM-based NPCs—that are integrated within games such as the *AI people* (GoodAI, 2024 [expected]) and *1001 Nights* by Yuqian Sun [1106]—are trained to generate dialogue interactions with the player and corresponding behavioral patterns. This is usually achieved by fine tuning LLMs to associate input language to predefined number of agent actions. We further detail the narrative generation aspects of LLMs (and other methods) in such games in Chap. 9.

6.9 Social Simulation Games

Social simulation games are games in which players explore social interactions with multiple other game characters that are predominantly controlled by AI. Such games are often viewed as a subgenre of simulation video games in which the player controls one or more virtual characters with the aim to explore the characters' *life* in association with the other characters *living* in this virtual world. The genre was popularized when the first game of *The Sims* series (Electronic Arts, 2000–2024) was out in 2000. *Animal Crossing* (Nintendo, 2001) is another popular game of this genre that was released the year after. Similarly to *The Sims*, *Animal Crossing* turned into a popular game series of nearly two decades of production; the latest game of the series being *Animal Crossing: New Horizons* (Nintendo, 2000). The popularity of these game series gave rise to diverse game titles featuring social simulations of various sorts. A recent example is *Disney Dreamlight Valley* (Gameloft, 2023) in which players 'live' in a magical valley and interact with a number of *Disney* and *Pixar* characters.

The AI systems in *The Sims* have gone through multiple iterations as the series has progressed, and are arguably an important part of its popularity. Interestingly, the NPCs need to have exactly the right level of self-sufficiency; if they are too good at taking care of themselves, the game will get uninteresting to play as the player would be unnecessary, and if they are too stupid, the game will become tedious to play as the NPCs would need to be micro-managed. The hybrid AI system in *The Sims* games is built around utility-based AI. NPCs in *The Sims* have a set of needs, which are prioritized similarly to Maslow's hierarchy of needs, where needs such as hunger take precedence over e.g., gossiping. Each NPC will autonomously try to satisfy its most pressing needs, and the games feature an architecture where various objects and other NPCs can broadcast their ability to satisfy various needs; the *Sims* can then navigate towards their best source of need fulfillment using standard path-planning. Figure 6.15 shows an example of player-NPC interaction that involves the current emotional state of the player and the NPC, the navigation needs of the player's *Sim* and the possible dialogue options for the player.

Creating believable social simulations is a key element of successful social simulation games [42]. As social simulation games predominantly employ NPCs in social interactions with the player, AI is tasked to help them act **believably** and in a **human-like** fashion. Agents of these games are also often required to behave in an emotionally expressive manner while acting as the player character's companions, friends or lovers. AI in social simulation games is, therefore, necessary for modeling NPC behavior and playing the game as an NPC but not for winning; rather for the experience of play. AI methods can also guide NPC's social and individual needs and make them behave in an emotion-aware fashion by detecting the player's affect or sentiment.

There is no wonder why years of active research have been dedicated to the aforementioned social NPC tasks. This is where AI research with a focus on NPCs playing for experience [42, 1174] meets research on *affective agents* [489, 620, 1287] and *computational narrative* [619], among other fields. (Here we focus primarily on

NPC in-game behavior as work related to story or narrative generation is covered in Part III.) The usual approach followed in early days of designing social NPCs is the construction of *top-down* (ad-hoc designed) agent architectures that represent their various cognitive, social, emotive and behavioral abilities. The focus has traditionally being on both the modeling of the agent's behavior and its appropriate expression under particular in-game contexts. A popular way of constructing a computational model of agent behavior is to base it on a theoretical cognitive model such as the Ortony-Clore-Collins (OCC) model of emotions [26, 307, 316, 399, 838], which represents human-like decision making, appraisal and coping mechanisms dependent on a set of perceived stimuli. In a recent study, Antunes et al. [28] used ChatGPT as a prompting mechanism to support socially intelligent interactions with agents. The agents combine the belief-desire-intention model [375]—similar to the BDI model used for controlling the creature of *Black and White* (EA, 2000)—and the OCC model to represent, respectively, their cognitive abilities and emotional appraisal.

To assess the believability of NPCs we tend to observe the various ways it can be manifested in a particular game [1174]. Believability can be expressed, for instance, via the NPC's behavior and decision making, its body stance, its facial expression, its generated language, or simply put, a combination of all aforementioned aspects. One would thus argue that expressive and believable conversational agents such as *Greta* [892] and *Rea* [183], or *virtual humans* [1111] should be considered in the design of social simulation games for enhancing their believability of their NPCs. Beyond social simulation games, the use of such character models has been dominant in the domains of intelligent tutoring systems [227], embodied conversational agents [26,

Fig. 6.15 A screenshot from *The Sims 4* (Electronic Arts, 2014) displaying the emotional state of the player (i.e., embarrassed) and the NPC *Sim* it interacts with (i.e., angry). Dialogue actions are also displayed around the player (e.g., compliment outfit) in this example. Gameplay screenshot taken by the authors

182], and affective agents [400]. Agent architecture systems and social simulation models such as these control the NPCs of the *Crystal Island* game series [968, 969, 977], the expressive agents of *Prom Week* [729] and *Façade* (Procedural Arts, 2005) [721], the agents of the *World of Minds* [318] game, and the FAtiMA [281] enabled agents of *My Dream Theater* [170]. In a recent modification of a commercial-standard game [413], *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011) was enriched with a number of different socially expressive NPCs based on the *Comme il Faut* [731] social AI tool of *Prom Week*. An emotion-enriched version of *Comme il Faut* was also integrated in a mod of *Conan Exiles* (Funcom, 2021) [778]. Findings from these papers suggest that players tend to spend more time with the modified NPCs compared to the vanilla NPCs that are shipped with the game; the *Comme il Faut* enabled NPCs were also perceived as more believable and less predictable.

The ever growing popularity of LLMs across every aspect of game AI research, unsurprisingly, has influenced research in social simulation games. A prominent and influential example is a recent paper by Park et al. [851] showcasing the emergent behaviors of 25 LLM-guided generative agents that populate a sandbox environment, named *Smallville*, that is inspired by *The Sims*. Each LLM guiding the actions of an agent is prompted with a short biography characterizing the agent, including the agent's name, age, interests, hobbies etc. The LLM is enriched with a complete record of the agent's experiences. Agent experiences are used to synthesize the agent's reflections and those reflections are eventually turned into behavior plans. In that way, each agent can generate a stream of actions that is influenced by their prescribed (prompted) biographies. Users of *Smallville* can just observe the agents performing their actions in this virtual town or even intervene and influence the agents' plans. A human evaluation of *Smallville* via crowdsourcing shows that agents can generate believable behaviors, both individually and socially. Even more impressively, agents self-organize and demonstrate emergent social behaviors such as collectively planning the necessary actions required to throw a successful party [851].

Concluding this section it is worth pointing out that the design of social simulation games often involves a particular set of underlying **learning objectives** with an educational purpose. An example is the narrative-centered social simulation game series *Crystal Island* designed for effective science learning [968, 977, 1127]; that game features emotion-aware and expressive NPCs that guide the learning process of the player. The learning objective can, instead, be the training of social skills such as conflict resolution and social inclusion through games; *Village Voices* [558] (see Fig. 6.16a), *My Dream Theater* [170] (see Fig. 6.16b), and *Prom Week* [729] are indicative examples of such soft skill training games. Finally, the core gameplaying objective can be an AI-based one such as learning the basic principles of supervised learning and reinforcement learning in the *ArtBot* game [1226]. The interested reader in games for AI and machine learning education is referred to a relatively recent survey on the topic [379].

(a) *Village Voices*. Screenshot image adopted from [558].

(b) *My Dream Theater*. Screenshot image adopted from [169].

Fig. 6.16 The *Village Voices* and the *My Dream Theater* social simulation games for conflict resolution. *Village Voices* realizes experiential learning of conflict in social, multi-player settings, whereas *My Dream Theater* offers a single-player conflict management experience in a social setting. In the former game the AI takes the role of modeling conflict and generating appropriate quests for the players, whereas in *My Dream Theater* AI takes the role of controlling expressive agent (NPC) behaviors

6.10 Further Reading

The methods used for playing games have an extensive body of literature that is covered in detail in Chap. 3 and this chapter. The different game genres that we covered in this chapter contain a corresponding literature the interested reader could use as a starting point for further exploration. In this section we put an emphasis on seminal work and recommend additional readings and resources.

Beyond the papers listed in the corresponding section, the interested reader is referred to Chap. 13 of [306] for algorithmic advances of AI employed in card games. When it comes to board AI gameplaying, the Ludii general game system [881] and the tabletop games framework [50, 360] define excellent landing frameworks. Edelkamp's book [306] dedicates an entire chapter (Chap. 15) to general gameplaying with a focus on MCTS. The *decision transformer* [203] and *multi-game decision transformer* [615] offer sophisticated and high performing methods for arcade-style games that are potentially applicable to any real-time 2D video game. If the reader is interested in the state of the art of AI players for racing games they are referred to the original article of GT Sophy [1258]. We recommend the recent work of Pearch and Zhu [856] when it comes to imitation learning in FPS games. A good starting point for interested readers in *Minecraft* gameplaying AI is *Project Malmo* [516]. Refer to a recent survey paper [1068] covering various attempts made to train AI to play football, if your research interest is in team sports games. For the interested reader in text-based games, the *Text Game Research List*[19] features a comprehensive list of papers on interactive text-based virtual environments, agents, and related areas.

[19] https://www.textgames.org/.

Finally, for readers interested in social simulation games, Marsella et al. [708] cover the most popular computational models of emotion that could be incorporated in game NPCs, and Azad and Martens [42] offer a throughout taxonomy and survey of simulated models of social interactions that define the backbone of this game genre.

6.11 Exercises

In this section we provide a general exercise that allows you to apply any of the algorithms covered in Chap. 3 in a single domain. The purpose of the exercise is for you to get familiar with those methods before testing them in more complicated and complex domains and tasks.

As discussed in Sect. 6.3.2, the *Ms Pac-Man vs. Ghost Team* competition is a contest held at several AI conferences around the world in which AI controllers for Ms Pac-Man and the ghost team compete for the highest ranking. For this exercise, you will have to develop a number of Ms Pac-Man AI players to compete against the ghost team controllers included in the software package. This is a simulator entirely written in Java with a well-documented interface. While the selection of two to three different agents within a semester period has been shown to be a good educational practice we will leave the final number of Ms Pac-Man agents to be developed on you or your class instructor.

The website of the book contains code for the game and a number of different sample code classes in Java to get you started. All AI methods covered in Chap. 3 are applicable to the task of controlling Ms Pac-Man. You might find out, however, that some of them are more relevant and efficient than others. So which methods would work best? How do they compare in terms of performance? Which state representation should you use? Which utility function is the most appropriate? Do you think you can make the right implementation decisions so that your Ms Pac-Man plays at a professional level? How about at a world champion, or even at a superhuman level?

Why Ms Pac-Man?

Some of our readers might object to this choice of game and think that there should be more interesting games AI can play with. While *Ms Pac-Man* is very old it is arguably a classic game that is still fun and challenging to play, as well as a simple testbed to start experimenting with the various AI methods and approaches introduced in this chapter and the previous one. *Ms Pac-Man* is simple to understand and play but is not easy to master. This controversial element of play simplicity combined with problem complexity makes *Ms Pac-Man* the ideal testbed for trying out different AI methodologies for controlling the main character, Ms Pac-Man. Another exciting feature of this game is its non-deterministic nature. Randomness not only augments the fun factor of the game but it also increases the challenge for any AI approach

considered. As discussed, another argument for the selection of *Ms Pac-Man* is that the game and its variants have been very well studied in the literature, tested through several game AI competitions, but also covered for years on AI (or game AI) courses in several universities across the globe.

6.12 Summary

This book part dedicated to AI gameplaying concluded with a detailed review of the studies and the relevant methodologies on a *game genre* basis. Specifically, we saw how AI can play board and card games, real-time 2D and 3D games, dungeon crawlers, puzzle games, strategy games, text-based games and social simulation games. We saw that different game genres require different behavioral and cognitive skill-sets from players and—as a result—map to different AI needs.

As a general observation we note that there is not a single *best* type of algorithm out there that plays well the different genres we discussed. There are several factors that make this selection a tricky one and usually the resulting "best" algorithm is a tailor-made method to a specific game, and a specific mode of interfacing with that game. The type of gameplay observations available (e.g., player actions vs. pixels) will normally determine the AI representation chosen. The purpose of AI playing this game (e.g., to win vs. for experience) and the role (as a player or NPC) will guide the design of the method's utility. Finally, the availability of a forward model, computational resources, and time for an AI to take a decision are a few more important factors influencing the use of search, planning, RL, supervised learning or evolutionary learning methods (or even hybrids of those). We hope that after going through the numerous examples we detail in this chapter your AI algorithmic decisions for a particular game use case will be very much informed!

Moving on to the next part of the book we explore the core principles, methods and uses of AI for the procedural generation of game content.

Part III
Generate

Chapter 7
Procedural Content Generation

Abstract This chapter introduces procedural content generation (PCG), and presents a comprehensive taxonomy of content types, methods, and roles. PCG can be used for a large variety of content types, including quests, textures, terrains, items, characters, and game rules. However, 2D and 3D levels and maps are the most common types of generated content. PCG methods can be used on their own or to assist players and designers, and during game development as well as during the game's runtime. The various methods used differ in many respects, among others how controllable and how fast they are, and whether they give guarantees as to the playability of the content.

Procedural content generation [1023] is an area of game AI that has seen an explosive growth of interest. While games that incorporate some procedurally generated content have existed since the early 1980s—in particular, the dungeon crawler *Rogue* (Toy and Wichmann, 1980) and the space trading game *Elite* (Acornsoft, 1984) are early trailblazers—research interest in academia has really picked up within the last decade and a half.

Simply put, PCG refers to methods for generating game content either autonomously or with only limited human input. Game **content** is what is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc. Typically, NPC behavior and the game engine itself are not thought of as content. Probably the most common current usage of PCG is for generating visuals, levels and terrains, but in the future we might see widespread generation of all kinds of content, possibly even complete games. In this book, the PCG methods we cover tend to focus on *functional* rather than *decorative* content; this distinction will be explored further in this chapter.

Seeing PCG from an AI perspective, content generation problems are AI problems where the solutions are content artifacts (e.g., levels) that fulfill certain constraints (e.g., being playable, having an entrance and an exit) and/or maximize some metrics (e.g., length, difference in outcomes between different strategies). As we will see, many AI methods discussed in Chap. 3 can be used for PCG purposes, including evolutionary algorithms, neural networks, reinforcement learning and LLMs; those are covered in detail in Chap. 8. There are also a number of methods that are commonly used for PCG that are not typically thought of as AI; those are also presented

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_7

in Chap. 8. This chapter instead is devoted to the core principles, notions and roles of PCG in games. We start with discussing why you want to use PCG at all; just like for playing games, there are some very different motivations for generating content. Then in Sect. 7.2 we present a general taxonomy for content types, PCG methods, and possible roles in games.

7.1 Why Generate Content?

Perhaps the most obvious reason to generate content is that it could help human designers or artists creating that content. Ever since computer games were invented, the number of person-months that go into the development of a successful commercial game has kept increasing.[1] It is common for a major game to be developed by hundreds of people over a period of several years. This leads to a situation where fewer games can be developed (or can be profitable) leading in turn to less risk-taking and less diversity in the games marketplace. A game development company that could *assist* some of their artists and designers with algorithms would have a competitive advantage, as games could be produced faster and cheaper while preserving quality. This argument was made forcefully by legendary game designer Will Wright in his talk *The Future of Content* at the 2005 Game Developers Conference, a talk which helped reinvigorate interest in procedural content generation. More recently, the advent of LLMs and diffusion models have precipitated a hype cycle for generative AI in game development. To the best of our knowledge—as AI and games researchers and co-founders of a start-up that develops **AI-assistive tools**—PCG can assist in the processes of art and content production, thus directly contributing to the reduction of around 20%-40% of a game's cost on average. Obviously these are rough estimates that are affected by several factors including, but not limited to, the type of game, the developer team size, the company's structures and product pipelines.

Beyond assistance in their productivity, content generation, especially embedded in intelligent design tools, can foster the **creativity** of individual human creators. Humans, even those of the "creative" vein, tend to imitate each other and themselves. Algorithmic approaches might come up with radically different content than a human would create, through offering an unexpected but valid solution to a given content generation problem. Some evidence for this is already available in the literature [634, 1279, 1317]. Plenty of additional evidence can be found in the increasing use of PCG methods as assistive tools in video game development practice as we will see in the remainder of this book part. This makes it possible for small teams without the resources of large companies, and even for hobbyists, to create content-rich games by freeing them from worrying about details and drudge work while retaining overall directorship of the games. PCG can then provide a way for *democratizing* game design by offering reliable and accessible ways for everyone to make better games in less time.

[1] At least, this is true for "AAA" games, which are games sold at full price worldwide.

Both of these arguments assume that what we want to make is something like the games we have today. But PCG methods could also enable completely **new types of games**. To begin with, if we have software that can generate game content at the speed it is being "consumed" (played), there is in principle no reason why games need to end. For everyone who has ever been disappointed by their favorite game not having any more levels to clear, characters to meet, areas to explore, etc., this is an exciting prospect.

Even more excitingly, the newly generated content can be tailored to the tastes and needs of the player playing the game. By combining PCG with player modeling, for example through measuring the response of players to individual game elements, we can create **player-adaptive games** that seek to maximize the enjoyment of players (see the roles of PCG in Sect. 7.2.3). The same techniques could be used to maximize the learning effects of an educational game, or balance your favorite adventure game.

Finally, a completely different but no less important reason for developing PCG methods is to *understand design and creativity*. Computer scientists are fond of saying that you do not really understand a process until you have implemented it in code (and the program runs). Creating software that can competently generate game content could help us understand the process by which we "manually" generate the content, and clarify the affordances and constraints of the design problem we are addressing. This is an iterative process, whereby better PCG methods can lead to better understanding of the design process, which in turn can lead to better PCG algorithms enabling a **co-creativity** process between designers and algorithms [1279].

7.2 Taxonomy: Procedural Content Generation

With the variety of content generation problems, methods and approaches that are available, it helps to have a structure that can highlight the differences and similarities between them. In the following, we introduce a revised version of the taxonomy of PCG that was originally presented by Togelius et al. [1176]. It consists of a number of dimensions, where an individual method or solution should usually be thought of as lying somewhere on a continuum between the ends of that dimension. Beyond the taxonomy for the **content** types (Sect. 7.2.1) and the properties of the PCG **methods**—detailed in Chap. 8—we provide an outline of the **roles** a PCG algorithm can take, which we cover extensively in Sect. 7.2.3.

7.2.1 Content

We identify the *type*, the *functionality*, and the *spatiality* of the generated outcome as the three core dimensions that relate to content in this taxonomy (see also Fig. 7.1).

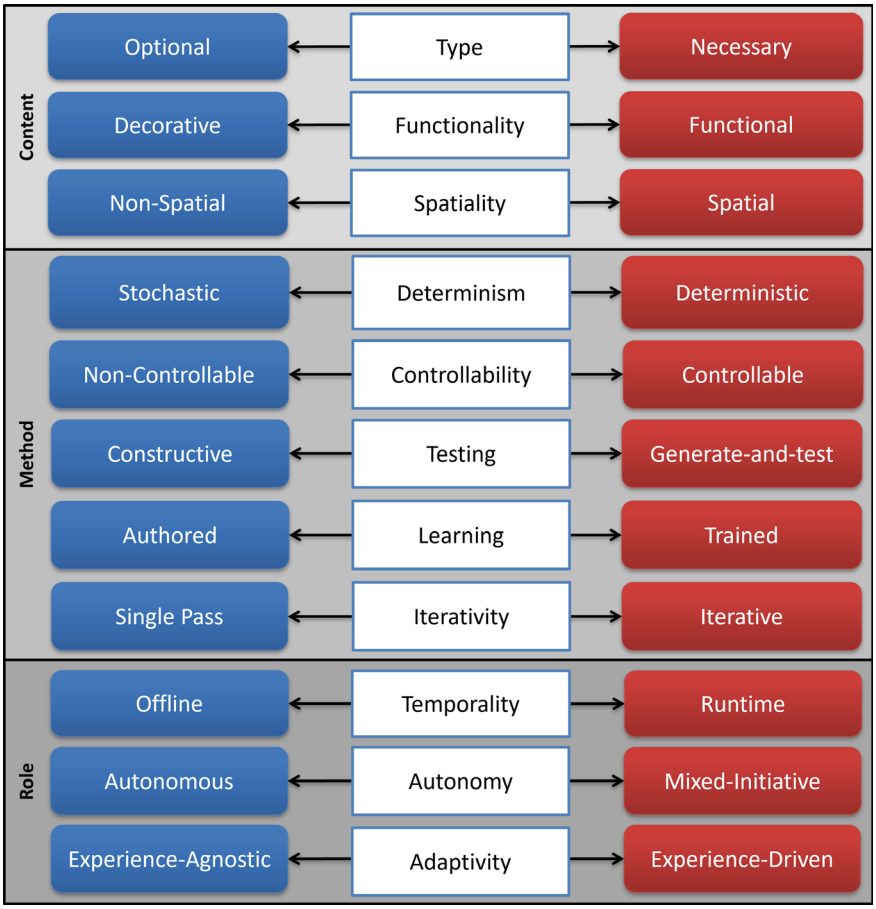


Fig. 7.1 An illustration of the PCG taxonomy discussed in Sect. 7.2

Type: Necessary Versus Optional

PCG can be used to generate **necessary** game content that is required to be traversed or interacted with for the completion of a game, or it can be used to generate **optional** content that can be discarded or exchanged for other content. The main distinguishing feature between necessary and optional content is that necessary content should always be correct while this condition does not hold for optional content. Necessary needs to be consumed or passed as the player makes their way through the game, whereas optional content can be avoided or bypassed. An example of optional content is the generation of different types of weapons in *Borderlands* (2k, 2009), or the various reward items existent within the box tiles of *Super Mario Bros* (Nintendo, 1985). Necessary content can be the main structure of the levels in *Super Mario Bros*, or the collection of certain keys required to pass to the next level

in *Doom* (id Software, 1993). Generally, if it's a side quest or something that comes out of a loot box, it's optional. We detail the deployment of PCG methods across different content types in Chap. 9.

Functionality: Decorative Versus Functional

Outside of game development, the types of content generated by most generative AI models are images (e.g., by GANs and diffusion models) and text (e.g., by transformers in the form of LLMs). These types of content have what can be thought of as graceful degradation; if a picture is overlaid with some noise, is distorted, or part of it is missing, you can still appreciate much of it. Similarly, if a text has spelling or grammar errors, or even misses a sentence or two, you will probably be able to read most of it. This is also true of some types of game content, such as textures, skyboxes, sound and visual effects of any sort, and 3D models of "background" content such as bushes and other plants that are not meant to be interacted with.

In contrast, much of game content has a *functional* role. The functional requirements of the content means that it can be broken in a way that simple images or textures cannot. Typically, the functional requirements come about through the player's interaction with the content. For example, game levels need to be completable. But one or a few items added or removed (for example, the path may be blocked or the key to the door might be missing) might make it impossible to complete the level, however hard you try. Even if the broken level looks almost identical to a completable level, it is entirely different as it is failing its functionality requirement. In this sense, a game level is more like program code than like natural language text. Not all functionality is binary, however; for example, the functionality requirements for an RTS map include that resource access should be sufficiently challenging and fair.

This axis of our taxonomy thus distinguishes between **functional** content, which has functional roles, and **decorative** content. The word decorative here is not in any way meant as a pejorative. Masterpieces such as the *Mona Lisa* or the *Erdtree* in *Elden Ring* (Bandai Namco Entertainment, 2022) are decorative by this definition, even though they can be really important to the game. It simply means that it does not need to fulfill a function where it enables certain interactions. One way of thinking about it is that if you played a game blind, with no visual or aural input, only functional content would matter. Another way of thinking about it is that decorative content can often be produced without knowing the details of the mechanics of the game. In this book, we are mainly concerned with procedural generation of functional content, as much of decorative content generation falls within generative AI in general, and is not necessarily specific to games.

Spatiality: Non-spatial Versus Spatial

Many of the methods we will discuss in the next two chapters are specifically tailored to creating two- or three-dimensional levels, maps, or other artifacts. Those types of content are indeed very common, and most video games have some kind of spatial content. Other types of content, such as stories, music, and rules, do not by themselves

have any *spatiality* (though they can be intimately tied to a map, a level, or another object). In our taxonomy, the spatiality dimension ranges from **spatial** (levels, maps, 3D and 2D objects) to **non-spatial** (stories, music). There are types of content that have less than full spatiality as well; for example, quests are often represented as graphs that fit within a spatial representation of a level.

An important reason for the existence of this dimension in the taxonomy is that PCG methods often transfer between the more spatial types of content, but less so between the non-spatial types. For example, while every game has different mechanics, a method for generating 2D levels for one game can often be reconfigured to work for another game, or even for 3D levels or 2.5D maps. Many of the methods described in the next few chapters are tailored to spatial content, but some work for various types of non-spatial content as well.

7.2.2 Methods

PCG algorithms can be classified according to a number of properties such as their level of controllability, determinism and so on. This section outlines the five dimensions across which PCG methods can be classified. Figure 7.1 offers an illustration of the taxonomy that we discuss in this section.

Determinism: Stochastic Versus Deterministic

Our first distinction with regards to PCG methods concerns the amount of randomness in content generation. The right amount of variation in outcomes between different runs of an algorithm with identical parameters is a design decision. **Stochasticity** allows an algorithm (such as an evolutionary algorithm) to offer great variation, necessary for many PCG tasks, at the cost of controllability. While content diversity and expressivity are desired properties of generators, the effect of randomness on the final outcomes can only be observed and controlled after the fact. Completely **deterministic** PCG algorithms, on the other hand, can be seen as a form of data compression. A good example of deterministic PCG is the first-person shooter *.kkrieger* (.theprodukkt 2004), which manages to compress all of its textures, objects, music and levels together with its game engine in just 96 KB of storage space.

Controllability: Controllable Versus Non-controllable

The generation of content by PCG can be **controlled** in different ways or remain **non-controllable**. At one extreme, you would only supply a random seed for the pseudorandom number generator, effectively having no control over the output. More likely, you would want to supply a set of parameters that control the content generation along a number of dimensions. Random seeds were used when generating the world in *Minecraft* (Mojang, 2011), which means the same world can be regenerated if the same seed is used [1251]. A vector of content features was used in [1024] to

generate levels for *Infinite Mario Bros* (Persson, 2008) that satisfy a set of feature specifications. In the age of LLMs, there has been much work on trying to create content generators that are controlled by text prompts [365, 1099, 1146].

The dimensions of controllability and stochasticity are mostly orthogonal. One can find content generators that are highly controllable yet completely deterministic, for example based on cellular automata. Many content generators are stochastic and controllable, for example search-based ones. And many of the ad-hoc constructive generators found in roguelikes are stochastic but not controllable. However, a completely deterministic generator with no parameters at all would reduce to an algorithm that would always produce the same artifact. (Still, this may be useful, if you think of PCG as a form of data compression.)

Testing: Constructive Versus Generate-and-test

Another distinction may be made between algorithms that can be called **constructive** and those that can be described as **generate-and-test**. A constructive algorithm generates the content once, and is done with it; however, it needs to make sure that the content is correct or at least "good enough" as it is being constructed. An example of this approach is using fractals or cellular automata to generate terrains or grammars to generate levels (also refer to the corresponding PCG method sections in Chap. 8). A generate-and-test algorithm, instead, incorporates a testing mechanism. After a candidate content instance is generated, it is tested according to some criteria (e.g., is there a path between the entrance and exit of the dungeon, or does the tree have proportions within a certain range?). If the test fails, all or some of the candidate content is discarded and regenerated, and this process continues until the content is good enough. Evolutionary algorithms are a form of systematic generate and test algorithms, and the corresponding PCG paradigm is search-based PCG [1176]. Beyond the search-based paradigm we dedicate several sections of the next chapter of the book (see Chap. 8) to important generate-and-test PCG methods.

Learning: Authored Versus Trained

A fourth dimension in our taxonomy of PCG methods considers the existence or lack of an underlying learning process for content generation. On one end of the spectrum we meet methods that are entirely **authored** and do not involve any machine learning tasks. Methods such as cellular automata and L-systems belong to this category. On the other end of the spectrum, we meet methods that generate content via machine learning techniques. Any machine learning method—such as Markov models, neural network variants or reinforcement learning methods—that is able to generate game content fits within this category. The PCG via Machine Learning framework [419, 1104] is detailed in Sect. 8.5.

Iterativity: Single Pass Versus Iterative

The final dimension of our PCG method taxonomy considers the iterativity of a method. PCG methods can be classified as **single pass** when they generate content once and they don't iterate over their generated outcome. Examples of such methods include pre-trained neural networks or GANs. Alternatively, methods can be classified as **iterative** when they are required to iterate over their generated content. Approaches relying on cellular automata, diffusion models, reinforcement learning and LLMs that are prompted several times in a row are indicative examples of iterative PCG methods. AI methods may adopt an iterative approach to their training (such as backpropagation of errors or policy updates) or to their generated outcomes (such as the autoregressive nature of a transformer); the iterativity dimension of our taxonomy considers only the generative part. Search-based methods are always iterative, but many iterative methods are not search-based.

7.2.3 Roles

The generation of content algorithmically may take different roles within the domain of games. We can identify three axes across which PCG roles can be placed: *players*, *designers* and *time*. We envision PCG systems that consider designers while they generate content or they operate interdependently of designers; the same applies for players. Ultimately any of those PCG systems can run in design (**offline**) or play time (**runtime**), Fig. 7.2 visualizes the key roles of PCG in games across the dimensions of designer, player, and time, corresponding, respectively, to the dimensions of *autonomy*, *adaptivity* and *temporality* of our PCG taxonomy.

Regardless of the generation method used, game genre, content type or any other role, PCG can act either *autonomously* or as a *collaborator* in the design process. We refer to the former role as **autonomous** generation and the latter role as **mixed-initiative** generation. Further, we cover the **experience-driven** PCG role by which PCG algorithms consider the player experience in whatever they try to generate. As a result, the generated content is associated to the player and their experience. If PCG, instead, does not consider the player as part of the generation process it becomes **experience-agnostic**. The following sections outline the characteristics of each of the six PCG role dimensions with a particular emphasis on the mixed-initiative and the experience-driven roles.

Temporality: Runtime Versus Offline Generation

PCG techniques can be used to generate content in **runtime**, as the player is playing the game, allowing the generation of endless variations, making the game infinitely replayable and opening the possibility of generating player-adapted content, or **offline** during the development of the game or before the start of a game session. The

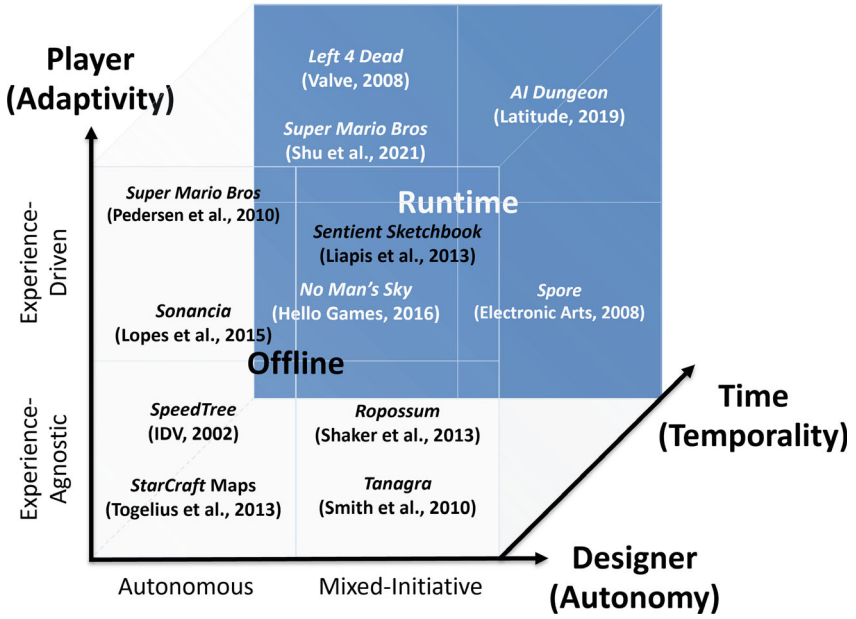


Fig. 7.2 The eight key PCG roles in games across the dimensions of designer (autonomy), player (adaptivity) and time (temporality). For each combination of roles the figure lists a number of indicative examples of tools, games or studies covered in this chapter. Examples of *offline* and *runtime* generation are depicted in black and white text, respectively

use of PCG for offline content generation is particularly useful when generating complex content such as environments and maps; several examples of that was discussed at the beginning of the chapter. An classic example of the use of experience-driven runtime content generation can be found in the game *Left 4 Dead* (Valve, 2008), a first-person shooter game that provides dynamic experience for each player by analyzing player behavior on the fly and altering the game state accordingly using PCG techniques [22, 107]. Roguelikes and roguelites, such as *Hades* (Supergiant, 2020), all build on runtime PCG.

A trend related to runtime content generation is the creation and sharing of player-generated content to an increasing number of games. Early days of player generated content can be found in games such as *LittleBigPlanet* (Sony Computer Entertainment, 2008) and *Spore* (Electronic Arts, 2008) providing a content editor (level editor in the case of *LittleBigPlanet* and the *Spore* Creature Creator) that allows players to edit and upload complete creatures or levels to a central online server where they can be downloaded and used by other players. Player-AI co-generated content can be found in recent games empowered by LLMs such as *AI Dungeon* (Latitude, 2019).

With respect to the other four roles of PCG in games, runtime generation is possible in any role within player adaptivity and designer autonomy. Runtime PCG, however, is more prominent when combined with autonomous generators compared

to mixed-initiative ones, unless player-AI co-creation occurs as part of a game as in *AI Dungeon* (Latitude, 2019). On the other hand, the offline generation of content can occur both autonomously and in an experience-agnostic manner. Further it is the dominant way to generate content in a mixed-initiative fashion (e.g., as in *Sentient Scketchbook* [640]) whereas it is less relevant for experience-driven generation as this PCG role occurs primarily on runtime by definition.

Autonomy: Mixed-initiative Versus Autonomous Generation

AI-assisted game design refers to the use of AI-powered tools to support the game design and development process. This is perhaps the AI research area which is most promising for the development of better games [1268]. In particular, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives.

We identify AI-assisted game design as the task of creating artifacts via the interaction of a human initiative and a computational initiative. The computational initiative is a PCG process and thus, we discuss this co-design approach under the heading of procedural content generation. Although the term **mixed-initiative** lacks a concrete definition [821], in this book we define it as the process that considers both the human and the computer *proactively* making content contributions to the game design task although the two initiatives do not need to contribute to the same degree [634, 1279]. Mixed-initiative PCG thus differs from other forms of co-creation, such as the collaboration of multiple human creators or the interaction between a human and non-proactive computer support tools (e.g., spell-checkers or image editors) or non-computer support tools (e.g., artboards or idea cards). The initiative of the computer can be seen as a continuum between the *no initiative* state, leaving full control of the design to the designer and having the computer program simply carry out the commands of the human designer, to the *full initiative* state which yields an autonomously creative system. Any state between the two is also possible as we will see in the examples covered next and as depicted in Fig. 7.3.

While the process of AI-assisted game design is applicable to any creative facet within game design [643] it is level design that has benefited the most from it. Within commercial-standard game development, we can find AI-based tools that allow varying degrees of computer initiative. On one end of the scale, level editors such as the *Garden of Eden Creation Kit* (Bethesda, 2009) or game engines such as the *Unreal Development Kit* (Epic Games 2009) leave most of creative process to the designer but they, nevertheless, boost game development through automating interpolations, pathfinding and rendering [1279]. On the other end of the computer initiative scale, PCG tools specialized on e.g., vegetation—*SpeedTree* (IDV, 2002)—or FPS levels—*Oblige* (Apted, 2007)—only require the designer to set a small amount of generation parameters and, thus, the generation process is almost entirely autonomous.

Within academia, the area of AI-assisted game design tools has seen significant research interest in recent years [599, 1291] with contributions mainly to the level design task across several game genres including platformers [1059], strategy games [199, 601, 639–641, 1279] (see Fig. 7.4a), open world games [1052], racing

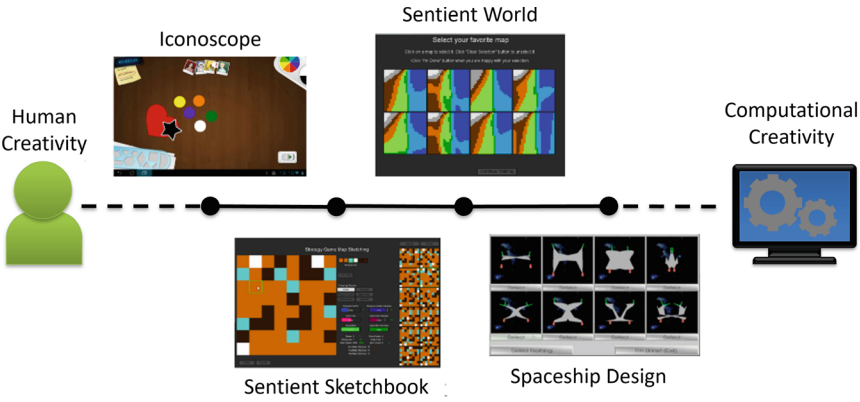


Fig. 7.3 The mixed-initiative spectrum between human and computer initiative (or creativity) across a number of mixed-initiative design tools discussed in this section. *Iconoscope* is a mixed-initiative drawing game [631], *Sentient Sketchbook* is a level editor for strategy games [640], Sentient World is mixed-initiative map editor [641] and Spaceship Design is a mixed-initiative (mostly computer initiative) tool powered by interactive evolution [638]. Adapted from [635]

games [179], casual puzzle games [682, 1022] (see Fig. 7.4b), horror games [661], first-person shooters [825], educational games [155, 631], mobile games [802], and adventure games [540]. The range of mixed-initiative game design tools expands to tools that are designed to assist with the generation of complete game rulesets such as the MetaGame [860], the RuLearn [1150] and the Ludocore [1057] tools to tools that are purposed to generate narratives [19, 800, 1110] and stories within games [597].

Any PCG approach could potentially be employed for mixed-initiative game design. The dominant methods that have so far being used, however, rely on evolutionary computation following the search-based PCG paradigm. Even though evolution, at fist sight, does not appear to be the most attractive approach for real-time processing and generation of content, it offers great advantages associated, in particular, with the stochasticity of artificial evolution, diversity maintenance and potential for balancing multiple design objectives. Evolution can be constrained to the generation of playable, usable, or, in general, content of particular *qualities* within desired design specifications. At the same time, it can incorporate metrics such as novelty [644] or surprise [404], for maximizing the *diversity* of generated content and thus enabling a change in the creative path of the designer [1279]. Evolution can be computationally costly, however, and thus, interactive evolution is a viable and popular alternative for mixed-initiative evolutionary-based generation (e.g., see [179, 638, 641, 825]).

Beyond artificial evolution, another class of algorithms that is relevant for mixed-initiative content generation is constraint solvers and constraint optimization. Methods such as answer set programming [124, 646] have been used in several AI-assisted level design tools including *Tanagra* [1059] for platformers and *Refraction* [155] for educational puzzle games. Artificial neural networks can also perform certain tasks in a mixed-initiative manner, such as performing "autocomplete" or level repair [504]

(a) *Sentient Sketchbook*

(b) *Ropossum*

Fig. 7.4 Examples of mixed-initiative level design tools. *Sentient Sketchbook* **a** offers map sketch suggestions to the designer via artificial evolution (see rightmost part of the image); the suggestions are evolved to either maximize particular objectives of the map (e.g., balance) or are evolved to maximize the novelty score of the map. In *Ropossum* **b** the designer may select to design elements of *Cut the Rope* (Chillingo, 2010) game levels; the generation of the remaining elements are left to evolutionary algorithms to design. Images used with permission

through the use of deep learning approaches such as stacked autoencoders. The goal here is to provide a tool that "fills in" parts of a level that the human designer does not want or have time to create, and correcting other parts to achieve further consistency. With the rise of ML-based generative methods we observe a rapid adoption of such approaches for mixed-initiative PCG in games. Recent studies go beyond the constructive and search-based paradigm and realize mixed-initiative design via text-to-level surrogate models [365, 751] and reinforcement learning [272] among other methods.

The role of **autonomous** generation is arguably the most dominant PCG role in games. The earlier parts of this chapter—and the majority of studies covered in the remainder of this book part—are dedicated to extensive discussions of PCG systems that do not consider the designer in their creative process. As a result we will not cover this PCG role in further detail here. What is important to be discussed, however, is the fuzzy borderline between mixed-initiative and autonomous PCG systems. It might be helpful, for instance, to consider autonomous PCG as the process by which the role of the designer starts and ends with an offline setup of the algorithm. For instance, the designer is only involved in the parameterization of the algorithm as in the case of *SpeedTree* (IDV, 2002). One might wish, however, to further push the borderline between autonomous and mixed-initiative generation and claim that generation is genuinely autonomous only if the creative process reconsiders and adapts the utility function that drives the content generation—thereby becoming creative in its own right. A static utility function that drives the generation is often referred to as *mere* generation within the **computational creativity** field [643].

While the line between autonomy and collaboration with **designers** is still an open research question, for the purposes of this book, we can safely claim that the PCG process is autonomous when the initiative of the designer is limited to algorithmic parameterizations or instantiations before the generation starts. Examples include prompting, such as when a designer interacts with a text-to-image model, setting the mutation rate of a search-based PCG method, or defining parameters of wave function collapse level generator.

Adaptivity: Experience-driven Versus Experience-agnostic Generation

Games offer one of the most representative examples of rich and diverse content creation applications and are elicitors of unique user experiences. In that regard, the **experience-driven** PCG (EDPCG) [1289, 1290] framework views game content as the *building block* of games and the generated games as the potentiators of player experience. One could define EDPCG as a generic and effective approach for the optimization of user experience via the adaptation of the experienced content. According to the experience-driven role of PCG in games **player experience** is the collection of affective patterns elicited, cognitive processes emerged and behavioral traits observed during gameplay [1287].

By coupling player experience with procedural content generation, the experience-driven perspective offers a player-centered role to PCG. Since games are composed of game content that—when played by a particular player—elicits experience patterns,

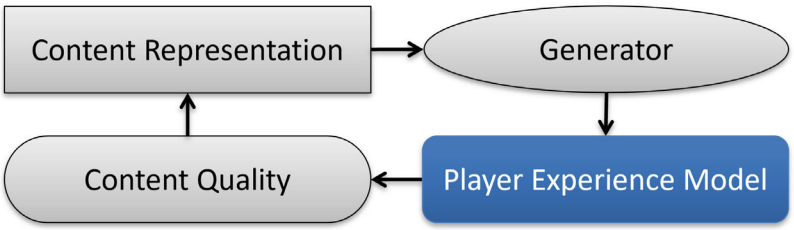


Fig. 7.5 The four key components of the experience-driven PCG framework [1289]. The core distinction of this framework compared to other PCG frameworks is that it considers a player experience model (depicted as a blue box)

one needs to assess the quality of the content generated (linked to the experience of the player), search through the available content, and generate content that optimizes the experience for the player (see Fig. 7.5). In particular, the key components of EDPCG are as follows:

- **Player experience modeling**: player experience is modeled as a function of game content and the player. One may use theory or data-driven methods to derive this function from multiple observations of the player as interacting with a game (or even the designer as interacting with a generator) such as behavioral patterns, facial expressions, physiological signals and so on. A detailed presentation of methods that one can use to derive experience models is presented in Part IV of this book.
- **Content quality**: the quality of the generated content is assessed and linked to the modeled experience. A number of methods are possible here and vary from ad-hoc designed measures, to AI agents that evaluate the content all the way to methods that request human feedback in implicit or explicit manners.
- **Content representation**: content is represented accordingly to maximize search efficacy and robustness. The way content is represented may vary from pixels on the screen (direct representation) all the way to the seed of a random number generator (indirect representation).
- **Content generator**: the generator searches through the generative space for content that optimizes the experience for the player according to the acquired model.

Each component of EDPCG has its own dedicated literature and the extensive review of each is covered in other parts of the book. In particular, player experience modeling is detailed in Part IV whereas the remaining three components of the framework will be covered in the presentation of the search-based PCG framework (see Sect. 8.2 of the PCG methods chapter). Detailed surveys and discussions about EDPCG are available in [1289, 1290].

Left 4 Dead (Valve, 2008) is an example of the use of experience-driven PCG in a commercial game where an algorithm is used to adjust the pacing of the game on the fly (in **runtime**) based on the player's *emotional intensity*. In this case, adaptive PCG is used to adjust the difficulty of the game in order to keep the player engaged [107].

Adaptive content generation can also be used with another motive such as the generation of more content of the kind the player seems to like. This approach was followed, for instance, in the *Galactic Arms Race* [432] game where the weapons presented to the player are evolved based on her previous weapon use and preferences. In another EDPCG study, El-Nasr et al. [311, 315] implemented a direct fitness function—derived from visual attention theory—for the procedural generation of lighting. The procedural Zelda game engine [441], a game engine designed to emulate the popular *The Legend of Zelda* (Nintendo, 1986–2023) action-RPG game series, is built mainly to support experience-driven PCG research.

Another example is the work by Pedersen et al. [859], who modified an open-source clone of the classic platform game *Super Mario Bros* (Nintendo, 1985) to allow for personalized level generation. The realization of EDPCG in this example is illustrated in Fig. 7.7. The first step was to represent the levels in a format that would yield an easily searchable space. A level was represented as a short parameter vector describing the number, size and placement of gaps which the player can fall through, and the presence or absence of a switching mechanic. The next step was to create a model of player experience based on the level played and the player's playing style. Data was collected from hundreds of players, who played pairs of levels with different parameters and were asked to rank which of two levels best induced each of the following user states: fun, challenge, frustration, predictability, anxiety, boredom. While playing, the game also recorded a number of metrics of the players' playing styles, such as the frequency of jumping, running and shooting. This data was then used to train neural networks to predict the examined user states using **evolutionary preference learning**. Finally, these player experience models were utilized to optimize game levels for particular players [1024]. Two examples of such levels can be seen in Fig. 7.6.

It is worth noting—as discussed in Chap. 10—that one may wish to further improve the models of experience of Mario players by including information about the player beyond gameplay [48] such as their head pose [1018] or facial expressions [97]. Instead of relying solely on the search-based paradigm EDPCG can be coupled with RL-based content generators of the PCGRL family. Recent studies introduced the experience-driven PCG via RL (EDRL) framework that is able to generate *fun* Super Mario Bros levels in an online fashion [1038, 1239]. Inspired by

(a) Human

(b) World-Champion AI

Fig. 7.6 Example levels generated for two different Mario players. The generated levels maximize the modeled *fun* value for each player. The level on top is generated for one of the experiment subjects that participated in [859] while the level below is generated for the world champion agent of the Mario AI competition in 2009

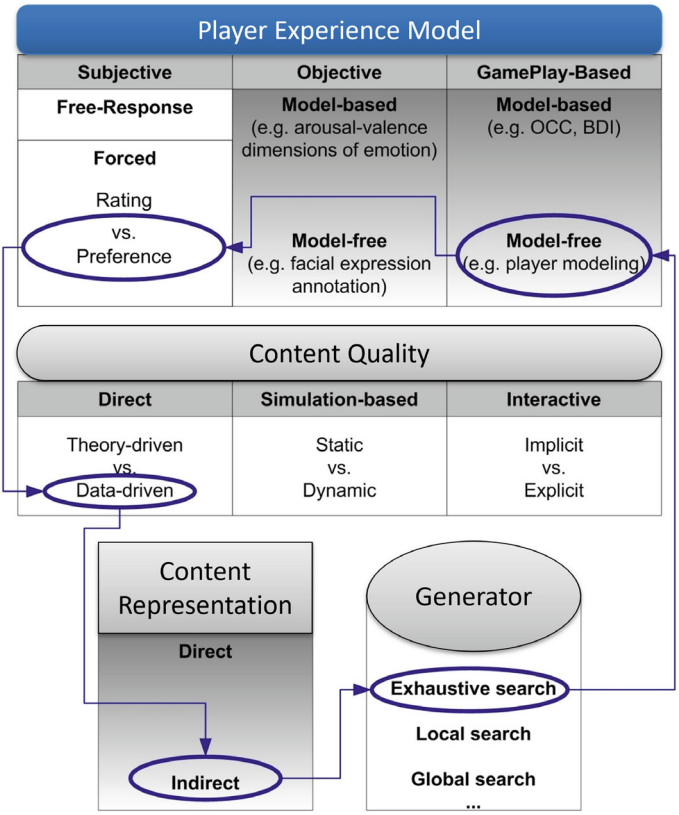


Fig. 7.7 Detailing the EDPCG framework through an example. The gradient grayscale-colored boxes represent a continuum of possibilities between the two ends of the box while white boxes represent discrete, exclusive options within the box. The blue arrows illustrate the EDPCG approach followed for the *Super Mario Bros* (Nintendo, 1985) example study [859, 1024]: Content quality is assessed via a direct, data-driven evaluation function which is based on a combination of a gameplay-based (model-free) and a subjective (pairwise preference) player experience modeling approach; content is represented indirectly and exhaustive search is applied to generate better content

the *theory of fun* [581], the notion of fun is quantified as moderate degrees of content diversity that, in turn, shapes the reward function used by EDRL.

On the other end of the adaptivity spectrum we meet **experience-agnostic** PCG methods. By experience-agnostic we refer to any PCG approach that does not consider the role of the player in the generation of content. But where should we set the boundary of involvement? When do we consider a player as part of the generation process and when don't we? While the borderline between experience-driven and experience-agnostic is not trivial to draw we define any PCG approach whose content quality function does not include an experience model or it does not interact with the player in any way during generation as experience-agnostic. As with the role of

autonomous PCG, the remaining two chapters of this part will take us through several examples of content generation that do not involve a player or a player experience model. To avoid being repetitive we will refer the reader to the PCG studies already covered in this chapter—and the remainder of this book part—that are outside the definition of experience-driven PCG.

7.3 Further Reading

Extensive versions of most of the material covered in this chapter can be found in dedicated chapters of the PCG in Games Textbook [1023]. In particular, all the methods and types of PCG in games are covered in further detail (see Chaps. 1–9 of [1023]). Considering the roles of PCG, the mixed-initiative and the experience-driven role of PCG are, respectively, detailed in [634] and [1025]. In addition to [634] the framework named *mixed-initiative co-creativity* [1279] provides a theoretical grounding for the impact of mixed-initiative interaction on the creativity of both designers and computational processes. Further, the original articles about the experience-driven PCG framework can be found in [1289, 1290].

7.4 Summary

This chapter viewed AI as a means for generating content in games. We defined procedural content generation as the algorithmic process of creating content in and for games and we explored the various benefits of this process. We then provided a general *taxonomy* about game content and its generation methods. We ended this chapter with a discussion on the possible roles a generator can take in games which are determined by temporality (offline versus runtime) and the degree to which they involve the designer (autonomous versus mixed-initiative) and/or the player (experience-agnostic versus experience-driven) in the process.

The next chapter of this part is dedicated to the various *methods* one can use to generate content in games. We detail methods that vary across the dimensions of our taxonomy: determinism (e.g., grammars versus evolution), controllability, testing capacities (i.e., constructive generation versus generate-and-test generation), learning capabilities (e.g., neural networks versus fractals) and iterativity (e.g., GANs versus large language models). Algorithms are placed and discussed under the categories of constructive methods (such as cellular automata and fractals), search-based, solvers, quality-diversity, self-supervised machine learning, and reinforcement learning.

Chapter 8
Methods for Generating Content

Abstract This chapter describes commonly used methods for PCG, both methods primarily used for runtime generation in roguelikes and other PCG-heavy games, and methods that are currently mostly explored in research labs. The chapter is organized according to the type of method. Constructive methods, which are diverse, content-specific, and generally fast, are presented first. Search-based methods, presented next, represent the application of evolutionary computation methods such as generic algorithms and quality diversity to PCG problems. The chapter ends with a thorough coverage of newer PCG methods built on machine learning, including supervised, self-supervised, and reinforcement learning.

There are many different algorithmic approaches to generating content for games. While many of these methods are commonly thought of as AI methods, others are drawn from graphics, theoretical computer science or even biology. The various methods differ also in what types of content they are suitable to generate. In this chapter, we discuss a number of PCG **methods** that we consider important and include constructive algorithms (Sect. 8.1), search-based (Sect. 8.2), quality-diversity (Sect. 8.3), solver-based (Sect. 8.4), machine learning (Sect. 8.5) and reinforcement learning methods (Sect. 8.6).

PCG methods presented here follow, in part, the AI method categorization we have adopted in Part I. An important clarification regards the terms *procedural content generation* and **generative AI** in games. While these terms point to largely overlapping sets of methods for creation of game content, generative AI (as a more recent term) focuses on a more narrow set of machine learning methods, and often the use of methods that have not been adapted specifically for games. If readers are interested primarily in generative AI methods in games—according to this definition at least—they are referred to Sects. 8.5 and 8.6 of this chapter.

8.1 Constructive Methods

What distinguishes constructive methods from other PCG methods is their lack of an underlying content *testing* mechanism. As we saw earlier in the PCG taxonomy presented in Chap. 7 a constructive algorithm generates content once, and it stops

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_8

there. Constructive methods, however, need to make sure that the content they generate is correct (or at least "good enough") while it is constructed. **Generate-and-test** PCG methods instead test the content they generate and they keep re-generating it to match some predefined quality specifications. Popular examples of constructive PCG methods that we cover in this section include grammars, cellular automata, noise, fractals, and the more recently popular wave function collapse method.

8.1.1 Grammar-Based Methods

Grammars are fundamental structures in computer science that also have many applications in procedural content generation. In particular, they are very frequently used for producing plants, such as trees, which are commonly featured in many different types of games. However, grammars have also been used to generate missions and dungeons [288, 289], rocks [264], underwater environments [3] and caves [704]. In these cases, grammars are employed as a constructive method, creating content without any evaluation functions or re-generation. Grammars can also be combined with search-based methods, so that the grammar expansion is represented as a genotype-to-phenotype mapping.

A (formal) **grammar** is a set of *production rules* for rewriting strings, i.e., turning one string into another. Each rule is of the form (symbol(s)) $\rightarrow$ (other symbol(s)). Here are some example production rules:

1. $A \rightarrow AB$
2. $B \rightarrow b$

Expanding a grammar is as simple as going through a string, and each time a symbol or sequence of symbols that occurs in the *left-hand side* of a rule is found, those symbols are replaced by the *right-hand side* of that rule. The process starts from a seed string called *axiom*. For example, if the initial axiom string is A, in the first rewriting step the A would be replaced by AB by rule 1, and the resulting string will be AB. In the second rewriting step, the A would again be transformed to AB and the B would be transformed to b using rule 2, resulting in the string ABb. The third step yields the string $ABbb$ and so on. A convention in grammars is that uppercase characters are nonterminal symbols, which are on the left-hand side of rules and therefore rewritten further, whereas lower-case characters are terminal symbols which are not rewritten further.

Starting with the axiom A the first few expansions of the two aforementioned production rules look as follows:

A
AB
ABA
ABAAB
ABAABABA

ABAABABAABAAB
ABAABABAABAABABAABABA
ABAABABAABAABABAABABAABAABABAABAAB

This particular grammar is an example of an **L-system**. L-systems are a class of grammars whose defining feature is parallel rewriting, and was introduced by the biologist Aristid Lindenmayer in 1968 explicitly to model the growth of organic systems such as plants and algae [650]. With time, they have turned out to be very useful for generating plants in games as well as in theoretical biology.

One way of using the power of L-systems to generate 2D (and 3D) artifacts is to interpret the generated strings as instructions for a turtle in **turtle graphics**. Think of the turtle as moving across a plane holding a pencil, and simply drawing a line that traces its path. We can give commands to the turtle to move forwards, or to turn left or right. For example, we can define the L-system alphabet {F, +, –, [,]} and then use the following key to interpret the generated strings:

- F: move forward a certain distance (e.g., 10 pixels).
- +: turn left 30 degrees.
- −: turn right 30 degrees.
- [: push the current position and orientation onto the stack.
-]: pop the position and orientation off the stack.

Bracketed L-systems can be used to generate surprisingly plant-like structures. Consider the L-system defined by the single rule $F \rightarrow F[-F]F[+F][F]$. Figure 8.1 shows the graphical interpretation of the L-system after 1, 2, and 3 rewrites starting from the single symbol F [822]. Minor variations of the rule in this system generate different but still plant-like structures, and the general principle can easily be extended to three dimensions by introducing symbols that represent rotation along the axis of drawing. For this reason, many standard packages for generating plants in game worlds are based on L-systems or similar grammars. For a multitude of beautiful examples of plants generated by L-systems refer to the book *The Algorithmic Beauty of Plants* by Prusinkiewicz and Lindenmayer [900].

There are many extensions of the basic L-system formalism, including nondeterministic L-systems, which can help with increasing diversity of the generated content, and context-sensitive L-systems, which can produce more complicated patterns. Formally specifying L-systems can be a daunting task, in particular as the mapping between the axiom and rules on the one hand and the results after expansion on the other are so complex. However, search-based methods can be used to find good axioms or rules, using for example desired height or complexity of the plant in the evaluation function [822]. Alternatively, L-systems can be used in conjunction with transformer architectures [614]. Deep neural networks with attention mechanisms can learn distributions of production rules and tree geometries encoded as L-strings, thereby, replacing the L-system rule-based generation process.

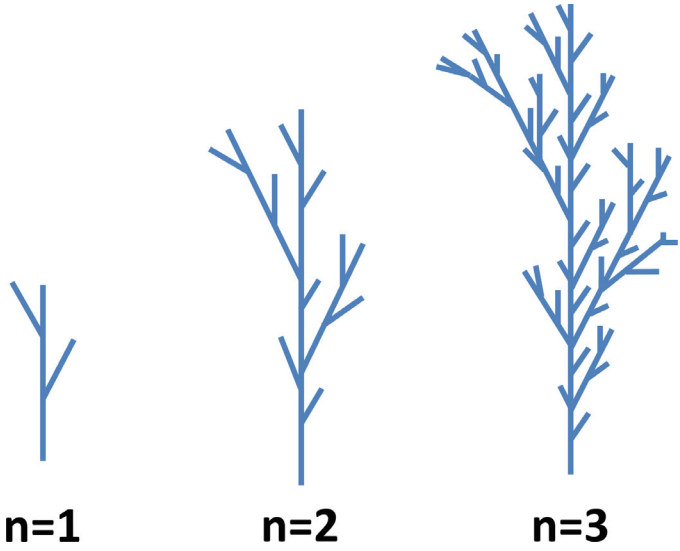
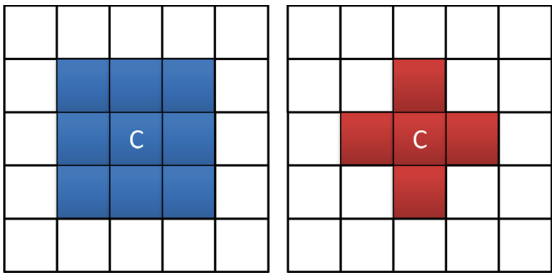


Fig. 8.1 Three rewrites of the bracketed L-system $F \rightarrow F[-F]F[+F][F]$. Image adapted from [822]

8.1.2 Cellular Automata

A cellular automaton (CA, plural: cellular automata) is a discrete model of computation which is widely studied in computer science, physics, complexity science and even some branches of biology. Cellular automata can be used to computationally model biological and physical phenomena such as growth, development, patterns, forms, or even emergence. While CA have been the subject of extensive study, the basic concept is very simple: cellular automata are a set of cells placed on a grid that change through a number of discrete time steps according to a set of rules; these rules rely on the current state of each cell and the state of its neighboring cells. The rules can be applied iteratively for as many time steps as we desire. The conceptual idea behind cellular automata was introduced by Ulam and von Neumann [807, 1224] back in the 1940s; it took about 30 more years, however, for us to see an application of cellular automata that showed their potential beyond basic research. That application was the two-dimensional cellular automaton designed in Conway's *Game of Life* [230]. The *Game of Life* is a zero-player game; its outcome is not influenced by the player's input throughout the game and it is solely dependent on its initial state (which is determined by the player).

A cellular automaton contains a number of cells represented in any number of dimensions; most cellular automata, however, are either one-dimensional (vectors) or two-dimensional (matrices). Each cell can have a finite number of **states**; for a binary cellular automaton, the cell can be *on* or *off*. A set of cells surrounding each cell define its **neighborhood**. The neighborhood defines which cells around



(a) Moore (b) von Neumann

Fig. 8.2 Two popular types of 2D neighborhoods used in cellular automata. Depicted neighborhoods of cell C are of size 1

a particular cell affect the cell's future state and its size can be represented by any integer number greater than 1. For one-dimensional cellular automata, for instance, the neighborhood is defined by the number of cells to the left or the right of the cell. For two-dimensional cellular automata, the two most common neighborhood types are the **Moore** and the **von Neumann** neighborhoods depicted in Fig. 8.2. The former neighborhood type is a square consisting of the cells surrounding a cell, including those surrounding it diagonally; for example, a Moore neighborhood of size 1 contains the eight cells surrounding each cell. The latter neighborhood type, instead, forms a cross of cells which are centered on the cell considered. For example, a von Neumann neighborhood of size 1 consists of the four cells surrounding the cell, above, below, to the left and to the right. (see Fig. 8.2).

At the beginning of an experiment (at time $t = 0$) we initialize the cells by assigning a state for each one of them. At each time step t we create a new generation of cells according to a rule or a mathematical function which specifies the new state of each cell given the current state of the cell and the states of the cells in its neighborhood at time $t - 1$. Normally, the rule for updating the state of the cells remains the same across all cells and time steps (i.e., it is static) and is applied to the whole grid.

Cellular automata have been used extensively in games for modeling environmental systems like heat and fire, rain and fluid flow, pressure and explosions [348, 1113] and in combination with influence maps for agent decision making [1114, 1115]. Another use for cellular automata has been for thermal and hydraulic erosion in procedural terrain generation [824]. Of particular interest for the purposes of this section is the work of Johnson et al. [515] on the generation of infinite cave-like dungeons using cellular automata. The motivation in that study was to create an infinite cave-crawling game, with environments stretching out endlessly and seamlessly in every direction. An additional design constraint was that the caves are supposed to look organic or eroded, rather than having straight edges and angles. No storage medium is large enough to store a truly endless cave, so the content must be generated at runtime, as players choose to explore new areas. The game does not scroll but instead presents the environment one screen at a time, which offers a time window of a few hundred milliseconds in which to create a new room every time the player

exits a room. The CA method used the following four parameters to control the map generation process:

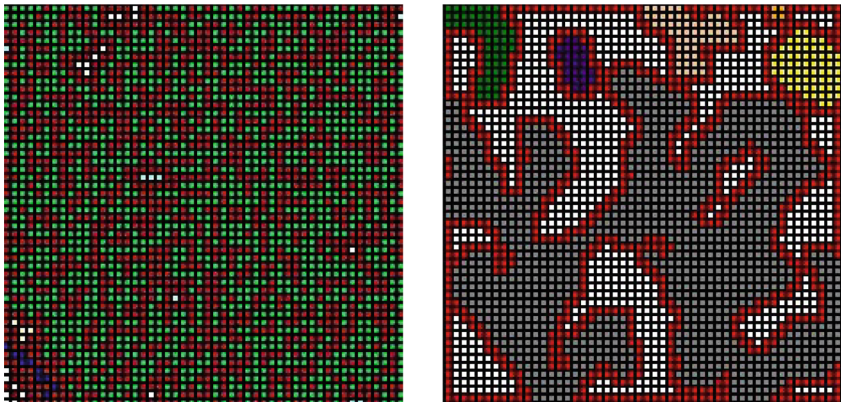
- a percentage of rock cells (inaccessible areas) at the beginning of the process;
- the number of CA generations (iterations);
- a neighborhood threshold value that defines a rock;
- the Moore neighborhood size.

In the dungeon generation implementation presented in [515], each room is a 50×50 grid, where each cell can be in one of two states: *empty* or *rock*. Initially, the grid is empty. The generation of a single room is as follows.

1. The grid is "sprinkled" with rocks: for each cell, there is probability (e.g., 50%) that it is turned into rock. This results in a relatively uniform distribution of rock cells.
2. A number of CA generations (iterations) are applied to this initial grid. For each generation the following simple rule is applied to all cells: a cell turns into rock in the next iteration if at least T (e.g., 5) of its neighbors are rock, otherwise it will turn into free space.
3. For aesthetic reasons the rock cells that border on empty space are designated as "wall" cells, which are functionally rock cells but look different.

The aforementioned simple procedure generates a surprisingly lifelike cave-room. Figure 8.3 shows a comparison between a random map (sprinkled with rocks) and the results of a few iterations of the cellular automaton. But while this process generates a single room, a game would normally require a number of connected rooms. A generated room might not have any openings in the confining rocks, and there is no guarantee that any exits align with entrances to the adjacent rooms. Therefore, whenever a room is generated, its immediate neighbors are also generated. If there is no connection between the largest empty spaces in the two rooms, a tunnel is drilled between those areas at the point where they are least separated. A few more iterations of the CA algorithm are then run on all nine neighboring rooms together, to smooth out any sharp edges. Figure 8.4 shows the result of this process, in the form of nine rooms that seamlessly connect. This generation process is extremely fast, and can generate all nine rooms in less than a millisecond on a modern computer.

A similar approach to that of Johnson et al. is used for dungeon generation in the *Galak-Z* (17-bit, 2016) game [13]. In that game, cellular automata generate the individual rooms of levels and the rooms are tied together via a variation of a Hilbert curve, which is a continuous fractal space-filling curve [448]. *Galak-Z* shows an alternative way of combining CA with other methods for achieving the desired map generation result.



(a) A random map (b) A map generated with cellular automata

Fig. 8.3 Cave generation: Comparison between a CA and a randomly generated map. The CA parameters used are as follows: the CA runs for four generations; the size of the Moore neighborhood considered is 1; the threshold value for the CA rule is 5 ($T = 5$); and the percentage of rock cells at the beginning of the process is 50% (for both maps). Rock and wall cells are represented by white and red color respectively. Colored areas represent different tunnels (floor clusters). Images adapted from [515] with permission

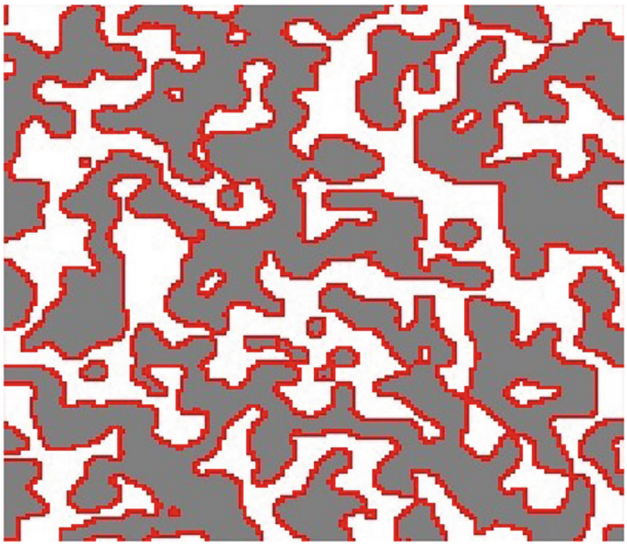


Fig. 8.4 Cave generation: a 3×3 base grid map generated with CA. Rock and wall cells are represented by white and red color respectively; gray areas represent floor. Moore neighborhood size is 2, T is 13, number of CA iterations is 4, and the percentage of rock cells at the initialization phase is 50%. Image adopted from [515]

In summary, CA are very fast constructive methods that can be used effectively to generate certain kinds of content such as terrains and levels (as e.g., in [515]), but they can also be potentially used to generate other types of content. The greatest benefits a CA algorithm can offer to a game content generator is that it depends on a small number of parameters and that it is intuitive and relatively simple to grasp and implement. However, the algorithm's constructive nature is the main cause for its disadvantages. For both designers and programmers, it is not trivial to fully understand the impact that a single parameter may have on the generation process, since each parameter affects multiple features of the generated output. While the few parameters of the core algorithm allow for a certain degree of controllability, the algorithm cannot guarantee properties such as playability or solvability of levels. Further, it is not possible to design content that has specific requirements, e.g., a map with a certain connectivity, since gameplay features are disjoint from the control parameters of the CA. Thus, any link between the CA generation method and gameplay features would have to be created through a process of trial and error. In other words, one would need to resort to preprocessing or a generate-and-test approach.

8.1.3 Noise and Fractals

One class of algorithms that are very frequently used to generate heightmaps and textures are noise algorithms, many of which are fractal algorithms, meaning that they exhibit scale-invariant properties. Noise algorithms are usually fast and easy to use but lack in controllability.

Both textures and many aspects of terrains can fruitfully be represented as two-dimensional matrices of real numbers. The width and height of the matrix map to the x and y dimensions of a rectangular surface. In the case of a texture, this is called an **intensity map**, and the values of cells correspond directly to the brightness of the associated pixels. In the case of terrains, the value of each cell corresponds to the height of the terrain (over some baseline) at that point. This is called a **heightmap**. If the resolution with which the terrain is rendered is greater than the resolution of the heightmap, intermediate points on the ground can simply be interpolated between points that do have specified height values. Thus, using this common representation, any technique used to generate noise could also be used to generate terrains, and vice versa—though they might not be equally suitable.

It should be noted that in the case of terrains, other representations are possible and occasionally suitable or even necessary. For example, one could represent the terrain in three dimensions, by dividing the space up into **voxels** (cubes) and computing the three-dimensional voxel grid. An example is the popular open-world game *Minecraft* (Mojang, 2011), which uses unusually large voxels. Voxel grids allow structures that cannot be represented with heightmaps, such as caves and overhanging cliffs, but they require a much larger amount of storage.

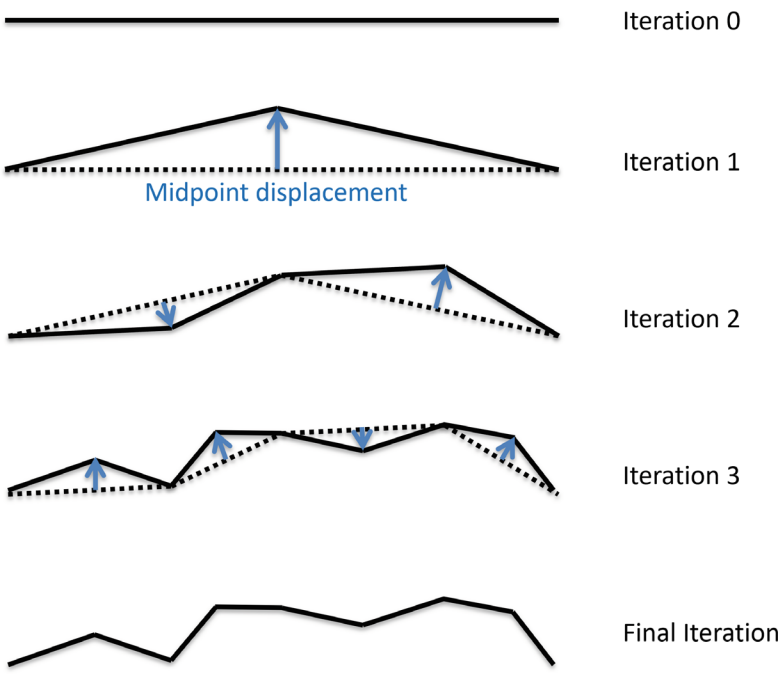


Fig. 8.5 The Midpoint Displacement algorithm visualized

Fractals [302, 824] such as **midpoint displacement algorithms** [74] are in common use for real-time map generation. Midpoint displacement is a simple algorithm for generating two-dimensional landscapes (seen from the side) by repeatedly subdividing a line. The procedure is as follows: start with a horizontal line. Find the midpoint of that line, and move the line up or down by a random amount, thus breaking the line in two. Now do the same thing for both of the resulting lines, and so on for as many steps as you need in order to reach sufficient resolution. Every time you call the algorithm recursively, lower the range of the random number generator somewhat (see Fig. 8.5 for an example).

A useful and simple way of extending the midpoint displacement idea to two dimensions (and thus creating two-dimensional heightmaps which can be interpreted as three-dimensional landscapes) is the **Diamond-Square algorithm** (also known as "the cloud fractal" or "the plasma fractal" because of its frequent use for creating such effects) [349]. This algorithm uses a square 2D matrix with width and height $2^n + 1$. To run the algorithm you normally initialize the matrix by setting the values

of all cells to 0, except the four corner values which are set to random values in some chosen range (e.g., [−1, 1]). Then you perform the following steps:

1. *Diamond step*: Find the midpoint of the four corners, i.e., the most central cell in the matrix. Set the value of that cell to the average value of the corners. Add a random value to the middle cell.
2. *Square step*: Find the four cells in between the corners. Set each of those to the average value of the two corners surrounding it. Add a random value to each of these cells.

Call this method recursively for each of the four subsquares of the matrix, until you reach the resolution limit of the matrix (3×3 sub-squares). Every time you call the method, reduce the range of the random values somewhat. The process is illustrated in Fig. 8.6.

There are many more advanced methods for generating fractal noise, with different properties. One of the most important is **Perlin noise**, which has some benefits over Diamond Square [873]. These algorithms are covered thoroughly in books that focus on texturing and modeling from a graphics perspective [302].

8.1.4 Wave Function Collapse

Wave function collapse (WFC) is an algorithm blending aspects of constraint satisfaction and machine learning that converts tiny images used for its training into functional content for games. The vanilla version of WFC was invented by the indie game developer Gumin in 2016 [414], and has strong similarities to the earlier *Model Synthesis* method by Paul Merrell [752, 753]. Since its inception, the method has seen a significant adoption and use from small-scale independent game development projects all the way to larger scale game productions such as *Bad North* (Raw Fury, 2018) and *Townscaper* (Oskar Stålberg, 2021). Given that this iterative method features constraint satisfaction and machine learning components it could be placed either under the solver-based or the ML-based PCG category. We decided, however, to place it under the constructive methods section as WFC normally does not evaluate content for any particular objective (i.e., strictly speaking not a search-based or a PCGML method) and does not attempt to satisfy any global constraints such as feasibility of playability but rather local ones such as connectivity (i.e., strictly speaking not a solver-based method either).

WFC generates content that is *locally similar* to an input bitmap provided by the designer. In that sense one of the core functionalities of WFC resembles the way cellular automata work as we saw earlier. “Local similarity” could mean, for instance, that the generated output should contain only those patterns of pixels that

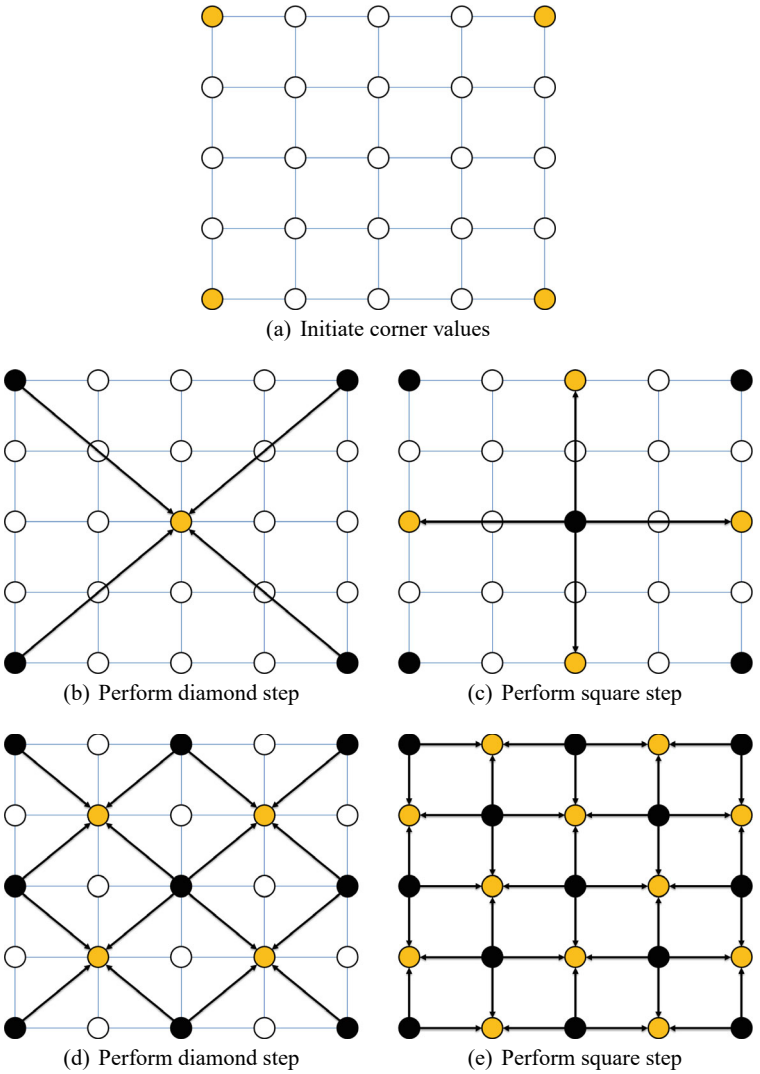


Fig. 8.6 The Diamond-Square algorithm visualized as a simple example of five steps. Images adapted from a figure by Christopher Ewin, licensed under CC BY-SA 4.0. White circles indicate cells that have not been altered yet and whose values are normally set to zero. Yellow circles indicate cells considered currently by the algorithm whose values are adjusted within a predefined range (e.g., $[-1, 1]$). Black circles depict cells whose values have been adjusted already during a diamond or a square step

are present in the input or that the distribution of input patterns should be similar to the distribution of patterns over many generated outcomes.

The initialization process of WFC is loosely inspired by quantum mechanics [548]. Specifically, the algorithm is given a number of tiles (or it is given an input image through which WFC identifies the unique tiles existent) and a set of adjacency constraints for those tiles. For example a designer may provide a set of road and field tiles and also provide constraints such as that a road tile cannot be connected to a field tile. Alternatively, a designer may provide an input bitmap image and a patch size N and ask WFC to derive the unique N×N patches (i.e., tiles) existent and also derive the adjacency constraints based on the color of the derived tiles. Figure 8.8 offers an illustrative example of the WFC process as a whole; the images and visualization concept depicted are inspired by Stephen Sherratt's blog.[1] Given the tiles and the adjacency constraints the output of WFC is essentially a grid of a desired dimension consisting of cells of tiles. Tiles are placed into cells, and each cell maintains a list of tiles that it may contain given the constraints imposed by its neighboring cells. Based on the original implementation introduced by Gumin [414] but also inspired by [548], next we list the core steps of the algorithm:

Given size of patch N and an input image.

1. Observe the input.
   a. Identify unique tiles existent on the input image (if those are not given already).
   b. If not given, count N×N patterns by classifying the identified tiles into patterns via a convolution of their local neighborhood [548].
   c. You might want to augment the derived patterns via rotations, reflections or other standard image data augmentation methods.
   d. Create a matrix of pattern adjacency constraints based on how patterns overlap with neighboring tiles.
2. Create the *wave*: an array with the length of the output image. Each element of the wave represents a state of N×N patterns in the output.
3. Initialize the wave in the completely unobserved state.
4. Repeat steps 5-6 until termination condition:
5. **Observation**: select a wave element (i.e., an N×N pattern or a cell) to collapse. Selection is based on a heuristic such as the cell with the lowest entropy of potential tiles (i.e., the cell with the least number of tile options). Collapse the chosen element into a definite state. The tile chosen is among those allowed for that cell according to the adjacency constraints. If all cells have been observed or there are cells without available tiles to choose from, go to 7.

[1] Available at: https://www.gridbugs.org/wave-function-collapse/.

6. **Propagation**: The collapse initiates a propagation process that starts from the neighboring cells, gradually eliminating tile selections based on the adjacency constraints. The propagation process stops when there are no more alterations to be made.
7. **Termination condition**: either the wave has collapsed (all wave elements are in a completely observed state) and you return the output, or a cell does not have any allowed tile choices left after the propagation step and you don't return any output image.

What makes WFC a particularly powerful algorithm is that the collapsing step (i.e., which cell to collapse and which tile to collapse it into) can be selected by a human. This allows WFC to form part of mixed-initiative tools for level design [14, 601] and even allow for the real-time creative level design as in *Townscaper* (Oskar Stålberg, 2021). What is also important to note is that WFC can even operate with no input image thereby helping designers work with specific tiles; in *Townscaper* (Oskar Stålberg, 2021) for instance WFC generates the adjacency matrix by matching the tile vertices [1139] (see Fig. 8.7). For a demonstration see this 2D demo by Oskar Stålberg[2] in which WFC generates the adjacency matrix based on the color on the edges. The process resembles the principles of the *Wang Tiles* [1234] by which a

Fig. 8.7 *Townscaper* (Oskar Stålberg, 2021) is a seminal example of level generation via WFC. The game first procedurally generates quadrilateral grids to potentially yield more realistic looking buildings. Every time a player wishes to place a new building block at a particular tile WFC is used to determine which building tiles are available based on the shape of the current level created by the player. Screenshot taken by the authors

[2] https://oskarstalberg.com/game/wave/wave.html.

number of square tiles—modeled visually with a color on each side—are arranged side by side with matching colors.

8.2 Evolutionary Algorithms and Search-Based PCG

The search-based approach to PCG [1176] has been very popular in academic research on game content generation. In search-based procedural content generation, an evolutionary algorithm or some other stochastic search or optimization algorithm is used to search for content with the desired qualities. The basic metaphor is that of design as a search process: a good enough solution to the design problem exists within some space of solutions, and if we keep iterating and tweaking one or many possible solutions, keeping those changes which make the solution(s) better and discarding those that are harmful, we will eventually arrive at the desired solution.

The metaphor of search has been used to describe the *manual* design process in many different disciplines: for example, Will Wright—designer of *SimCity* (Electronic Arts, 1989) and *The Sims* (Electronic Arts, 2000)—described the game design process as search in his talk at the 2005 Game Developers Conference. In other specialized domains such as architecture, the design process can be conceptualized as search and implemented as a computer program [100, 1254]. In this section, we focus on the non-metaphorical use of search, where it is performed by a computer algorithm.

The core components of the search-based approach to solving a content generation problem are the following:

- A **search algorithm**. This is the "engine" of a search-based method. Often relatively simple evolutionary algorithms work well enough, however sometimes there are substantial benefits to using more sophisticated algorithms that take e.g., constraints into account, or that are specialized for a particular content representation.
- A **content representation**. This is the representation of the artifacts you want to generate, e.g., levels, quests or winged kittens. The content representation could be anything from an array of real numbers such as a chromosome or a latent vector to a graph to a string. The content representation defines (and thus also limits) what content can be generated, and determines whether effective search is possible; see also the discussion about representation in Chap. 2.
- One or more **evaluation functions**. An evaluation function is a function from an artifact (an individual piece of content) to a number indicating the quality of the artifact. The output of an evaluation function could indicate e.g., the playability of a level, the intricacy of a quest or the aesthetic appeal of a winged kitten. An evaluation function could even assess the degree of novelty [644] or surprise [404] of the generated content. Crafting an evaluation function that reliably measures the aspect of game quality that it is meant to measure is often among the hardest

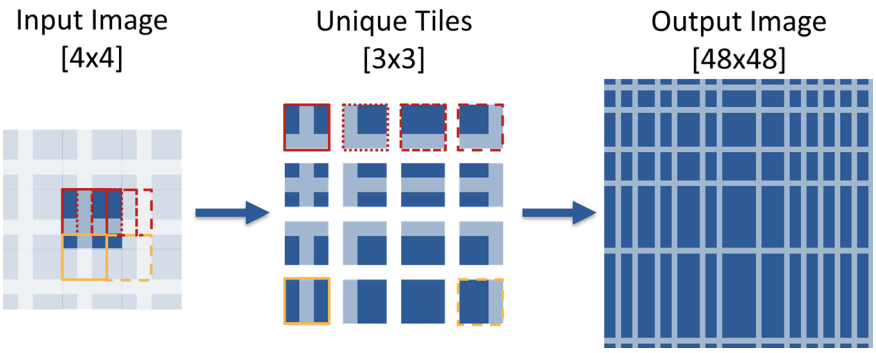


Fig. 8.8 Example illustrating the process of generating an output image from a given input image via WFC. In this example the designer provides a 4×4 pixel bitmap image and requests a 48×48 output image that is generated through patches of size 3. During its initialization process WFC derives the 3×3 unique tiles (patches) available by observing the image similarly to a convolutional filter (i.e., WFC detects all 3×3 tiles available in this 4×4 input image). When the process observes pixels beyond the edge of the image (see for instance the 3rd, 4th, 12th and 16th tile in the example provided) the method assumes that the image repeats forever across all directions (observe the underlying pale colored image on the left). The adjacency constraints in this example are derived based on color: blue and light blue edges can only touch their own colors. Light blue pixels could represent roads and blue pixels could represent fields for instance. Once the 16 unique tiles have been detected the steps of *observation* and *propagation* iterate to generate the output image of predetermined size 48×48 (i.e., the wave). More details on these steps can be found in the WFC pseudocode presented in Sect. 8.1.4

tasks in developing a search-based PCG method. Refer also to the discussion about utility in Chap. 2.

Let us look at some of the choices for **content representations**. To take a very well-known example, a level in *Super Mario Bros* (Nintendo, 1985) might be represented in any of the following ways:

1. Directly, as an ASCII map, where each character in the genotype corresponds to one "block" in the phenotype (e.g., bricks, question mark blocks, etc.). The different blocks of the level can also be represented as text symbols that feed a large language model that represents the level as in [1100].
2. Somewhat indirectly, as a list of the positions and properties of the different game entities such as enemies, platforms, gaps and hills (an example of this can be found in [1019]).
3. More indirectly, as the latent vector of any trained representation of *Super Mario Bros* levels. Examples include the latent vector of a GAN as in [1038, 1223], or the latent space of an autoencoder [750].
4. Even more indirectly, as a repository of different reusable patterns (such as collections of coins or hills), and a list of how they are distributed (with various transforms such as rotation and scaling) across the level map (an example of this can be found in [1071]).

5. Very indirectly, as a list of desirable properties such as the number of gaps, enemies, or coins, the width of gaps, etc. (an example of this type of representation can be found in [1024]).
6. Most indirectly, as a random number seed.

While it clearly makes no sense to evolve random number seeds (it is a representation with no locality whatsoever) the other levels of abstraction can all make sense under certain circumstances. The fundamental tradeoff here is between cost and **locality**. On the one hand, more direct, more fine-grained representations are more expensive computationally but tend to have higher locality, which implies that the utility values (e.g., the fitness) of neighboring points in the search space are closer to each other. On the other hand, less direct, more coarse-grained representations probably yield lower locality but come with smaller search spaces. Smaller search spaces of high locality are in general easier to search. However, larger search spaces would (all other things being equal) allow for more different types of content to be expressed, or in other words increase the *expressive range* of the generator.

The third important choice when designing a search-based PCG solution is the **evaluation function** known as the *fitness function* under the evolutionary computation paradigm. The evaluation function assesses all candidate solutions, and assigns a score (a fitness value or evaluation value) to each candidate. This is essential for the search process; if we do not have a good evaluation function, the evolutionary process will not work as intended and will not find good content. It could be argued that designing an entirely accurate content quality evaluation is "**AI-complete**" in the sense that to really understand what a human finds fun, you actually have to be a human or understand humans in depth. However, we can get pretty far with well-designed domain-specific heuristics. In general, the evaluation function should be designed to model some desirable quality of the artifact, e.g., its playability, regularity, entertainment value, etc. The design of an evaluation function depends to a great extent on the designer and what they think are the important aspects that should be optimized and how to formulate that.

In search-based PCG, we can distinguish between three classes of evaluation functions: direct, simulation-based, and interactive.

- **Direct** evaluation functions map the generated content (or features extracted from it) directly to a content quality value and, in that sense, they base their fitness calculations directly on the phenotype representation of the content. No simulation of the gameplay is performed during the mapping. Some direct evaluation functions are hand-coded, and some are learned from data. Direct evaluation functions are fast to compute and often relatively easy to implement, but it is sometimes hard to devise a direct evaluation function for some aspects of game content. Example features include the placement of bases and resources in real-time strategy games [1166], the size of the ruleset in strategy games [687] or the current mood of the game scene based on visual attention theory [311]. The mapping between features and fitness might be contingent on a model of the playing style, preferences or affective state of players. Such forms of fitness evaluations have been used to personalize player experience in games. In [1018, 1024], for example, the

authors trained neural networks to predict the player experience (such as challenge, frustration and enjoyment) given a playing style and the characteristics of a level as inputs. These trained neural networks were used as surrogate models of level quality yielding levels that, for example, would maximize enjoyment while the minimizing frustration for the player. Or the exact opposite, if you wish.

- **Simulation-based** evaluation functions use AI agents that play through the generated content in order to estimate its quality. Statistics are calculated about the agents' behavior and playing style and these statistics are then used to score game content. The type of the evaluation task determines the area of proficiency of the AI agent. If content is evaluated on the basis of playability, e.g., the existence of a path from the start to the end in a maze or a level in a 2D platform game, then AI agents should be designed that excel in reaching the end of the game. On the other hand, if content is optimized to maximize a particular player experience, then an AI agent that imitates human behavior should generally be adopted. An example study that implements a human-like agent for assessing content quality is presented in [1157] where neural-network-based controllers are trained to drive like human players in a car racing game and then used to evaluate the generated tracks. Each track generated is given a fitness value according to statistics calculated while the AI controller is playing. Another example of a simulation-based evaluation function is measuring the average fighting time of bots in a first-person shooter game [180]. In that study, levels were simply selected to maximize the amount of time bots spent on the level before killing each other. Similarly, Cachia et al. [158] used the same combat time metric (alongside other static evaluations of the map) to evolve two-floor shooter levels.
- **Interactive** evaluation functions evaluate content based on interaction with a human, so they require a human "in the loop". Examples of this method can be found in the work by Hastings et al. [432], who implemented this approach by evaluating the quality of the personalized weapons evolved implicitly based on how often and how long the player chooses to use these weapons. Figure 8.9 presents two examples of evolved weapons for different players. Cardamone et al. [179]

(a) (b)

Fig. 8.9 Examples of evolved weapons in *Galactic Arms Races*. Images obtained from [431] with author permission

also used this form of evaluation to score racing tracks according to the users' reported preferences. Ølsted et al. used the same approach to design levels for first-person shooter games [825]. The former study is an example of an *implicit* collection of data, i.e., where players do not report anything about their preferences; in the latter study, however, players' preferences are collected *explicitly*. The challenge of explicit data collection is that, if not well integrated, it requires the gameplay session to be interrupted. This method, however, provides a reliable and accurate estimator of player experience, as opposed to implicit data collection, which is usually noisy and based on assumptions. Hybrid approaches are sometimes employed to mitigate the drawbacks of these two methods by collecting information across multiple modalities such as combining player behavior with eye gaze and/or skin conductance. Early applications of this approach can be found in biofeedback-based camera viewpoint generation [714], level generation [661, 1018] and visuals generation in physical interactive games [1276].

Search-based methods have extremely broad applicability, as global search and optimization methods can be used to construct almost any kind of game content. This generality, however, comes with a number of drawbacks. One is that it is generally a rather slow method, requiring the evaluation of a large number of candidate content items. The time required to evolve a good solution can also not be precisely predicted, as there is no runtime guarantee for evolutionary algorithms. This might make search-based PCG solutions unsuitable for time-critical content generation, such as when you only have a few seconds to serve up a new level in a game. It should also be noted that the successful application of search-based PCG methods relies on judicious design choices when it comes to the particular search algorithm, representation and evaluation function.

As we will see in what follows, there are several algorithms that are generally better suited than evolutionary algorithms for generating game content of specific types. However, none of them have the *versatility* of the search-based approach, with abilities to incorporate all kinds of objectives and constraints. As we will see, many other algorithms for content generation can also be combined with search-based solutions, so that evolutionary algorithms can be used to search the parameter space of other algorithms such as **latent variable evolution** [1038, 1223].

8.3 Quality-Diversity Methods

Throughout its long history in video game development PCG algorithms have been been engineered to generate content that is not only playable and fun—i.e., content of high *quality*—but also *diverse*, meaning that it does not all look and play the same. As we saw with earlier PCG methods, game content normally needs to satisfy a number of playability constraints, such as reachable platforms in a platformer level generator, before it can be deployed in a game. To be assessed of high quality, content also needs to be entertaining and challenging enough.

While maintaining high degrees of quality, PCG methods are also required to generate entirely new experiences for the player. Players interacting with game content generators expect the algorithms to be able to produce novel and surprising content like an awkwardly looking new weapon, a new level with weird aesthetics never seen before and so on. The expectations on content diversity are so high that may force players to leave the game if such expectations are not met. For example, the initial player backlash against *No Man's Sky* (Hello Games, 2016) was partly due to the perceived similarity in the generated worlds of the game [689].

It should be obvious by now that content of low quality may make the game unnecessarily easy or challenging (at best), or impossible to complete (at worst), while content with insufficient diversity can lead to player fatigue. This calls for a different PCG framework that can emphasize quality and diversity at the same time. The paradigm of *procedural content generation through quality-diversity* (PCGQD) in [403] can be viewed as a subset of search-based PCG [1175, 1176] that relies on quality-diversity algorithms. As detailed in Sect. 3.3.2 quality-diversity is a novel family of evolutionary search algorithms that simultaneously maintain both the quality and the diversity of their solutions. QD algorithms such as **MAP-Elites** [781] reward divergence (e.g., novelty or surprise) while maintaining control of the solutions' quality through constraints. In the remainder of this section we will outline the core components of QD algorithms and we will identify the strengths of PCGQD over other PCG methods. We will also go through a few indicative case studies of the use of QD algorithms for the procedural generation of game content.

PCGQD algorithms may employ two main strategies to guide evolutionary search towards improving the **quality** of generated content. First, a QD algorithm can reward quality via a local competition between individuals within the same niche. Algorithms adopting this method, such as *novelty search via local competition* [618], rely on a distance metric that decides which content candidates a generated individual compares itself with. Second, quality can be assessed through a pre-determined set of constraints that each generated content solution needs to satisfy, as in *constrained novelty search* [644]. Such constraints can be either "hard" (e.g., can a player complete this game level or not) or "soft" (e.g., the ideal number of turns in this racing track is 3).

When it comes to the **diversity** component of PCGQD we can identify two main ways an algorithm can achieve it. First, diversity can be achieved by rewarding the distance in the behavior space of generated individuals. The individual is some type of content and its behavior represents any property that characterizes the content such as the length of a generated level or the height of a generated building. Normally one needs to design a distance function that determines how different the behavior of the content under assessment is from the rest of content that has been generated already. The individuals we compare against can be the current population and an archive of past individuals, as in novelty search [617], or a predicted behavior of the population as in surprise search [405], among other options. Second, diversity can be achieved through the partitioning of the behavioral space. One may use for instance N different dimensions that characterize the behavior of generated content that is discretized as a grid (or a map) of cells. Each cell in such a grid represents a different area in the

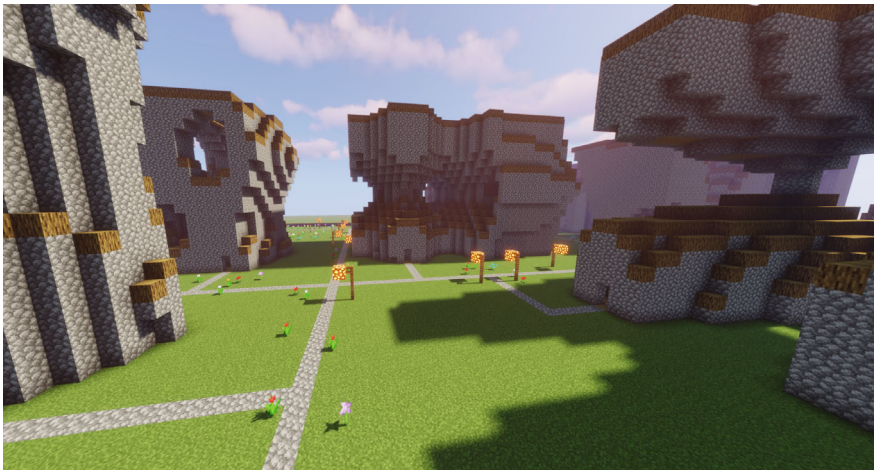


Fig. 8.10 Representation of *Minecraft* buildings organized into a settlement. The buildings are generated by the PCGQD method introduced in [63] combining constrained novelty search and 3D autoencoders. Image used with author permission

behavior space of the generated content and contains all individuals that have certain properties (e.g., lengthy levels with linear progression). A popular QD algorithm that builds on this behavior partitioning concept is MAP-Elites [781]. Among the many differences between the two diversity approaches the most notable is that behavior partitioning allows designers to control and illuminate the design space of whatever is generated.

The advantages of PCGQD are many compared to other PCG frameworks. First, QD methods are very efficient as they are able to produce large sets of high quality and diverse content in one run. Second, QD methods can consider more than one dimension of designer interest directly addressing the challenge of designing a single objective function that defines content quality. Third, PCGQD offers online **expressivity analysis** [1058] as a byproduct of the search for highly diverse and high-quality content even in a single run. Forth, QD approaches can empower the designer by offering control of both the criteria of quality and the behavioral dimensions of the generated artifacts, thereby, making QD approaches particularly suitable for mixed-initiative content design (see Sect. 7.2.3). Finally, QD approaches such as MAP-Elites are able to illuminate the search space and visualize it as a feature map, which can, in turn, assist developers to explore and better analyze the generator's outcomes. For more details about the palette of PCGQD algorithms available and their benefits for content generation the interested reader is referred to [403].

As an indicative application of PCGQD in games we present a recent study [63] that evolved and generated novel buildings in *Minecraft* (see Fig. 8.10). The method relies on the DeLeNox algorithm [633] and assesses the novelty of buildings via the latent space of 3D autoencoders representing those buildings. DeLeNox alternates the phases of exploration and transformation. In the *exploration* phase the

algorithm evolves multiple populations of neural networks, in particular compositional pattern producing network (CPPNs) [1078], via constrained novelty search. A set of repair and constraint functions are imposed to ensure that generated buildings adhere to structural rules and constraints, thereby ensuring content quality. In the *transformation* phase, the 3D autoencoder is retrained with datasets containing novel buildings. The alternating exploration-transformation phases allow the PCGQD algorithm to identify new areas of the generative space by reshaping the boundaries of the autoencoder's latent vector.

8.4 Solver-Based Methods

Another approach to PCG which is also based on the idea of content generation as search in a space of artifacts is the solver-based family of methods. Solver-based methods for PCG use constraint solvers, such as those used in **logic programming**, to search for content artifacts that satisfy a number of constraints.

Constraint solvers allow you to specify a number of constraints and find on or (usually) many artifacts that satisfy these constraints; some solvers require you to specify the constraints mathematically, others use a logic language for specification. Behind the scenes they can be implemented in many different ways. While there are some solvers that are based on evolutionary computation, it is more common to use specialized methods, such as reducing the problem to a **SAT** (satisfiability) problem and using a SAT solver to find solutions. Many such solvers progress not through evaluating whole solutions, but searching in spaces of partial solutions. This has the effect of iteratively pruning the search space: eliminating parts of the search space repeatedly until only viable solutions are left. This marks a sharp difference with the search-based paradigms, and also suggests some differences in the use cases for these classes of algorithms. For example, while evolutionary algorithms are **anytime algorithms**, i.e., they can be stopped at any point and some kind of solution will be available (though a better solution would probably have been available if you had the algorithm run for longer), this is not always the case with constraint satisfaction methods. Unlike with evolutionary algorithms, it is possible to prove bounds on the worst-case complexity of SAT solvers and the algorithms that depend on them. However, while the worst-case complexity is often shockingly high, such algorithms can be very fast when applied (judiciously) in practice.

An early example of solver-based methods is the level generator for Smith and Whitehead's *Tanagra* **mixed-initiative** platform level design tool [1060]. At the core of the tool is a constraint-based generator of platform game levels. The constraint solver uses a number of constraints on what constitutes a solvable platform level (e.g., maximum jump length, and distance from jumps to enemies) as well as constraints based on aesthetic considerations, such as the "rhythm" of the level, to generate new platform game levels or level segments. This generation happens very fast and produces good results, but is limited to linear levels. character's gameplay moves. Another example is the work by El-Nasr et al. [311, 315] on procedural generation of

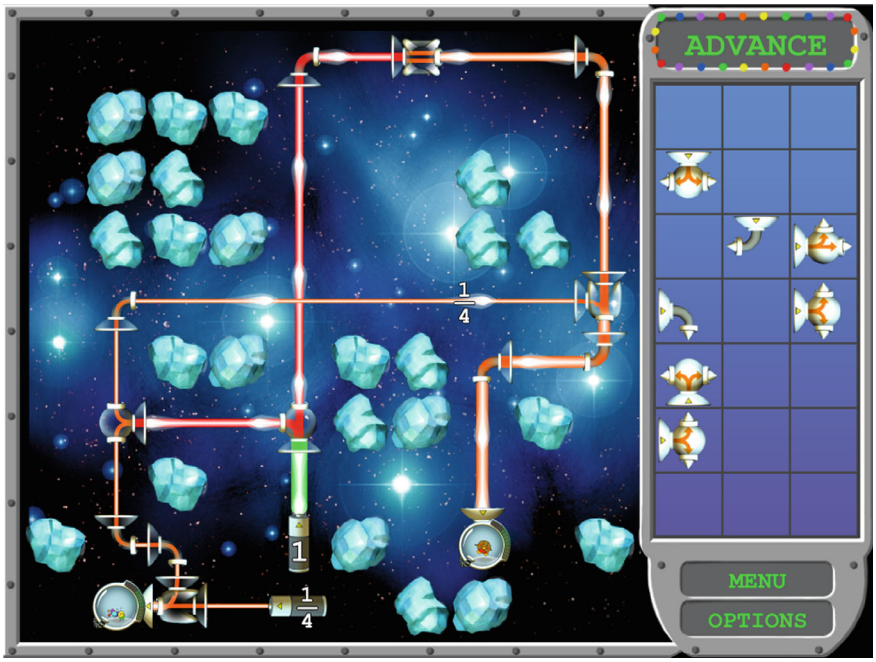


Fig. 8.11 A screenshot from the *Refraction* educational game. A solver-based PCG method (ASP) is used to generate the levels and puzzles of the game. Further details about the application of ASP to *Refraction* can be found in [155, 1053]. Image used with author permission

lighting. The system developed in those studies configures and continuously modulates the lighting in a scene with the aim to enhance player experience by using constraint non-linear optimization to select the best lighting configuration.

One approach to constraint solving which can be particularly useful for PCG because of its generality is **Answer Set Programming** (ASP) [1056]. ASP builds on AnsProlog, a constraint programming language which is similar to Prolog [124]. Complex sets of constraints can be specified in AnsProlog, and an ASP solver can then be used to find all models (all configurations of variables) that satisfy the constraints. For PCG purposes, the model (the set of parameters) can be used to describe a world, a story or similar, and the constraints can specify playability or various aesthetic considerations. An example of an ASP application for level and puzzle generation is the *Refraction* game (see Fig. 8.11). Another early example of ASP-based PCG is *Anza Island* [226], a simple sandbox game demonstrating the versatility of ASP for content generation. The player's goal in that game is to create logical constraints guiding an AI agent to play a level that is also procedurally generated via ASP. For a good introduction to and overview of the use of ASP in PCG you may refer to [805, 1056].

Generally solver-based methods can be suitable when the whole problem can be encoded in the language of the constraint solver (such as AnsProlog). It is generally

complicated to include simulation-based tests or any other call to the game engine inside a constraint satisfaction program. An alternative if simulation-based tests need to be performed is evolutionary algorithms, which can also be used to solve constraint satisfaction problems. This allows a combination of fitness values and constraints to drive evolution [404, 637, 644].

8.5 PCG via Machine Learning

As machine learning has made its way into absolutely everything over the past decade, this is also true for PCG. In particular, one can train generators on *existing content*, to be able to produce more content. In this section we cover the family of PCG algorithms introduced as *PCG via Machine learning* (PCGML) by Summerville et al. [1104] back in 2017. The more recently popular term **generative AI** as applied to game content largely overlaps with the PCGML paradigm. After all, PCGML investigates the generative capacity of algorithms that adopt machine learning methods and train themselves on (large) content corpora to be able to generate new content.

PCGML is inspired by the recent breakthroughs in deep learning, where network architectures such as **generative adversarial networks** [391] **variational autoencoders** [569], large scale **transformer** architectures , and **diffusion models** have attained extremely visually appealing results in learning to produce images and videos of e.g., bedrooms, cats or faces. PCGML is also inspired by earlier results where both simpler learning mechanisms such as Markov chains and more complex architectures such as recurrent neural networks have learned to produce text and music after training on some corpus.

While these kinds of generative methods based on machine learning work well for some types of content—most notably music, images, and some types of text—many types of game content pose additional challenges. A key difference is that most game content has strict structural constraints to ensure playability. These constraints differ from the structural constraints of text or music because of the need to play games in order to experience them. A level that structurally prevents players from finishing it is not a good level, even if it is visually attractive; a strategy game map with a strategy-breaking shortcut will not be played even if it has interesting features; a game-breaking card in a collectible card game is merely a curiosity; and so on. Thus, the domain of game content generation poses different challenges from that of other generative AI domains. The same methods that can produce "mostly correct" images of bedrooms and horses, that might still have a few impossible angles or vestigial legs, are less suitable for generating mazes which must have an exit. This is one of the major challenges in using machine learning for PCG. The other main challenge is that for many types of game content, there simply isn't enough existing content to train on. This is, however, an active research direction where much progress might be achieved in the next few years.

The core difference between PCG via machine learning and approaches such as search-based PCG is that the content is created *directly* (e.g., via sampling) from

models which have been trained on game content. While some search-based PCG approaches use evaluation functions that have been trained on game content—for instance, the work of Shaker et al. [1027] or Liapis et al. [633]—the actual content generation is still based on search. Next, we present some popular machine learning methods that can be used for content generation in games. For consistency purposes the methods we outline are coupled with particular level generation studies in variants of *Super Mario Bros*. We first cover PCGML methods that take advantage of small-scale ML representations such as ***n*-grams**, **Markov models** and shallow **artificial neural networks**. We then proceed with methods from the deep ML family of algorithms including deep networks, **GANs** and **LLMs**.

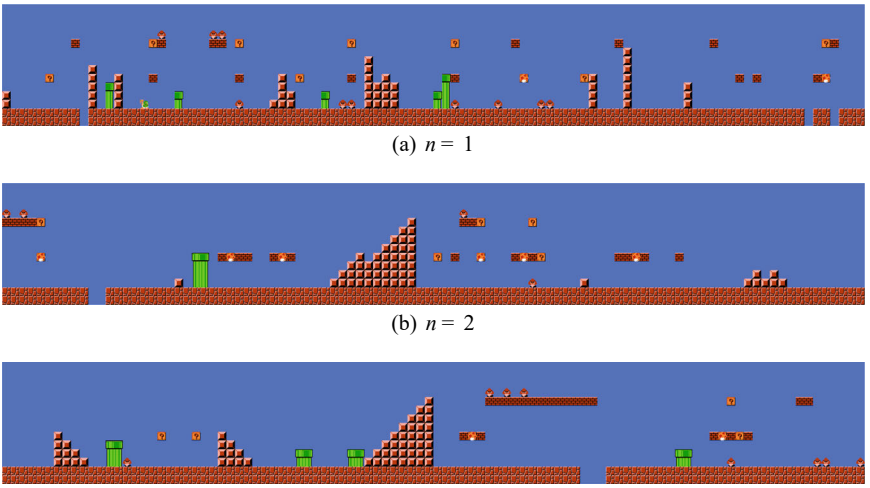
The taxonomy of methods presented in [419] classifies PCGML methods as *constraint*-based such as methods that are trained to learn constrains, *probabilistic* such as Markov random fields and Bayesian networks, ANN-based including any neural network approach, *grid*-based such as GANs, *sequence*-based such as recurrent ANNs and transformer models, and finally RL-based including methods from the TD family all the way to deep RL. While we adopt and follow a similar taxonomy of methods we do not cover all of them in this book (e.g., the probabilistic category is not covered here). Instead in this book we outline a selected number of methods and provide a number of examples that can assist the reader further. For more examples of early work in PCGML the reader is referred to [1104] whereas for a more recent and detailed presentation of ML methods for PCG, the reader is referred to [419, 654].

8.5.1 n-grams and Markov Models

For content that can be expressed as one- or two-dimensional discrete structures, such as many game levels, methods based on Markov models can be used. One particularly straightforward Markov model is the n-gram model, which is commonly used for text prediction. The n-gram method is very simple—essentially, you build conditional probability tables from strings and sample from these tables when constructing new strings—and also very fast.

Dahlskog et al. trained n-gram models on the levels of the original *Super Mario Bros* (Nintendo, 1985) game, and used these models to generate new levels [259]. As n-gram models are fundamentally one-dimensional, these levels needed to be converted to strings in order for n-grams to be applicable. This was done through dividing the levels into vertical "slices," where most slices recur many times throughout the level [258]. This representational trick is dependent on there being a large amount of redundancy in the level design, something that is true in many games. Models were trained using various n values, and it was observed that while $n = 0$ creates essentially random structures and $n = 1$ creates barely playable levels, $n = 2$ and $n = 3$ create rather well-shaped levels. See Fig. 8.12 for examples of this.

Summerville et al. [1103] extended these models with the use of **Monte Carlo tree search** to guide the generation. Instead of solely relying on the learned conditional probabilities, they used the learned probabilities during rollouts (generation of whole



(a) $n = 1$

(b) $n = 2$

(c) $n = 3$

Fig. 8.12 Mario levels reconstructed by n-grams with n set to 1, 2, and 3, respectively [259]. Images used with author permission

levels) that were then scored based on an objective function specified by a designer (e.g., allowing them to bias the generation towards more or less difficult levels). The generated levels could still only come from observed configurations, but the utilization of MCTS meant that playability guarantees could be made and allowed for more designer control than just editing of the input corpus. This can be seen as a hybrid between a search-based method and a machine learning-based method. Snodgrass and Ontañón trained two-dimensional Markov Chains—a more complex relative of the n-gram—to generate levels for both *Super Mario Bros* (Nintendo, 1985) and other similar platform games, such as *Lode Runner* (Brøderbund, 1983) [1062].

8.5.2 Neural Networks

Given the many uses of neural networks in machine learning, and the many different neural network architectures, it is little wonder that neural networks are also highly useful for machine learning-based PCG. Following on from the *Super Mario Bros* (Nintendo, 1985) examples in the previous section, Hoover et al. [471] generated levels for that same game by extending a representation called functional scaffolding for musical composition (FSMC) that was originally developed to compose music. The original FSMC representation posits 1) music can be represented as a function of time and 2) musical voices in a given piece are functionally related [469]. Through a method for evolving neural networks called *NeuroEvolution of Augmenting Topologies* [1081], additional musical voices are evolved to be played simultaneously with

an original human-composed voice. To extend this musical metaphor and represent *Super Mario Bros* levels as functions of time, each level is broken down into a sequence of tile-width columns. The height of each column extends the height of the screen. While FSMC represents a unit of time by the length of an eighth-note, a unit of time in this approach is the width of each column. At each unit of time, the system queries the ANN to decide a height to place a tile. FSMC then inputs a musical note's pitch and duration to the ANNs. This approach translates pitch to the height at which a tile is placed and duration to the number of times a tile-type is repeated at that height. For a given tile-type or musical voice, this information is then fed to a neural network that is trained on two-thirds of the existing human-authored levels to predict the value of a tile-type at each column. The idea is that the neural network will learn hidden relationships between the tile-types in the human-authored levels that can then help humans construct entire levels from as little starting information as the layout of a single tile-type.

8.5.3 GANs

GANs [391] do not learn to imitate samples of content, as any method from the supervised learning family. Instead, GANs, as covered in Sect. 3.6.4, operate on the principles of **adversarial learning** by which they aim to generate content that could pass as original (or real). Training a GAN requires access to a content corpus, such as a series of game levels. Specifically, GANs train two separate networks simultaneously: 1) the Generator (G) network that is fed with random noise and generates fake content and 2) the Discriminator (D) network that is trained to distinguish between fake and real content from a corpus. The generator tries to fool the discriminator while the discriminator tries to not get fooled in a somewhat symbiotic (co-evolving) relationship. It is common that both networks have the same architecture and it is also common to employ a fully connected convolutional network for both. Once both networks are trained, the discriminator is normally thrown away and the generator can be used to create new content such as a game level. The question that arises here is: how does one guide the generator of the GAN to create content based on some design specifications or aesthetic criteria? There are many ways and, arguably, one of the most popular approaches is to search through the latent vector of the generator and attempt to influence what the generator creates (see Fig 8.13). Any method that is able to search through a latent vector and evaluate the generated outcomes is applicable such as exhaustive search, metaheuristic search, open-ended evolution and even RL exploration. Here we focus on the method named **latent variable evolution** [106] that employs evolutionary search on the latent vector of a GAN's generator (represented as a genome) for the purpose of generating content that maximizes a particular fitness function.

As an indicative example of latent variable evolution we outline the successful use of the method for the generation of *Super Mario Bros* levels in [1223]. A GAN is first trained to generate levels for the game using levels from the *Video Game Level*

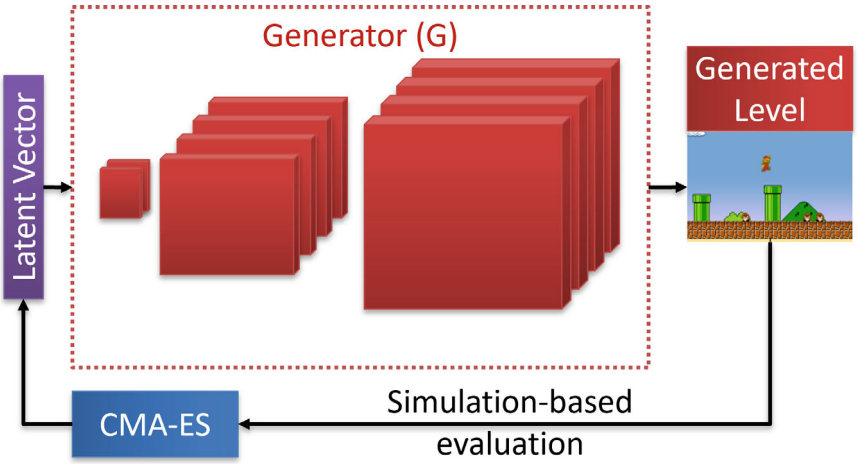


Fig. 8.13 The latent variable evolution phase of MarioGAN. An evolutionary algorithm (EA) searches through the latent vector of a trained generator (G) to optimize certain properties of the generated levels as specified by the fitness function of the EA. Image recreated from [1223]

Corpus [1105]. For details on GAN training the reader is referred to the AI methods chapter (see Sect. 3.6.4). Once GAN training is complete the *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [426] evolutionary algorithm is used to search for ideal latent vectors (i.e., inputs to the GAN) as shown in Fig. 8.13. During latent variable evolution, the generated levels are evaluated through a number of fitness functions that reward certain distributions of tile types (e.g., how many pipes, gaps or blocks a level should have). This general approach can be taken further by using a quality-diversity algorithm such as a version of MAP-Elites rather than a single objective evolutionary algorithm, yielding a method that generates a diverse set of Mario levels that vary according to specified behavioral descriptors [347].

8.5.4 LLMs

Any PCG method that generates content from ML models that are trained on existing content, strictly speaking, fits under the PCGML paradigm [1104]. Neither the original PCGML framework [1104] nor the more recent and extended version of PCGML in 2022 [419] considered large language models given that research interest in LLMs scaled up soon after the release of the GPT-2 model in 2019 [920]. We outline the use of LLMs for game content generation in this section as LLMs for content creation can be largely viewed as a PCGML method. While the use of LLMs for game content creation—at the time of writing—is an area under significant exploration, LLMs already showcase a high potential for future PCG applications in games [365]. For more details on LLMs, the reader is referred to Sect. 3.6.7.

LLMs, and more broadly LMMs, can be prompted with text or other modalities such as images and sounds and generate text or a sequence of symbols that may represent various content types. As we saw in the PCGML methods covered earlier several studies have already demonstrated that it is possible to generate tile-based game levels in a reliable fashion. In the era before LLMs one would normally employ a sequence-based predictor—such as an LSTM [1102]—to model and generate sets of tiles as sequences.

The use of LLMs for game level generation follows a similar pipeline: a game level can be represented as a sequence of tiles [1146, 1303] or a sequence of positional embeddings or rooms [364]. Then a GPT variant (or any other LLM from a different algorithmic family) can be fine-tuned on a large corpus of existing game level content such as *Sokoban* (Thinking Rabbit, 1982) levels [1146], *Super Mario Bros* (Nintendo, 1985) levels [1099], manually crafted game level rooms (or floor layouts) [364] or game themes (including stories, characters, locations, and visual assets) in dungeon crawling card games [1303]. At generation time the fine tuned model can be sampled (or prompted) to produce novel levels or level layouts (see Fig. 8.15). Recent studies also investigated the LLM-based generation of 3D buildings in *Minecraft* [485], as well as the rules and the levels of simple maze games [481].

In another recent indirect use of LLMs for game generation Google DeepMind researchers [138] showcased that it is possible to generate and play short platformer levels in a general fashion when prompted with text and other modalities such as images. The method introduced, namely *Genie*, first converts different text prompts into images. *Genie* then converts those images to interactive, playable environments through transformer-based architectures (ViT variants) that are trained in an unsupervised fashion from thousand hours of gameplaying videos available over the Internet.

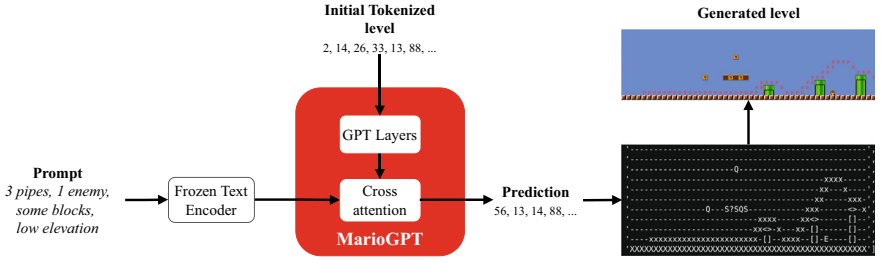
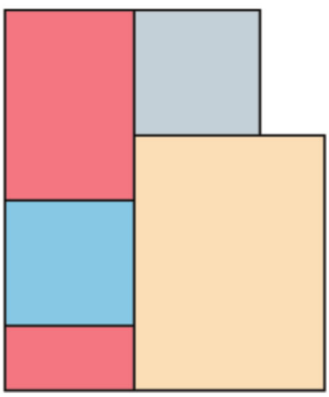
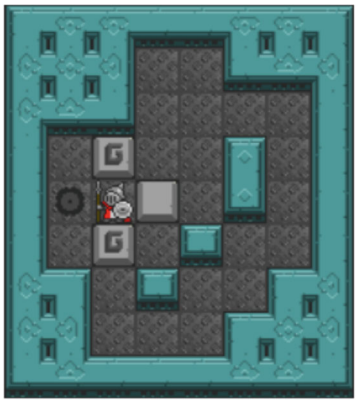


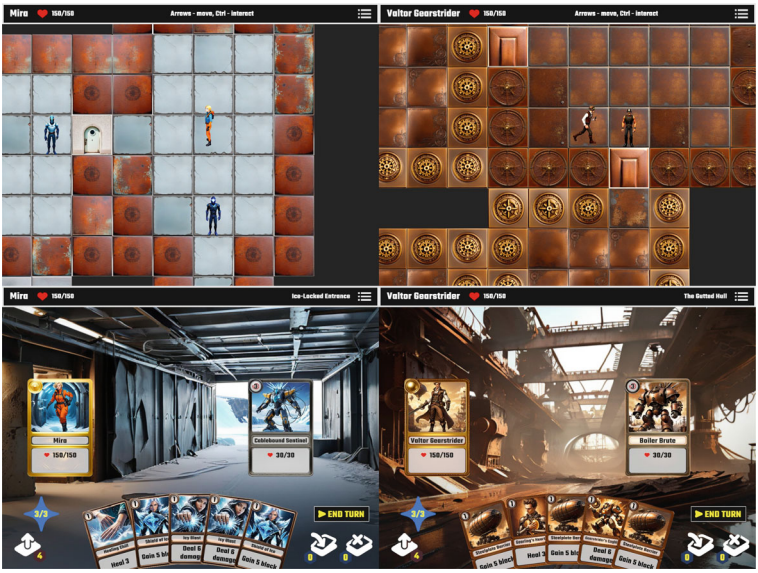
Fig. 8.14 A high-level perspective of the MarioGPT architecture. MarioGPT is a fine-tuned model of the distilled GPT-2 language model. The autoregressive model is trained to predict next token sequences. Levels are represented as tokenized strings of columns flattened into a single vector. To make it possible to prompt the model via text a frozen BART [622] text encoder. The output of the text encoder and the actual level sequence feeds the cross-attention layers of the GPT-2 architecture. Image adopted from [1099] and used with permission



(a) Architext



(b) Sokoban



(c) CrawLLM

Fig. 8.15 Examples of LLM-based level generation. **a** A level (floor) layout generated by a fine-tuned GPT-J model prompted as *a house with two bedrooms and one bathroom* [364]; red and blue boxes correspond to bedrooms and bathrooms, respectively. **b** A level for the puzzle game *Sokoban* generated by GPT-3, visualized with the Griddly tileset [53]. **c** The *CrawLLM* re-theming LLM-based generator that adapts game assets—such as stories, characters, locations, and visual assets—to new settings and narratives [1303]. Images used with permission

To remain consistent with the PCGML case studies we covered earlier we detail the application of LLMs for the generation of *Super Mario Bros* (Nintendo, 1985) levels. The method named MarioGPT fine-tuned a GPT-2 model on a rather small corpus of *Super Mario Bros* (Nintendo, 1985) levels [1099]. Each level was represented as a sequence of text symbols that would be used to train the model (see Fig. 8.14). Strictly speaking MarioGPT is a hybrid PCG algorithm that combines ML and search. In particular, it trains LLMs models whose latent representations are searched through for the generation of higher quality content, as specified by a fitness function. Generated levels of the corpus are mutated by combining existing level segments and samples from the trained GPT model. Even though the method has access to less than 20 initial levels, the mutation approach acts as a data augmentation method that yields large and diverse corpus of playable levels.

8.6 Reinforcement Learning

A major issue with supervised and self-supervised learning is, as noted previously, data scarcity. While some methods are more data-efficient than others, the fundamental issue is that in many cases where you want to generate game content, you do not have much game content to train a generator on. Luckily, there are other machine learning methods that can be of help here. In particular, there is reinforcement learning, where a policy is learned from interaction with the environment. *Procedural content generation via reinforcement learning* is often shortened to PCGRL [559], which perhaps does not roll of the tongue, but at least it's only five letters.

Usually, RL in games is thought of in the context of learning to play a game. Here, gameplaying is seen as a sequential decision task, and a policy is learned to maximize return (discounted sum of rewards) over many episodes of the game. But content generation can also be seen as a sequential decision problem. From this perspective, a content generation policy is learned by taking sequences of edit actions that modify some kind of content artifact, such as a level. Rewards are based on how the edits improve the quality of the artifact; see Fig. 8.16.

Concretely, this can take many different forms. We can illustrate this with the setting from the 2020 paper by Khalifa et al. [559] that popularized PCGRL. Here, the task is to generate square levels of 14×14 tiles. Levels can be generated for either of three different games: a binary "game" (really just a maze), a generic dungeon patterned on *The Legend of Zelda*, and the box-pushing puzzle *Sokoban*. Binary has two tile types (wall and floor), Sokoban has four and Zelda has six.

In the most popular mode of applying PCGRL to these games, the so called "narrow" configuration, a cursor or agent scans over the level multiple times, like an old-fashioned CRT monitor, considering each tile in order. For each tile, a neural network (the policy) gets all or part of the level centered on the tile as input, and decides whether to change the tile to another tile type. If it does take an action to change the tile type, it then gets a reward based on the change in estimated level quality. Level quality is estimated using cheap calculations mainly based on path

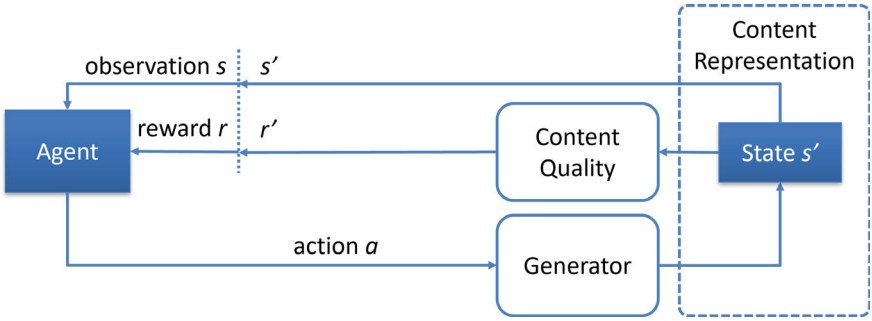


Fig. 8.16 General overview of the PCGRL framework. An RL agent generates content by taking an action a given the current the observation space s. The content representation (i.e., the environment in RL terms) evaluates the generated content for quality and returns an immediate reward r for this action. Image inspired by [559, 1038]

length, for example for the Zelda game the key and door should be accessible from the starting location at particular path length, and enemies should be spread out. It is important here that the reward calculations should not be too computationally complex, as dense rewards seem to be necessary for PCGRL to work well.

Other ways of structuring the action space in two-dimensional PCGRL include the "turtle" and the "wide" modes. The turtle mode functions similarly to the narrow mode, with a cursor that focuses on one tile at a time, but here the policy decides not only whether to make an edit and what tile type to use, but also which direction to go next. Watching a trained PCGRL agent editing a level using the turtle mode gives the impression of a game agent intentionally traversing the game level, occasionally making edits as it goes. The wide mode does away with the idea of a movable cursor that determines where edits can be made; instead, the policy can choose any position in the level at each time step. It can still only change one tile at a time.

Other recent approaches to RL-based content generation include the use of adversarial RL for the procedural generation of platformer and racing game levels [385] as well as levels for simulation games such as *Roller Coaster Tycoon 2* (Infogrames Interactive, 2002) [167]. RL offers a wide variety of content for dissimilar player styles but it often comes with a certain lack of control over the generated content. To address this controllability issue the generated content often passes through a playability test or a repair function [58, 1239].

Recent advancements have extended the capabilities of the PCGRL framework. By introducing reward shaping and conditional inputs, the framework enables goal-aware generators to produce controllably diverse, user-responsive content while maintaining high quality and exploring designer-specified dimensions [300]. Its expansion into 3D domains, such as *Minecraft*, demonstrates the ability to account for the increased complexity of 3D environments, including factors like gravity and multidimensional movement, to generate diverse and functional levels [510]. Additionally, the integration of Jax for parallel GPU-based simulation improves training efficiency and enhances scalability, enabling the generation of larger levels

and incorporating features like randomized level sizes and frozen pivotal tiles to improve control and generalization [301]. These developments collectively advance PCGRL's flexibility and applicability in procedural content generation.

Experience-driven PCGRL (EDRL)

But PCGRL is not dependent on tile-based content representations or single choices. It is also not depended on objectively defined reward functions about playability and balance. A recent framework introduced by Shu et al. [1038] named ED(PCG)RL, EDRL in short, combined elements of experience-driven PCG and PCG via reinforcement learning. EDRL is able to teach RL designers to generate endless playable levels in an online manner so that they elicit certain experiences for the player, designed in the form of reward functions. Specifically, EDRL was tasked to design *Super Mario Bros* levels by generating and concatenating level segments while considering the diversity among them (see Fig. 8.17). The EDRL agents learn to maximize a reward function that quantifies Koster's notion of *fun* [581] by moderating the degree of diversity across level segments and possible playtraces [1239]. The *fun*

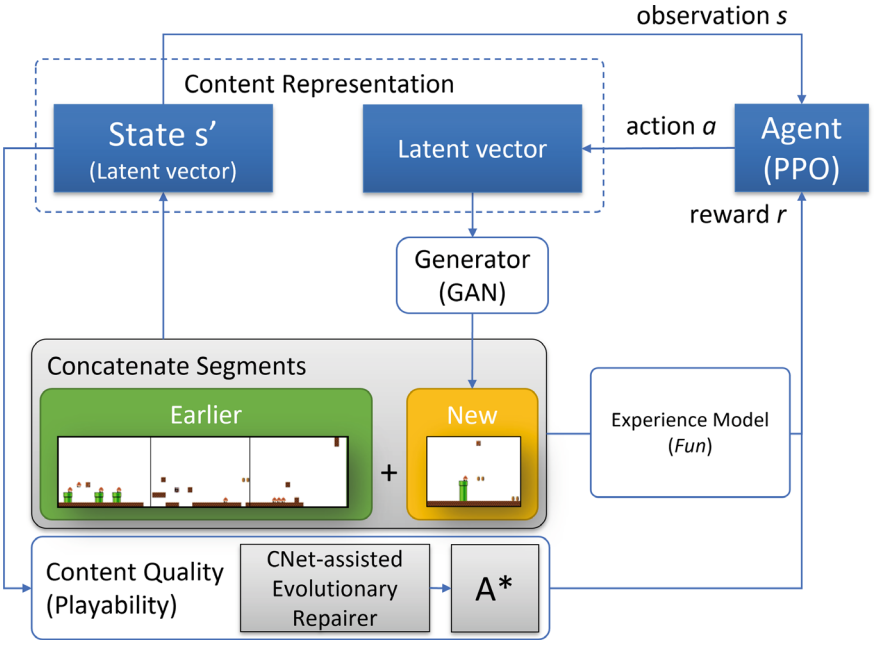


Fig. 8.17 General overview of the ED(PCG)RL, EDRL in short, framework implemented on *Super Mario Bros*. EDRL interweaves elements of both the EDPCG and the PCGRL frameworks. White rounded boxes and blue boxes depict components of EDPCG [1289] and PCGRL [559], respectively. Content representation (i.e., environment in RL terms) is a common component across the two frameworks and is depicted with a dashed line. Image adapted from [1038] and used with permission

reward function was cross-verified recently through extensive user studies reported in [1238]. The quality of the generated levels is assessed for correctness (via a neural net-assisted evolutionary level repairer) and playability (through AI-based testing).

EDRL is capable of generating endless, playable *Super Mario Bros* levels with varying degrees of *fun* and playability at **runtime** as actions taken apply changes in the latent space of a GAN; see Fig. 8.17. The GAN is pre-trained to generate level segments for *Super Mario Bros*. The RL-based trained policy observes the latent space representation of a level segment and takes an action by feeding the GAN with a new latent vector. Taking actions in the latent space of a learned model enables PCGRL to operate rapidly with a range of different underlying representations, and is in no way restricted to 2D tile-based content. In principle PCGRL, and EDRL, can be generalized to any game that can be represented as a sequential design process and features a built-in compressed representation of its content.

8.7 Further Reading

Introductions to many of the methods covered in this chapter can be found across dedicated chapters of the PCG in Games Textbook [1023]; see Chaps. 1–9, in particular. In what follows we list additional books, papers and resources for each PCG method we presented in this chapter.

Starting with constructive PCG methods and specifically noise and fractals, the interested reader is referred to books with an emphasis on texturing and modeling methods from a graphics perspective such as [302]. The capabilities and various uses of the WFC algorithm for content generation in games are detailed in the recent survey paper by Karth and Smith [548].

The search-based family of algorithms introduced in [1175, 1176] over a decade ago has seen multiple applications across various forms of content generation tasks. The diversity of such applications is demonstrated by two indicative recent papers: in the first one, search-based PCG is compared to RL for experience-driven racing track generation [58] and in the second one, it is used for modeling and generating *game economies* (i.e., the ways resources in games are created, transformed, or exchanged) [957]. PCG via quality diversity is detailed in [405] with recent applications varying from platformer level generation [792] to the generation of text descriptions and cover images for hypothetical video games [1304]. When it comes to solver-based methods, the papers of Smith with Mateas [1056] and Nelson [805] offer good introductions to the use of ASP for procedural content generation tasks.

Moving on to ML methods for PCG [1104], the interested reader is referred to [419, 654] for recent and detailed presentations. If one is particularly interested in the use of LLMs as a PCG method we recommend the recent survey paper by Gallotta et al. [365]. Finally, further details about RL methods as mechanisms for content generation in games can be found in [559, 1038].

8.8 Summary

This chapter covered the key methods and frameworks used for the procedural generation of game content. In addition to the constructive methods covered in the first part of this chapter the remaining methods generate and evaluate the content they create in an iterative fashion. Beyond the search-based PCG methods we outlined the quality diversity PCG and solver-based paradigm, and we presented a number of machine learning methods for the generation of game content. In particular we focused on both self-supervised learning and reinforcement learning methods and corresponding frameworks. The last chapter of this part is dedicated to the types of content a PCG method can generate and the ways such content can be assessed and evaluated.

Chapter 9
Procedural Content Generation by Content Type

Abstract The final chapter of the PCG part of the book discusses content generation from the perspective of the type of content that is being generated. This chapter includes a large number of examples. The first section discusses levels, the perhaps most common type of artifact to be procedurally generated both in research and deployment. This is followed by examples of generating visuals, audio, and narrative. We then discuss the generation of game rules, and complete games. Finally, we cover the important topic of how to evaluate the quality of a content generator.

The final chapter of Part III takes the perspective of content types, and presents examples of generating some common and uncommon types of game content. In particular, we go through the possible content types that a PCG algorithm can generate in a game. Liapis et al. [643] identified six creative domains (or else **facets**) within games that we will follow for our discussion in this section. These are level architecture, audio, visuals, rules (game design), narrative, and gameplay. In this chapter we will cover the first five facets and but exclude the gameplay facet. Creative gameplay is directly associated with play and as such is covered in Part II. Beyond the creative facets we review independently we also discuss the potential use of PCG for complete game generation. We conclude the chapter with a dedicated section on the evaluation of PCG.

9.1 Levels

The generation of levels is by far the most popular use of PCG in games. Levels can be viewed as **necessary** content since every game has some form of spatial representation or virtual world within which the player can perform a set of actions. Levels are also expected to be **functional** if played, or merely **decorative** if they are generated during ideation phases of design. The properties of the game level, in conjunction with the game rules, frame the ways a player can interact with the world and determine how the player can progress from one point in the game to another. The game's **level design** contributes to the challenges a player faces during the game. While games would often have a fixed set of mechanics throughout, the way a level

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_9

is designed can influence the gameplay and the degree of game challenge. For that reason, some PCG researchers have argued that levels coupled with game rules define the absolutely necessary building blocks of any game; in that regard the remaining facets covered in this chapter are optional [630]. The variations of possible level designs are endless: a level representation can vary from simple two-dimensional illustrations of platforms and coins—as in the *Super Mario Bros* (Nintendo, 1985) series—to the constrained 2D space of *Candy Crush Saga* (King, 2012), to the three-dimensional and large urban spaces of *Assassin's Creed Odyssey* (Ubisoft, 2018) and *Call of Duty* series (Activision, 2003–present), to the 2D structures of the *Angry Birds* series (Rovio, 2009-2022), and finally to the voxel-based open gameworld of *Minecraft* (Mojang, 2011).

We can view the procedural generation of game levels from the lens of **procedural architecture**. Similarly to architecture, level design needs to consider both the *aesthetic* properties and the *functional* requirements of whatever is designed within the game world. Depending on the game genre, functional requirements may vary from a reachable end-goal for platform games, to a challenging gameplay in driving games such as *Forza Motorsport* (Turn 10 Studios 2005), to waves of gameplay intensity as in *Pac-Man* (Namco, 1980), *Left 4 Dead* (Valve, 2008), *Resident Evil 4* (Capcom, 2005) and several other games. A procedural level generator also needs to consider the aesthetics of the content as the level's aesthetic appearance may have a significant impact not only on the visual stimuli it offers to the player but also on navigation. For example, a sequence of identical rooms can easily make the player disoriented—as was intended in the dream sequences of *Max Payne* (Remedy, 2001)—while dark areas can add to the challenge of the game due to low visibility or augment the player's arousal as in the case of *Amnesia: The Dark Descent* (Frictional Games, 2010), *Outlast* (Red Barrels, 2013) [1262], *Nevermind* (Flying Mollusk, 2015) and *Sonancia* [661]. When the level generator considers larger, open levels or gameworlds then it draws inspiration from urban and city planning [680], with edges to constrain player freedom and landmarks to orient the player and motivate exploration [643]—as in the *Grand Theft Auto* series (Rockstar Games, 1997–2021)—and districts to break the world's monotony—as in *World of Warcraft* (Blizzard Entertainment, 2004) which uses highly contrasting colors, vegetation and architectural styles to split the world into districts that are suitable for characters of different level ranges.

As we have discussed earlier, the generation of levels in a procedural manner is clearly the most popular and possibly the oldest form of PCG in the game industry. We already mentioned the early commercial use of PCG for automatic level design in games such as *Rogue* (Toy and Wichman, 1980) and the Rogue-inspired *Diablo* (see Fig. 9.1) series (Sierra Entertainment and Blizzard Entertainment, 1997–2023), and the world generation examples of *Civilization VI* (2K, 2016) and *Minecraft* (Mojang, 2011). The level generation algorithms traditionally used in commercial games are constructive, in particular in games where players can interact with and change the game world via play. Players of *Spelunky* (Mossmouth, LLC, 2008), for instance, are allowed to modify a level which is not playable (i.e., the exit cannot be reached) by blowing up the blocking tiles with a bomb provided by the game level. **Wave**

Fig. 9.1 The procedurally generated dungeons of *Diablo V* (Blizzard Entertainment, 2023), one of the most characteristic examples of level generation in commercial games. The *Diablo* series (Sierra Entertainment and Blizzard Entertainment, 1997–2023) is a relatively recent descendant of *Rogue* (Toy and Wichmann, 1980) (i.e., rogue-like) role-playing game characterized by dungeon-based procedurally generated game levels. Screenshot taken by the authors

function collapse [548] is another constructive PCG method that gained popularity in indie games like *Bad North* (Raw Fury, 2018) and *Townscaper* (Oskar Stålberg, 2021) enabling level design aspects such as terrain generation and procedural building design.

The academic interest in procedural level generation [1023] and generative AI for level design has been increasing for the last 15 years or so and has produced a large volume of papers [632]. Most of the academic studies described in this part are focused on levels. The interested reader may refer to those for examples of level generation across various methodologies, PCG roles and game genres.

9.2 Visuals

Games are, by definition, visual media unless the game is designed explicitly to not have visuals—there are a scarce few non-visual games, e.g., the *Real Sound: Kaze no Regret* (Sega, 1997) adventure audio game. The visual information displayed on the screen conveys messages to the player which are dependent on the graphical style, color palette and visual texture. Game visuals are both very popular content types (used in almost all games) and very easy to integrate with the remaining content facets. Even though visuals are required to fit within the aesthetic style of the game and satisfy certain constraints such as animation speed, sprite size, and visibility,

Fig. 9.2 The caricaturized and highly-emotive visuals of the game *Limbo* (Playdead, 2010). Screenshot taken by the authors

they are, perhaps, the content type with the fewest functionality constraints. Visuals in games can vary from simple abstract and pixelized representations as the 8-bit art of early arcade games, to caricaturized visuals as in *Limbo* (Playdead, 2010) (see Fig. 9.2), to photorealistic graphics as in the *FIFA* series (EA Sports, 1993–2022) [507].

Within the game industry PCG has been used broadly for the generation of any of the aforementioned visual types. Arguably, the complexity of the visual generation task increases the more the resolution and the **photorealism** of the desired output increases. There are examples of popular AI-based generative tools, however, that can successfully output photorealistic 2D and 3D visuals, that have been available for visual generation for over 20 years now. These include tools such as *SpeedTree* (Interactive Data Visualization, Inc., 2002) for vegetation and *FaceGen* (Singular Inversions, 2001) for faces. Moreover, methods such as **Perlin Noise** [873] have been used for visuals and texture generation in games like *Doom* (id Software, 1993) for more than 30 years now.

With the rise of large scale multimodal models and stable diffusion in recent years a number of visual generation tools have revolutionarized the asset creation process in games. Some indicate examples include large media models like DALL-E 2 [922] that combine **Contrastive Language-Image Pre-training** (CLIP) [920] and diffusion models for the generation of visuals that are conditioned to specific text prompts. *Midjourney* (Midjourney, Inc., 2022) [758] generates high quality images from natural language descriptions using variants and combinations of DALL-E [922] and *Stable Diffusion* [961] models. The latter is a deep learning, text-to-image (and

image-to-image) model trained by *Stability AI*.[1] The various stable diffusion models that currently exit are based on variants of diffusion methods [961] and can be fined-tuned using the efficient LoRa training algorithm [483]. *Leonardo Ai*[2] is an indicative game visual asset creation tool that—at the time of writing this book— can generate a variety of assets such as characters and environments based on text prompts and using underlying models from the aforementioned families of algorithms.

Within academia notable early examples of visual generation include the study of decorative ornamentation in buildings like castles, palaces, and temples that might exist in games [1248]. Other early attempts to game visuals generation include the creation of diverse and believable rocks via a combination of L-systems and evolutionary algorithms [264]. Similar methods have been used for the procedural generation of biologically-plausible corals and seabeds [3]. A number of authors have been inspired by theories about "universal" properties of beauty [29] to generate visuals of high appeal and appreciation [638]. The PCG algorithm in that study generates spaceships based on their size, simplicity, balance and symmetry, and adapts its visual output to the taste of the visual's designer via interactive evolution. The PCG-assisted design process referred to as **iterative refinement** [641] is another way of gradually increasing the resolution of the visuals a designer creates by being engaged in an iterative and creative dialog with the visuals generator. More recent approaches in mixed-initiative visuals generation have instead focused on constructive methods such as **wave function collapse** [548] for the generation of tile-based textures [601]. Beyond the generation of in-game entities a game visuals generative algorithm may focus on general properties of the visual output such as pixel shaders [477], lighting [311, 315], brightness and saturation, which can all influence the overall appearance of any game scene.

A number of games that are designed around PCG principles such as *Petalz* [944, 945] (flower generation), *Galactic Arms Race* [432] (weapon generation; see also Fig. 8.9) and *AudioInSpace* [468] (weapon generation) feature visual generation. In all three games, visuals are represented by neural networks that are evolved via interactive evolution. Within the domain of weapon particle generation another notable example is the generation of surprising yet balanced weapons—with predefined mesh and particle effects—for the game *Unreal Tournament III* (Midway Games, 2007). Weapon generation in that study was driven by **constrained surprise search** [404], an algorithm that maximizes the surprise score of a weapon but at the same time imposes designer constraints to it so that it is balanced.

Beyond image generation and creation of game environment properties, at the time of writing this book we see frequent breakthroughs in **video generation**. Frame-to-frame (video) generation can be used for the creation of game trailers or cut scenes in a game and hence can be viewed as a visuals generation task. Recent open-source image-to-video models such as *stable video diffusion* [96] and state-of-the-art closed-source video generation tools such as Open AI Sora[3] are expected to influence the

[1] https://stability.ai/.

[2] https://leonardo.ai/.

[3] https://openai.com/sora/.

creation of videos in games. One could instead view video generation, partially, as a game generation task. There is a certain overlap between video generation and neural game engines as problem domains, as in both cases the model needs to learn to predict the dynamics of the world to some extent. It is important to note that the majority of current video generation models lack basic information about the game context and game world constraints of any type as they merely generate frame sequences based on a text or image prompt. The *Genie* [138] playable video generator is one of the few attempts towards that direction. The reader is referred to Sect. 9.6 for further details related to game generation.

Finally, it is worth mentioning the use of deep learning based methods to enhance and post-process the game's graphical output in various ways. Neural networks can be trained to **upscale** or upsample low-resolution images. This can now be done in real-time on games on many consumer devices, enabling for example older games to be displayed at a level of detail that they were not designed with [749]. Beyond upscaling, **neural style transfer** has started to become possible to do in real-time. In neural style transfer, a picture or video is translated to a different style. For example, turning the stylized and somewhat cartoonish graphics of *Grand Theft Auto V* (Rockstar Games, 2013) into photorealistic video [937]. One can even transfer graphics in between games of the same genre such as converting visuals of *Fortnite* (Epic Games, 2017) to visuals of another battle royale game like *PUBG: Battlegrounds* (Krafton, 2017) using GAN variants [1185]. These methods can also be applied to do 2D games, for example efficiently swapping the look and feel of platformers [990]. In one possible future, the traditional rendering pipeline is radically simplified, just outputting solid geometric objects, with deep learning turning it into a images of the requisite style; for example *GAN Theft Auto* turns a procedurally generated 3D city into naturalistic images [256].

9.3 Audio

Even though audio can be seen as optional content it can affect the player directly and its impact on the player's experience is apparent in most games [224, 369, 371]. Audio in games has reached a great level of maturity as demonstrated by the audio categories of the annual *BAFTA Game Awards*, the three *MTV Video Music Awards* (2004–2006) for best video game soundtrack [643] and, more recently, the two *Grammy Awards* for best score soundtrack for video games and other interactive media (2023–2024). Most games that feature audio also feature some form of a soundtrack which may contain music combined with a set of sound effects. The audio types one meets in games may vary from fully orchestrated soundtrack (background) music, as in *Skyrim* (Bethesda, 2011), to sound effects, as the dying or pellet-eating sounds of *Pac-Man* (Namco, 1980), to the voice-acted sounds of *Fallout 3* (Bethesda, 2008). Most notably within **indie game development**, *Proteus* (Key and Kanaga, 2013) features a mapping between spatial positioning, visuals and player interaction, which collectively affect the sounds that are played. Professional tools such as the

procedural audio engine *MetaSounds* of Unreal Engine 5 (Epic Games, 2020) offer designers with complete control over digital signal processing graphs for the generation of audio. The long-standing *sfxr* (Tomas Pettersson, 2007) and *bfxr*[4] sound generation tools provide procedural sound components to audio designers, demonstrating a commercial interest in and need of procedurally generated audio for over two decades.

At a first glance, the generation of game audio, music and sounds might not seem to be particularly different from any other type of audio generation outside games. Games are interactive, however, and that particular feature makes the generation of audio a rather challenging task. When it comes to the definition of **procedural audio** in games, a progressive stance has been that its procedurality is caused by the very interaction with the game. For instance, game actions that cause sound effects of music can be considered as procedural audio creation [223, 370].

The constraints imposed on audio generation seem to be relatively soft compared to, say, level generation. While audio generation should abide to the aesthetic style, the current mood, the scene's emotional tone, and the player and NPC actions at any given moment, an inappropriate audio generation will not necessarily break the game; it may, however, break the player's experience entirely. Ideally, the game audio must be able to adapt to the current game state and the player behavior. As a result adaptive audio is a grand challenge for composers since the combinations of all possible game and player states could be largely unknown [475]. Another difficulty for the autonomous generation of adaptive music, in particular, is that it requires real-time composition and production; both of which need to be embedded in a game engine. An increasing number of games involve some kind of procedural soundtrack, and some research projects have focused on music generation able to adapt to affective shifts in real-time [891, 1010]. Beyond academia, a few commercial-standard efforts have been made in adaptive audio tools. For instance *Melodrive*[5] is one of the first AI systems that composes emotion-aware music in real time by adapting to the media environment aiming to match the mood and the aesthetic style of the video.

Recent years have seen impressive developments in audio generation via deep learning [125] and large media models with some degree of controllability [280]. Examples of such systems include the music generation tools developed under the *Flow Machines* project [846] or the open-source tools of Google Magenta Studio[6] that rely on several AI generative methods: from integrated models of classic signal processing with deep learning [323] to GANs for music synthesis [322]. Most of the available music generation models, however, are not particularly designed to perform well in games. Beyond industrial generative AI audio tools, academic work on procedural game music and sound generation has seen substantial advancements in the last decade: from the experience-driven procedural music generation framework [891] over a decade ago, to the mood-conditioned in-game music generator *MetaCompose* [1010] that relies on multi-objective evolutionary optimization, to the

[4] https://www.bfxr.net/.

[5] https://melodrive.itch.io/.

[6] https://magenta.tensorflow.org/demos.

more recent *MINUET* tool [703] that procedurally generates music conditioned on sentiment analysis of textual narrative segments.

Generally speaking, sound can be either **diegetic** or **non-diegetic**. A sound is diegetic if its source is within the game's world. The source of the diegetic sound can be visible on the screen or can be implied to be present at the current game state. Diegetic sounds include characters, voices, sounds made by items on-screen or interactions of the player, or even music represented as coming from instruments available in the game. A non-diegetic sound, on the other hand, is any sound whose source is outside the game's world. As such non-diegetic sounds cannot be visible on-screen or even implied to be off-screen. Examples include commentaries of a narrator, sound effects which are not linked to game actions, and background music.

A PCG algorithm can generate both diegetic and non-diegetic sounds including music, sound effects and commentaries. Examples of non-diegetic sound generation in games include the *Sonancia* horror sound generator that tailors the tension of the game to the desires of the designer based on the properties of the game level [661]. The mapping between tension and sounds in *Sonancia* has been derived through **crowdsourcing** [663]. Similarly to *Sonancia*—and towards exploring the creative space between audio and level design—*Audioverdrive* generates levels from audio and audio from levels [465]. Notably within diegetic audio examples, Scirea et al. [1009] explores the relationship between procedurally generated music and narrative. Studies have also considered the presence of game characters on-display for the composition of game soundtracks [131] or the combination of short musical phrases that are driven by in-game events and, in turn, create responsive background audio for strategy games [475].

Finally, it is worth mentioning that there are games featuring PCG that use music as the input for the generation of other creative domains rather than music per se. For example, games such as *Audio Surf* (Fitterer, 2008) and *Vib Ribbon* (Sony Entertainment, 2000) do not procedurally generate music but they instead use music to drive the generation of levels. *AudioInSpace* [468] is another example of this: a side-scrolling space shooter game that does not generate music but uses the background music as the basis for weapon particle generation via artificial evolution. *Audioverdrive*, as already mentioned, performs content generation in a bidirectional fashion between two facets: it uses audio to generate levels but it also uses levels as input for the generation of audio [465].

9.4 Narrative

Many successful games rely heavily on their narratives; the clear distinction however, between such narratives and traditional stories is the interactivity element that is offered by games. Now, whether games can tell stories [527] or games are instead a form of narrative [1] is still an open research question within **game studies**, and beyond the scope of this book. The study of computational (or procedural) narrative focuses on the representational and generational aspects of stories as those can

be told via a game. Stories can play an essential part in creating the aesthetics of a game which, in turn, can impact affective and cognitive aspects of the playing experience [836].

By breaking the game narrative into subareas of game content we can find core game content elements such as the game's plotline [380, 939], but also the ways a story is represented in the game environment [149, 1200]. The coupling of a game's representation and the story of the game is of vital importance for player experience. Stories and plots are taking place in an environment and are usually told via a **virtual camera** lens. While the placement and movement of a virtual camera can arguably be viewed as a visuals generation task, the control of cameras—similarly to films—is normally coupled with the creation of narratives and stories in games. Unsurprisingly, the behavior of the virtual camera—viewed as a parameterized element of computational narrative—can drastically influence the player's experience. That can be achieved via an affect-based cinematographic representation of multiple cameras as those used in *Heavy Rain* (Quantic Dream, 2010) or through an affect-based automatic camera controller as that used in the Maze-Ball prototype game [1285]. Choosing the best camera angle to highlight an aspect of a story can be seen as a multi-level optimization problem, and approached with combinations of optimization algorithms [151]. Games such as *World of Warcraft* (Blizzard Entertainment, 2004) use cut scenes to raise the story's climax and lead the player to particular player experience states

The creation or semi-automatic generation of stories and narratives belongs to the area of **interactive storytelling**, which can be viewed as a form of story-based PCG. The story can adjust according to the actions of the player targeting personalized **story generation** (e.g., see [184, 948] among others). Ultimately, game worlds and plot point story representations can be co-generated as demonstrated in a number of studies (e.g., see [430]).

Computational narrative methods for generating or adapting stories or narratives are traditionally built on planning algorithms, and **planning** is therefore essential for classic narrative generation [1300]. The space of stories can be represented in various ways, and the representations in turn make use of dissimilar search and planning algorithms, including traditional optimization, evolutionary search [17] and reinforcement learning approaches [184, 201, 803] all the way to sophisticated transformer-based architectures and modern large language models [352]. Over 20 years ago, Cavazza et al. [184] introduced an interactive storytelling system built with the Unreal game engine that used **Hierarchical Task Network** planning to support story generation and anytime user intervention. Young et al. [1300] introduced an architecture called *Mimesis*, primarily designed to generate intelligent, plan-based character and system behavior at runtime with direct uses in narrative generation. Finally the IDtension engine [1120] dynamically generated story paths based on the player's choices; the engine was featured in *Nothing for Dinner*,[7] a 3D interactive story aiming to help teenagers living challenging daily life situations at home.

[7] http://nothingfordinner.org.

Fig. 9.3 A screenshot from the *Façade* (Procedural Arts, 2005) [721] game featuring its main characters: Grace and Trip. The seminal design and AI technologies of *Façade* popularized the vision of interactive narrative as it showcases the use of natural language processing and adaptive story generation in games. Played from a first-person perspective, the player interacts with Trip and Grace via text input assuming the role of a long-time friend

Similarly to traditional approaches of narrative and text generation, *interactive storytelling* in games relies heavily on stored knowledge about the (game) world. Games that rely on narratives—such as *Heavy Rain* (Quantic Dream, 2010)—may include thousands of lines of dialog which are manually authored by several writers. To enable interactive storytelling the game should be able to select responses (or paths in the narrative) based on what the player will do or say, as in *Façade* (Procedural Arts, 2005) [721] (see Fig. 9.3). To alleviate, in part, the burden of manually representing world knowledge, data-driven approaches can be used. For instance, one may crowdsource actions and utterance data from thousand of players that interact with virtual agents of a game and then train **virtual agents** to respond in similar ways using n-grams [834]. Or instead, one might design a system in which designers collaborate with a computer by taking turns on adding sentences in a story; the computer is able to provide meaningful sentences by matching the current story with similar stories available on the cloud [1110].

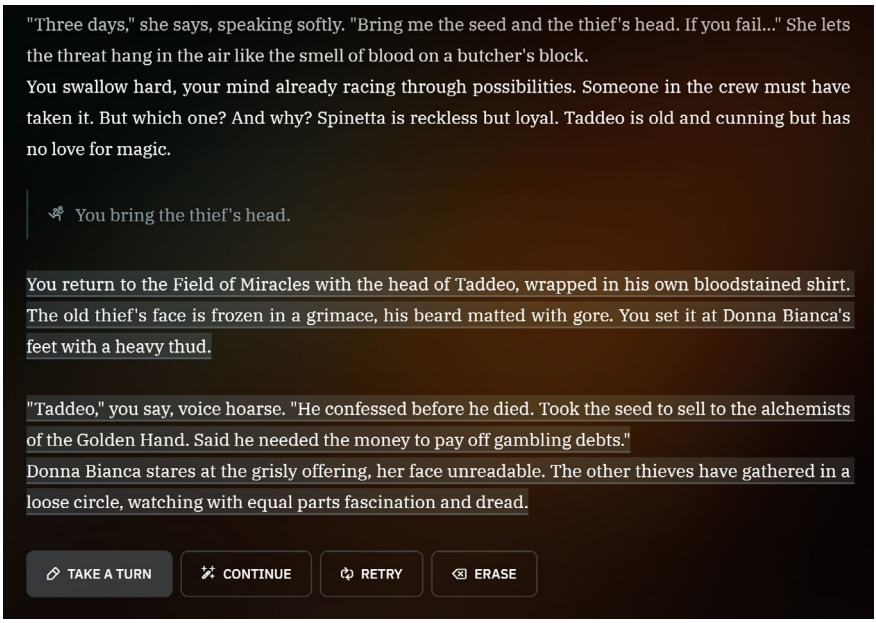


Fig. 9.4 A screenshot from Nick Walton's *AI Dungeon* text-adventure game. Screenshot taken from author's own gameplay

Given the textual nature of stories and narratives, unsurprisingly the recent breakthroughs of large language models and contemporary methods in text generation have also influenced and gradually dominated the generation of narrative in games [352, 629]. One might argue that the first (or at least the first to the best of our knowledge) text-based adventure game which was based on LLMs is the *AI Dungeon* (Latitude, 2019) [486] chat-based interactive storytelling game. *AI Dungeon*, which was originally powered by a fine-tuned version of GPT-2, allows the player to take actions through text prompts.[8] The game has evolved substantially since its first version (see Fig. 9.4) and has gradually incorporated more recent and effective LLMs that the player can pick before they start playing. Kobold AI[9] is another LLM-based text-adventure game example, out of the many available at the time of writing. A growing number of these LLM-games incorporate text-to-image models [961] in an attempt to co-create narratives with their corresponding visuals.

Another important aspect of interactive narrative and story-based PCG is their deep interrelation to **believable agents**. Narrative generation benefits from and influences the use of believable agents that interact with the player and are interwoven in the story plot. The narrative can yield more (or less) believability to

[8] https://play.aidungeon.com/.

[9] https://koboldai.net.

agents and thus the relationship between agents and the story they tell is important [184, 669, 880, 1322]. In that sense, the computational narrative of a game may define the arena for believable agent design. Research at the crossroads of believable agents and story-based PCG has also influenced the design and development of games. Starting from popular independent attempts like *Façade* (Procedural Arts, 2005) [721] (see Fig. 9.3), *Prom Week* [730] and *Nothing for Dinner* to the commercial success of *The Elder Scrolls V: Skyrim* (Bethesda Softworks, 2011), *Heavy Rain* (Quantic Dream, 2010) and *Mass Effect* (Bioware, 2007) narrative has traditionally been among the key factors of player experience and immersion; particularly in narrative-heavy games as the ones aforementioned.

Early examples of sophisticated computational narrative techniques crossing over from academia to commercial-standard products include the storytelling system *Versu* [329] which was used to produce the game *Blood & Laurels* (Emily Short, 2014). More recent efforts include prototype games that are inspired by *The Sims* (Electronic Arts, 2000) [851] simulating a virtual village with several villagers each equipped with an LLM. Their text generation capacities allows them to communicate with other agents and players and take actions that are provided and processed in a text format. The text-based interaction among the agents of this world allows emergent social phenomena to occur. The *AI people* (see Fig. 9.5) prototype game from *GoodAI* is another noteworthy example. Similarly to [851], the conversational NPCs of *AI people* are empowered by LLMs which, in turn, allows them to be socially interactive and take actions that are relevant to a text-based conversation they have with a player [389]. Beyond dialogue and NPC behavior we observe some early attempts on the use of LLMs for the generation of NPC *chatter*—i.e., idle dialogue text that does not require an answer. Findings presented in [783] suggest that the

Fig. 9.5 Screenshot of *AI people*. A player can interact with other NPCs via text. This text-based interaction affects the behavioral patterns of the NPCs and the social relationships developed among the agents of this world. Used with permission from [389]

vast majority of generated NPC responses are relevant to the context of a given NPC persona.

For the interested reader the *interactive fiction database*[10] contains an up-to-date detailed list of games built on the principles of interactive narratives and fiction. Finally note that the various ways AI can be used to play text-based adventure games and interactive fiction are covered in Sect. 6.8.

9.5 Rules and Mechanics

The game rules frame the playing experience by providing the conditions of play—for instance, winning and losing conditions—and the actions available to the player (game mechanics). Rules constitute necessary content as they are in a sense the core of any game, and a game's rules pervade it.

For most games, the design of their ruleset largely defines them and contributes to their success. It is common that the rule set follows some standard design patterns within its genre. For example, the genre of platform games is partly defined by running and jumping mechanics, whereas these are rare in puzzle games. Evidently, the genre constrains the possibility (design) space of the game rules. While this practice has been beneficial—as rule sets are built on earlier successful paradigms—it can also be detrimental to the creativity of the designer. It is often that players themselves can create new successful game variants (or even sub-genres) by merely altering some rules of an existing game. A popular example is the modification of *Warcraft III* (Blizzard, 2002). The player in that game was able to control a single "hero" unit which gradually gave rise to a new, popular game genre known as Multiplayer Online Battle Arenas (MOBA).

Early approaches to rule generation take a search-based approach, and are thus dependent on some way of evaluating a set of rules [587, 1165]. However, accurately estimating the quality of a set of game rules is very hard. Game rules differ from most other types of game content in that they are almost impossible to evaluate in isolation from the rest of the game. While levels, characters, textures, and many other types of content can to some extent be evaluated outside of the game, looking at a set of rules generally gives very little information of how they play. For a human, the only way to truly experience the rules of a game is to play the game. For a computer, this would translate to simulating gameplay in some way in order to evaluate the rules. In this sense, rules can be said to be more similar to program code than they are to e.g., pictures or music.

So how can simulated playthroughs be used to judge the quality of the rulesets? Several ideas about how to judge a game depending on how agents play it have been introduced. The first is **balance**; for symmetric two-player games in particular, balance between the winning chances of the two players is a generally positive feature of a given ruleset [466, 542]. Another idea is **outcome uncertainty**, meaning that

[10] http://ifdb.tads.org/.

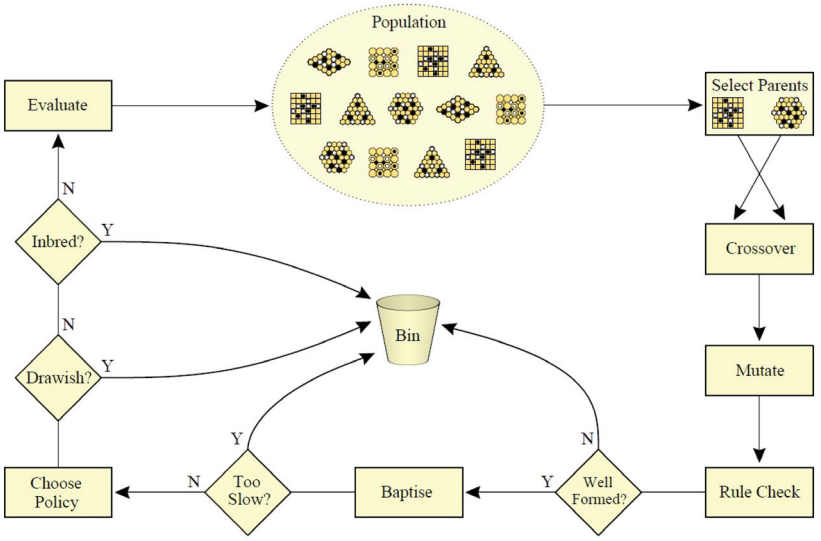
any particular game should be "decided" as late as possible [136]. Yet another idea is **learnability**: a good game, including its ruleset, is easy to learn and hard to master. In other words, it should have a long, smooth learning curve for the player, as learning to play the game is part of what makes it fun. This idea can be found expressed most clearly in Koster's *Theory of Fun* [581], but can also be said to be implicit in Schmidhuber's theory of artificial curiosity [1002] and in theories in developmental psychology [339]. Within work in game rule generation, attempts have been made to capture this idea in different ways. One way is to use a reinforcement learning agent to try to learn to play the game; games where the agent improves the most over a long time score the best [1170]. Another way of capturing this idea is to use several agents of different skill levels to try to play the game. Games score better when they maximize the performance difference between these agents [813]. This idea is also related to the idea of **depth** in games, which can be seen as the length of the chain of heuristics that can be acquired in a game [604].

Perhaps the most successful example of game rule generation within academic research is *Ludi* [136]. *Ludi* follows a search-based PCG approach and evolves **grammars** that represent the rules of board games (see Fig. 9.6). The fitness function that drives the rule generation is composed by several metrics that estimate good design patterns in board games, such as game depth and rule simplicity. A successfully designed game that came out of the *Ludi* generator is named *Yavalath* [134] (see Fig. 9.6). The game is played on a 5×5 hexagonal board by two or three players. *Yavalath*'s winning and losing conditions are very simple: you win if you make a line of four tokens, whereas you lose if you make a line of three tokens; the game is a draw if the board fills up. The *Ludi* system was later developed into the much more extensive *Ludii* environment, with a more expressive game description language, with which more than 1,000 games have been specified [881]. A combination of LLMs and search-based PCG has been used to create novel, playable and interesting games within this language [1148]. *Ludii* is discussed further in Chap. 6.

Another example of a board game that is created through a rule generation process, but without a domain-specific language, is *Sumplete* (Hey, Good Game, 2023). *Sumplete* is a logic-based, combinatorial number-placement puzzle game—primarily inspired by *Sudoku*—created in a mixed-initiative fashion using ChatGPT.[11] The core difference between *Sudoku* and *Sumplete*, is that each row and column has a target sum value associated with it. In contrast to *Sudoku*, the goal of a *Sumplete* player is to identify which numbers to delete so that the sum of each column and row matches its corresponding target sum.

One of the earliest examples of *video game* rule generation is Togelius and Schmidhuber's experiment with generating simple Pac-Man-like games [1170]. In that study rules are evolved to maximize the learnability of player agents as measured via simulated playthroughs. *Mechanic Miner* [238] is another early evolutionary computation tool that assists the design of simple game mechanics for puzzle platform games through simulated playthroughs. Another example is the work by Nielsen et al. in which game rules are represented using the **video game description language** [814].

[11] https://openai.com/chatgpt/.



(a) The *Ludi* game rule generator

(b) A deluxe version of the *Yavalath* game

Fig. 9.6 The *Ludi* game rule generator **a** and its game *Yavalath* **b**. Image **a** is adapted from [1176]. Image **b** is published with permission by Cameron Browne and Néstor Romeral Andrés

Using Answer Set Programming (a solver-based method) [124] rather than search-based methods, rules have been generated for simple 2D games that, however, respect constraints such as playability (i.e., the victory condition is attainable) [1055]. It is fair to say that none of these attempts to generate video games have been able to produce *good* games, i.e., games that anyone (except the creator of the system that creates the games) would want to play. This points to the immense challenge in accurately estimating the quality of a video game rule set. One of the reasons this seems to be a more challenging problem than estimating the quality of a board game rule set is the time dimension, as human reaction time and ability to estimate and predict unfolding processes play an important role in the challenge of many video games. For more examples and in-depth analysis on rule and mechanics generation the interested reader is referred to the "rules and mechanics" chapter of the PCG book [806].

9.6 Games

Game generation refers to the use of PCG algorithms to computationally design new complete games. The vast majority of PCG studies so far, however, have been very *specific* to a particular game facet or domain. It is, for instance, either a level that is generated or the audio for some level but rarely both. Meanwhile it is surprising to think that the relationship between the different facets is naturally interwoven. A characteristic example of the interwoven nature among game facets is given in [643]: player actions—viewed as a manifestation of game rules—are usually accompanied by corresponding sound effects such as the sound of Mario jumping in *Super Mario Bros* (Nintendo, 1985). Now let us think of a PCG algorithm that introduces a new rule to the game—hence a new player action. The algorithm automatically constrains the sound effects that can be associated to this new action based on a number of factors such as the action's duration, purpose and overall contribution to the game plot. Actions and sounds appear to have a cause and effect (or hierarchical) relationship and a PCG algorithm would naturally prioritize the creation of the action before it generates its sound. Most relationships between facets, however, are not strictly hierarchical or unidirectional. For example, a game level can be successful because of a memorable landmark as much as the gameplay it affords [643]. Similarly, the narrative of a game relies on a multitude of factors including the camera placement as well as the visuals and the sounds.

The game generation topic has attracted a growing interest in recent years even though the relationship between the different game facets is considered rarely. Most game generation projects focus on a single facet of a game and do not investigate the interaction between different facets. The rule generator for Pac-Man-like games [1170], for instance, evolves different rules for different colored agents but it does not evolve the agents' color to indicate different playing strategies. Similarly to the ghosts of *Pac-Man* (Namco, 1981) we could imagine that red represents an aggressive behavior whereas orange represents a passive behavior.

Among the few notable and early attempts of game generation, *Game-O-Matic* [1183] is a game generation system that creates games representing ideas. More specifically, Game-O-Matic includes an authoring tool in which a user can enter entities and their interactions via **concept maps**. The entities and interactions are translated, respectively, into game visuals and game mechanics. One of the first preliminary discussions on how multi-facet integration might happen is offered by Nelson and Mateas [804]. In their paper they present a system for matching sprites (visuals) to very simple WarioWare-style mechanics. Their system is somewhat similar to *Game-O-Matic*, but works a bit differently: instead of the designer specifying verbs and nouns they want a game to be about, they give the system constraints on how verbs and nouns relate in the game (for example, a chasing game needs a "prey" sprite that is something that can do things like "flee" or "be hunted" and so on). The system then uses **ConceptNet**[12] and **WordNet**[13] to generate games that fit these constraints.

Arguably one of the most elaborate examples of game generation is ANGELINA by Cook et al. [232, 233, 235–237]. ANGELINA[14] is a game generator that has seen several developments over the years and is currently capable of evolving the rules and levels of the game, collecting and placing pieces of visuals and music (that are relevant to the theme and the emotive mood of the game), giving names for the games it creates and even creating simple commentaries that characterize them. ANGELINA is able to generate games of different genres—including platformer games (see Fig. 9.7) and 3D adventure games—some of which have even participated in **game design** competitions [233].

More recently, Cook introduced *Puck* [231], an automated game designer that builds on the principle of continuous and human realistic (i.e., slow) creativity processes which are based on exhaustive content generation methods [1093].[15] Beyond PCG systems that are trained to act autonomously, game generators can be integrated in a **mixed-initiative** interactive framework [1279] by which AI and human designers take part in a co-creative process of designing rules for a game. An indicative example is the *Mechanic Maker* series of tools which allow users to make 2D games by interacting with AI agents that learn game rules through human demonstration [980]; in a more recent version, *Mechanic Maker* 2.0, employs RL agents for assessing any generated set of rules [388].

A somewhat separate strand of game generation research concerns systems that create games from data, particularly openly accessible data. These systems, or the games they generate, have been referred to as *Data Games* [355]. This includes systems such as *Open Data Monopoly*, which generates Monopoly boards based on demographic and geographic information and the player's chosen valuation of demographic indicators; the game you play therefore becomes of exploring your own values [354]. Simpler data games, such as *Bar Chart Ball*, are essentially playable

[12] http://conceptnet.io/.

[13] https://wordnet.princeton.edu/.

[14] http://www.gamesbyangelina.org/.

[15] *Puck* is freely available online: https://gamesbypuck.itch.io/puck.

Fig. 9.7 ANGELINA's Puzzling Present game. The game features an invert gravity mechanic that allows a player to overcome the high obstacle on the left and complete the level. Image obtained with permission from http://www.gamesbyangelina.org/

data visualizations. In Bar Chart Ball, your selection of what aspects of what data to visualize is the one and only game mechanic, and mastery of the data allows you to guide the ball proficiently [1158]. A more complex data game is *Data Agent*, a system that creates murder mystery games featuring real people and locations from Wikipedia, OpenStreetMaps, and other public data sources [57, 406]. All of these systems use various AI methods in the game generation, typically including evolutionary algorithms to select which elements of vast troves of data to create game content from.

Other systems try to build games from information and suggestions provided by the player. An early example of this *A Rogue Dream*, which takes a single word as input and generates a "dream" that becomes a roguelike game [234]. More recently, the *Word2World* system uses LLMs to create a narrative drive 2D world with quests and characters from a prompt [797].

The systems above make some initial, yet important, steps towards game generation and they attempt to interweave the different domains within games in a meaningful way, mostly in a hierarchical fashion. However, PCG eventually needs to rise to the challenge of tackling the compound generation of multiple facets in an *orchestrated* manner [630, 636, 1165]. An indicative early study on the fusion of more than one generative facet (domain) in games is the one performed by Karavolos

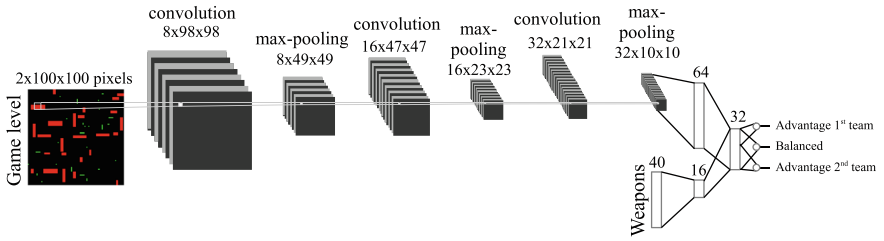


Fig. 9.8 A convolutional neural network (CNN) architecture used for fusing levels and weapons in first-person shooters. The network is trained to predict whether a combination of a level and a weapon would yield a balanced game or not. The CNN can be used to orchestrate the generation of a balanced level given a particular weapon and vice versa. Image adopted from [541] with permission

et al. [541]. The study employs surrogate-based evolution to derive the common generative space—or the common patterns—of game levels and weapons in first-person shooters. The aim of this **orchestration** process between level design and game design is the generation of level-weapon couplings that are balanced. The unknown mapping between level representations and weapon parameters is learned by a deep convolutional neural network which predicts if a given level with a particular set of weapons will be balanced or not. Balance is measured in terms of the win-lose ratio obtained by AI bots playing in a **deathmatch** scenario. Figure 9.8 illustrates the architecture used to fuse the two domains. In a more recent study [542] a similar CNN surrogate model learned to balance FPS levels and FPS game rules. Such a model is thus able to adjust in-game character classes and levels (or both) towards designer-specified targets. For the interested reader in the orchestration process we further elaborate on the topic in the last chapter of this book; some early discussions on this vision can also be found in [630, 636].

9.7 Evaluating Content Generators

Creating a generator is one thing; evaluating it is another. Regardless of the method followed, generators should be evaluated on their ability to achieve the desired goals of the designer. Arguably, the generation of any content is trivial; the generation of **valuable** content for the task at hand, on the other hand, is a rather challenging procedure.

9.7.1 Content Evaluation: Why Is It Difficult?

But what makes the evaluation of content so difficult? First, it is the diverse, stochastic and **subjective** nature of the *users* that experience the content. Whether players or designers, users have dissimilar personalities, gameplay aims and behavior, emotional responses, intents, styles and goals [639]. When designing a PCG system it is critical to remember that we can potentially generate massive amounts of content for designers to interact with and players to experience. It is thus of utmost importance to be able to evaluate how successful the outcomes of the generator might be across dissimilar users: players and designers. While content generation is a cheap process relying on algorithms, design and gameplay are expensive tasks relying on humans who cannot afford the experience of bad content. Second, content quality might be affected by *algorithms* and their underlying **stochasticity**, for instance, in evolutionary search. Content generators often exhibit non-deterministic behavior, making it very hard to predict a priori what the outcomes of a particular generative system might be. The unpredictable hallucinations of large multimmodal models are also representative of the challenges one might face when they attempt to generate content that satisfies certain quality criteria.

9.7.2 Content Properties: Function Versus Aesthetics

Particular properties of content can be *objectively* defined and tested whereas other properties of it can only be assessed *subjectively*. It is only natural to expect that functional properties of content quality can be objectively defined whereas a large part of its aesthetics can only be defined subjectively. For instance, playability of a level is a functional characteristic that can be objectively measured—e.g., an AI agent manages to complete the level; hence it is playable. Balance and symmetry can also be objectively defined to a degree through estimates of deviation from a norm—it may be a score (balance) or a distance from a central choke point in the map (symmetry). There are games, however, for which content balance, symmetry and other functional properties are not trivially measurable. And of course there are several aspects of content such as the comprehensibility of a narrative, the pleasantness of a color scheme, the preference for a room's architectural style or the graphics style, and the experience of sound effects and music that are not objectively measured.

Functional, objectively defined, content properties can be expressed either as metrics or as constraints that a generator needs to satisfy. **Constraints** can be specified by the content designers or imposed by other game content already available. For instance, let us assume that a well-designed generated strategy game level needs to be both balanced and playable. Playability can form a simple binary constraint: the level is playable when an AI agent manages to complete it; it is non-playable otherwise. Balance can form another constraint by which all items, bases and resources are accessible to similar degrees by all players; if equal accessibility is below a

threshold value then the constraint is not satisfied. Next, let us suppose we wish to generate a new puzzle for the map we just generated. Naturally, the puzzle needs to be compatible with our level. A PCG algorithm needs to be able to satisfy these constraints as part of its quality evaluation. **Constraint satisfaction** algorithms such as the **feasible-infeasible two-population genetic algorithm** [640, 644], constrained divergent search rewarding content *value* but also content *novelty* [644], *surprise* [404, 405] or *diversity* [403, 627], and constraint solvers such as **answer set programming** [1056] are able to handle this. The generated results are within constraints, thereby valuable for the designer. Value, however, may have varying degrees of success and this is where alternative methods or heuristics can help, as we cover in the section that follows.

9.7.3 How Can We Evaluate a Generator?

Generally speaking, a content generator can be evaluated in three ways: directly by the **designer** or indirectly by either **human players** or **AI agents**. Designers can directly observe properties of the content generator and take decisions based on data **visualization** methods. Human players can play and test the content and/or provide feedback about the content via subjective reporting. AI agents can do the same: play the content or measure something about the content and report it to us in the form of a quality metric, or metrics. Clearly, machines cannot *experience* the content but they can, instead, simulate playing it and provide us estimates of content experience. The overall evaluation process can very well combine and benefit from any of the aforementioned approaches. In the remainder of this section we cover the approaches of data visualization, AI automated playtesting and human playtesting in further detail.

Visualization

The visualization approach to content quality assessment is associated with a) the computation of meaningful metrics that can assign measurable characteristics to content and b) ways to visualize these metrics. The task of metric design can be viewed as the equivalent of utility function design (see Chap. 2). As such, designing good content generation quality metrics in an ad-hoc manner involves a degree of practical wisdom. Metrics, for instance, can be based on the **expressive range** of the generator under assessment, so-called expressivity metrics [1016, 1058]. The analysis of a generator's expressivity gives us access to the potential overall quality of the generator across its full range of generative space. The generative space can then be visualized as **heatmaps** or alternative graphical representations such as 2D or 3D scatter plots (see Fig. 9.9 for an example). It is one thing if a level generator is able to create only a few meaningful or playable levels and another if the generator is robust

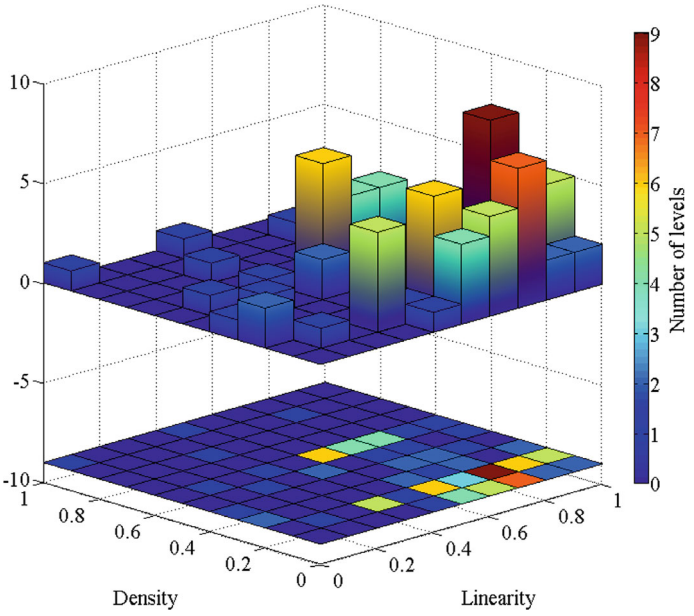


Fig. 9.9 The expressive range of the *Ropossum* level generator for the metrics of linearity and density. Adapted from [1016]

and consistent with respect to the playability of its generated levels. It is also one thing if our generator is only able to generate levels with very specific characteristics within a narrow space of its expressive range and another if our level generator is able to express a broad spectrum of level properties, yielding uniformly covered expressive ranges. Such information can be directly visible on the illustrated heatmaps or scatter plots. Alternatively, **data compression** methods can be used directly on the generated content and offer us 2D or 3D representations of the generative space, bypassing the limitations of ad-hoc metric design. An example of this approach is the use of autoencoders for compressing the images produced by the DeLeNoX autonomous content generator [633].

Alternative ways for illuminating the expressive capacity of content generators can be adopted from **quality diversity** methods for PCG introduced in [405]. An example of such an illumination process can be seen in Fig. 9.10 depicting a map of mazes as generated from a MAP-Elites variant—named *Monte Carlo Elites* [1015]. The map showcases the fittest individuals (i.e., mazes) generated by the algorithm across two supposedly important features for maze design: average corridor length and number of corners. Design features as such can be selected by the level designer and the expressivity of the generator can then be evaluated both through observation and via evaluation metrics such as the **QD score** [781].

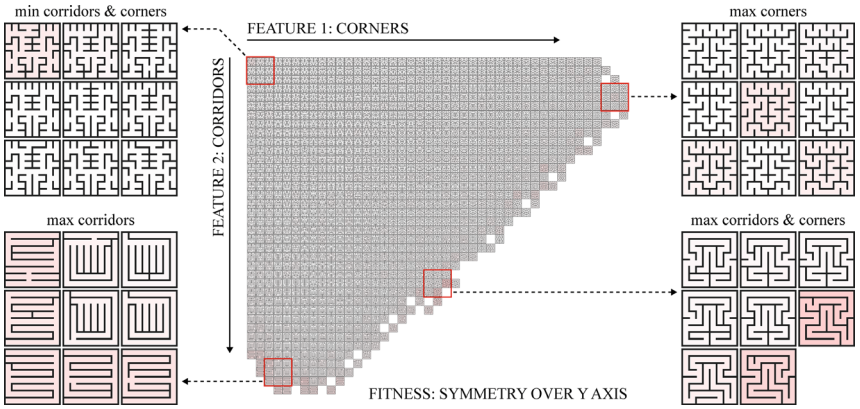


Fig. 9.10 Feature map of a quality diversity algorithm used for illuminating the generative capacity of a QD maze generator across two features: maximize corridors and corners. The fitness used in this example is the maze's symmetry over the Y axis. The quality of a maze (i.e., its fitness) is represented by its whiteness. Image inspired by [1015] and created by the authors of the paper

AI Players

Using AI to **playtest** our generator is a safe and relatively cheap way to rapidly retrieve quality metrics for it without relying on human playtesting. In the same way that a search-based PCG method would use AI to simulate the content before generating it, AI agents can test the potential of a generator across a number of metrics and return us values about its quality. The metrics can be in the form of classes—for instance, successful or unsuccessful tests performed for completing an area of a level—scalar values—e.g., a level's balance—or even ordinal values—e.g., the rank of the level in terms of challenge. The relevance of the metrics to the generator's quality is obviously dependent on the designer's ad-hoc decisions. Once again, designing appropriate metrics for our AI agent to compute is comparable to the challenge of designing any *utility* function. An interesting approach to AI-based testing is the use of **procedural personas** [60, 459, 461]. These are data-driven inferred models of dissimilar play styles that potentially imitate the different styles of human play. In a sense, procedural personas provide a more human-realistic approach to AI-based testing of a generator. Finally, by visualizing particular game artifacts or simulating them through the use of AI agents we can understand what is possible within our content space such as the possibility space of generated platformer levels. We can also infer how rules and functions operate in whatever we generate such as balance and playability. Ultimately, we can test to which degree the data we extract through AI agents relates to data we extract from human playtesting [801].

Human Players

The most obvious approach to evaluate the content experienced by players is to *explicitly* ask them about it. Playtesting is regarded as an expensive way to test a generator but it can be of immense benefit for content quality assurance, in particular for those aspects of content that cannot be measured objectively—e.g., aesthetics and playing experience. A **game user study** can involve a small number of dedicated players that will play through various amounts of content or, alternatively, a **crowdsourcing** approach can provide sufficient data to machine learn content evaluation functions (see [208, 624, 1027] among others). Data obtained can be in any format including classes (e.g., a binary answer about the quality of a level), scores (e.g., the likeliness of a sound) or ranks (e.g., a preference about a particular level). It is important to note that playtesting can be complemented by **annotations** coming from the designers of the game or other experts involved in content creation. In other words, our content generator may be labeled with both first-person (player) and third-person (designer) annotations. Further guidelines about which questionnaire type to use and advice about the design of user study **protocols** can be found in Sect. 2.4 and Part IV that follows.

9.8 Further Reading

An early analysis of the creative facets (content types) available in games and their corresponding content generators is introduced in [643] for the interested reader. The opportunities and challenges of orchestrating the generation process of more than one creative facet is detailed in [636]. As already mentioned at the beginning of this chapter, a great resource for PCG research across all content types covered here (and beyond) is the publication database of the *PCG workshop*[16] that runs since 2010 [632]. The database features hundreds of papers dedicated to all content types, dominated by level and world generators. In this section we also list some indicative frameworks, books and survey papers that have focused on various methodologies for level generation such as search [1176, 1289], ML [419], reinforcement learning [559, 1239], deep learning [654], quality diversity [405], LLMs [365] and survey papers on the recent history of academic PCG [632]. Finally, the topic of PCG evaluation is covered in the final chapter of the PCG in Games textbook [1023].

9.9 Exercises

PCG offers endless opportunities for generation and evaluation across the different creativity domains in games and across combinations of those. As an initial step we

[16] https://www.pcgworkshop.com/.

would recommend the reader to start experimenting with maze generation, platformer level generation, and level generation via RL and then test their methods across various PCG problems (as outlined in the following sections). The website of the book contains details regarding all provided frameworks and potential exercises.

9.9.1 Maze Generation

Maze generation is a very popular type of level generation and relevant for several game genres. In the first exercise we recommend that you develop a maze generator using both a constructive and a search-based PCG approach and compare their performance according to a number of meaningful criteria that you will define. The reader may use the Unity 3D open-access maze generation framework which is available at: http://catlikecoding.com/unity/tutorials/maze/. Further guidelines and exercises for maze generation can be found at the book's website.

9.9.2 Platformer Level Generation

The platformer level generation framework is based on the *Infinite Mario Bros* (Persson, 2008) framework which has been used as the main framework of the Mario AI (and later Platformer AI) Competition since 2010. The competition featured several different tracks including gameplay, learning, Turing test and level generation. For the exercises of this chapter the reader is requested to download the level generation code from the Mario AI framework (https://github.com/amidos2006/Mario-AI-Framework) and apply constructive and generate-and-test methods for the generation of platformer levels. The levels need to be evaluated using one or more of the methods covered in this book. Further details and exercises with the Mario AI level generation framework can be found at the book's website.

9.9.3 Level Generation via RL

A recent framework students may use to explore the idea that levels can be designed via RL processes is the *OpenAI GYM environment for PCG via Reinforcement Learning* available at: https://github.com/amidos2006/gym-pcgrl. Students are tasked to pick any of the possible game genres—including puzzle, platformers, and dungeon crawlers—and test the level generation capacity of RL across various reward functions, content representations and RL hyperparameters.

9.9.4 PCG Benchmark

A new PCG framework—launched at the time of writing—allows the testing and comparison of generative methods across a variety of PCG problems in games. The *PCG benchmark* (available at: https://github.com/amidos2006/pcg_benchmark) follows the design methodology of OpenAI Gym making it easy to use, test, and expand for new problems. Currently it supports more than 12 problems in content generation: from creating 2D levels and rule sets for simple arcade games, to creating isometric buildings, all the way to creating playable arcade dungeon crawler games. Students may pick one of the several PCG problems supported, select an appropriate PCG method to solve it and evaluate to which degree the generated content passes the *quality*, *diversity*, and *controllability* functions provided by the framework.

9.10 Summary

This chapter detailed the various types of content that can be generated in games. From levels and visuals, to audio and narrative, to rules and finally entire games. For each content type we presented indicative methods and studies from academic PCG work and also outlined popular tools and commercial games that feature such content generators. The chapter ends with a discussion on the important topic of evaluation, the challenges it brings, and a number of evaluation approaches one might consider such as data visualization, AI and human players.

We have so far covered the most traditional use of AI in games (Part II) and the use of AI for generating parts of (or complete) games (this part). The next part is dedicated to the human player and the ways we can use AI to model aspects of their behavior and their experience.

Part IV
Model

Chapter 10
Player Modeling

Abstract This chapter introduces the subfield of player modeling, one the three constituent parts of AI and games as defined in this book. The chapter starts with laying out the various reasons for wanting to model game players, and defining player modeling by comparison with other tasks such as player imitation. This is followed by a comprehensive taxonomy of player modeling, focused on the role of the model, the methodological approach taken, and the types of input to the model and output from it. The discussion of methodological approaches introduces the distinction between top-down models, which are informed by theories from fields such as psychology, game studies, and neuroscience, and bottom-up theories which are purely driven by machine learning. Several theories of player experience are discussed in some detail.

Games, gameplay experience and learning are tightly associated. Good games are designed to elicit great experiences. In response, games are played and mastered because they are able to elicit great experiences. A great amount of learning occurs in the interaction between games and gameplay as players improve their skills during play motivated by their experiences. Thus, the affective, cognitive and behavioral responses of human players during gameplay are the driver of both the gameplaying and the learning that occurs through it.

While Part II focused on AI players, this part of the book is dedicated to *human* players of games and the use of AI for modeling them. This area of research is often called **player modeling** [1054, 1288]. We take player modeling to mean the detection, prediction and expression of human player characteristics that are manifested through cognitive, affective and behavioral patterns while playing games. In the context of this book, player modeling studies primarily AI models of players. By *model* we refer to a mathematical representation—it may be a rule set, a vector of parameters, or a set of probabilities—that captures the underlying function between the characteristics of the player, their interaction with the game, and their response to that interaction. Given that every game features at least one player (with some notable exceptions [93]), and that player modeling affects work on gameplaying and content generation, we consider the modeling of player behavior and experience as a core use of AI in games [1268, 1291]. In the remainder of this chapter we first discuss reasons why AI should be used to model players (Sect. 10.1) and then define the core ingredients of player modeling (Sect. 10.2). In Sect. 10.3 we provide a high

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_10

level **taxonomy** of player modeling *roles*, tasks and use cases of player modeling in games (Sect. 10.3.1), *approaches* (Sect. 10.3.2), and aspects with regards to the model's *input and output* (Sect. 10.3.3).

10.1 Why Model Players?

The study of human players is, by nature, a multidisciplinary field founded in psychology, cognitive science, human computer interaction, affective sciences and data sciences. **Psychology** is central to the study of players as it offers a plethora of methods for the analysis of human behavior, cognition and emotions. Cognitive science has long studied models of human cognition and it is as relevant to player modeling as it is foundational to AI [975]. Certain sub-fields of **human computer interaction** are tightly associated with and influence the advancement of player modeling. Particularly, the fields of **affective computing** [878], **user modeling** and **user experience research** focus on modeling and simulating human behavior, human experience and perceptions of emotions (i.e., human affect).

Player modeling is related to all aforementioned fields (and more) but with a particular focus: it studies humans in the domain of games. Notably, games can yield dynamic and complex player experiences, the manifestations of which cannot be captured trivially by standard methods in empirical psychology, affective computing or cognitive modeling research. The high potential that games have in affecting players is mainly due to their ability to place the player in a continuous mode of interaction, which, in turn, elicits complex cognitive, affective and behavioral responses. Thus, the study of the player may not only contribute to the design of improved forms of human-computer interaction, but also can advance our knowledge of human behavior and experience at large.

As mentioned earlier, every game features at least one user—the player—who controls some aspect of the game environment. The **player character** could be visible in the game as an **avatar** or a group of entities [162], or could be invisible as in many puzzle games and casual games. Avatar control may vary from the relatively simple, such as movement in an orthogonal grid, to the highly complex, such as having to decide several times per second between hundreds of different possibilities in a 3D world. The interaction between the player(s) and the game is dynamic, real-time and in many cases fast-paced. The interaction is also *rich* in the sense that many modalities of interaction may be involved and that the information exchange between the game and the player may both be fast and entail large amounts of data for us to process. If the game is well-designed, the interaction is also highly engaging for the player. Since great amount of information that can be extracted through this interaction and used for creating models of the player, the game should be able to learn much about the person playing it (perhaps as much as the person learns about the game while playing it). In fact if one wishes to be fair, there is no reason why the model should not know more about how you play than you do. Given these intricacies, understanding and modeling the interaction between the player and the

game can be seen as a *holy grail* of game design and development. Designing the interaction and the emergent player experience *right* results in successful games that manage to elicit unique experiences.

10.2 What Is Player Modeling?

To clarify the differences between AI and human players in terms of the modeling task let us consider the following two scenarios. Player models are either built on demonstrations provided by humans or AI-controlled agents in the game (see rows of Fig. 10.1). The models can either be used for playing the game or modeling players (see columns of Fig. 10.1). Ultimately, the question of why you might want to employ an AI to model players of a game can be reduced to two more specific questions.

Is the Model Used for Playing the Game? It is very common for one to model gameplaying behavior with the purpose of building an AI gameplaying agent (see left column of Fig. 10.1). The main drivers for doing so is to build efficient agents that can win a game, agents that can play and experience the game as humans do, or agents that will test a game thoroughly. The demonstration data the model is trained

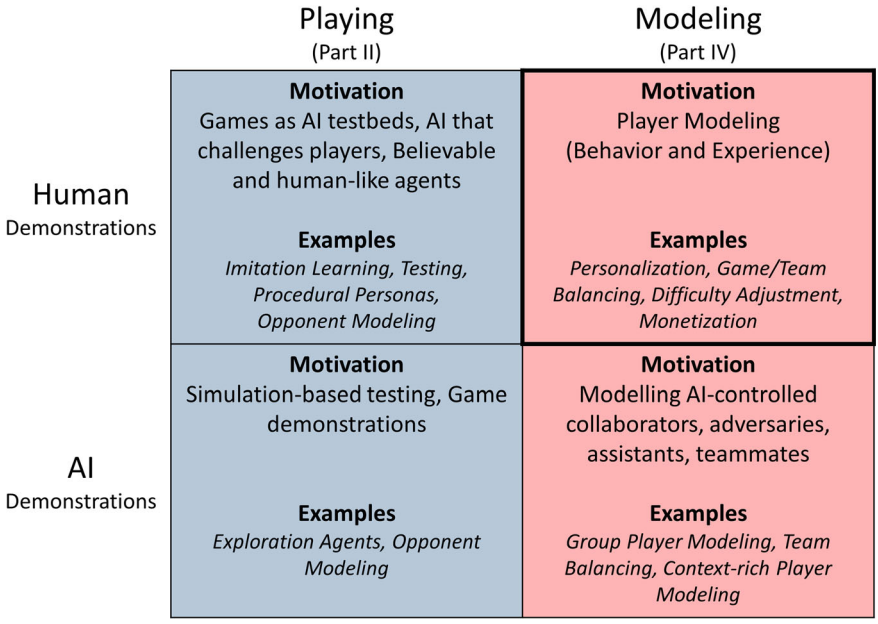


Fig. 10.1 Defining player modeling. The two possible goals (playing, modeling) AI can aim for and the two data sources (human, AI) it can model. We provide a summary of motivations and some indicative examples for each of the four options. Player modeling covered in this part of the book primarily studies humans as modeled from human demonstrations (top right outlined box)

upon (AI or human demonstrations) determines the type of algorithm and the purpose of the trained model. If human demonstrations are available an AI bot can be trained to imitate the way humans play or even surpass the performance of the best available human gameplay performance. Examples include AI agents that were trained to imitate expert play trajectories for *Starcraft II* (Blizzard Entertainment 2010) [1219] or *Atari 2600 Games* [203]. Sometimes the terms player modeling and **opponent modeling** [90, 358, 991] are used interchangeably when a human player is modeled. Opponent modeling, however, is a more narrow concept referring to predicting behavior of an **adversarial player** when playing to win in an imperfect information game like **Poker** [90] or *StarCraft* (Blizzard Entertainment 1988) [829]. We may want to model collaborators or simply players in general. If human demonstrations are not available a model might instead observe data that is demonstrated from AI-controlled agents available in a game environment. Modeling such data could yield exploratory agents (e.g., via reinforcement learning) that consider the behavior of their teammates, assistants or adversaries to shape their gameplaying policy.

Independently of where training data is coming from (AI-controlled NPCs or humans) when a model of any player is primarily considered by an AI algorithm for *playing* the game then we can safely assume that the model primarily enhances the capacities of the gameplaying agent. Any use of models for gameplaying is not core to what we consider in this part of the book. All studies that utilize human player or NPC models for playing a game are detailed in Part II, instead. The blue colored boxes of Fig. 10.1 illustrate that distinction.

Does the Model Use AI Demonstrations?

One could arguably detect behavioral, emotional or cognitive aspects of both human players and non-human (AI-controlled) players (notwithstanding the actual existence of emotions in the latter). This distinction can be observed across the two rows of Fig. 10.1. Modeling the experience of an AI-controlled NPC would seem to be a futile exercise, as one can hardly say that an NPC possesses actual emotions or cognition. Modeling the behavior of an NPC is also generally of little interest, at least if one has access to the game's code: a perfect model for the NPC already exists. NPC modeling, however, can be a useful testbed for player modeling techniques, for instance, by comparing the model derived from human players with the hand-crafted one. More interestingly, it can be an integral component of AI that adapts its behavior in response to the dynamics of the NPCs—as in [47]. One could also imagine that, if the NPC is very computationally complex, there might be situations where you want a computationally cheaper, if imperfect, model of it.

While all earlier examples describe models trained on NPC data that are used to play games (see bottom left box of Fig. 10.1) it is also possible to use demonstrations for the purpose of modeling the AI-controlled NPCs (see bottom right box of Fig. 10.1). Modeling NPCs such as teammates, collaborators, or adversaries could be useful for human player modeling if the derived models can enhance the predictive capacity of the model. This use of AI demonstrations for modeling can also be helpful for game balancing and modeling of player groups that contain AI players as

in *Left 4 Dead* (Valve 2008). When data from AI entities are considered in addition to human demonstrations one expects that player models of higher predictive power will likely emerge since the context for the model is richer. One can thus view this use of AI demonstrations as a form of data augmentation for player modeling.

While the challenges faced in modeling AI-controlled NPCs are substantial, the issues raised from the modeling of human players define a far more complex and important problem for the understanding of player behavior and experience. In this part of the book, we focus on aspects that can be detected from, modeled from, and expressed in games with human players [1288]. We thus explicitly exclude the modeling of NPCs from further discussion. In our definition, player modeling is modeling of a **human player**.

Defining Player Modeling

If you have answered negatively both aforementioned questions then you are reading the right part of this book. Within the chapters included in this part we are concerned with AI that considers observations of *human* players for the purpose of *modeling* them; refer to the top right box of Fig. 10.1. We also make a distinction between player modeling [200, 476] and **player profiling** [1288]. The former refers to modeling complex *dynamic* phenomena during gameplay interaction, whereas the latter refers to the categorization of players based on *static* information that does not (or will not likely) alter during gameplay. Information of static nature includes personality, cultural background, gender and age. We put an emphasis on the former, but will not ignore the latter, as the availability of a good player profile may contribute to the construction of reliable player models.

In summary, player modeling—as we define it in this book—is the study of computational means for the modeling of a player's experience or behavior which is based on theoretical frameworks about player experience and/or data derived from the interaction of the player with a game [1268, 1288]. Player models are built on dynamic information obtained during game-player interaction, but they could also rely on static player profiling information. Unlike studies focusing on taxonomies of *behavioral* player modeling—e.g., via a number of dimensions [1054] or direct/indirect measurements [1030]—we view player modeling in a holistic manner including cognitive, affective, personality and demographic aspects of the player. Moreover, we exclude approaches that are not directly based on human-generated data or not based on empirically-evaluated theories of player experience, human cognition, affect or behavior.

Here we introduce the core components of player modeling that will serve as our basis for the remaining chapters of Part IV. Generally speaking, and irrespective of the application domain, computational models are characterized by three core components: the input the model will consider, the computational model per se, and the output of the model (see Fig. 10.2). The model itself is a computational mapping between the input and the output. The mapping is hand-crafted, derived from data, or a mix of the two. In this chapter we will go through a taxonomy of roles, approaches and important aspects of the model input and model output. Chapter 11 will detail the

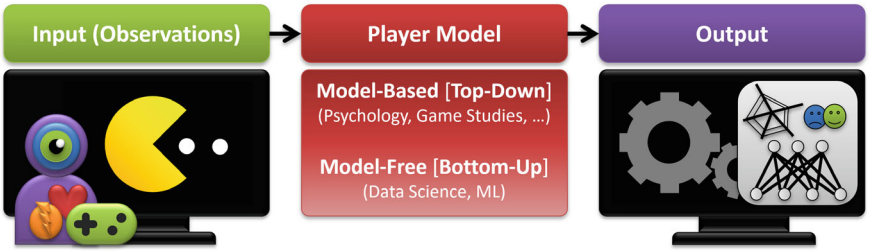


Fig. 10.2 A high level overview of a player model. The model senses inputs from the player (observations) such as gameplay patterns, physiological signals or the visual feed, and predicts labeled outputs related to player behavior (e.g., completion times), experience (e.g., player motivation) or even unlabeled player embeddings. In this chapter we focus on a taxonomy of player modeling and primarily discuss its roles (Sect. 10.3.1), the distinction between model-based and model-free approaches (Sect. 10.3.2) and important aspects of a model's input and output (Sect. 10.3.3)

various options for the input and the output of the model as well as the AI methods one can use to model players depending on the sensed input and output. Chapter 12 will detail the aspects of players a model can represent through a number of use cases. Part IV does not intend to provide an exhaustive review of player modeling studies under the above definition, but rather an introduction and a high-level taxonomy that explores the possibilities with respect to the modeling approach, the model's input and the model's output.

10.3 Taxonomy: Player Modeling

There are numerous ways a player can interact with a game, various AI methods that can capture this interaction, multiple disciplines dedicated to the study of human behavior and play, and multimodal ways we can observe players. In this section we introduce a taxonomy (as in the earlier parts of the book) so that we highlight the differences and emphasize the similarities among all these aspects. Our player modeling taxonomy contains a number of key dimensions that affect the design and deployment of a player model. Similarly to the taxonomies introduced for gameplaying and PCG, some dimensions are binary while some represent aspects that usually lie somewhere on a continuum of that dimension. We start discussing the six possible *roles* of PM through a dedicated taxonomy (Sect. 10.3.1). We then discuss a critical dimension related to the player modeling *approach* used (Sect. 10.3.2) and we conclude the presentation of the taxonomy with aspects related to the *input and output* of the player model (Sect. 10.3.3) that affect the use of the model, the method employed and its ultimate role. As with the gameplaying and PCG taxonomies presented earlier, Fig. 10.3 illustrates all aspects of the player modeling taxonomy we discuss here.

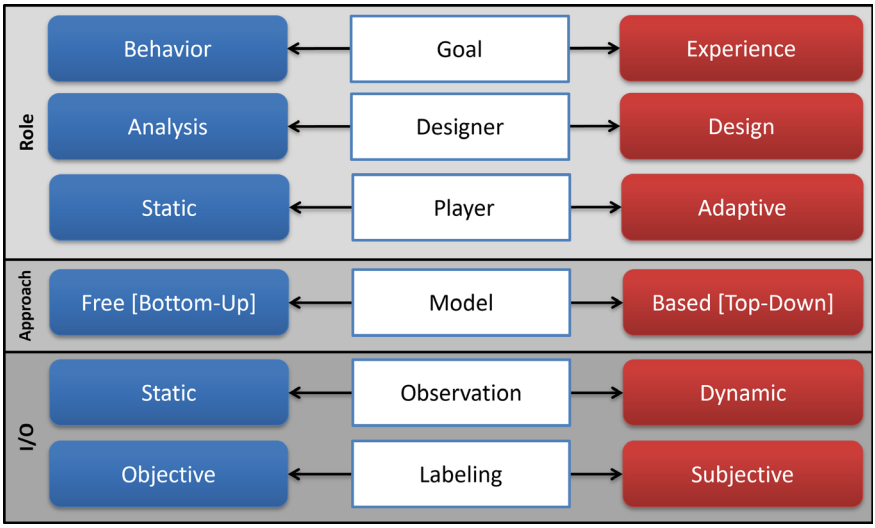


Fig. 10.3 An illustration of the player modeling taxonomy discussed in Sect. 10.3. The taxonomy includes the possible *roles* (top box) and *approaches* (middle box) of player modeling, and the *input and output* (bottom box) aspects a player model may consider

10.3.1 Roles

By now we have situated player modeling as the study of AI methods for modeling human players based on their observations. But what are the potential goals of player modeling? Answers to that question will give rise to corresponding roles player models can take in games. We can argue that the *primary goal* of player modeling is to *understand* how the interaction with a game is experienced by individual (or groups of) players. Thus, while games can be utilized as an arena for eliciting, evaluating, expressing and even synthesizing experience, the main aim of the study of players in games is the understanding of players' cognitive, affective and behavioral patterns. Indeed, by the very nature of games, one cannot dissociate games from player experience.

Given that the core aim of a player model is to uncover phenomena associated with players we can make an important distinction of the roles of such models in games: roles associated primarily to player **behavior** vs. roles associated primarily to player **experience**; see Fig. 10.4. At this point it is important to note that while player behavior and player experience are interwoven notions there is a subtle difference between them. Player behavior points to *what a player does* in a game whereas player experience refers to *how a player feels* during play. The feeling of one's gameplay experience is clearly associated with what one does in the game; player experience, however, is primarily concerned with affective and cognitive aspects of play as opposed to mere reactions of gameplay which refer to player behavior. Independently of the type (behavior or experience) the model can be used during the

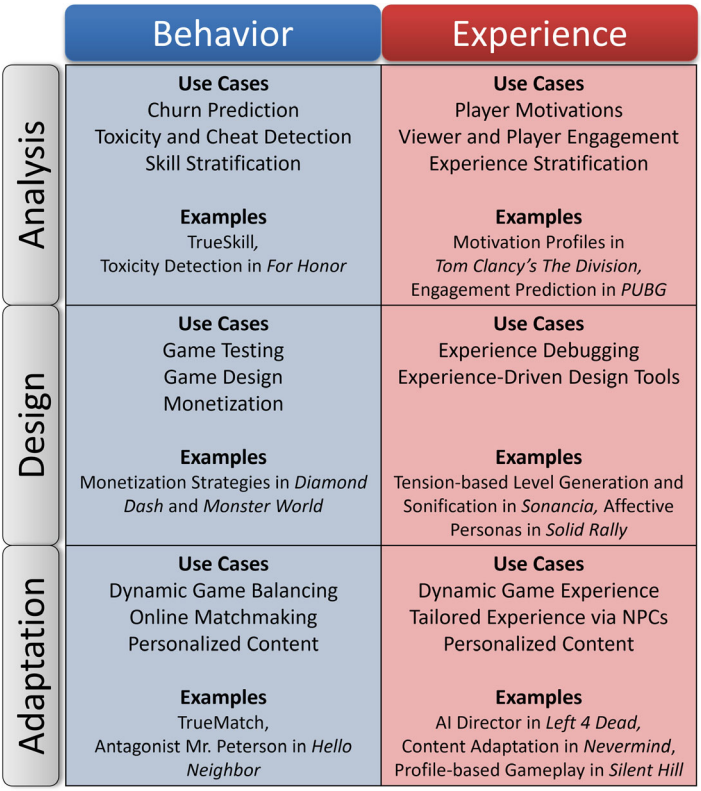


Fig. 10.4 The six key **roles** of player modeling associated with the *goal* of player behavior opposed to player experience modeling, across the tasks of *analysis*, *design* and *adaptation* as identified in our taxonomy. Each role (box) contains a number of popular use cases and indicative examples, some of which are detailed in this part of the book. This figure can be seen as zooming in on the top right corner of Fig. 10.1 which defines what player modeling is

analysis or *design* phases and even during play (*adaptation*). Figure 10.4 illustrates the 6 core roles of player modeling as discussed in this section and lists a number of popular use cases under each role. The taxonomy presented in Fig. 10.3 lists these six roles under three dimensions. The *goal* of a player model is split between that of player behavior and that of player experience, the *designer* dimension either points to the analysis of the model per se or the integration of the player model into the design process (e.g., as a tool) and finally the *player* dimension refers to models that are either used during play (*adaptive*) or not (*static*). A static model can serve any the two designer roles (analysis and design) and hence we don't dedicate any further discussion on this aspect. In the remainder of this section we present the six roles of player modeling across the *goal* dimension of our taxonomy.

Player Behavior Roles

The first reason why one would want to use AI to model players is for understanding something about their **behavior**. This subarea of player modeling is concerned with structuring observed player behavior even when no labels of experience are available—for instance, by identifying player types or predicting player skill via game and player **analytics** [295, 312, 314]. A popular distinction in data derived from games [312, 314] is the one between **player metrics** and **game metrics**. The latter is a superset of the former as it also includes metrics about the game software (system metrics) and the game development process as a whole (process metrics). System metrics and process metrics are important aspects of modern game development that influence decision making with respect to procedures, business models, and marketing. In this book, however, we focus on player metrics. The interested reader may refer to [312, 314] for alternative uses of metrics in games and the application of analytics to game development and research—i.e., **game analytics**.

We can identify three main roles associated with modeling player behavior. First, derived knowledge about the in-game behavior of the player can be used for **behavior analysis** per se. Modeling aspects of player behavior may provide answers to questions such as "when will this player stop playing?" or "how often will that player get stuck in that area of the level?" or "which item type will this player pick in the next room?". Note that if the purpose of modeling is to develop non-player characters with believable, **human-like** behavioral characteristics, and to understand human play per se (through creating generative models of it) then the task becomes one of player *imitation*, which is detailed in Part II. Player modeling in this role is primarily tasked with the *prediction* of unwanted behaviors such as cheat detection [535] or toxicity [178] or, alternatively, behaviors related to churn [871, 872, 973] and skill such as the *TrueSkill* performance ranking system used in competitive games since *Halo 3* (Microsoft Game Studios 2007) [444].

Behavior *prediction* is achieved primarily via supervised learning (including self and semi-supervised learning) or reinforcement learning. In addition to predictive methods borrowed from the supervised learning family of algorithms, unsupervised approaches such as **clustering** [291] and *association mining* [714] are key for the analysis of player behavior. The aim of clustering is the classification of player behaviors within a number of clusters depending of their behavioral attributes. Association mining, instead, is useful in instances where frequent patterns or sequences of actions (or in-game events) are important for determining how a player behaves in a game, or across other players or games.

The second role of player behavior models is that of **behavior design**. Once predictive models of player behavior are in place they can be used to improve **game testing** and game design procedures and customer support, and yield better monetization [312, 314] and retention strategies in casual games such as *Diamond Dash* (Wooga 2011) [973], or ultimately lead to the redesign of (parts of) the game. Finally, models can enable **behavior-adaptive games** as modeled player behaviors can be associated with in-game actionable decisions that would improve the playing experience or the game per se. Popular examples of this role include the adjustment of

NPC behaviors for dynamic game **balancing** [25, 875, 1222] in games like *Hello Neighbor* (tinyBuild 2017) and online **matchmaking** tools such as the *TrueMatch* system [768] that ensure appropriate levels of game balance. In addition to in-game agent behaviors, game content can be offered in a personalized manner to (clusters of) players in runtime with the aim of improved player experience and monetization. Chapter 12 covers a number of specific uses cases associated with the player modeling roles outlined here.

Player Experience Roles

The second reason that drives the use of AI for modeling game players is for understanding something about the players' **experience** during play. Models of player experience are often built using machine learning, typically supervised learning, methods. The training data here consists of some aspect of the game or player-game interaction, and the targets are labels derived from some assessment of player experience, gathered for example from physiological measurements or questionnaires [1286, 1287].

Similarly to player behavior models, we can identify three primary roles when a model is tasked to target player experience (see Fig. 10.4). The first role is that of **experience analysis** with the aim to observe, monitor, and analyze, for instance, the underlying playing motivations [740], the viewers' engagement of streamed games such as *PUBG* (Krafton 2017) [741] or the player's emotional intensity patterns over particular content types [693] among many other use cases. The second role is to test, debug and improve the experience of players, namely the role of **experience design**. The design of a player's experience can occur autonomously as an *experience debugging* tool such as the affective personas of the *Solid Rally* game [60] or in a mixed-initiative fashion through which a designer prompts a level generator with different tension curves as in the sonification process of *Sonancia* levels [661]. The third role for a player experience model is to enable **experience-adaptive games** such as the tension-based placement of items and opponent NPCs in *Left 4 Dead* (Valve 2008) or the adaptive gameplay of *Silent Hill: Shattered Memories* (Konami 2010) driven by a player's profile. Once predictors of player experience are derived they can be taken into account for designing the in-game experience. That can be achieved by adjusting the behavior of non-player characters (see Part II) or by adjusting the game environment via experience-driven PCG [1286, 1289] as in *Nevermind* (Flying Mollusk 2015) and *Super Mario Bros* (Nintendo 1985) [1020, 1038] (see Part III). Chapter 12 covers specific uses cases associated with these three player experience roles.

10.3.2 Approaches

At this point we have defined what player modeling is (and what it is not) and we have identified its key roles for improving both player experience and game production. A high-level classification of the available approaches for player modeling

can be made between **model-based** (or top-down) and **model-free** (or bottom-up) approaches [1288, 1289] (see Fig. 10.3). These definitions are inspired by the analogous classification in RL by which a world model is available (i.e., model-based) or not (i.e., model-free); **hybrid** approaches between them can naturally exist. The gradient red color of the player model box in Fig. 10.2 wishes to illustrate the endless ways one may combine top-down with bottom-up approaches. One for instance can think of a theoretical framework (model-based) of tension that conditions the outputs of a bottom-up surrogate model of tension. Alternatively, one could think of a top-down method that captures player engagement through an ad-hoc formulation that is then cross-validated and refined via human feedback. The remainder of this section presents the key elements of and core differences between the two ends of the continuum.

10.3.2.1 Model-Based (Top-Down)

In a model-based (or top-down) [1288] approach a player model is built on a theoretical framework. As such, researchers follow the *modus operandi* of the humanities and social sciences, which hypothesize models to explain phenomena. Such hypotheses are usually followed by an empirical phase in which it is experimentally determined to what extent the hypothesized models fit observations; however, such a practice is not the norm within player experience research. While user experience has been studied extensively across several disciplines, in this book we identify three main disciplines we can borrow theoretical frameworks from and build models of players. We start from a player's brain (**neuroscience**), we then focus on a player's emotional and behavioral manifestations while playing games (**psychology and affective sciences**) and we finally look at the context of this human activity (**game studies, game design and games user research**). We conclude this section with a short discussion over other disciplines that can influence the design of a player model.

Neuroscience

Try to imagine a futuristic setting for player modeling. Most likely you would intuitively think of games that automatically recognize a player's needs, desires, preferences and emotions directly thought their brain activity; the "Playtest" episode of the *Black Mirror* science fiction TV series builds on this very scenario. In such a scenario a top-down player model would be able to associate gameplaying behavior and experience to certain patterns of brain activity.

As futuristic as it may sound, however, a number of studies have already relied on the working hypothesis of an underlying mapping between the brain, its neural activity and player experience. While this relationship is not as well explored and the presumptive mapping seems to be largely unknown, some interesting findings are already in place. For example, in-game interest has been associated with activity in the visual cortex and the release of endomorphin whereas the sense of achievement has been linked to dopamine levels [69]. According to [69], neuroscientific

evidence suggests that the reward systems of games are directly associated with the dopamine-based reward structures in the brain and that dopamine is released during gameplay [576]. Further, pleasure has been associated with areas in the brain responsible for decision making, thereby revealing the direct links between gameplay experience and decision making [960]. Pleasure has also been associated with uncertain outcomes or uncertain rewards [1034] as well as with interest and curiosity [79], which are all key elements of successful game design. Stress is also tightly coupled with player experience given its clear association with anxiety and fear. Stress can be both monitored via **physiology** and regulated via game design. The testosterone levels of players have also been measured in association to digital game activities [725] and findings reveal particular patterns of competition in games as testosterone factors. Finally, it appears that trust between players in a social gaming setup could be measured indirectly via oxytocin levels [580].

The degree to which current findings from neuroscience are applicable to player experience research is largely unknown since access to neural activity and brain hormone levels remains a rather intrusive process at the time of writing. A large body of literature in cognitive neuroscience, however, has investigated the relationship between video game playing and associated brain activity. A key unsurprising finding is that the various video game genres have dissimilar impact on cognitive processing. The vast majority of such research has so far put an emphasis on the different perceptual, attentional, and cognitive effects shooter games have in contrast to other game genres [260]. Recent findings suggest that the relationship between personality, hormones, and game content is far from trivial or clear. Playing a violent video game, for instance, can even have a stress-reducing effect depending on personality traits and one's psychological needs [1228].

Practically speaking, manifestations of brain activity such as the brain's electrical waves—measured through electroencephalography (EEG) on the scalp—or more indirect manifestations such as stress and anxiety—measured through skin conductance—can give us access to approximates of brain activity. While such data can be hard to get and could disturb the gameplaying experience, it can be used for modeling the experience of play and even control games directly via brain-computer interfaces (BCI) [474] as discussed later in this chapter.

Psychology and Affective Sciences

Let us now zoom out from a player's brain and instead observe the ways brain activity is manifested psychologically and physiologically. Beyond hormones, brain activity and cognitive functioning, a top-down player modeling approach may instead rely on existing theoretical models of human behavior and experience. Among the several disciplines that can serve such models in this section we outline selected frameworks from psychology and the study of emotion that can be deployed directly in games for the top-down analysis of player experience. In particular we focus on fundamental theoretical contributions from the psychology of entertainment and the study of affective interaction in games.

Dominant theories of emotion [607] such as the **cognitive appraisal theory** [356, 1001] can be key for the analysis of player experience. Cognitive appraisal assumes

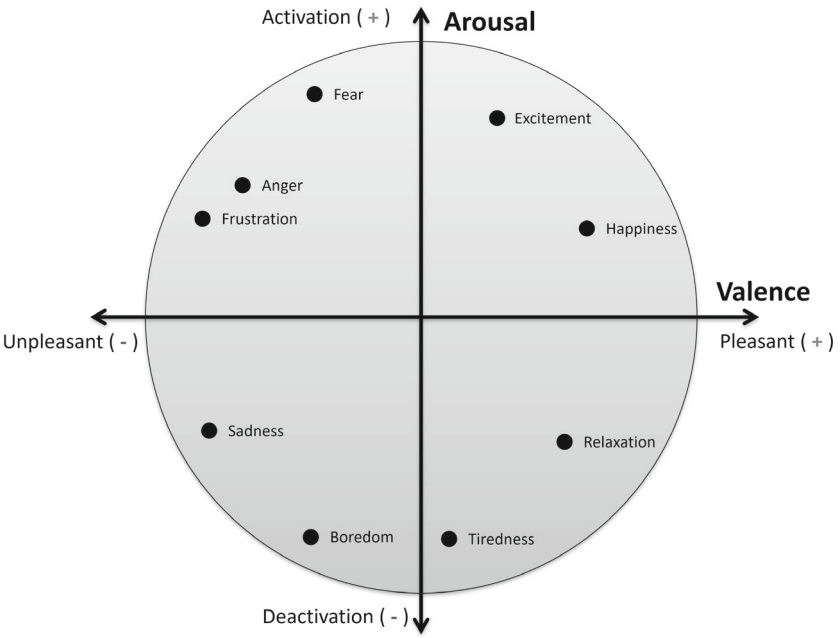
that emotions are the outcomes of a cognitive process. For instance, one could design an ad-hoc computational model that maps the current observation of an NPC (e.g., shooting happening nearby) with a set of actions (e.g., reload and run away) and corresponding elicited emotions (e.g., the NPC's level of fear is heightened). The GAMYGDALA game engine [897] is a representative example of such a top-down player emotion model that is built on the cognitive appraisal model by Ortony, Clore, and Collins (OCC) [838].

The player model may instead rely on well-established representations of emotion as studied within affective sciences. A popular such representation is Ekman's mapping of particular universal facial expressions of emotions to a number of culturally-independent emotional states. Early works identify six such emotional states that can be expressed universally via facial expressions [309]: *anger*, *disgust*, *fear*, *surprise*, *happiness* and *sadness*; more recent studies suggest that *contempt* is also such a universally expressed emotional state [310]. Using Ekman's model one can theoretically infer the emotional state of a player based on their facial expression, if the latter is recorded through a camera. In practice, as we will see later on in this book, there are many challenges associated with this type of non-verbal emotion detection.

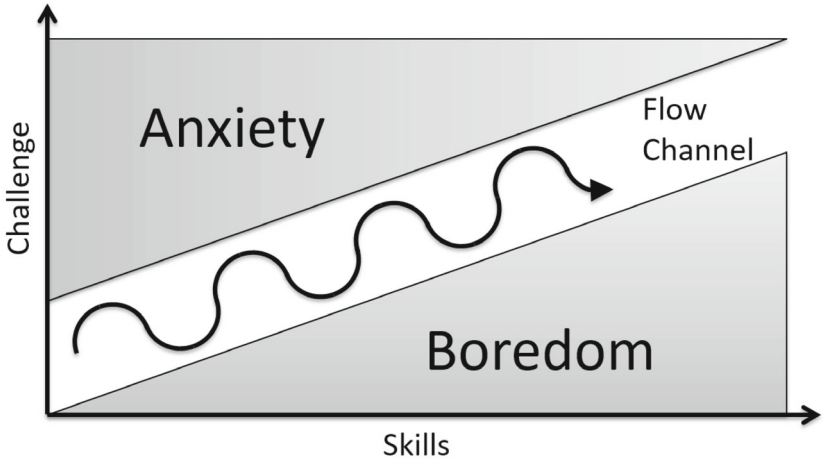
Ekman's model views emotions as a predetermined set of discrete classes. While convenient for particular use cases, such representations might not be as fine-grained for modeling players. Alternatively, affect models that represent emotion through continuous dimensions may be used. The most popular of such representations is the circumplex core affect model of Russell [974] (see Fig. 10.5a). The two dimensions of this circumplex model, namely **arousal** and **valence** [334], can theoretically represent any player experience manifestation in games: valence refers to how pleasurable (positive) or unpleasurable (negative) the emotion is, whereas arousal refers to how intense (active) or lethargic (inactive) that emotion is. A more elaborated version of the circumplex model of affect developed by Mehrabian and Russell [738] includes the dimension of **dominance**. The added dimension of the *Pleasure (Valence), Arousal, Dominance* (PAD) model represents the degree of control one feels over their emotion. For example, while both fear and anger are unpleasant emotions of high arousal—and close to each other on the arousal-valence model; see Fig. 10.5a—fear is a submissive emotion whereas anger is a dominant one.

If one opts to build a top-down player model—independently of the theoretical model considered—they are required to map emotional manifestations of players directly to specific player states. For instance, one may view player experience as a psychophysiological phenomenon [1284] and link a player's increased heart rate to high arousal and, in turn, to high levels of in-game excitement or frustration. Indicatively, in-game arousal has been correlated to physiological activity such as as heart rate and skin conductance in *Left 4 Dead* (Valve 2008) [22] and other first-person shooters [293]. Therefore, one could, in principle, use the correlating aspects of such physiological signals as proxies of player experience in a top-down fashion.

Beyond established theories of emotion, model-based approaches can also be inspired by a general cognitive-behavioral theoretical framework such as the *theory of mind* [898] for modeling aspects of social interactions in games, as in [746]. Popular indicative frameworks for deriving user models in games include the **usability**



(a) Russell's two-dimensional circumplex model of affect. The figure contains a small number of representative affective states (black circles).



(b) An illustration of the *flow channel*.

Fig. 10.5 Two popular frameworks used for modeling users and their experience in games: **a** the arousal-valence circumplex model of affect and **b** the flow channel concept

theory [495, 811], the **belief-desire-intention model** [119, 375], the OCC cognitive model [838] and Skinner's behavioristic approach [1051] with its links to reward systems in games. We can also draw inspiration from social sciences and linguistics to map player experience to lexical aspects of gameplay interaction (e.g., chatting). Natural language processing, opinion mining and **sentiment analysis** normally rely on affective and sociological aspects of textual communication [840, 849]. As a complementary approach to top-down models, language models such as GPT variants can be utilized for their abilities to act as computational models of affect [1125].

Of particular importance to player modeling is the concept of **flow** by Csikszentmihalyi [249–251]. Flow has been a popular psychological construct for modeling player experience in a top-down fashion [1115]. When in a state of flow (or else, state of "happiness") during an activity we tend to concentrate on the present moment, we lose our ability of reflective self-consciousness, we feel a sense of personal control over the situation or activity, our perception of time is altered, and we experience the activity as intrinsically rewarding. Analogously the optimal experience during play has been associated with a fine balance between boredom and anxiety, also known as the **flow channel** (see Fig. 10.5b). Given its direct relevance to player experience, flow can been adapted and incorporated for use in game design and for the understanding of player experience [790, 1112, 1115].

The underlying motivations of game playing can also provide fundamental theoretical frameworks for the construction of top-down player models. The well-established **self-determination theory** (SDT) is the most representative positive psychology theory for the analysis of human motivation [271]. The theory has been adopted across a diverse number of domains that study human behavior including video games [901, 942]. SDT considers two types of factors that may influence motivation: intrinsic and extrinsic factors. The former type of factors rely on intrinsic properties of the activity such the drive to master the game, whereas the latter type of factors are instantiated primarily through external reward schemes and expectations such as the in-game score and unlocked achievements. All such factors are well represented in games since games can both offer rewards and objectives that promote extrinsic motivation directly and also facilitate intrinsic motivation to players [901]. Specifically, the three basic psychological needs underlying intrinsic motivation in games have been identified as *competence*, *autonomy* and *relatedness* [976]. Competence refers to the sense of accomplishment for the mastery of an action and it is associated with self-efficacy and progression. Within games, competence is manifested via the interactions players must learn to master in order to complete the game. Autonomy refers to our sense of control and it is manifested in games through player tactics and strategic decisions. Finally, relatedness refers to the sense of belonging and interacting with others. Relatedness manifests itself through our interactions with other player or non-player characters. The theory of self-determination has formed the basis of several studies in player experience research [139, 740] and it has influenced the design of player experience questionnaires such as the *Ubisoft Perceived Experience Questionnaire* (UPEQ) [43] and the *Player Experience of Need Satisfaction* (PENS) questionnaire [976].

Another very popular psychological construct that may form the basis of a top-down player model is that of **presence**. Presence can be defined as the feeling of a mediated experience and has been linked to a number of in-game factors such as the game narrative [942]. However, the feeling of presence, or associated constructs such as immersion, are not directly mapped to a set of emotions or particular in-game contexts and are not considered core psychological needs. Nevertheless, our in-game presence (or immersion) can arguably explain the driving force of our playing motivations and it may help us analyze our gameplaying experience [162, 660]. A top-down modeling approach that relies on a theoretical framework of presence would, for instance, directly map aspects of a player's spatial navigation behavior to levels of player immersion. Tools that attempt to measure presence through self-reporting and could be used as alternative bottom-up approaches include the UPEQ [43] and PENS [976] questionnaires.

Game Studies, Game Design and Games User Research

Instead of observing the human brain, or even the physiological or bodily reactions of the player during play, one could turn to the analysis of the interaction context (play) and the domain (games). The analysis and study of player experience has provided a growing number of frameworks one could consult for a top-down player model. Such theoretical models of user experience in games are often met in game studies, game design and game user research studies.

An indicative model that has been used extensively in player modeling work is Malone's three core design dimensions—defined as *challenge*, *curiosity* and *fantasy*—that collectively contribute to the design of 'fun' games [698]. Challenge refers to the uncertainty of achieving a goal due to factors such as a variable difficulty level, multiple level goals, hidden information, and randomness. Curiosity refers to the player's feeling of uncertainty with respect to what will happen next. Finally, fantasy is the ability of the game to show (or evoke) situations or contexts that are not actually present. These three dimensions have been quantified, operationalized and successfully evaluated in prey-predator games [1271], physical games [1274, 1280], preschooler games [537] and racing games [1156].

Bartle's [65, 66] classification of player types within games can be used indirectly for modeling players. Bartle's original work analyzes players of multiplayer online games such as MMORPGs and multi-user dungeons (MUDs) and the derived taxonomy was not intended to be used outside these game genres. The idea, however, that players can be classified according to their in-game behavior has inspired the design of player taxonomies and archetypes across several other game genres [1178]. Bartle identifies four **archetypes** of players he names *killers* (i.e., players who focus on winning and are engaged by ranks and leaderboards), *achievers* (i.e., players who focus on achieving goals quickly and are engaged by achievements), *socializers* (i.e., players who focus on social aspects of games such as developing a network of friends) and *explorers* (i.e., players who focus on the exploration of the unknown). Various other methodologies have also been followed to derive player experience archetypes for particular classes of games [68, 1178, 1295].

Another popular game design view over player experience is Koster's **theory of fun** [581]. Koster relates the concept of *fun* with learning in games: the more you learn the more you tend to play a game. According to his theory you stop playing a game that is way too easy (no learning of new skills) or way too hard (no learning either). Notions of Koster's fun—quantified as moderate levels of difficulty or divergence—have been used as fitness or reward functions for the procedural generation of personalized racing tracks [1156] and platformer levels [1038, 1239]. An alternative to Koster's perspective on *fun* is the four "fun" factor model of Lazzaro [608]. The four fun factors are named *hard fun* (e.g., playing to win and seeing how good I am at it), *easy fun* (e.g., playing to explore new worlds and game spaces), *serious fun* (e.g., playing to feel better about myself or get better at something that matters to me) and *people fun* (e.g., playing as an excuse to invite friends over, or having fun merely by watching them play). Such factors could be considered, for instance, for the design of stratified player models that satisfy the *fun* needs of particular players.

Other popular and interconnected views of player experience from a **game design** and **game studies** perspective include the notion of the 'magic circle' in games [981] and the theoretical model of involvement [162]. The latter is a notable multifaceted model for capturing player immersion, defined as the player's sensation of inhabiting the virtual environment presented. The model is composed of six types of player involvement: affective, kinesthetic, spatial, shared, ludic, and narrative. Some limited attempts have been made to study and validate the player experience of immersion in an experimental setting [447]. More research will thus be required to successfully quantify each one of the six facets of player immersion to metrics that can reliably capture the corresponding player immersion aspects.

With a careful analysis of the models we covered in this section and their subcomponents one could coherently argue that there is one underlying theoretical model of player experience after all. While it is not the intention of this book to thoroughly discuss the interconnections between the aforementioned models, it is worth pointing out a number of indicative examples of our envisaged overarching player experience model. An explorer (Bartle), for instance, can be associated with the easy fun factor of Lazzaro and the curiosity dimension of Malone. Further, the achiever archetype (Bartle) can be linked to the serious fun factor (Lazzaro). Accordingly, a killer archetype (Bartle) maps to the hard fun factor (Lazzaro), the challenge dimension of Malone's model, and a number of flow aspects. Finally, a socializer player profile (Bartle) could be associated with people fun (Lazzaro) and, in turn, to the shared involvement facet of Calleja [162].

Even though the literature on theoretical models of experience is rather rich, one needs to be cautious with the application of such theories to games (and game players) as the majority of the models have not been derived from or tested on interactive media such as games. Calleja [162], for instance, reflects on the inappropriateness of the concepts of 'fun' and 'magic circle' (among others) for games. At this point it is worth noting that while ad-hoc designed models can be an extremely powerful and expressive they need to be cross-validated empirically to be of practical use for

computational player modeling; however, such practices are not as common within the broader area of game studies and game design.

Research on the intersections of *user experience* (UX) and *player experience* (PX)—namely *games user research* (GUR)—can also shed light on the relationship between user states of games and games themselves [292]. Normally these relationships are in form of correlations between player reports about their experience (e.g., via dedicated questionnaires) and game (or player) observational data such as psychological or behavioral analytics [293]. The general field of GUR also focuses on the associations across player states as reported by players. Through such a study, the player state *enjoyment* has been identified as a core, yet highly diluted, component of player experience that cannot be quantified as easily [739]. Further findings suggest that player enjoyment appears to be distinct from flow as it may be elicited independently of challenge and cognitive processing. As the outcome of most GUR studies is predominately a computational relationship between aspects of player experience and its manifestation (e.g., player behavior, bodily responses) one could devise player models on such relationships. Think for instance of an ad-hoc player model that outputs a player's fear as a linear mapping of particular heart rate variability features.

Other Disciplines

The list of areas that can influence the top-down design of player models can be extended to any discipline that studies human behavior and experience. Here we briefly touch upon the discipline of **behavioral economics** and the science of decision making.

What is foundational to human behavior is decision making and what is critical to decision making is *judgment*. Behavioral economics have long attempted to address the question of how we judge and value different options and then decide to take one (or more). A fundamental aspect of behavioral economics is the principle that humans act as *irrational agents* when they take decisions. According to Kahneman et al. [534] there are a number of reasons why we humans act irrationally. These reasons were named *heuristics* and the underlying theory was named *judgment heuristics theory*. Importantly the judgment heuristics theory was inspired by and built on the bounded rationality theory of Simon [1045], a fundamental theoretical framework for AI. Decision making heuristics as such can be used not only for the understanding of play in games but also for the ad-hoc design of player behavioral models.

10.3.2.2 Model-Free (Bottom-Up)

In this section we will outline player modeling approaches that are theory-agnostic and primarily rely on data. Such **model-free** approaches refer to the data-driven construction of an unknown mapping (model) between a player **input** (or observation) and a **player state** or a set of such states. Any manifestation of player affect or behavioral pattern could define the input of the model (see more in Sect. 11.1). A

player state, on the other hand, is any representation of the player's behavior or experience. This is essentially the output of the computational model which is covered thoroughly Chap. 12.

Model-free approaches follow the *modus operandi* of the exact sciences, in which observations are collected and analyzed to generate models without a strong initial assumption on what the model looks like or even what it captures. Player data and potentially labels of player states are collected and used to derive the model. Any **machine learning** technique—see Chap. 3—or statistical approach can be used for the construction of the mapping between the input and the output of the model. While supervised learning (and its variants) is the dominant player modeling paradigm, reinforcement learning is applicable when a reward function can characterize aspects of playing behavior or experience which can train an RL model. Unsupervised learning is applicable when target outputs (labels) are not available but, alternatively, data is used for the analysis of playing behavior. Examples include studies in which player actions, goals, intentions, and affective states are modeled and predicted for the purpose of believable gameplay, adaptive gameplay, interactive storytelling or even the improvement of a game's monetization strategy [60, 686, 837, 991, 1141, 1321]. We refer the reader to Chap. 11 for more details regarding the various AI algorithms that are applicable depending on the availability of human demonstrations and labels of player states.

The first player modeling studies that rely solely on player data can be traced back to the early years of the AI and games field. A selected list of such studies includes player modeling attempts in first-person shooters [1143, 1144], racing games [1156] and variants of *Pac-Man* (Namco 1980) [1281]. More recently the growing deployment of game telemetry tools started offering researches large sets of game and player data. This data availability has opened up the horizons of behavioral data mining in games—i.e., **game data mining**—and gradually introduced the subfield of **game data science** [295, 314]. Studies that attempt to identify different behavioral, playing and action patterns within a game are well summarized in [312, 314] and include [70, 291, 740, 1132, 1135, 1245], among many others. The availability of large data sets and sophisticated deep learning methods of recent years has also influenced the study of player modeling. Indicatively, models from the CNN family have been employed on game frames and game audio to infer aspects of player experience [695, 888], adversarial deep learning has been used to predict game outcomes for automatically balancing games [931], and supervised learning models have managed to predict a player's personality using in-game and player data [421]. Recently we have also witnessed early attempts of multi-player experience prediction in games such as *Hearthstone* (Blizzard Entertainment 2014) [723]. For thorough reviews of recent player modeling studies with an emphasis on player behavior and player experience, refer to [314, 1286], respectively.

10.3.2.3 Hybrids

The space between model-based and a model-free approaches can be viewed as a continuum along which any player modeling approach might be placed. While a completely model-based approach relies solely on a theoretical framework that maps a player's responses to game stimuli, a completely model-free approach assumes there is an unknown function between modalities of user input and player states that a machine learner (or a statistical model) may discover, but does not assume anything about the structure of this function. Relative to these extremes, the vast majority of studies in player modeling may be viewed as **hybrids** that combine elements of the two approaches. The continuum between top-down and bottom-up player modeling approaches is illustrated with a gradient color in Fig. 10.2.

10.3.3 Input and Output

The final part of our taxonomy contains a critical aspect for both the input (observation of the model and the output (see Fig. 10.3). We outline these two aspects here.

Observation: Dynamic Versus Static

Observations of human players can be split across several dimensions and types but one core dimension is whether what we observe is of **static** nature or **dynamic** nature. The former refers to any data of the player and their game that vary temporally such as their physiological responses, their in-game responses, the current game state, and their skill. On the other hand, the latter refers to any data that contains static information of the player that normally includes aspects of their profile including game preferences and preferred modes of play. One might argue that several aspects of a player profile can be potentially dynamic (e.g., age and style) but we argue back that normally those aspects change at a much slower rate compared to the dynamicity of other player observations. We detail the core differences between static and dynamic observations that may serve as a model's input in Sect. 11.1.

Labeling: Objective Versus Subjective

The final distinction we make in our player modeling taxonomy is the one between **objective** and **subjective** labels. The former refers to labeled data that is objectively defined (to the degree possible) and usually represents aspects of play such as level completion times, items purchased and used, or player performance in a deathmatch FPS scenario. The latter, instead, refers to labels of subjectively defined concepts such as reported experience, preferences and opinions about own play, or opinions about other players. We discuss the difference of these two ends of the spectrum in depth in Chap. 11. Chapter 2 also dedicates a relevant section on human feedback.

10.4 Further Reading

For an extensive reading on game and player analytics (including visualization, data preprocessing, data modeling and game domain-dependent tasks) and dame data science we refer the reader to the edited books by El-Nasr et al. [312, 314]. The edited volume of Drachen et al. [292] is recommended as a good starting point if the reader has an interest in core concepts and methodologies of *games user research*. When it comes to further reading on player modeling, two early papers in this field offer complementary perspectives and taxonomies and a thorough discussion on what aspects of players can be modeled and the ways players can be modeled: the survey papers of Smith et al. [1054] and Yannakakis et al. [1288].

10.5 Summary

This chapter first defined player modeling, and then offered a taxonomy of roles, approaches and considerations regarding the input and the output of the model. The core reasons why AI should be used for modeling players is either to derive something about the players' experience (how they feel in a game) or for us to understand something about their behavior (what they do in a game). Once such computational models are constructed they can be used as predictive models per se or in conjunction with gameplaying agents and content generation methods.

Player behavior and player experience can be modeled in a top-down or a bottom-up approach (or a mix of the two). Top-down (or model-based) approaches have the advantage of solid theoretical frameworks usually derived from other disciplines or other domains than games. Bottom-up (or model-free) instead rely on data from players and have the advantage of not assuming anything about players. Such methods only assume that player experience and behavior are associated with demonstrations left by the player and that these demonstrations are representative of the phenomenon we wish to explain.

The next chapter of this part puts an emphasis on the specific methods one can use to build player models depending on the amount and type of data available and the role the player model is tasked to perform. Such data-driven models sense player observations (input) and are asked to predict certain labels (outputs). We thus first cover the various options with regards to the sensed input and the output of the model, that both determine the player modeling method to be used.

Chapter 11
From Observations to Models of Players

Abstract This chapter discusses the nuts and bolts of player modeling from the three perspectives of input, output, and algorithm. Input is subdivided into gameplay input, objective input (including physiological signals), game context, and player profile. The model's output, or that which is predicted, can be of various types. Some player models simply learn embeddings that can be used to segment players, other models predict high-level player behavior or various aspects of player experience. The chapter also contains a discussion of principles and tools for labeling data for player modeling. A multitude of machine learning methods can be used to model players, and this chapter discusses supervised learning, reinforcement learning, and unsupervised learning, as well as manual model authoring.

Chapter 10 went through the basic definitions of player modeling, the possible roles a player model can take and a high level taxonomy of approaches for player modeling. In this chapter we go through the entire pipeline (methodology) of obtaining a player model, from observations of players to their models, building upon the *data-driven* approach. We first cover the options one has when selecting what the player model will *sense* (input) from the player and then cover the options one has in terms of possible model outputs and the tools to obtain them. In particular, Sect. 11.1 details the types of data a player model can consider for its **input** including gameplay data, objective data such as a player's bodily and facial expression reactions, contextual information about the game, and profile information about the player. Section 11.2 instead, takes us through the different types of **output** a player model might represent (behavior, experience, embedding) which are associated with the role of the player model and the availability of labeled data. Given the challenges of obtaining reliable and useful labels for modeling players, we dedicate Sect. 11.3 to a discussion around good practices and appropriate tools for manual data labeling.

The player observations and output data collectively determine the AI **method** one should use, including the appropriate utility and representation of the method, for building the player model. To complete the entire player modeling pipeline—from inputs and outputs to models—in Sect. 11.4 we discuss the application of learning paradigms which we deem appropriate for player modeling. Specifically we cover supervised learning variants, reinforcement learning, and unsupervised learning methods to model players. For completeness we also briefly discuss authoring

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_11

(model-based) approaches one may use when not considering a machine learning route. All aforementioned AI paradigms and corresponding methods are detailed in Chaps. 2 and 3, respectively.

11.1 What Is the Model's Input Like?

What could a player model observe, sense or consider? One could argue that any information about the decision making patterns and the emotional manifestations of a player during the game could be relevant. Such information can be observed through variations of gameplay patterns, or sensed via alterations in a player's attention or level of focus. Any changes in the player's physiology, facial expression, posture, and speech could also be sensed and considered by a player model. Monitoring all these alterations of multimodal player data would seem critical information for constructing the player's model. Any player model would ultimately need to rely on such data as it represents aspects of the overall gameplaying experience. Data of this type—that may feed the player model—can be obtained directly through the game engine or with the help of sensors. These sensors are either available directly through the various gameplaying platforms or they are integrated into the game indirectly. An example for a direct integration is a gaze tracker that you use to play a game; indirect integration instead, would be a skin conductance sensor of a smartwatch that sends data to the game you are playing on your smartphone.

But there is more to it. Players are not playing in a silo; they are affected by the game itself and other players, so it feels like the current game state is also important for a player model. Beyond data that we can monitor during play there is data that does not alter that much over time (or even if it does, the change is slow) that could inform a player model. Our personalities, ages, and game preferences, for instance, could affect the way we play and feel while we are playing.

We build upon earlier studies on player modeling [1281, 1286] and we argue that the model's input can be of three types of *dynamic* (primarily temporal) observations: (1) anything that a player is doing in a game environment gathered from **gameplay** data such as behavioral data of any type, user interface selections, preferences, or in-game actions; (2) **objective** data collected as responses to game stimuli such as physiology, speech and body movements; and (3) the **game context** which comprises any player-agent interactions but also any type of game content viewed, played, and/or created such as the audiovisual footage of a game. In addition to the dynamic input types a model may also consider *static* **player profile** information such as their personality and their demographics. The four categories are detailed in the remainder of this section, but they are also illustrated as green boxes in Fig. 11.1 and are summarized in Table 11.1.

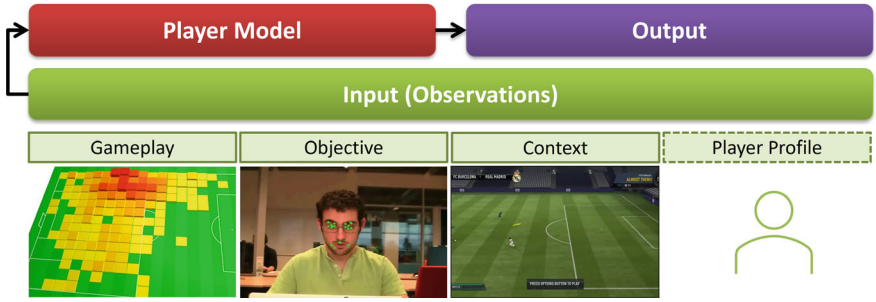


Fig. 11.1 The various options for the **input** (observations) of a player model as discussed in Sect. 11.1. The observations of the model may include gameplay or objective data, the game context, and information about the player's profile. The dotted box indicates that the player profile is the only input type containing static information

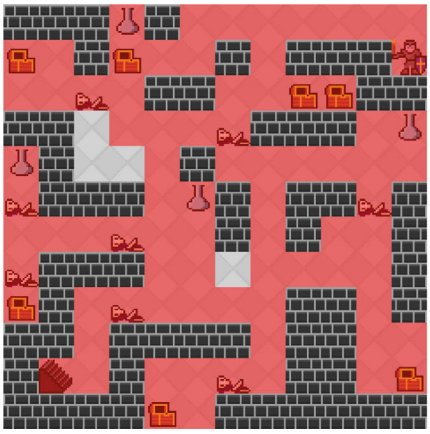
Table 11.1 List of possible player model inputs (observations)

Input	Categories	Options
Gameplay		Ad-hoc gameplay features, preferences, in-game actions
Objective	Physiology	EDA, ECG, EEG, EMG
	Body	Facial expression, head pose, gestures, body movement, eye gaze
	Language	Speech, text
Context	AI Agent	Navigation, expression, exploration, verbal/non-verbal interaction
	Content	Visuals, audio, narrative, game rules, level
	Game profile	Genre (shooter, platformer, racing, strategy, adventure, etc.)
		Platform (mobile, AR, VR, console, desktop)
		No. of players (single, one-and-a-half, two-player, multi-player)
		Observability (fully observable vs. partially observable game)
		Stochasticity (deterministic vs. non-deterministic game)
		Time (real-time vs. continuous)
		Action Space (2 to many player actions)
Player profile	Individual	Personality, demographics, game preferences, game expertise
	Social	Linked data, social media, games played, platform player profile

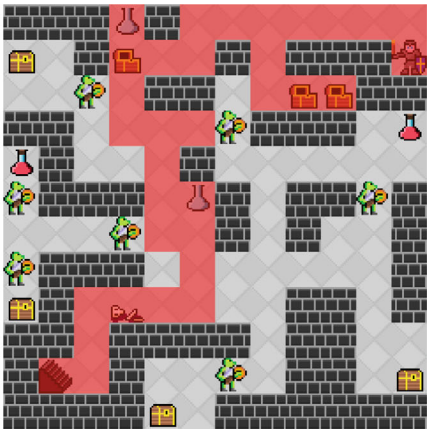
11.1.1 Gameplay Input

Given that games may affect the player's cognitive processing patterns and cognitive focus we assume that a player's actions and preferences are linked directly to their overall experience. Consequently, one may infer the player's current experience state by analyzing patterns of their interaction with the game [228, 401]. It should thus seem obvious that gameplay data is relevant to a player model, to say the least.

Gameplay input refers to anything a player does in a game environment which is collected via in-game logs of any type such as user interface selections, preferences, or in-game actions. In particular, gameplay includes any aspect that can be derived from the interaction between the player and the game directly. These observable and interpretable aspects of gameplay—also broadly known as **player metrics** [312, 314]—include detailed attributes of the player's behavior derived from responses to game elements such as objects, NPCs, game levels, or user menus. Popular examples of attributes that are directly linked with gameplay include spatial locations of players and key events viewed as **heatmaps** [294], trajectories or aggregated descriptive data [314, 744] and communication with other players or an audience [741]. Some indicative examples of heatmaps in the *MiniDungeons*[1] [459, 460] puzzle game are illustrated in Fig. 11.2. Both general measures (such as performance and time spent



(a) In this example the player acts as a completionist: killing all monsters, drinking all potions and collecting all treasure.



(b) In this example the player acts as a speedrunner: reaching the exit, avoiding any monsters and only collecting potions and treasures that are near the path to the exit.

Fig. 11.2 Two example heatmaps (human playtraces) in the *MiniDungeons* game. *MiniDungeons* is a simple turn-based rogue-like puzzle game, implemented as a benchmark problem for modeling the decision-making styles of human players [459]. Images used with permission

[1] https://github.com/sentientdesigns/minidungeons.

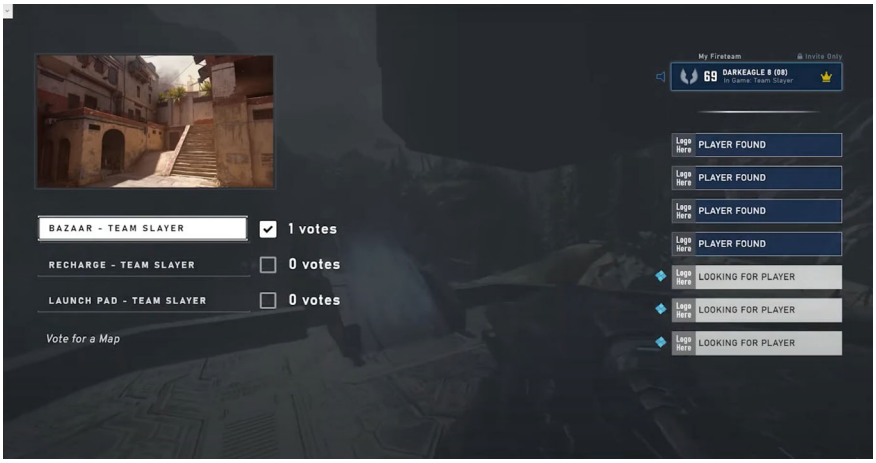


Fig. 11.3 A screenshot from the matchmaking process of *Halo Infinite* (Microsoft Game Studios 2021). The game features *TrueSkill II* for player skill assessment and *TrueMatch* for matchmaking. Screenshot taken by the authors.

on a task) and game-specific measures (such as the weapons selected in a shooter game [432, 1192]) can define relevant and appropriate player metrics.

One of the most popular uses of gameplay-based player models is for game (and player) balancing. Two of the most widely used **game balancing** systems are Microsoft's *TrueSkill* and *TrueMatch*. The former is a performance ranking system used in competitive games since *Halo 3* (Microsoft Game Studios 2007) [444]. *TrueSkill* is based on the underlying logic of the Elo rating method for measuring the relative skill levels of players in zero-sum games such as Chess but repurposed for multi-player video games. The *TrueSkill* rating of the player depends on a predetermined set of performance measures for a particular game such as the kill-death ratio in first-person shooters. The *TrueMatch* system [768] matches team players and adversarial players together in online multiplayer game settings to ensure appropriate levels of game balance. This RL-based matchmaking system may operate with any skill-rating algorithm even though it has been predominantly used in conjunction with *TrueSkill II* [769] in games like *Gears 5* (Xbox Game Studios 2019) and *Halo Infinite* (Xbox Game Studios 2021); see Fig. 11.3.

Gameplay Input: Considerations

A key limitation of the gameplay input is that the player experience is only observed *indirectly*. Gameplay metrics primarily map to behavioral patterns of play that are not necessarily aligned with a particular playing experience. For example, a player that shows limited interaction through their logged gameplay data could be either planning their next quest, talking to their friend over the phone or even feeling bored with the game. However, gameplay metrics can still reveal, in part, certain player

experiences such as the presence of boredom, frustration or even toxic behavior. We focus on a number of such use cases in Chap. 12.

It is also important to note that player models derived from gameplay data do not necessarily generalize across all players and games. Certain statistics that hold for player populations, may provide little information for individual players. Therefore, it is crucial that such models are fine-tuned to the needs of players, and manifest gameplay experiences of player personas (player types) [176] and ultimately individual players. One might even argue that gameplay data is not even relevant for particular games or players as the ad-hoc design of the gameplay attributes might not be useful for capturing certain aspects of player experience. One cannot know a priori whether the completion time of a level or the in-game score are relevant to predict player motivation [740]. Furthermore when one attempts to use pure player metrics to estimate a player's experience and make the game respond in an appropriate manner to these perceived experiences, it is advisable to keep track of the player's feedback to the game responses, and adapt the game when the feedback indicates that the player experience was gauged incorrectly.

11.1.2 Objective Input

Video game players are presented with a wide palette of experience stimuli during their game. Such stimuli vary from simple auditory and visual events (such as sound effects and textures) to complex narrative structures, cinematographic views of the game world and emotively expressive game agents. Any player response to these stimuli may, in turn, cause changes in the player's physiology, reflect on the player's facial expression, posture and speech, and alter the player's attention and focus level. Observing these changes during play and monitoring them via appropriate sensors would seem to be beneficial for modeling players. We refer to this type of signals caused as physiological or bodily responses to in-game stimuli as **objective** inputs to a player model.

Player models built on objective inputs attempt to *sense* and incorporate multiple modalities of player activity beyond gameplay. In Table 11.1 we identify three main objective ways of sensing player experience—physiology, body and language—which we detail in the remainder of this section.

The first category of objective input, **physiology**, refers to physiological signals such as electrodermal activity (EDA), electrocardiogram (ECG), electromyogram (EMG), and electroencephalogram (EEG) [298, 855]. It is widely evidenced that arousing or tense events cause dynamic changes in both sympathetic (increase) and parasympathetic (decrease) nervous systems whereas low arousal states (e.g., relaxing or resting) increase the activity on the parasympathetic nervous system. As mentioned earlier, such activity may cause observable alterations in one's facial expression, head pose, electrodermal activity, heart rate variability, blood pressure, pupil dilation [159, 1031] and so on. Recent years have seen a significant volume of studies that explore the interplay between physiology and gameplay by investi-

gating the impact of different gameplay stimuli on dissimilar physiological signals ([293, 699, 700, 736, 790, 924, 1145, 1177] among others). Such signals are usually obtained through ECG [1285], photoplethysmography [1177, 1285], galvanic skin response (GSR) [462–464, 700], respiration [1177], electroencephalography [21, 440, 815, 928], and electromyography (see Table 11.1).

The second category of objective input, namely **body**, refers to signals that are expressed as bodily reactions including facial expression, head pose, gestures, eye gaze, haptics, and body movement [407, 1101]. One may use sensing devices to track the player's *bodily expressions* at different levels of detail and infer the real-time emotional responses from the gameplay stimuli. The core assumption of such objective input modalities is that particular bodily expressions are linked to expressed emotions and cognitive processes. Objective input modalities, beyond physiology, that have been explored extensively include facial expressions [30, 154, 398, 538, 1308], muscle activation (typically face) [229, 274], body movement and posture [36, 89, 286, 538, 1201], haptics [835], gestures [478], and eye movement [36, 784]. Signals that rely on **language** communication including speech and text define the third and last category of objective input. Language communication may occur among game players or between games and players. Verbal cues may have different degrees of influence on gameplay. Text and speech responses can be completely *spontaneous* when, for instance, a player manifests fear verbally due to a jump scare in a horror game. A recent study analyzed verbal narration of YouTube streamers while playing the horror game *Outlast* (Red Barrels 2013) [1262] and, unsurprisingly, revealed the high impact of jump scares on player experience in the game. Text and speech, instead, can influence gameplay indirectly as used by players of multi-player FPS or MOBA games for team coordination purposes; still the use of language in these cases is *optional*. The role of language can be even more direct in text-based games such as *Façade* (Procedural Arts 2005) and *AI Dungeon* (Latitude 2019) where text is *necessary* for playing the game. If available as a modality of interaction in a game, speech [54, 517, 519, 536, 1221, 1262], and text [237, 657, 849] may offer reliable detectors of sentiment and playing experience in games.

Model-free approaches are common when objective modalities of players are considered for modeling players. Experiment participants are normally asked to annotate facial (or head pose) images of players or verbal interactions with particular experience states (see [888, 1018, 1027] among others) in a crowdsourcing fashion. The approach is also common in studies of psychophysiology in games [1177, 1284, 1285].

At the moment of writing there are few examples of commercial games that utilize objective input from players. One particularly interesting example is *Nevermind* (Flying Mollusk 2015), a **biofeedback**-enhanced adventure horror game that adapts to the player's stress levels by increasing the level of challenge it provides: the higher the stress the more the challenge for the player (see Fig. 11.4). A number of sensors which detect heart activity are available for affective interaction with *Nevermind*. *The Journey of Wild Divine* (Wild Divine 2001) is another biofeedback-based game designed to teach relaxation exercises via the player's blood volume pulse and skin conductance. It is also worth noting that AAA game developers such as Valve have

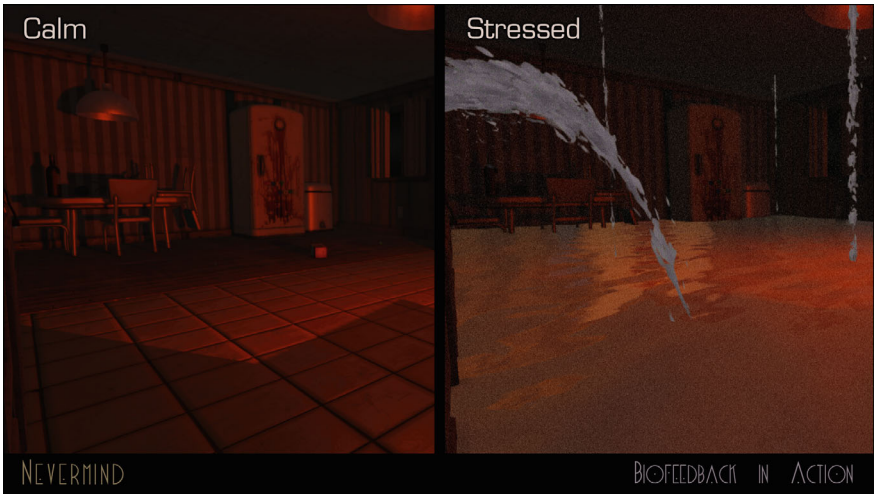


Fig. 11.4 A screenshot from *Nevermind* (Flying Mollusk 2015). The game supports several off-the-shelf sensors that allow the audiovisual content of the game to adapt to the stress levels of the player. Image obtained from Erin Reynolds with permission.

experimented with player physiology input for the personalization of games such as *Left 4 Dead* (Valve 2008), though they are not part part of the published game [22].

Objective Input: Considerations

While objective measurements can be a very informative way of assessing the player's state during the game, their limitations are not negligible. Usually most of the sensors are not available in a player's living room (i.e., in the wild) and most of those sensors are intrusive, thereby affecting gameplay at large. Moreover, the signals obtained are usually noisy due to environmental conditions and hardware limitations. In what follows, we discuss these limitations in more depth.

Sensing the multimodal data of a player in a reliable fashion has been a core challenge for the advancement of player modeling. Despite the continuous development of new sensing technology and annotation tools for players, research on player modeling still heavily relies on custom-made solutions [490]. The reasons for such hesitations are many; we cover the two most important ones here. First, game playing is not a static but rather a physically demanding activity—even when the player uses a controller of traditional consoles. This essentially means that any sensor used for player data collection has to be unobtrusive and thus offer unlimited degrees of freedom and comfort to the player. Second, as games generally impose a high level of cognitive load on the user, we expect a certain loss of emotional expressivity at times. There is growing evidence, for instance, suggesting that some emotion detection methods—such as face-based affect recognition—might be less appropriate when applied to certain types of players and games [965, 1018]. Beyond the general limitations of objective sensing there are a number of *specific considerations*

for each type of objective observation of players. Many **visual cues** obtained through a camera, for instance facial expressions, body posture and eye movement, require a well-lit environment. Such settings are often not present when one plays games in their dark living room. Moreover, camera-based sensors might be rightfully seen as privacy hazards by some players. Even though highly unobtrusive, the majority of the vision-based affect-detection systems currently available have additional limitations when asked to operate in real-time [1308]. An exception to this rule is body posture, which can both be effectively detected nowadays and can provide us with meaningful estimates of player experience in games where bodily expression matters [572]. However, the appropriateness of camera-based input modalities for games is questionable since experienced players tend to stay still while playing games likely due to habituation effects [35, 965, 1018]. Other types of camera-based objective measures appear to be implausible within commercial-standard game development. **Pupillometry** and **gaze tracking**, for instance, are very sensitive to distance from screen, and variations in light and screen luminance, which collectively make them rather impractical for use in a game application. The recent evolution of **virtual reality** (VR) devices, however, gives eye gaze sensing technologies entirely new opportunities and uses within games [1039]; a notable example of a VR headset that features reliable eye tracking technology is PlayStation VR2.

As a response to the limitations of camera-based measurements, **language**—expressed as speech or text—offers highly accessible, real-time efficient and unobtrusive modalities with great potential for gaming applications. Modalities based on language are primarily applicable to games where speech or text forms a control modality (as e.g., in conversational games [537, 1297]), collaborative games that naturally rely on speech or text for communication across players (e.g., in collaborative team-based first-person shooters), or games that rely on natural language processing such as text-based adventure games or interactive fiction (see Sect. 6.8).

When it comes to the players' **physiology**, existing hardware for EEG, respiration and EMG require the placement of body parts such as the head, the chest or parts of the face on the sensors, making those physiological signals rather impractical and highly intrusive for most games. On the contrary, recent sensor technology advancements for the measurement of electrodermal activity (skin conductivity), photoplethysmography (blood volume pulse), heart rate variability and skin temperature have made those physiological signals more attractive for the study of player emotion in games. As we cover extensively next, real-time recordings of these can nowadays be obtained via comfortable wristbands and stored in a personal computer or a mobile device via wireless connections [1284, 1286]. A number of VR headsets also feature EDA sensors that can be integrated directly to any VR game setting.

Despite its limited adoption to date, sensor technology can support research on player modeling through a continuous development of new sensing, data collection and player experience annotation tools that eliminate many of the traditional challenges of collecting data from human players. As listed in [1286] some indicative examples of reliable vendor-based solutions for EEG, EDA, HR, and eye-tracking

include Emotiv,[2] Empatica,[3] Intel Realsense,[4] Plux,[5] Polar,[6] Shimmer,[7] and Tobii,[8] among many others; most of these vendors offer processing software as well. Complementary to the above, conventional cameras can be used in conjunction with vendor-based software such as Affdex [735] or open source tools such as OpenFace [51] for facial recognition. Beyond game platforms that can integrate the aforementioned sensors—such as console, desktop and mobile—recent VR platforms open a promising direction for rich multimodal data collection in games. VR Headsets such as Valve Index,[9] HTC Vive,[10] and the PlayStation VR2, may be used for multimodal player data collection including gaze and physiology.

The physiological sensors that are most widespread among consumers these days are almost certainly those on smart watches. Apple alone has sold hundreds of millions of Apple Watches, which each contain a multitude of relevant sensors, including electrical and optical heart sensors, temperature, gyroscope, and accelerometer. Other manufacturers offer similar products. Interestingly, we have not seen much (even any) use of these sensors for player modeling. It stands to reason that someone should integrate smartwatch sensor-based player modeling into real-time game feedback.

11.1.3 Game Context

The game's context is the third category of input a player model may consider. Game **context** refers to the game's genre, the platform used, and the game's characteristics that collectively define the momentanous state of the game during play and excludes any gameplay aspects; those are already covered as part of the gameplay input (see Sect. 11.1.1). Clearly, our gameplay affects some aspects of the game context and vice versa but the two can be viewed as separate entities. Viewing this relationship from an analytics lens, the game context can be seen as a form of *game metrics* (see Sect. 10.3.1), opposed to gameplay which is a form of player metrics.

In this section, we discuss the various categories of game context that may affect a player's behavior and experience listed on Table 11.1. Specifically, the context of the game environment, the game genre, the form of interfacing, the number of players, the game content, potential social aspects of the game, and the overall objective of the game are foundational and they impact any other observation considered for a player model.

[2] https://www.emotiv.com/.
[3] https://www.empatica.com/.
[4] https://www.intelrealsense.com/.
[5] https://www.pluxbiosignals.com/.
[6] https://www.polar.com/.
[7] https://shimmersensing.com/.
[8] https://www.tobii.com/.
[9] https://store.steampowered.com/valveindex.
[10] https://www.vive.com/.

There are two types of *dynamic* game context that we can observe during a game: a player's interaction with AI agents and game content (see Table 11.1). Under the **AI agent** category, we fit any observable information related to AI-controlled agents that might be available in the game. Examples include agent facial expression, verbal, and non-verbal agent behavior, social agent behavior, and individual agent behavioral patterns that can affect a player's experience in games with strong social NPC presence such as *The Sims* series (Electronic Arts 2000–2024). Game **content** instead refers to any content type in a game not related to AI agent behaviors. Building on the taxonomy of Liapis et al. [636, 643] each game is viewed as a synthesis of creative facets available including visuals, audio, and narrative.

In addition to any observation related to the current state of the gameplay, context includes characteristics of the game we name **game profile**. Following the taxonomy introduced in Sect. 4.2 and adopted in [1286] the core game characteristics that fall under the game profile category include: the number of players, the observability of play, the stochasticity of the game, the time granularity, and the action space for the player (see Table 11.1). It is important to note that any information regarding the game profile is *static* meaning that no aspect of it can be altered by the player or the game. As a result, the game profile cannot act as a dynamic stimulus during play. One could think of games that change their genre or their mode of interaction (e.g., from VR to desktop) [108] but those alterations are rare. One could also think of *general* player modeling attempts across games of the same genre or even across games of dissimilar genres [166, 744] where game profile information varies across games and needs to be considered. The game profile is also tightly linked to a player's profile as we will discuss in the next section.

All game context types of input listed in Table 11.1 can affect the experience of play and are met in various configurations in games. As a matter of fact, it is rarely the case that only a single player experience elicitor (e.g., an image, or a sound) is active at a time. Instead, in-game stimuli are always orchestrated [636] and offer rich and multimodal experiences to players. As an example of such an orchestration process think of a text-based adventure game played with an expressive AI companion; the game is placed in a dungeon featuring horrifying visual and audio effects, a set of appropriate virtual camera movements and shots, and dim lighting.

Game Context: Considerations

The importance of the game context for modeling players is so obvious that can be summarized in a sentence: the lack of game context yields unreliable models of players. Player models ought to be conditioned to, and thus limited by, the game context. Let's think through an analogy here. When we wish to understand the emotional state of our discussants in a meeting we observe the social signals available (e.g., particular facial expressions) but it always helps if we can comprehend the underlying social and cultural context. In that same line of thought, any player reactions cannot be dissociated from the stimulus (or the game context) that elicited them, so naturally, player states are always linked to the game context. (External factors beyond the game context can affect a player's reactions during play but those are beyond the

control capacity of a player model.) As a result, player models that do not take context into account run a risk of inferring erroneous states for the player. For example, an increase in galvanic skin response can be linked to different high-arousal affective states such as frustration and excitement. It is very hard to tell however, what the heightened galvanic skin response "means" without knowing what is happening in the game at the moment. In another example, a particular facial expression of the player, recorded though a camera, could be associated with either an achievement in the game or a challenging moment, and needs to be triangulated with the current game state to be understood. Evidently, such dualities of the underlying player state may be detrimental for the design of the player model.

Very few studies have explored modalities of player input in isolation of the game context. Indicatively, a recent study derived models of player experience based solely on physiology for the purpose of comparison against models that fused aspects of in-game video and audio [695]. Good practice in player modeling, however, commands that any reactions of the players is triangulated with information about the current game state. The game context—naturally combined (or **fused**) with other input **modalities** from the player—has been used extensively in the literature for the prediction of different affective and cognitive states relevant to playing experience [229, 437, 693, 713, 714, 736, 737, 859, 927, 956, 1024]. The majority of the aforementioned studies fuse modalities of a player's physiology with in-game events and level attributes of arcade-like games.

11.1.4 Player Profile

A player profile includes all the information about the player which is *static* and it is not directly linked to the gameplay. This may include information on player personality (such as expressed by the Five Factor Model of personality [240, 733]), culture dependent factors, general demographics such as gender and age, and game demographics such as earlier experience with a particular game or game genre, or even preferences of particular games. A player's profile may be used as input to the player model to complement the captured in-game behavior with general attributes about the player. Such information may lead to more precise predictive models about players.

In the taxonomy presented in Table 11.1 we distinguish between *individual* aspects of a player profile such as one's **personality** and age and *social* aspects such as the player's network graph. Certain social aspects of a player's profile such as a console's online player profile are arguably more dynamic than a player's individual aspects. Somewhere between the highly dynamic in-game behavior and the static profile information about the player we may also consider *linked data* retrieved from web services that is not associated with gameplay per se. This type of metadata is primarily related to online social activity beyond the game under investigation and may include social media posts, emoticons, emojis [333], tags used, places visited, game reviews written, other games played, achievements unlocked, or any relevant

semantic information extracted from diverse content available online. The benefit of adding such information to player models has so far seen limited use in games [978]. In contrast to current player modeling approaches the use of massive amounts and dissimilar types of content across linked online sources would enable the design of player models which are based on user information stored across various online datasets, thereby realizing semantically-enriched game experiences. For example, sentiment-analyzed textual reviews [840, 849] of players from game review sites such as Metacritic[11] or GameRankings[12] can be used as input to a model. This model can then be used to create game content which is expected to appeal to the specific parts of the community, based, for instance, on demographics, skill or interests collected from the player's in-game achievements or favored games.

A popular way of integrating player profile information to a player model is via some form of **clustering** method that assigns a player to a predefined subgroup of the player population with certain characteristics. A player model for instance might consider players of matching skill one's geographic proximity (so as to avoid high ping times) as in *TrueMatch* [768]. Alternatively for game balancing purposes, a player is likely to be grouped together with other players from the same region and similar achievements unlocked in a game. After the player group is identified the game can act accordingly by offering experiences that are likely to be desired by the group. Such an approach has been widely used in player modeling research [741, 1141, 1274]. Game personalization systems that consider player profiles tend to adopt methods from recommender systems—such as those of Netflix [387]—for improving user retention and engagement based on player preferences. In the era of large language models, information of player profiles available online may already pretrained in LLMs, such as those of the GPT family, and could be fused with other multimodal data including frames of games and in-game reviews. While at the time of writing we cannot identify any studies on LLM-based player profiling, we foresee the use of LLMs as efficient player profiling methods complementary to existing techniques [365].

Player Profile: Considerations

While gender, age [1131, 1295], nationality [88] and player expertise level [165] have already proven important factors for profiling players the role of personality remains somewhat contentious. On the one hand, the findings of van Lankveld et al. [1206, 1207], for instance, reveal that gameplay behavior does not necessarily correspond to a player's behavior beyond the game. On the other hand, Yee et al. [1296] have identified strong correlations between player choices in *World of Warcraft* (Blizzard Entertainment 2004) and the personalities of its players. Strong correlations have also been found between the playing style and personality in the first-person shooter *Battlefield 3* (Electronic Arts 2011) [1132]. Recent work on personality modeling reveals that certain metrics related to a player's personality can be predicted in a

[11] http://www.metacritic.com.

[12] http://www.gamerankings.com/.

customized version of the role-playing game *Fallout: New Vegas* (Bethesda Softworks 2010) [421]. In general, we need to acknowledge that there is no guaranteed one-to-one mapping between a player's in-game behavior and personality, and that a player's personality profile does not necessarily indicate what the player would prefer or like in a game [1288]. Finally, the labeling and grouping of players under a specific player profile might not be representative (at best), misleading and even detrimental; we touch upon such ethical considerations in Chap. 15.

11.2 What Is the Model's Output Like?

The **output** of a player model is usually a representation of the player's state and refers to a player characteristic that we wish to model. The reader is also referred to Fig. 11.5 for illustrations of such relationships. In this section we detail three options for the output of the model, each relying on different types of human feedback and serving different purposes. First, if we wish to model the *experience* of the player the output is provided predominately through manual labeling, annotations, or human demonstrations of player experience. Second, if instead we wish to model aspects of player *behavior* the output is predominately based on in-game actions of players, human demonstrations or labels of behavior. Finally, it may very well be that we have no labeled data available and hence the model learns to predict a player *embedding*. The following Sect. 11.2.1 explores the condition where the model has no labeled data as output. Sections 11.2.2 and 11.2.3 discuss the particularities of the output, respectively, for the purpose of behavior and experience modeling.

11.2.1 Learning Player Embeddings

Labeling player data is a costly and tedious exercise that is often bypassed or engineered in the interest of simplicity. Obtaining human feedback in efficient and reliable ways remains an unresolved issue within [1323] and beyond [989] games research. Consequently very often we are faced with datasets where target outputs about player behavior or experience states are not available. One can think of the countless videos of gameplay—available through various online platforms—that are not labeled in any way. Player metrics may suffer from the same limitation of unlabeled data.

When labeled data is not available modeling of players must rely on any form of self-supervised or unsupervised learning [291, 295, 314, 425] for the construction of a player embedding (see Fig. 11.5). The input of a player embedding can be any observation available from the player including gameplay, context, objective and profile data. According to the self-supervised learning paradigm [49] detailed in Chap. 2, a player model can be trained without depending on human feedback. The player model instead is trained on the unlabeled data and learns to generate its training signal. Self-supervised methods for modeling players include generative

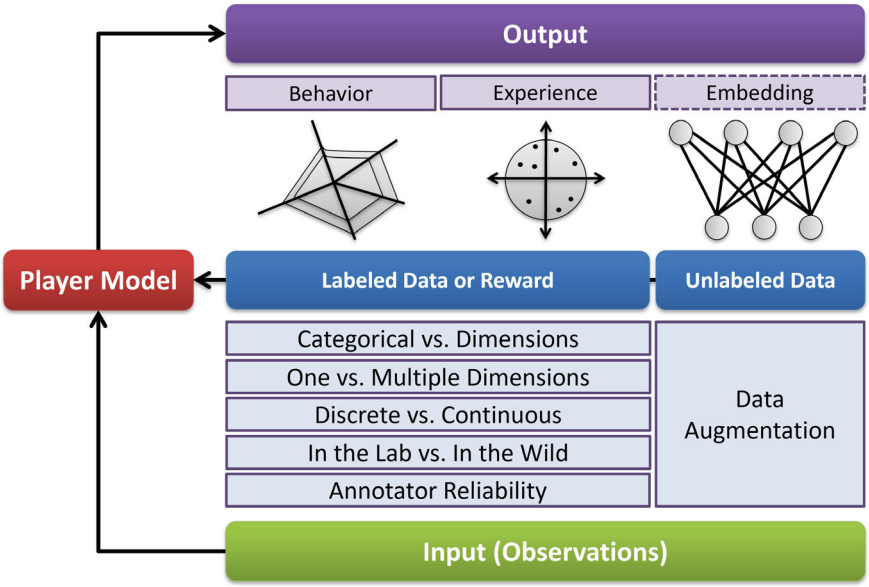


Fig. 11.5 Player modeling viewed from an **output** perspective as discussed in Sect. 11.2. Emphasis is put on the three possible output types a player model might have (behavior, experience, embedding) depicted with purple boxes. The dotted box indicates that player embedding is the only output type that does not rely on labeled data. Each output type is conditioned by a number of labeling principles (i.e., the light blue boxes under labeled data or rewards, and unlabeled data) that we discuss in this chapter. While we touch upon data augmentation in player embeddings in Sect. 11.2.1, in Sect. 11.3 we detail the principles of manual data labeling (related to player behavior and experience).

methods such as autoencoders, contrastive methods such as contrastive learning and adversarial methods such as GANs. Player modeling tasks associated with such methods are detailed in Sect. 11.4.5.

The unsupervised learning learning paradigm (introduced in Chap. 2) focuses on fitting a model to observations by discovering associations of the input and without having access to a target output. The input is generally treated as a set of random variables and a model is built through the observations of associations among the input vectors. Unsupervised learning as applied to modeling players involves tasks such as *clustering* and *frequent pattern mining* of which are detailed in Sect. 11.4.6.

Given that data in this player modeling output type is unlabeled there are no major concerns in relation to labeling principles. However, one important aspect of self-supervision in particular is **data augmentation** [1036]. There are many available methodologies for augmenting small datasets, commonly used in computer vision tasks, as briefly discussed in Part 1. Such methods may apply a number of transformation methods on player model observations (e.g., images of players or frames of games) such as image cropping, rotation, blurring and distortion, but also various brightness, contrast and saturation alterations. The process manages to augment

small training sets to larger datasets, thereby, enhancing the representation capacity of self-supervised learning methods. In principle, however, one needs to be careful with over-augmenting unlabeled datasets of players as doing so might backfire in several ways. First, data augmentation may increase the computational cost of training the player model beyond the anticipated budget. Second, one might introduce noise to the training corpus through the augmentation process that will, in turn, yield player models of worse generalization capacities. Finally, data augmentation is not a panacea. Augmenting a dataset is not always appropriate (or necessary) when, for instance, the available player observation corpus is already diverse enough.

11.2.2 Modeling Player Behavior

As discussed in the previous section, if no labeled data is available then player modeling is largely associated with a self-supervised of unsupervised learning paradigm. If labeled data about a player's behavior is available instead, then player modeling is tasked with the prediction of a particular behavioral state (or a set of player states). Independently of the behavioral downstream task under consideration player behavior modeling relies primarily on gameplay inputs such as the position of a player and the game context such as the current frame of the game. Player behavior models could be enhanced with player profile data but are not dependent on objective observations of the player. Such multimodal signals of players are usually considered in player experience modeling covered in the next section.

From a high-level behavioral perspective a game player makes a number of decisions in a sequential manner. Some of these decisions are actions that affect gameplay and the remaining in-game actions are those that do not affect gameplaying directly. Such AI models consider the moment-to-moment game state, that is available at a frequency of frame rates, learn to predict and output player actions at a rate that allows them to play the game. For example, we can learn to imitate the moves of a player on a frame-by-frame basis by comparing the play traces of AI agent and trajectories provided by human players as e.g., achieved in *Super Mario Bros* (Nintendo 1985) [784, 837] or Atari games [203]. An AI model that is able to predict gameplay actions could, in principle, be considered a player model but, strictly speaking, it is not. Instead it is an AI agent that is able to play a game as it predicts and infers sequences of gameplay actions (i.e., a gameplay AI agent). We have already made this distinction clear at the introductory chapter of this part and, thus, we detail generative AI models that play games in Part III.

So what can a player behavior model do if it doesn't predict the actions of gameplay, after all? A player model may predict any player state that is not directly linked to gameplay. In other words, a player model is not able to play a game but it may be able to predict any aspect of playing behavior beyond moment-to-moment actions. The target output of such a behavioral model is normally an aggregated feature of player behavior over time, or a behavioral pattern. Examples of such labeled data include player metrics such as game completion times, win rates, churn [314, 973] but

also metrics that are retrieved from team-based collaborative or competitive games such as game balance [875].

The dominant machine learning paradigm for player behavioral modeling is supervised learning (see Sect. 11.4.2. Generally speaking the model learns to predict the labeled data of the downstream task at hand. Variants of semi-supervised learning (weak supervision) can be adopted if the labeled dataset is generally small. Moreover reinforcement learning is applicable to the modeling task if aspects of player behavior can be expressed as reward functions. Models deriving from such a process may help us illuminate aspects of playing behavior and address questions related to gameplay such as the following: When will my pool of players stop playing this game? How good will this player be in that new level? Will that combination of teams yield a balanced game [875]? Is this game level played as intended? How can we predict toxicity through a player's in-game behavior [178]? Beyond providing data-informed answers about gameplay, player behavior models can enhance the capacity of an AI gameplaying agent or influence the generative expressivity of an AI algorithm. Think, for instance, of a racing game AI agent that consults a model of player tension for its decision making [60]. Alternatively think of a level generator that evaluates and adjusts the level's difficulty for a certain type of player, as in [1223].

It is important to note that a player embedding can be fused (or combined) with a player behavior model in many ways. A popular way to combine the two is to first derive an embedding (e.g., a latent vector of a CNN, player clusters, frequent patterns of player decisions) and then use that embedding as the observation of a player behavior model. A player model of behavior can serve all core roles identified in Chap. 10, including the analysis of the model per se, the use of the model for design purposes and the runtime use of the model to adapt the game to player behavioral needs.

11.2.3 Modeling Player Experience

To model the experience of the player one needs access to such experience labels that may include motivation factors for playing a level, challenge or difficulty of play, emotional states such as frustration and anxiety, or degrees of tension, engagement, valence or arousal. Those labels ideally need to be as close to the **ground truth** of experience as possible. The ground truth (or gold standard) in psychology and affective sciences [878] refers to a hypothesized and unknown label, value, or function, that best characterizes and represents an subjectively defined construct such as a player's experience. Labels of that sort are normally provided through manual annotation which is a laborious and costly process. Human feedback is however necessary given that we are required to predict subjective notions such as the emotional states of the player. The validity of such predictions can be questioned as there might be numerous factors contributing to a deviation between a label and the actual underlying player experience [1286].

As player experience models are trained to predict labels, the task of experience detection is largely viewed as a supervised learning problem [1286] through which observable attributes of the player (model's input) are mapped to the player's state (model's output). Any supervised learning method can be used for inferring such a mapping. Similarly to player behavior modeling weak supervision can be used when labeled data is not sufficient whereas reinforcement learning can be used when labeled data can shape a reward function.

11.3 Manual Labeling: Principles and Tools

After introducing the three possible output types of a player model in this section we discuss the critical, yet often overlooked, aspect of *manual labeling* and we review its underlying principles and good practices for player modeling; also refer to the blue boxes of Fig. 11.5. When faced with a player model *embedding* task we are generally faced with a set of unlabeled observations. Hence, one would argue that no particular labeling practices apply beyond the potential challenges of data augmentation we covered in Sect. 11.2.1. One might claim, however, that the labeling process of any player clusters, patterns, or latents that emerge deserves a careful analysis and discussion. While assigning labels on any embedding can be a challenging task it is a process that normally follows the construction of the player model. We consider such labeling processes an integral part of model analysis and we showcase it through a player clustering use case in *Tomb Raider: Underworld* (EIDOS interactive 2008); see Sect. 12.1. We also note that the obvious general principles governing collection, pre-processing, visualization and evaluation apply to all data (labeled or not) but are beyond the scope of this book.

Moving on to models of *behavior*, the labeling process is performed on in-game actions that a player model will try to predict. Normally those actions are objectively defined features of a player's behavior such as time spent on a task, purchasing history, menu selections, and daily usage of particular content. We thus consider labeling of player behavior primarily as an autonomous process. There are however several aspects of player behavior that are subjective in nature and require manual labeling such as the playing style, playing aesthetics, and the underlying motivations and intentions of a behavioral pattern. The general principles and tools we cover in this section apply to such behavioral labels too.

In addition to the subjective aspects of player behavior, player *experience* requires manual labeled data. The output of any player experience model is usually a set of particular states (e.g., happy), a scalar (e.g., the emotional dimensions of arousal and valence), or an ordinal relationship (e.g., tension is higher now than before). Providing labels representing the output of the model is predominately achieved through a subjective annotation process by the players themselves (first-person) or by others (third-person) such as game experience designers, peers or external observers. Those annotated labels ideally need to be as close to the ground truth of playing experience as possible. Refer to Chap. 2 and its dedicated section on human feedback (Sect. 2.4).

Manually annotating gameplay is a challenge in its own right with respect to both the human annotators involved and the human feedback protocol chosen [755, 1269, 1270, 1282]. First, the annotators need to be skilled enough to be able to approximate the actual experience, aesthetics or playing style well. Second, there are still many open research questions left for the design of the ideal human labeling collection protocol. For instance, who will end up doing the labeling: the person experiencing the gameplay or others? When it comes to timing, should the labeling be done during play, before or after? Will the labeling involve states (discrete representation) or instead involve the use of intensity dimensions (continuous representation)? Are labels provided in discrete time periods or continuously as demonstrations? Should the annotators be asked to rate a stimulus in an absolute fashion (e.g., frustration is 0.8) or, instead, rank it in a relative fashion (e.g., frustration is higher now)? Answers to the above questions yield different data annotation protocols and, inevitably, varying degrees of data quality, validity and reliability.

Section 2.4 has already addressed a number of the aforementioned questions that are general to AI algorithms that wish to employ and consider human feedback. These include the options of free response vs. forced choice labels, the first-person vs. the third-person protocol options, and the label data types available. In this section instead we focus on questions specific to player modeling that are usually raised in subjective labeling of player states. Moreover we detail the available tools that could be used for collecting player labels in games as adopted from [1286]. Table 11.2 summarizes all the tools we discuss.

Categorical Labels Versus Dimensions

A key question of manual annotation is how subjective notions of experience or play can be best represented and labeled. Player experience can potentially be represented as a number of different states (**categorical**) or, alternatively, as a set of **dimensions** as scalar values. On the one hand, categorical labeling is practical as a means of representing subjective aspects of play since the labels can form individual items (e.g., "excited", "annoyed", "believable" etc.) in a questionnaire, making it easy to ask the annotator to pick one (e.g., as in [1027]). Dimension-based labeling, on the other hand, appears to be advantageous for two key reasons. First, player states such as *tension* or *immersion* are hard to capture with words or linguistic expressions that have fuzzy boundaries. Second, states do not allow for variations in *experience intensity* over time since they are binary: either the state is present or not. For example, the complex notions of *fun*, or even *engagement*, cannot be easily captured by their corresponding linguistic representation in a questionnaire or define well a particular *state* of a playing experience. Instead it seems natural to represent them as a continuum of experience intensity that may vary over time (i.e., as a trace). For these reasons we often observe low agreement among the annotators [243] when we represent the playing experience via discrete states.

One of the major trends we observe in Table 11.2 is a historical shift from categorical and more complex annotation methods towards simpler, dimensional labeling techniques. Early annotation tools such as ANVIL [570] or ELAN [1253] (released

Table 11.2 Popular manual labeling tools discussed in this chapter. Labels appear in square brackets. Tool properties appear in parentheses. Adopted and adjusted from [1286]

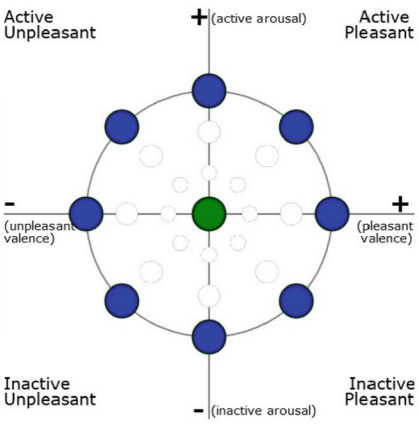
Tool	Categorical (C) versus Dimensions (D)	Discrete versus Trace
FeelTrace [244]	2D [Arousal-Valence]	Trace
EmuJoy [791]	2D [Arousal-Valence]	Trace
AffectButton [127]	3D [Pleasure-Arousal-Dominance]	Trace
ANNEMO [943]	1D (Configurable)	Trace
GTrace [245]	1D (Negative to Positive)	Trace
RankTrace [664]	1D (Unbounded)	Trace
DARMA [384]	2D (Configurable)	Trace
RCEA [1314]	2D [Arousal-Valence]	Trace
RCEA-360VR [1261]	2D [Arousal-Valence]	Trace
PAGAN (RankTrace) [742]	1D (Configurable, unbounded)	Trace
PAGAN (BTrace) [742]	1D (Configurable)	Discrete (configurable)
ANVIL [570]	C	Discrete
ELAN [1253]	C	Discrete
CARMA [383]	1D (Negative to Positive)	Discrete (configurable)
AffectRank [1282]	2D [Arousal-Valence]	Discrete
NOVA [442]	1D or C (Configurable)	Both

in 2001 and 2006, respectively) focused on measuring discrete categorical emotions. The shift towards continuous labels started with annotation tools such as FeelTrace [244], AffectButton [127], and *AffectRank* [1282], RankTrace [664] that allow annotators to label the affect dimensions of the *Circumplex Model of Emotions* [974] (see Fig. 10.5a). Figure 11.6 illustrates two such tools that are based on the arousal-valence core model of affect. Figure 11.7 depicts the RankTrace [664] continuous annotation tool which can be used for the annotation of a single dimension of a player's state (i.e., tension in this example). All three tools are accessible and of direct use for annotating subjective aspects of playing behavior and experience.

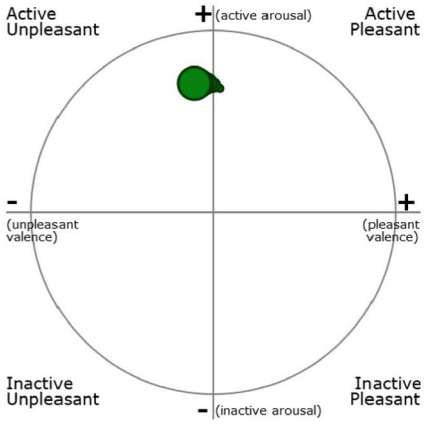
One Versus Multiple Dimensions Simultaneously

Once we opt to represent player states using the dimensional approach we need to decide whether one or more dimensions will be labeled simultaneously. While the two-dimensional annotation scheme is still popular nowadays we observe a gradual shift towards one-dimensional labeling tools in recent years. Table 11.2 lists GTrace [245], ANNEMO [943], CARMA [383], RankTrace [664], and PAGAN [742] as such tools. The core strength of one-dimensional labeling tools is the reduced cognitive load on annotators compared to multi-dimensional labeling where annotators are requested to split their attention across multiple dimensions and labels. Evidence

(a) Annotating facial expressions of players during play. The annotation is context-dependent as the video includes the gameplay of the player (see top left corner).



(b) *AffectRank*: A time-discrete annotation tool for arousal and valence.



(c) FeelTrace. A time-continuous annotation tool for arousal and valence.

Fig. 11.6 An example of third-person annotation based on videos of players and their gameplay using either **b** the *AffectRank* or **c** the FeelTrace annotation tool. *AffectRank* is freely available at: https://github.com/TAPeri/AffectRank

also suggests that focusing on one experience dimension at a time reduces noise and provides data of higher validity [245].

The benefits of multi-dimensional tools, however, are also many. Such tools can solicit labels at a higher rate as more than one notion can be labeled synchronously (e.g., arousal and valence in FeelTrace [244]). Moreover, as labels across different dimensions are collected simultaneously, the labels are less susceptible to recency

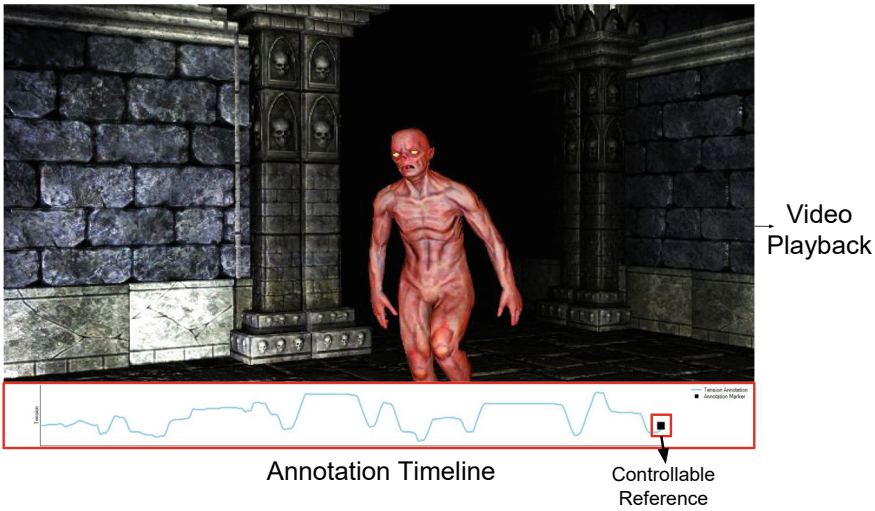


Fig. 11.7 The *RankTrace* annotation tool. In this example the tool is used for the annotation of tension in horror games. Participants play a game and then they annotate the level of tension by watching a video-recorded playthrough of their game session (top of image). The annotation trace is controlled via a wheel-like user interface. The entire annotation trace is shown for the participant's own reference (bottom of image). Image adopted from [664]

effects [326] compared to those obtained from repeated one-dimensional protocols. Indicative examples of such tools developed recently include the DARMA [384] and the RCEA variants [1261, 1314] (see Table 11.2).

Discrete Labels Vsersus Continuous Traces (Demonstrations)

Manual annotation can happen either within particular time intervals as discrete labels or continuously; refer to the third column of Table 11.2. Time-continuous **traces** (or **demonstrations**) have been popularized due to the existence of freely available tools such as FeelTrace [244] (see Fig. 11.6c) GTrace [245] and RankTrace [664, 742] which allow for continuous annotation of audiovisual content across the dimensions of arousal and valence. In addition to FeelTrace there are annotation tools like the continuous measurement system [754] and EmuJoy [791]; the latter is designed specifically for the annotation of music content. User interfaces such as wheels and knobs linked to the aforementioned annotation tools show further promise for the continuous annotation of a player's state in games [166, 216, 664, 742] (see Fig. 11.7). The continuous annotation process, however, requires more cognitive load than a time-discrete annotation protocol. The disadvantage of a higher cognitive load during the labeling process is that it yields disagreement among annotators and thus unreliable data for modeling players [276, 697].

As a response to the these limitations, **time-discrete** annotation provides data at particular intervals when the annotator feels there is a *change* in the player's state. And

changes are best indicated *relatively* rather than *absolutely*. *AffectRank*, for instance (see Fig. 11.6b), is a discrete, rank-based annotation tool that can be used for the annotation of any type of content (including images, video, text or speech) and provides annotations that are significantly more reliable than the annotations obtained from continuous annotation tools such as FeelTrace [1282]. The rank-based design of *AffectRank* is motivated by observations of recent studies in third-person video annotation indicating that "…humans are better at rating emotions in *relative* rather than absolute terms." [755, 1269, 1270, 1282]. Further, *AffectRank* is grounded in numerous findings showcasing the supremacy of ranks over ratings for obtaining annotations of lower inconsistency and order effects [716, 755, 1265, 1278, 1282, 1283].

A recent tool that builds on the relative-based annotation of *AffectRank* and allows for the annotation of any player state in a continuous yet unbounded fashion is *RankTrace* (see Fig. 11.7). The core idea behind *RankTrace* is introduced in [216]: the tool asks participants to watch the recorded playthrough of a play session and annotate in real-time the perceived intensity of a single dimension such as tension or engagement. The annotation process in *RankTrace* is controlled through a "wheel-like" hardware, allowing participants to meticulously increase or decrease emotional intensity by turning the wheel, similarly to how volume is controlled in a sound amplifier. Further, the general interfacing design of *RankTrace* builds on the one-dimensional GTrace annotation tool [245]. Unlike other continuous annotation tools, annotation in *RankTrace* is *unbounded*: participants can continuously increase or decrease the intensity as desired without constraining themselves to an absolute scale. This design decision is built on the *anchor* [1013] and *adaptation level* [443] psychology theories by which affect is a temporal notion based on earlier experience that is best expressed in relative terms [664, 1269, 1282]. The use of *RankTrace* has revealed the benefits of relative and unbounded annotation for modeling subjective player states like experience and style more reliably [664] and has also showed promise for the construction of general models of player emotion across games [166].

In the Lab Versus In the Wild?

Another critical question one needs to address is whether data will be collected in a university or **lab** setting or outside such controlled premises (i.e., **in the wild**). Most traditional labeling tools (see Table 11.2) require the presence of a researcher and the annotator in a lab which, in turn, limits the scope, the scale and ultimately the size of the dataset. In addition, the recent social reality brought on by the COVID-19 pandemic further necessitated the use of online methods for label collection. Indicatively, PAGAN [742] is one of the first frameworks developed with crowd-sourcing capacities in mind (see Fig. 11.8). Compared to earlier annotation tools, PAGAN is highly configurable and supports multiple different annotation methods including GTrace and RankTrace variants. While PAGAN tackles the issue of crowd-sourcing via an online platform, other annotation tools offer mobile integration (RCEA [1314]), VR integration (RCEA-360VR [1261]) or aim to alleviate the need for annotators through (semi) **automatic labeling** (e.g., NOVA [442]). At the time of writing we observe a shift towards crowd-sourcing and mobile integration in custom annotation tools [668,

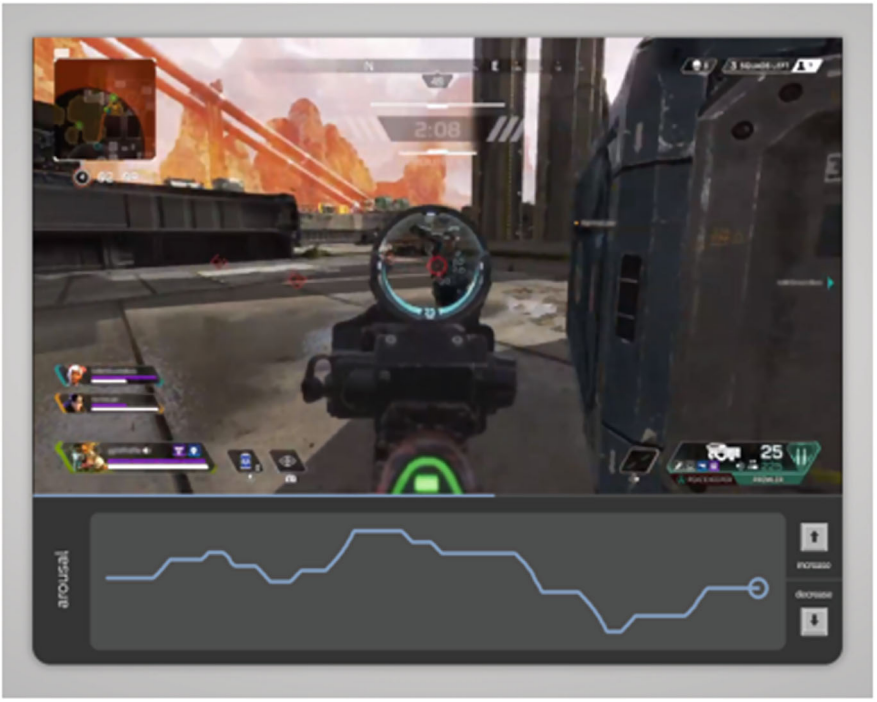
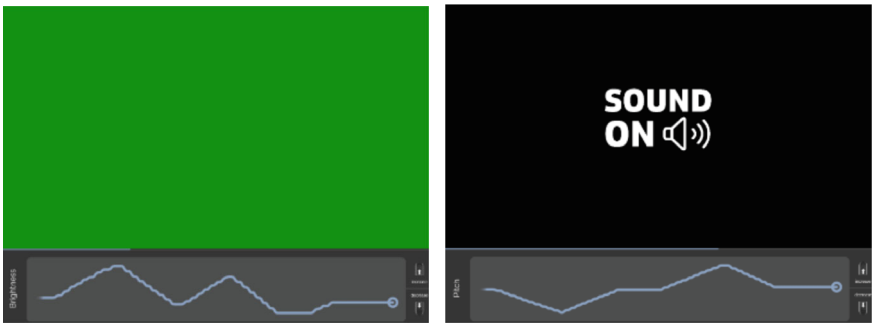


Fig. 11.8 RankTrace interface in the PAGAN platform annotating *Apex Legends* (Electronic Arts 2019) gameplay footage. No copyright infringement intended. Image adopted from [742]. PAGAN is currently available at: https://pagan.institutedigitalgames.com and https://github.com/davidmelhart/PAGAN

984] and we expect more reliable crowd-sourcing annotation tools to appear as the focus will continuously shift from the lab to real-world experiences. We also envisage that future annotation tools will gradually integrate few-shot [886] or even zero-shot learning methods that would, in part, automate the manual annotation process.

Knowing Your Annotator

Taking label collection to the wild comes with several disadvantages due to the limited control of the online labeling process. The most critical of those limitations is the reliability of unknown (likely untrained) annotators and the resulting validity of the labels one obtains ultimately. Several approaches have been introduced as a response to detect the reliability of the solicited labels but, generally speaking, the most popular of those either attempt to maximize the inter-rater agreement of the labels or process labels so as to remove any unnecessary subjective biases [1270]. A critical factor that is rarely investigated, however, is the reliability of the annotators per se. In fact, many aspects can contribute to the unreliability of a human annotator. Those include the lack of understanding of the task at hand or the labels described, inattentiveness,



(a) Visual (b) Auditory

Fig. 11.9 Tools assessing the quality of visual **a** and auditory **a** annotations. The QA tests employ the RankTrace tool through which annotators express **a** the level of color brightness or **b** audio pitch. The QA tools are available and accessible through the PAGAN annotation platform. Images adopted from [64] with author permissions

poor or faulty equipment, and several other uncontrollable parameters when label annotation operates in the wild.

There are multiple ways to identify unreliable annotators, primarily the ones you have not trained and you cannot really observe during the annotation task. To detect an unreliable annotator one needs to first define an objective ground truth, to the degree possible. Imagine for instance an objectively defined tension curve of an FPS game that increases when the player shoots at enemies. Once the ground truth is defined, any deviation from it could indicate that the corresponding annotator is not reliable. A common approach is to define a ground truth as the labels obtained from a set of experts (e.g., researchers). In [1323], for instance, labels about agent human-likeness were contrasted to crowd-sourced labels for validity. An alternative recent approach is to request all your annotators to take some quality assurance (QA) tests prior to their annotation task [64]. The QA will test the reliability of the annotator on objectively defined ground truths such as the color intensity of a video or the pitch of a sound; see Fig. 11.9. Findings suggest that such QA tools can easily distinguish between trained annotators and the best of untrained crowdworkers and, importantly, predict effectively the reliability of an annotator, thereby, saving on resources, effort and cost [64].

11.4 Methods for Player Modeling

Which AI algorithm should we use to model players given a set of player observations and (possibly) corresponding labels? There are many options out there. In this section we discuss the various AI methods available for deriving a player model including authoring, supervised learning variants (self- and semi-supervised learning included),

reinforcement learning and unsupervised learning. To showcase the difference among these methods let us suppose we wish to predict an aspect of a player's behavior such as their playing style; examples of playing style labels may include aggressive, achiever, content purchaser, trader, social, toxic and so on. If we don't have a dataset of player style observations or we don't wish to follow the machine learning route we can only author a player style model. If instead we wish to follow the data-driven approach we can only use unsupervised learning if no behavioral classes of style have been defined a priori and we don't have access to any objectively defined measures for playing style (i.e., unlabeled data). An alternative approach in this example of unlabeled data would be to use a self-supervised learning method and let it infer an embedding of player style. We can instead use supervised learning if, for example, we have labels of style and we wish to train a player model to predict a player's style. In that case we might already have an initial classification of players (either manually or via clustering) and we might wish to fit new players into these predefined classes. We can use reinforcement learning to derive policies that imitate different aspects of playing style provided as reward functions from the interaction with the environment. Finally, we may instead employ semi-supervised learning methods for modeling style when only a few of our player observations are labeled. Figure 11.10 situates the core paradigms we discuss here within the general player modeling framework.

11.4.1 Player Model Authoring

Imagine you wish to model a player's style or engagement level but you don't have access to any player data. Instead image that you have access to such data but you cannot afford to (or have any desire to) use any machine learning method. In such occasions the only way forward is to *author* the model directly. Doing so follows the model-based approach of player modeling we covered in the taxonomy of Chap. 10.

Modeling players in that way usually involves adopting a theoretical model such as the *theory of fun* [581], the OCC model [838] of the self-determination theory [271]. Modeling approaches vary from simple rule-based systems, to finite state machines, behavior trees and utility-based AI methods covered in Part 1. When one authors a utility system, a hierarchy of FSMs or BTs with a theoretical framework in mind, the result is a computational model of that framework for the players of the game under consideration.

The authoring approach for modeling players is very popular within the game industry. In principle, any rule, heuristic, FSM or utility that is designed to cluster players according to style, paying profile, or skill (among other characteristics) can be viewed as model authoring. We could argue that most games feature such an AI system as long as some (even a basic) form of player model is part of the design process. The model can then been used for monetization, balancing, testing and other purposes we already covered in Chap. 10. Starting with the most representative example of social interactions with NPCs, *The Sims* series (Electronic Arts 2000–2024) feature NPCs that consider the player's current emotional state, needs and

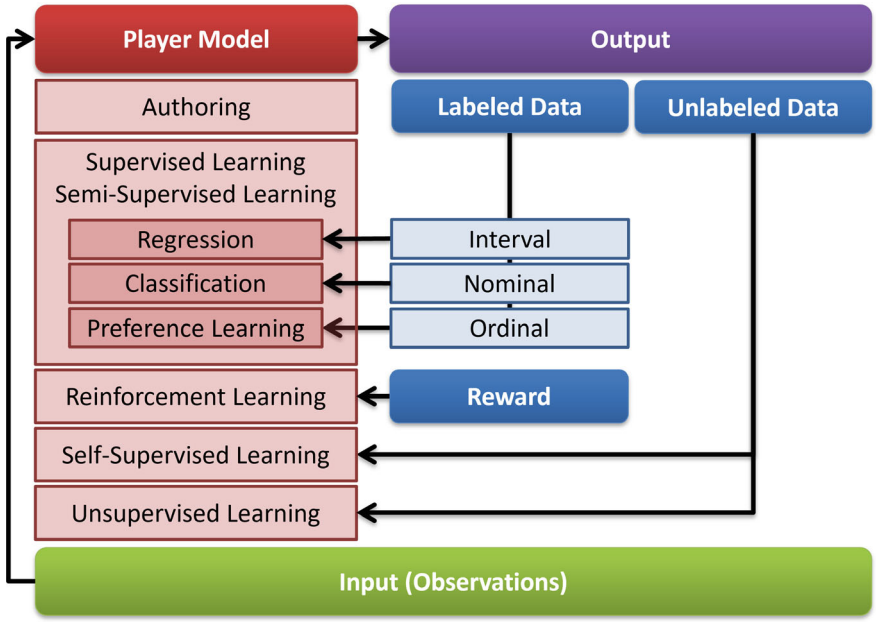


Fig. 11.10 The algorithmic components of player modeling discussed in Sect. 11.4. Model authoring is applied when no data is available or considered by the player model. If a data-driven approach is followed instead, emphasis is put on the possible training signals (labeled data, rewards, unlabeled data) and the corresponding AI paradigms covered in the section. **Labeled data** serve primarily supervised and semi-supervised learning methods to yield player models of behavior or experience. **Rewards**, if available, inform reinforcement learning processes that derive models of player behavior or experience. Finally, **unlabeled** data feed self-supervised and unsupervised learning methods that, in turn, yield an embedding as a player modeling output.

other behavioral patterns to take their decisions. Such ad-hoc models can be inspired by BDI [375] architectures—as the BDI model that controls the creature of *Black and White* (EA 2000)—and designed to follow a utility-based AI approach. Another popular example is the *AI Director* of *Left 4 Dead* (Valve 2008). The director assesses the current tension of the game through simple statistics of the player's performance, therefore matching player skill to in-game tension through an authored model. Then the director uses that model to adapt the game to the player's progression based on a predefined tension curve, which is authored too! Tracing back in time, one might claim that even early video games such as *Pac-Man* (Namco 1980) feature authored player models. The behavior of the ghosts in that game follows a rule-based system that models the degree of tension between the player and their opponents [500].

Several early academic studies in player modeling adopted an authoring approach for modeling fun and interestingness of gameplay in racing [1156] and Pac-Man-like games [1271]. The approach is still popular nowadays, however. Recent player modeling studies have modeled the level of *fun* a Mario player is having while playing through a simple heuristic: moderate degrees of level and gameplay diversity

make a *Super Mario Bros* (Nintendo 1985) level fun to play [1038, 1239]. That heuristic attempts to quantify the core principles of Koster's theory of fun [581] and it does a relatively good job capturing it as a follow up user study suggests [1238]. Similar player model authoring methods have been adopted by Roohi et al. [963, 964] mapping aspects of player experience to behavioral player patterns based on theoretical models of intrinsic motivation.

While authoring a player model can be effective when data is not available, it can also be limited by the expertise of the person (or the team) that designed them. In the remainder of this section we explore AI methods that rely on the model-free (data-driven) approach of player modeling. The methods assume that at least some player data (observations) is available.

11.4.2 Supervised Learning

Data-driven player modeling consists of finding a function that maps a set of measurable attributes of the player to a particular player state. Following the supervised learning approach this is achieved by automatically adjusting, the parameters of a model to fit a dataset that contains a set of input samples, each one paired with target outputs. The input corresponds to the list of measurable attributes while the output corresponds to the annotations of the player's states for each input sample we wish to predict. As mentioned already, labels may vary from behavioral characteristics, such as completion times of a level or player archetypes, to estimates of player experience, such as player frustration.

As we saw in Chap. 2 any supervised learning technique can be used in games for the analysis, the imitation and the prediction of player behavior, and the modeling of playing experience. The data type of the label (see Sect. 2.4) determines the output of the model and, in turn, the type of the machine learning approach that can be applied. In the remainder of this section we discuss the paradigms of **regression**, **classification** and **preference learning**, respectively, to learn from numerical (or interval), nominal and ordinal labels. Chapter 3 provides detailed presentations of these paradigms for the interested reader.

Regression

When the outputs a player model needs to approximate are interval values, the modeling downstream task is known as metric or standard regression. Any regression algorithm is applicable to the task, including linear or polynomial regression, support vector machines, and artificial neural networks of any sort. Refer to Sects. 3.5 and 3.6 for details on popular regression algorithms.

Regression algorithms are appropriate for **objectively-defined** tasks of player *behavior* including temporal aspects of play such as a game's score or the total playing time (i.e., churn) as in [613, 973] among many other studies. When the task, however, is to predict subjectively-defined labels, such as playing style or player

experience, caution needs to be put on the data analysis. While it is possible to use regression algorithms to learn the exact numeric ratings of style, believability, or experience, in general it should be avoided because regression methods assume that the target values follow a numerical scale. Ratings naturally define an ordinal scale [1269, 1270, 1278] instead. Prediction models trained to approximate a real-value representation of a rating—even though they may achieve high prediction accuracy values—do not necessarily capture the true reported playing experience because the ground truth used for training and validation of the model can been undermined by the numerous effects discussed in Sect. 2.4. We argue that the known fundamental pitfalls of **human feedback** and self-reporting outlined in [602, 755, 845, 1269, 1270] provide sufficient evidence against the use of regression for player experience modeling.

Regression methods are popular within studies of continuous experience labels (e.g., Nicolaou et al. [810]). Even though labels are provided as sequences of real-values, one must bear in mind that the aforementioned limitations are still present in continuous labeling. That said, there exist a few uses of regression in affect modeling that are methodologically sound with respect to the treatment of ratings. In particular, regression was used in [748] to learn the probability of a particular pose representing a discrete emotion. The probability was given as the number of annotators that tagged the posture with an emotion over the total number of annotations on it.

Classification

Classification is the appropriate form of supervised learning for player modeling when the labeled data represent a finite and non-structured set of classes. Classification methods can infer a mapping between those classes and player attributes. Available algorithms include shallow and deep artificial neural networks, decision trees, random forests, support vector machines, k-nearest neighbors, and ensemble learning among many others. Further details about a number of these algorithms can be found in Sects. 3.5 and 3.6.

Classes can represent aspects of playing behavior which need to be predicted, such as completion times (e.g., expressed as low, average or high completion time), **user retention** in a free-to-play game (e.g., expressed as weak, mild or strong retention) or player type in terms of **monetization** (e.g., non-monetizers vs. monetizers) or type of player toxic behavior as in [178]. Classes can alternatively represent player experience states such as an *excited* versus a *frustrated* player as manifested from facial expressions or *low*, *neutral* and *high* engagement states for a player. Classification is perfectly suited for modeling player experience if discrete annotations of experience (provided as target outputs) are nominal [255, 573].

Preference Learning

As an alternative to regression and classification methods, preference learning [359] methods are designed to learn from ordinal data such as ranks or preferences. Note that the training signal in the preference learning paradigm merely provides information for the **relative** relation between instances of the phenomenon we attempt to

approximate. Strictly speaking, the target outputs that follow an ordinal scale do not provide information about the intensity (regression) or the clusters (classification) of the phenomenon. However, one may wish to use regression analysis to predict an ordinal variable such as the player levels of preference from 1 to 7. In that case one can employ algorithms from the **ordinal regression** [1252] family where the objective is to predict a categorical scale with an underlying natural order between the labels [417]. Alternatively, a relative change or preference might be expressed as a class—such as the increase or decrease of viewer engagement over time—and treated as a classification problem.

Generally we could construct a player model based on in-game *behavioral preferences*. The information that a particular player, for example, prefers the mini-gun over a number of other weapons could form a set of pairwise preferences we could learn from. Alternatively we can build a model based on *experience preferences*. A player, for instance, reported that area X of the level is more challenging than area Y of the same level. Based on a set of such pairwise preferences we can derive a global function of challenge for that player.

As outlined in Chaps. 2 and 3 a large palette of algorithms is available for the task of preference learning. Many popular classification and regression techniques have been adapted to tackle preference learning tasks, including linear statistical models such as linear discriminant analysis and large margins, and non-linear approaches such as Gaussian processes [211], deep and shallow artificial neural networks [710], support vector machines [512] and preference transformers [562].

Preference learning has already been extensively applied for modeling aspects of players. For example, an increasing number of studies [60, 367, 710, 711, 740, 1276, 1285] have explored **linear discriminant analysis**, support vector machines, random forests and artificial neural network (shallow and deep variants) to predict player states expressed as pairwise preferences. Player states vary from affective constructs (e.g., viewer engagement, player enjoyment) and cognitive states (e.g., player motivation) to behavioral traits (e.g., player decisions). The derived surrogate models of player states have been employed across several game genres, such as platformers, racing, first person shooters, mobile, and horror games, to serve all possible roles a player model can take. Derived models can be used to analyze players, to drive the design process of a game, or even to enable player-adaptive games in runtime such as the AI director of *Left 4 Dead* (Valve 2008) or the adaptive antagonist in *Hello Neighbor* (tinyBuild 2017).

If ratings of player states are the only labels available we recommend their processing in an ordinal fashion, building on a recent study that attempted to model player motivation in *Tom Clancy's the Division* (Ubisoft 2016) [740]. The SVM model used in that study would map accurately in-game attributes—such as completion times and shooting patterns—to reported motivation ratings on the UPEQ questionnaire [43]. The motivation ratings, however, were converted to set of pairwise preferences that the SVM was tasked to learn via preference learning. As ratings express ordinal scales they can directly be transposed to any ordinal representation (e.g., pairwise preferences). For instance, given an annotator's rating indicating that a condition A felt 'slightly frustrating' and a condition B felt 'very frustrating', a preference

learning method can train a model that predicts a higher level of frustration for B than for A. In this way the modeling approach avoids introducing artifacts of what is the actual difference between 'very' and 'slightly' or the usage of the scale for this particular annotator. Further, the limitation of different subjective scales across users can be safely bypassed by transforming rating reports into ordinal relations on a per-annotator basis [1270]. Finally, the problem of the scale varying across time due to episodic memory still persists but can be minimized by transforming only consecutive reports. For example, given a report for three stimuli A, B and C provided in that order, the player model can be trained using only the relation between A and B, and between B and C, but dropping the comparison between A and C.

Summary: The Good, the Bad and the Ugly

The last section on supervised learning is dedicated to the comparison among the three downstream tasks—regression, classification and preference learning—for modeling players. Arguably the discussion is limited when the in-game *behavior* of players is predicted. If behavioral data about players follows interval, nominal or ordinal scales then naturally, regression, classification and preference learning should be applied, respectively. Behavioral labels, such as player actions, decisions, times spent across various tasks, are objective by nature which makes the task of learning less challenging. Given the subjective notion of player *experience* and aspects of playing behavior like style, cognitive reactions, and aesthetics, however, there are a number of caveats and limitations of each algorithm that need to be taken into account. Next, we discuss the comparative advantages of each and we summarize the key outcomes of supervised learning as applied to modeling the experience of players.

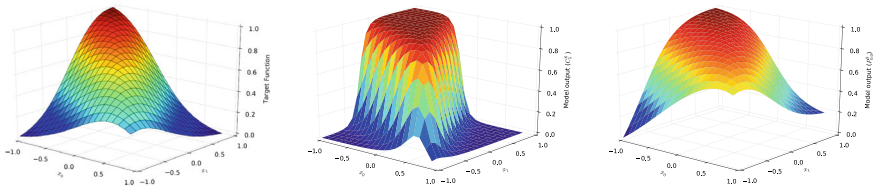
Regression Versus Preference Learning: Motivated by psychological studies suggesting that interval ratings misrepresent player experience [602, 755, 845], we will not dedicate ourselves an extensive comparison between preference learning and regression methods. The performance comparison between a regression and preference-learned model is also irrelevant as the former is arguably a priori incapable of capturing the underlying experience phenomenon as precisely as the latter. Such deviations from the ground truth, however, are not trivial to illustrate through a data modeling approach and thus the comparison is not straightforward. The main reason is that the objective ground truth is fundamentally ill-defined when a certain magnitude is used to label and characterize subjective notions such as player experience.

A viable approach that stands in between a preference learner and a regression model is **ordinal regression** [417]. While ordinal regression and preference learning are both used to tackle ordering tasks, the former uses regression analysis to predict labels that have a natural order between them such as an engagement scale (e.g., with labels 1–5), tension labels *low*, *medium*, and *high*, or game difficulty levels as in [350]. Preference learning instead, predicts relative preferences between pairs of data and can infer the global ranking of the phenomenon from those pairs. Compared to ordinal regression, preference learning is a more versatile, robust and scalable approach as it can still approximate ordinal regression problems and learn the underlying ranking of

the given labels by relying only on comparisons between labels [359]. By definition, ordinal regression is tasked to predict subjective labels, thereby introducing scaling biases to the player model. Moreover, ordinal regression is not applicable when preference data is the only data type available. For all aforementioned reasons we advise the use preference learning over regression and ordinal regression methods for modeling players.

Regression Versus Classification: Classes are easy to analyze and create player models from. Further, their use eliminates part of the inter-personal biases introduced with ratings. For these reasons classification should be preferred to regression for player experience modeling. We already saw that classification, instead of regression, is applied when ratings are available to overcome part of the limitations inherent in rating-based annotation. For instance, this can be achieved by transforming arousal ratings to *high*, *neutral* and *low* arousal classes [438, 693]. While this common practice in psychometrics eliminates part of the rating subjectivity it adds new forms of data biases inherent in the ad-hoc decisions to split the classes. Further, the analysis of player models across several case studies in the literature has already shown that transforming ratings into classes creates a more complicated machine learning downstream task [716, 1269, 1270]. Alternatively to regression or classification, ordinal regression can serve as a viable method that assumes ordinal relationship in between classes. Even though it comes with certain limitations covered earlier, ordinal regression builds classifiers that at least respect an ordinal relationship between labels such as the difficulty levels of a game [350]. We thus recommend that ordinal regression is preferred to classification whenever ordinal labels are available.

Classification Versus Preference Learning: Preference learning is the superior method for modeling experience when ranks or pairwise preferences are available. Even when ratings or classes are available, comparisons between classification and preference learning player models in the literature suggest that preference learning methods lead to more efficient, generic and robust models which capture more information about the ground truth [1269, 1270]. Indicatively, Crammer and Signer [247] compare classification, regression and preference learning training algorithms in a task to learn ratings. They report the supremacy of preference learning over the other methods based on several synthetic datasets and a dataset of movie ratings. In addition, extensive evidence already shows that preference learning better approximates the underlying function between input (e.g., experience manifestations such as gameplay) and output (e.g., annotations) [716]. Figure 11.11 showcases how much a preference learned model can approximate a hypothesized (artificial) ground truth, compared to a classification model trained on an artificial dataset. Compared to other ordinal learning methods such as ordinal regression, preference learning is the method of choice for improved reliability of the obtained models. The benefits of a preference learner over an ordinal regression model are discussed earlier in the comparison of the former against regression. In summary, preference learning via rank-based annotation controls for reporting memory effects, eliminates subjectivity biases and builds models that are closer to the **ground truth** of player experience [1282, 1283].



(a) Ground Truth (b) Classification (c) Preference learning

Fig. 11.11 A hypothesized (artificial) ground truth function (z-axis) which is dependent on two player attributes, x_1 and x_2 (**a**), the best classification model (**b**) and the best preference learned model (**c**). All images are retrieved from [716] with permission

Conclusion: Grounded in extensive evidence the selection of a supervised learning approach for modeling subjective notions of players appears to be obvious. To make our point even clearer, we draw an analogy from the popular spaghetti Western film *The Good, the Bad and the Ugly*, directed by Sergio Leone in 1966, and map the 3 learning paradigms to the 3 main characters of the film. Independently of how subjective notions of playing behavior and experience are labeled we argue that preference learning (i.e., *the good*) is a superior supervised learning method for the task at hand, classification (i.e., *the ugly*) provides a good balance between simplicity and approximation of the ground truth of whatever is modeled whereas regression (i.e., *the bad*) is based on rating annotations which are of questionable quality with respect to their relevance to the true subjective labels.

11.4.3 Reinforcement Learning

While it is possible to use reinforcement learning to model aspects of users during their interaction the RL approach for modeling players has been tried mostly in comparatively simplistic and abstract games [325] and has not seen much application in computer games. The key motivation for the use of RL for modeling players is that it can capture the relative valuation of game states as encoded internally by humans during play [1126]. At first glance, player modeling with RL may seem to be an application of RL for game playing, and we discuss this in Part II as part of the *play for experience* aim. In this section, we instead discuss this approach from the perspective that a policy learned via RL can capture internal player states with no corresponding absolute target values such as decision making, **learnability**, motivation, cognitive skills or emotive patterns. Further, those policies can be trained on player data such as human **play traces** of any sort including affect demonstrations. The derived player model depicts psychometrically valid, abstract simulations of a human player's internal cognitive or affective processes. The model can be used directly to interpret human play, or indirectly, it can be featured in AI agents which can be used as **playtesting** bots during the game design process, as baselines for

adapting agents to mimic classes of human players, or as believable, human-like opponents [60, 460].

Using the RL paradigm, we can construct player models via RL if a reward signal can adjust a set of parameters that characterize the player. The reward signal can be based either directly on the in-game behavior of the player—for instance, the decision taken at a particular game state—or indirectly on annotations (e.g., annotated excitement throughout the level) or objective data (e.g., physiological indexes showing player stress). The immediate reward function can be based on gameplay data if the model wishes to predict aspects of the gameplaying behavior such as playing style and behavioral aesthetics [41], or player engagement in terms of player churn and difficulty in terms of completion rates [962]. Alternatively, the reward can be based on any objective input from the player or a subjective report if the model attempts to predict aspects of player experience, such as the reported arousal reward of Barthet et al. [62] on a racing game. The representation of the RL approach can be anything from a standard Q table that, for instance, models the decision making behavior of a player (e.g., as in [460, 1126]) to a neural network that models in-game behavior [459, 962, 963], to a set of behavior scripts [1074] that are adjusted to predict gameplay behavior via RL, to hard-exploration algorithms like Go-Explore [304] that learn to predict the arousal level of a racing game at a particular game state [60].

Based on the roles a player model may have in a game production we view two ways of constructing models of players via RL. Models can be built **offline** (i.e., before gameplay starts) for the purposes of model analysis and model use during the design phases of the game (see Sect. 10.3.1. Alternatively, models can be built at **runtime** (i.e., during play) for player-adaptation purposes. We can also envision hybrid approaches by which models are first built offline and then are polished at runtime. Offline RL-based modeling adds value to our playtesting capacity via, for instance, *procedural personas* (see Part II) whereas runtime RL-based player modeling offers *dynamicity* to the model. Player modeling further adds on the capacity of the model to adapt to the particular characteristics of the player, thereby increasing the degree of personalization. We can think of models of players, for instance, that are continuously tailored to the current player by using the player's in-game annotations, behavioral decisions, or even physiological responses during the game.

RL-based player modeling approaches can be grouped in two classes: those that rely on human feedback and those that do not. Modeling methods that rely on human feedback either consider *expert* labels such as player behavior utilities specified by game designers [459] or, alternatively, model labels provided by the *players* themselves [61, 62]. RL-based methods that do not consider human labels would normally shape reward functions via a theoretical model that maps player experience to behavioral player patterns such as that of **intrinsic motivation** [963, 964].

The RL way of modeling players is gaining popularity over the years through a number of studies across different game genres. Indicatively, RL methods have been used for player modeling in educational games [809], in roguelike adventure games [459, 460] via evolutionary reinforcement learning, in first-person shooter games [1126] via inverse RL, in racing games [60] via the use of RL variants [304]

and in *Angry Birds Dream Blast* (Rovio Entertainment 2016) via deep RL variants [962, 963].

As detailed in Sect. 2.4 human feedback—provided in the form of ordinal labels—defines the dominant approach for shaping a reward model when RL seeks to optimize subjective notions such as the aesthetics [41] or the hypothetical feelings [60] of a gameplaying agent. These reward models can train RL agents to complete various tasks of subjective nature in games such as expressing their current in-game experiences and justifying their actions. Human feedback can also fine-tune LLMs via RL (see RLHF) [844] for similar purposes. The application of RL for modeling users beyond games has been quite active for the purposes of modeling web-usage data and interactions on the web [1124] or modeling user simulations in dialog systems [196, 376, 995]. Normally in such systems a statistical model is first trained on a corpus of human-computer interaction data for simulating (imitating) user behavior. Then reinforcement learning is used to tailor the model towards an optimal dialog strategy which can be found through trial and error interactions between the user and the simulated user. Fine-tuning LLMs via RLHF [844] is, at the time of writing, the most popular use or RL for modeling the opinions, preferences and intentions of users across any domain. While the application of RLHF within games is not as active currently we expect it to be in the near future and we touch upon the promise of that direction in Chap. 14.

11.4.4 Semi-Supervised Learning

The semi-supervised learning paradigm is relevant to player modeling when some observations of players are labeled but those labels are limited. As an example, one can think of very few labels of *engagement* left by players in a certain part of a particular level as in the *GameVibe* dataset [59]. Methods of self-supervised learning, as discussed in Part 1, become necessary when the cost of obtaining further labels is very high. This particularly applies to labels of player experience since the manual annotation process is generally tedious and costly [1286].

Potential ways of building player models using the semi-supervised learning paradigm include the *pseudo-labeling* and *self-training* methods as in [46] for labeling *Minecraft* (Mojang Studios 2011) gameplaying actions. In general these methods first build a model based on the labeled data using standard supervised learning methods. The trained model is then used to label the unlabeled data with pseudo labels. The most confident class of the model becomes the label for the data point. Finally the model is retrained on the resulting labeled dataset, containing real and the pseudo labels.

The semi-supervised learning paradigm has seen limited uses for building player models beyond gameplaying imitation learning tasks [46]. Indicatively, models trained through this paradigm have been employed for churn prediction in mobile games [655]. In that study semi-supervised learning is used to successfully predict churn on large-scale mobile games (using hundreds of millions of user-app interac-

tions) and it is tested on Samsung's *Game Launcher*. Importantly, the churn predictor both captures the dynamics between players and games and it is able to embed new players or new games not seen during training. In another more recent study, semi-supervised learning is used to identify players for sports analytics [1212]. Specifically, authors train a transformer network to recognize players through their jersey numbers from videos of the National Hockey League. The transformer architecture considers temporal sequences of player frames and predicts the probabilities of jersey numbers that are present in the frames. The semi-supervised training method used generates pseudo labels for jersey numbers present at any given frame and it seems to improve training speed and prediction accuracy. Beyond the aforementioned indicative studies we foresee a great potential of semi-supervised methods in the future, particularly considering the cost of obtaining labels via human feedback.

11.4.5 Self-Supervised Learning

The self-supervised learning (SSL) paradigm [49] can be used when labeled data, such as those obtained via human feedback, is not available. The SSL model instead uses solely the unlabeled data to generate its training signal. Self-supervised learning can be seen as both a form of supervised learning and a form of unsupervised learning. While the model predicts some aspects of player experience or behavior based on comparisons of observations (supervised), the model learns without labeled data (unsupervised); refer to Chap. 2 for more details. Independently of the SSL algorithm used, the outcome of the process is normally a **player embedding** of some sort (see Fig. 11.10) that may serve all player modeling roles: from the analysis of players and the support of the design process all the way to adaptive gaming.

The SSL paradigm is still in its infancy for modeling players. The paradigm however becomes more relevant and gains in popularity when a player model considers complex spatio-temporal observations such as in-game pixels. Early SSL studies have employed autoencoder variants to fuse in game actions and continuous physiological signals of players for the prediction of player experience states [715]. More recent SSL applications to player modeling include the use of contrastive learning methods to derive game state embeddings [1186, 1187] via the frames of the game, and user experience states [885] through facial expression recognition. In a recent study [886] on the *GameVibe* dataset [59], self-supervised video models were fine-tuned in the task of engagement modeling across several FPS game videos. Specifically, **few-shot learning** was used to fine-tune the pre-trained video models so that they predict changes of engagement in unseen FPS games of the dataset. The engagement model demonstrated high predictive capacity in unseen games despite it was only given a few samples of *high* versus *low* engagement (few-shot) from those games.

11.4.6 Unsupervised Learning

The aim of unsupervised learning (see Sect. 2.3.1) is to derive a model given a number of observations. There is no specified target output or reward signal and unlike self-supervised learning, the unsupervised learning model learns the inherent structure of the data. In unsupervised learning, the training signal is hidden internally in the interconnections among data attributes.

Independently of the method used, unsupervised learning derives a **player embedding** that can be used in support of all player modeling roles: for the analysis of players, for the design of better games and the runtime adaptation of games. This learning paradigm has mainly been employed to cluster players and mine associations between their attributes. While Chaps. 2 and 3 provide the general description of these unsupervised learning algorithms, in this section we focus on their specific application for modeling players.

Clustering

As discussed in Chap. 3, clustering is a form unsupervised learning aiming to find clusters in datasets so that data within a cluster is similar to each other and dissimilar to data in other clusters. When it comes to the analysis of user behavior in games, clustering offers a means for reducing the dimensionality of a dataset, thereby yielding a manageable number of critical features that represent user behavior. Relevant data for clustering in games include player behavior, navigation patterns, assets bought, items used, game genres played and so on. Clustering can be used to group players into archetypical playing patterns in an effort to evaluate how people play a particular game and as part of a user-oriented testing process [291]. Further, one of the key questions of user testing in games is *whether people play the game as intended*. Clustering can be utilized to derive a number of different playing or behavioral styles directly addressing this question. Arguably, the key challenge in successfully applying clustering in games is that the derived clusters should have an intelligible meaning with respect to the game in question. Thus, clusters should be clearly interpretable and labeled in a language that is meaningful to the involved stakeholders (designers, artists and managers) [291, 295]. In the case studies presented in Sect. 12.1 we meet the aforementioned challenges and demonstrate the use of clustering in the *Tomb Raider: Underworld* (EIDOS interactive 2008) game.

Frequent Pattern Mining

In Part 11 we defined frequent pattern mining as the set of problems and techniques related to finding patterns and structures in data. Patterns include sequences and itemsets. Both *frequent itemset mining* (e.g., Apriori [8]) and *frequent sequence mining* (e.g., GSP [1076]) are relevant and useful for player modeling. The key motivation for applying frequent pattern mining on game data is to find inherent and hidden regularities in the data. In that regard, key player modeling problems, such as player type identification and detection of player behavior patterns, can be viewed as frequent pattern mining problems. Frequent pattern mining can for example be used

to to discover what game content is often purchased together—e.g., players that buy X tend to buy Y too—or what are the subsequent actions after dying in a level—e.g., players that die often in the tutorial level pick up more health packs in level 1.

In an indicative study Chen et al. [207] utilized sequential information from 64 players of role-playing games (RPGs) and developed a sequential pattern mining model that uses gameplay action sequences as a predictor of a player's profile such as gender, game play expertise and personality traits. Their results show that compared to other player profile characteristics, game expertise is a dominant factor impacting in-game behavior in RPGs. In another relevant study, frequent sequence mining algorithms SPADE [1302] and GSP [1076] were used to derive sequential features of in-game behavior from 1560 players of a *Super Mario Bros* (Nintendo 1985) variant [1027]. Findings demonstrate that sequential features related to behavior (e.g., Mario moves right and then jumps) and game level content (e.g. a straight platform as a sequence of tiles) improve the predictive capacity of player experience models in that game.

11.5 Further Reading

If you are interested in the general principles, limitations and opportunities of human feedback—and associated literature—you are referred to Sect. 2.4. The relationship between psychology and its physiological manifestations has been studied extensively by now and the interested reader is referred to [27, 163, 927, 1284] among many. The reader is also referred to [955] for a recent survey of physiological sensors that can be used for modeling players. Another recent survey discusses the principles and tools of manual labeling for player experience modeling in more detail [1286].

Beyond input and output observations, all player modeling methods presented in this chapter are covered in detail in the background part of the book (see Part 1). Each method contains a set of indicative papers that the interested reader may wish to go through. For an introduction to supervised learning for modeling players' game data we recommend Chaps. 7 and 8 of [314]; Chap. 9 of that book puts a particular focus on neural networks as a predictive model. For clustering and frequent pattern mining methods of player data we recommend, respectively, Chaps. 6 and 11–12 of the same book. When it comes to the application of other (not as frequently used) methodologies we list a number of recent indicative papers that employ self-supervised learning [886], semi-supervised learning [655] and reinforcement learning [61, 62] for modeling aspects of players. Finally, interested readers in detailed surveys of player modeling are referred to [1054, 1286, 1288].

11.6 Summary

In this chapter we first detailed the various types of player information a model can consider and the various types of output it might represent. The input and the output of the model determine the method one can use to model players as covered extensively in this chapter. Specifically we explored the applications of ad-hoc authoring and supervised learning variants, but also reinforcement learning and unsupervised learning approaches. The use of a particular player modeling method is tightly coupled to the type and amount of data available and, importantly, it is depended on the particular use case for modeling players. The next and final chapter of this part is dedicated to representative use cases for player modeling as determined by the output type of the model.

Chapter 12
Player Modeling Exemplified

Abstract This chapter complements the more theoretical discussion of player modeling from the previous chapters by presenting a number of relevant and diverse examples of player modeling in some further detail. The examples presented in this chapter are drawn from purely academic studies as well as from AAA games. They range from learning embeddings to cluster players in *Tomb Raider: Underworld* (Eidos Interactive 2008), to predicting toxic player behavior in *For Honor* (Ubisoft 2017) to modeling player experience from pixel inputs in multiple games. The intent is to give a variety of examples that illustrate some of the rich variety of player modeling approaches that are available.

The first chapter of this part defined player modeling as a field and discussed its multifaceted role in games. Our high level overview of player modeling identified three main components that were then detailed in Chap. 11: the model's inputs (observations), the model's outputs and the methods we can use to draw the relationship between the two. As already outlined throughout this part, modeling of users in games can be classified into three main tasks: deriving an *embedding* of anything we observe from their play, modeling their *behavior*, or modeling their *experience*. The present chapter, which is the last chapter of this part, takes us through a number of concrete player modeling use cases across these three distinct tasks.

The examples we present shed some light on practical uses of player models but they are far from exhausting all the possible roles and use cases for player modeling. While this book could contain far more use cases, it is long enough as it is. Therefore, we opt to select seven representative ones based on a number of criteria. Those criteria include, but are not limited to, our familiarity with the use case, the impact of the work, the thoroughness of the analysis, and the usefulness of the case study from a pedagogical perspective. As seen in Fig. 12.1, the examples covered in this chapter vary in terms of AI methods and representations used, model observations, roles of player modeling and game genres considered. The games involved in the presented studies come both from research prototypes and commercial-standard games.

The structure of the chapter follows the three player modeling output tasks. Starting with Sect. 12.1 we detail a clustering use case for deriving player *embeddings* in a 3D platform-puzzle game. Then in Sect. 12.2 we focus on two uses cases of player

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_12

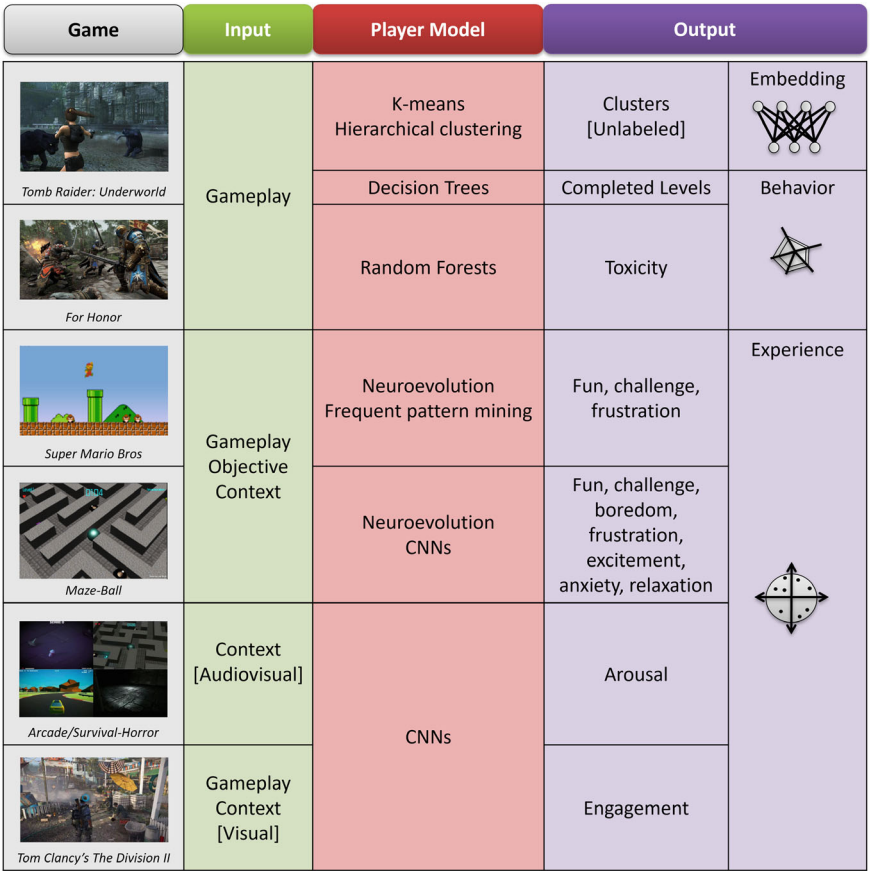


Fig. 12.1 The seven player modeling examples covered in this chapter. Each example (table row) lists (from left to right) the game(s) examined, the input types considered by the model, the player model per se, the output of the model and the output type

behavior prediction: completed levels (in a platform-puzzle game) and toxicity (in a MOBA game). Finally, in Sect. 12.3 we analyze four use cases of player *experience* models in platformer, arcade and survival-horror, and first-person shooter games.

12.1 Player Embeddings

In this section, we exemplify the process of deriving player embeddings via a representative use case. Our example is based on a series of studies on player modeling by Drachen et al. [291] and Mahlmann et al. [686] in the *Tomb Raider: Underworld*

(EIDOS interactive 2008) game. The analysis we present here focuses on the *clustering* of players [291]. Section 12.2 instead presents a follow up study on player behavior *prediction* in the same game [686].

Clustering Players in *Tomb Raider: Underworld*

Tomb Raider: Underworld (TRU) is a third-person perspective, advanced platform-puzzle game, where the player has to combine strategic thinking in planning the 3D-movements of Lara Croft (the game's player character) and problem solving in order to go through a series of puzzles and navigate through a number of levels (see Fig. 12.2).

The *dataset* used for this study includes entries from 25,240 players; 1365 of those that completed the game were selected and used for the analysis presented here. Note that TRU consists of seven main levels plus a tutorial level. Six features of gameplay behavior were extracted from the data and are as follows: number of deaths by opponents, number of deaths by the environment (e.g., fire, traps, etc.), number of deaths by falling (e.g., from ledges), total number of deaths, game completion time, and the times help was requested. All six features were calculated on the basis of completed TRU games. The selection of these particular features was based on the core game design of the TRU game and their potential impact on the process of distinguishing among dissimilar patterns of play.

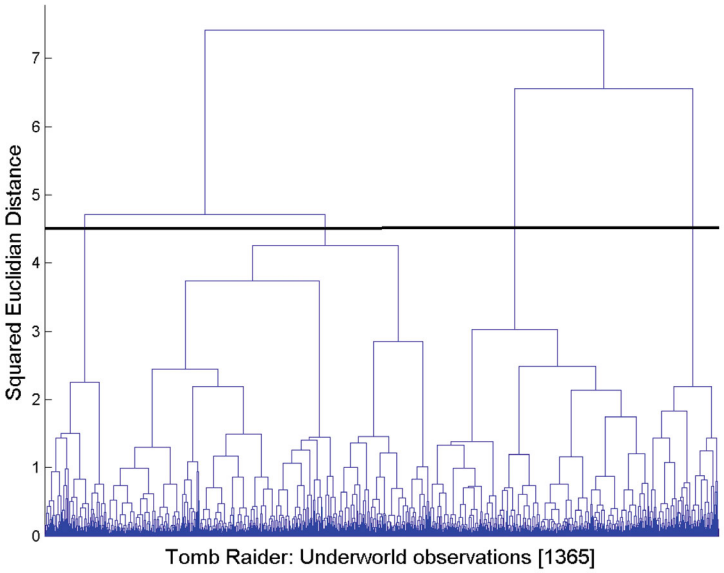
Fig. 12.2 A screenshot from the *Tomb Raider: Underworld* (EIDOS interactive 2008) game featuring the player character, Lara Croft. The game is used as one of the player embedding and behavior modeling case studies presented in this book. Image obtained from [291] with permission

Three different **clustering** techniques were applied to the task of identifying the number of meaningful and interpretable clusters of players in the data: k-means, hierarchical clustering and self-organizing maps. While the first two have been covered in Chap. 3 we will briefly outline the third method here. A **self-organizing map** (SOM) [577] creates and iteratively adjusts a low-dimensional projection of the input space via vector quantization. In particular, a large SOM called an emergent self-organizing map [1194] was used in conjunction with reliable visualization techniques to help us identify clusters. A SOM consists of neurons organized in a low-dimensional grid. Each neuron in the grid (map) is connected to the input vector through a connection weight vector. In addition to the input vector, the neurons are connected to neighbor neurons of the map through neighborhood interconnections which generate the structure of the map. SOM training can be viewed as a vector quantization algorithm which resembles the k-means algorithm. What differentiates SOM, however, is the update of the topological neighbors of the best-matching neuron. A best-matching neuron is a neuron for which there exists at least one input vector for which the Euclidean distance to the weight vector of this neuron is minimal. As a result, the whole neuron neighborhood is stretched towards the presented input vector. The outcome of SOM training is that neighboring neurons have similar weight vectors which can be used for projecting the input data to the two-dimensional space and thereby clustering a set of data through observation on a 2D plane. For a more detailed description of SOMs, the reader is referred to [577].

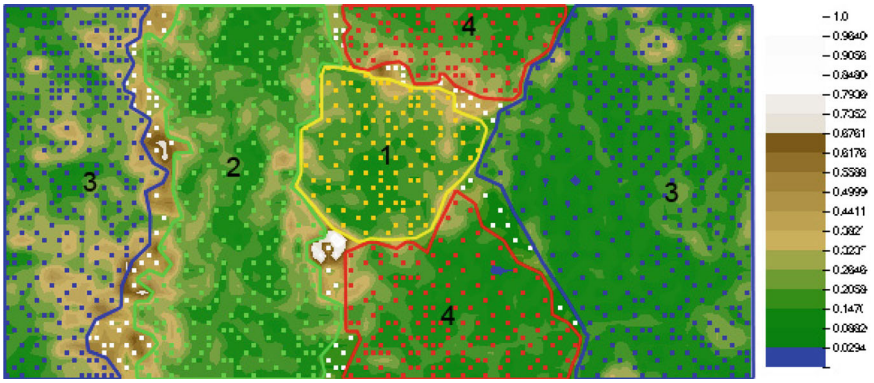
To get some insight into the possible number of clusters existent in the data, **k-means** was applied for all k values less than or equal to 20. The sum of the Euclidean distances between each player instance and its corresponding cluster centroid (i.e., quantization error) is calculated for all 20 trials of k-means. The analysis reveals that the percent decrease of the mean quantization error due to the increase of k is notably high when $k = 3$ and $k = 4$. For $k = 3$ and $k = 4$ this value equals 19 and 13% respectively while it lies between 7 and 2% for $k > 4$. Thus, the k-means clustering analysis provides the first indication of the existence of three or four main player behavioral clusters within the data.

As an alternative approach to k-means, **hierarchical clustering** was also applied to the dataset. This approach seeks to build a hierarchy of clusters existent in the data. The Ward's clustering method [1242] is used to specify the clusters in the data by which the squared Euclidean distance is used as a measure of dissimilarity between data vector pairs. The resulting **dendrogram** is depicted in Fig. 12.3a. As noted in Chap. 3 a dendrogram is a treelike diagram that illustrates the merging of data into clusters as a result of hierarchical clustering. It consists of many U-shaped lines connecting the clusters. The height of each U represents the squared Euclidean distance between the two clusters being connected. Depending on where the data analyst sets the squared Euclidean distance threshold a dissimilar number of clusters can be observed.

Both k-means and hierarchical clustering already demonstrate that the 1365 players can be clustered in a low number of different player types. K-means indicates there are for three or four clusters while the Ward's dendrogram reveals the existence of two populated and two smaller clusters, respectively, in the middle and at the



(a) Dendrogram of TRU data using Ward hierarchical clustering. A squared Euclidean distance of 4.5 (illustrated with a horizontal black line) reveals four clusters. Image adopted from [291].



(b) U-matrix visualization of a self-organizing map depicting the four player clusters identified in a population of 1,365 TRU players (shown as small colored squares). Different square colors depict different player clusters. Valleys represent clusters whereas mountains represent cluster borders. Image adopted from [291].

Fig. 12.3 Detecting player types from TRU data using hierarchical (**a**) and SOM (**b**) clustering methods

edges of the tree. Applying SOM, as the third alternative approach, allows us to cluster the TRU data by observation on a two-dimensional plane. The U-matrix depicted in Fig. 12.3b is a visualization of the local distance structure in the data placed onto

a two-dimensional map. The average distance value between each neuron's weight vector and the weight vectors of its immediate neighbors corresponds to the height of that neuron in the U-matrix (positioned at the map coordinates of the neuron). Thus, **U-matrix** values are large in areas where no or few data points reside, creating mountain ranges for cluster boundaries. On the other hand, visualized valleys indicate clusters of data as small U-matrix values are observed in areas where the data space distances of neurons are small.

The SOM analysis reveals four main classes of behavior (player types) as depicted in Fig. 12.3b. The different colors of the U-matrix correspond to the four different clusters of players. In particular, cluster 1 (9% of the TRU players) corresponds to players that die very few times, their death is caused mainly by the environment, they do not request help from the game frequently and they complete the game very quickly. Given such game skills these players were labeled as *Veterans*. Cluster 2 (22%) corresponds to players that die quite often (mainly due to falling), they take a long time to complete the game—indicating a slow-moving, careful style of play—and prefer to solve most puzzles in the game by themselves. Players of this cluster were labeled as *Solvers*, because they excel particularly at solving the puzzles of the game. Players of cluster 3 form the largest group of TRU players (46%) and are labeled as *Pacifists* as they die primarily from active opponents. Finally, the group of players corresponding to cluster 4 (17% of TRU players), namely the *Runners*, is characterized by low completion times and frequent deaths by opponents and the environment.

The results showcase how clustering of player behavior can be useful to evaluate game designs. Specifically, TRU players seem to not merely follow a specific strategy to complete the game but rather they fully explore the affordances provided by the game in dissimilar ways. The findings are directly applicable to TRU game design as clustering provides answers to the critical question of *whether people play the game as intended*. The main limitation of clustering, however, is that the derived clusters are not intuitively interpretable and that clusters need to be represented into meaningful behavioral patterns to be useful for game design. Collaborations between the data analyst and the designers of the game—as was performed in this study—is essential for meaningful interpretations of the derived clusters. The benefit of such a collaboration is both the enhancement of game design features and the effective **phenomenological debugging** of the game [291]. In other words, we make sure both that features of the game are not underused or misused and that the playing experience and the game balance are debugged.

12.2 Player Behavior

There are numerous downstream tasks a model can learn to predict within a player's behavior. In this section we outline two such representative tasks. The first one is the follow up study to the clustering use case in the *Tomb Raider: Underworld* (EIDOS interactive 2008) game. Instead of clustering TRU players, however, in this section

we present predictive models of their behavior [686]. The second example is the use of machine learning models for predicting toxic playing behavior in the multiplayer online battle arena game *For Honor* (Ubisoft 2017).

Predicting Player Behavior in *Tomb Raider: Underworld*

Building on the same set of TRU player data introduced in Sect. 12.1, a second study examined the possibilities of predicting particular aspects of playing behavior via supervised learning [686]. An aspect of player behavior that is particularly important for game design is to predict *when a player will stop playing*. As one of the perennial challenges of game design is to ensure that as many different types of players are facilitated in the design, being able to predict when players will stop playing a game is of interest because it assists with locating potentially problematic aspects of game design. Such information helps toward the redesign of a game's monetization strategy for maximizing user retention.

Data was drawn from the Square Enix Europe Metrics Suite and was collected during a two month period (1st December 2008–31st January 2009), providing records from approximately 203,000 players. For the player behavior prediction task it was decided to extract a subsample of 10,000 players. This provides a large enough and representative dataset for the aims of the study, while at the same time is manageable in terms of computational effort. A careful data preprocessing approach yielded 6430 players that were considered for the prediction task—these players had completed the first level of the game.

As in the TRU cluster analysis the features extracted from the data relate to the core mechanics of the game. In addition to the six features investigated in the clustering study of TRU, the extracted features for this study include the number of times the adrenaline feature of the game was used, the number of rewards collected, the number of treasures found, and the number of times the player changes settings in the game (including player ammunition, enemy hit points, player hit points, and recovery time when performing platform jumps). Further details about these features can be found in [686]. To test the possibility of predicting the TRU level the player completed last a number of **classification** algorithms are tested including logistic regression, multilayer perceptron backpropagation, variants of decision trees, Bayesian networks, and support vector machines. In this section we only outline the most interesting results from those reported in [686]. For all tested algorithms, the reported classification prediction accuracy was achieved through 10-fold cross validation.

Most of the tested algorithms had similar levels of performance, and were able to predict when a player will stop playing substantially better than the baseline. In particular, considering only gameplay of the first level of TRU (Level 1) classification algorithms reach an accuracy between 45 and 48%, which is substantially higher than the baseline performance (40%). When using additional features from the second level of the game (Level 2), the predictions are much more accurate—between 50 and 78%—compared to the baseline (45%). In particular, decision trees and logistic regression manage to reach accuracies of almost 78% in predicting on what level a player will stop playing. The difference in the predictive strength of using data from

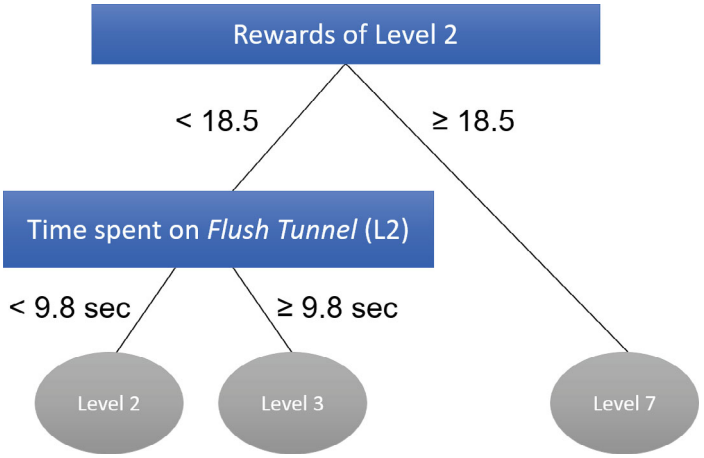


Fig. 12.4 A decision tree trained by the ID3 algorithm [904] to predict when TRU players will stop playing the game. The leaves of the tree (ovals) indicate the number of the level (2, 3 or 7) the player is expected to complete. Note that the TRU game has seven levels in total. The tree is constrained to tree depth 2 and achieves a classification accuracy of 76.7%

both the first and the second level of TRU as compared to only data from the first level is partly due to increased amount of features used in the latter case.

Beyond accuracy an important feature of machine learning algorithms is their transparency and their expressiveness. The models are more useful to a data analyst and a game designer if they can be expressed in a form which is easy to visualize and comprehend. **Decision trees**—of the form constructed by the ID3 algorithm [904] and its many derivatives—are excellent from this perspective, especially if pruned to a small size. For instance, the extremely small decision tree depicted in Fig. 12.4 is constrained to tree depth 2 and was derived on the set of players who completed both levels 1 and 2 with a classification accuracy of 77%. The predictive capacity of the decision tree illustrated in Fig. 12.4 is impressive given how extremely simple it is. The fact that we can predict the final played level—with a high accuracy—based only on the amount of time spent in the room named *Flush Tunnel* of Level 2 and the total rewards collected in Level 2 is very appealing for game design. What this decision tree indicates is that the amount of time players spend within a given area early in the game and how well they perform are important for determining if they continue playing the game. Time spent on a task or in an area of the game can indeed be indicative of challenges with progressing through the game, which can result in a frustrating experience.

Predicting Toxicity in *For Honor*

In this second example of player behavior modeling we examine the phenomenon of **toxic** behavior in multiplayer games. We thus ask the question whether it it possible to detect player toxicity just by observing in-game behavior. Further we will

Fig. 12.5 A screenshot from the *For Honor* (Ubisoft 2017) game featuring a typical fight between a samurai and knights. The game is used as one of the player behavior modeling case studies presented in this book. Image obtained from [178] with author permission

investigate the behavioral factors that might be able to assist machine learning methods to discover the unknown relationship between gameplay and toxic behavior. In the study presented in [178] and outlined here we examine whether it is possible to predict toxicity in the multiplayer online battle arena (MOBA) game *For Honor* (Ubisoft 2017). The study attempts to predict a player's toxicity (model output) by observing in-game behavior (model input) that have been labeled as toxic by Ubisoft community managers.

For Honor (Ubisoft 2017) is an action MOBA that takes place in a medieval fantasy setting (see Fig. 12.5). A player can select and control a hero from three different factions (representing knights, vikings, and samurai) with dissimilar skill-sets and class qualities. The game combines aspects of fighting games, third-person action and mass combat allowing for multiple action types. Players can walk and sprint, climb, roll, and attack opponents in many ways.

MOBA games like *For Honor* (Ubisoft 2017) can be "notoriously toxic" [178], most likely because they gather a number of prominent features that might contribute to a toxic environment. Those features include competitiveness, anonymity, counterfactual thinking (i.e., players that tend to blame others for unwanted events given the lack of full in-game observation), and negative social culture [596]. *For Honor* (Ubisoft 2017) is an ideal game to study and predict player toxicity as it both features all prominent factors and it has been specifically characterized for the toxicity of its player base [178].

Relevant to the behavior we attempt to predict, players of *For Honor* (Ubisoft 2017) might receive a *sanction* if they breach any of the clauses contained in the game's Code of Conduct including, but limited to, hate speech, use of offensive or

explicit names or images, scamming and phishing, and cheating. The game features an online system through which players can flag and report other players for any type of problematic behavior. The community managers of the game will then make a final decision based on the severity and the frequency of the events. The decision can be one of the following: (1) a warning, (2) a temporary ban from the game or (3) a permanent ban.

By 2021, over 20 million players had played *For Honor* (Ubisoft 2017). Surprisingly enough, only around 0.2% of those players ever received a sanction label; that is approximately 45 thousand players. Out of those 45k players the study analyses the data of 1793 players whose behavioral observations are recorded and available. The input of the player toxicity model considers behavioral features that are selected through a hybrid process. First, a team of domain experts pick critical behavioral features (top-down) and then a statistical data analysis (bottom-up) extracts additional features. The final set of 36 features contains aspects related to the play mode (e.g., playing against AI or practice), disengagement (e.g., average playtime spent running), away from keyboard (AFK) (e.g., ratio of rounds the player is labeled as AFK), performance (e.g., win rate), and chat actions (e.g., messages per minute).

To validate that toxic players are behaviorally distinguishable we can employ any supervised learning method that would predict the sanctioned players versus the unsanctioned ones via classification. Given the small size of this dataset one may employ a traditional (shallow) supervised learning method like SVMs or random forests. Using random forests, findings in [178] suggest that (a) it is possible to behaviorally distinguish sanctioned from unsanctioned players (b) predict the sanction severity (warned vs. banned) and even (c) predict the sanction type (offensive behavior vs. unfair advantage). Specifically, all models predict toxicity, its severity, and its type, with an accuracy of at least 82%, on average, over unseen players. Player modeling methods as the ones presented here are scalable and generalizable to other games beyond *For Honor* (Ubisoft 2017). Similar methodologies are also directly applicable to the detection of any labeled player behavioral pattern beyond toxicity.

12.3 Player Experience

The modeling of player experience involves learning a set of target outputs that approximate the experience (as opposed to the behavior) of the player. By definition, what is being modeled (experience) is of *subjective nature* and the modeling therefore requires target outputs that somehow approximate the ground truth of experience. A model of player experience predicts some aspect of the experience a player would have in some game situation, and learning such models requires labeled experiences. As mentioned, there are many ways this can be done, with approaches to player experience modeling varying regarding the inputs (from what the experience is predicted, e.g., physiology, level design parameters, playing style or game speed), the outputs (what sort of experience is predicted, e.g., fun, frustration, attention or immersion) and the modeling methodology.

In this section we will outline a few representative examples of player experience modeling. To best cover the material (methods, algorithms, uses of models) we rely on studies which have been thoroughly examined in the literature. We start with a variant of the popular *Super Mario Bros* (Nintendo 1985) game, a 3D prey-predator game named *Maze-Ball*, and a set of arcade-style games of various genres (horror adventure, maze-puzzle, racing, shooter) and we conclude this section with a player experience modeling study on a commercial-standard game: *Tom Clancy's the Division II* (Ubisoft 2020).

Modeling Player Experience in *Super Mario Bros*

Our first example builds upon the work of Pedersen et al. [858, 859] who modified an open-source clone of the classic platform game *Super Mario Bros* (Nintendo 1985) to allow for personalized level generation. This work is historically important because it set the foundations for the development of the **experience-driven procedural content generation** framework [1289] which constitutes a core research trend within procedural content generation (see also Part III).

The game used in this example is a modified version of Markus Persson's *Infinite Mario Bros* which is a public domain clone of Nintendo's classic platform game *Super Mario Bros* (Nintendo 1985). All experiments reported in this example rely on a model-free approach for the modeling of player experience. Models of player experience are based on both the level played (game context) and the player's playing style. While playing, the game recorded a number of behavioral metrics of the players, such as the frequency of jumping, running and shooting, that are taken into consideration for modeling player experience. Further, in a follow-up experiment [1018], the videos of players playing the game were also recorded and used to extract a number of useful visual cues such as the average head movement during play. The output (ground truth of experience) for all experiments is provided as first-person, rank-based reports obtained via **crowdsourcing**. Data was crowdsourced from hundreds of players, who played pairs of levels with different level parameters (e.g., dissimilar numbers, sizes and placements of gaps) and were asked to rank which of two levels best induced a number of player states. Across the several studies of player experience modeling for this variant of *Super Mario Bros* (Nintendo 1985) collectively players have been asked to annotate fun, engagement, challenge, frustration, predictability, anxiety, and boredom. These are the target outputs the player model needs to predict based on the input parameters discussed earlier.

Given the rank-based nature of the annotations the use of **preference learning** is necessary for the construction of the player model. The collected data is used to train artificial neural networks that predict the players' experiential states, given a player's state and a particular game context, using evolutionary preference learning. In **neuroevolutionary preference learning** [1267], a genetic algorithm evolves an artificial neural network so that its output matches the pairwise preferences in the data set. The input of the artificial neural network is a set of features that have been extracted from the data set. The genetic algorithm implemented uses a fitness function that measures the difference between the reported preferences and the relative magnitude

of the model output. Neuroevolutionary preference learning has been used broadly in the player modeling literature; the interested reader may refer to the following studies (among many): [712, 858, 859, 884, 1018, 1267, 1277].

The crowdsourcing experiment of Pedersen et al. [858, 859] resulted in data (gameplay and subjective reports of experience) from 181 players. The best predictor of reported *fun* reached an accuracy of around 70% on unseen subjective reports of fun. Indicatively, the input of the neural network *fun*-model consists of three features: the time Mario spends moving left, the number of opponents Mario kills from stomping, and the percentage of level played in the left direction. All three playing features appear to contribute positively for the prediction of reported *fun* in the game. The best-performing model for *challenge* prediction yields an accuracy of approximately 78%, whereas the best predictor for *frustration* reaches an accuracy of 89%. It is indeed an impressive finding that a player experience model can predict (with near-certainty) whether the player is frustrated by the current level by merely calculating the time Mario spent standing still or spent on its last life, the jump difficulty, and the deaths Mario had from falling in gaps.

The general findings of Pedersen et al. [858] suggest that good predictors for experience can be found if a preference learning approach is applied on crowdsourced reports of experience and gameplay data. The prediction accuracies, however, depend on the complexity of the reported state—arguably *fun* is a much more complicated and fuzzier notion to report compared to *challenge* or *frustration*.

In a follow up study by Pedersen et al. [859] the additional player states of *predictability*, *anxiety* and *boredom* were predicted with accuracies of approximately 77, 70 and 61%, respectively. The same player experience methodology was tested on an even larger scale, soliciting data from a total number of 780 players of the game [1027]. **Frequent pattern mining** algorithms were applied to the data to derive frequent sequences of player actions. Using sequential features of gameplay the models reach accuracies of up to 84% for *engagement*, 83% for *frustration* and 79% for *challenge*.

In addition to the behavioral characteristics the **visual cues** of the player can be taken into account as objective input to the player model. Such visual cues may include facial expressions such as a detected smile, but also head pose features such as distance from camera. In [1018] visual features were extracted from videos of 58 players, both throughout whole game sessions and during small periods of critical events such as when a player completes a level or when the player loses a life (see Figs. 12.6 and 12.7). The visual cues enhance the quality of the information we have about a player's affective state which, in turn, allows us to better approximate player experience. Specifically, fusing the gameplay and the visual reaction features as inputs to the artificial neural network studies have reported accuracies of up to 84, 86 and 84% for predicting reported *engagement*, *frustration* and *challenge*, respectively. The key findings of [1018] suggest that players' visual reactions can provide a rich source of information for modeling experience preferences and lead to more accurate models of player experience.

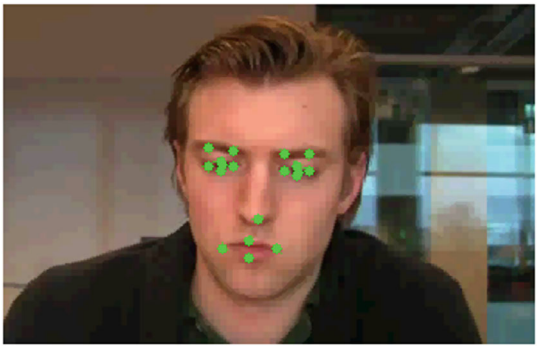
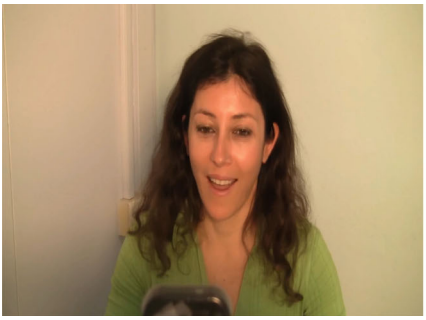
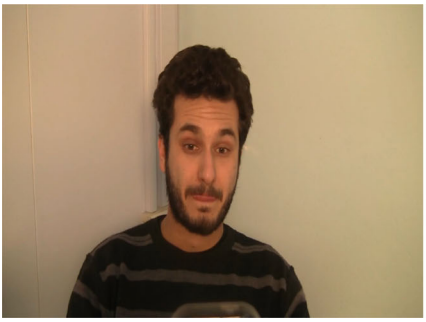


Fig. 12.6 Facial feature tracking for head movement. Image adopted from [1018]

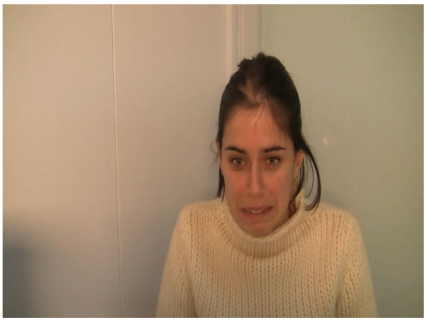


(a) Winning

(b) Losing



(c) Experiencing *challenge*



(d) Experiencing *challenge*

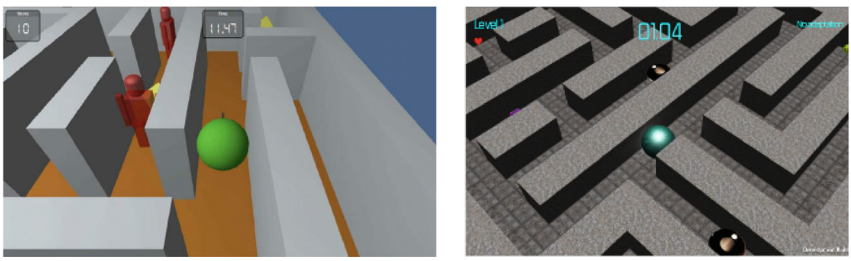
Fig. 12.7 Examples of facial expressions of *Super Mario Bros* (Nintendo 1985) players for different game states. All images are retrieved from the Platformer Experience Dataset [545]

Modeling Player Experience in *Maze-Ball*

Our second example for modeling player experience builds largely upon the extensive studies of Martínez et al. [710, 714, 715] who analyzed player experience using a simple 3D prey-predator game named *Maze-Ball* towards achieving affective-driven camera control in games. While the game is rather simple, the work on *Maze-Ball* offers a thorough analysis of player experience via a set of sophisticated techniques for capturing the **psychophysiological** patterns of players including preference learning, frequent pattern mining and deep CNNs.

Maze-Ball is a three-dimensional prey-predator game (see Fig. 12.8); similar to a 3D version of *Pac-Man* (Namco,1981). The player (prey) controls a ball which moves inside a maze hunted by 10 opponents (predators) moving around the maze. The goal of the player is to maximize her score by gathering as many tokens, scattered in the maze, as possible while avoiding being touched by the opponents in a predefined time window of 90 s. A detailed description of *Maze-Ball* gameplay can be found in [1285].

Gameplay attributes and physiological signals (skin conductance and heart rate variability) were acquired from 36 players of *Maze-Ball*. Each subject played a predefined set of eight games for 90 s, thus the total number of game sessions available is 288. Gameplay and physiological signals define the input of the player experience model. To obtain the ground truth of experience players self-reported their preference about game pairs they played using a rank-based questionnaire, in particular a 4-alternative forced choice (4-AFC) protocol [1285]. They were asked to rank the two games of each game pair with respect to *fun*, *challenge*, *boredom*, *frustration*, *excitement*, *anxiety* and *relaxation*. These annotations are the target outputs the player model will attempt to predict based on the input parameters discussed earlier. Several features were extracted from the gameplay and physiological signals obtained. These included features related to kills and deaths in the game as well as features associated with the coverage of the level. For extracting features from the physiological signals the study considered their average, standard deviation, and maximum and minimum values.



(a) Maze-Ball (b) Space-Maze

Fig. 12.8 Early Maze-Ball prototype (**a**) and a polished variant of the game (**b**) that features real-time camera. adaptation [575]

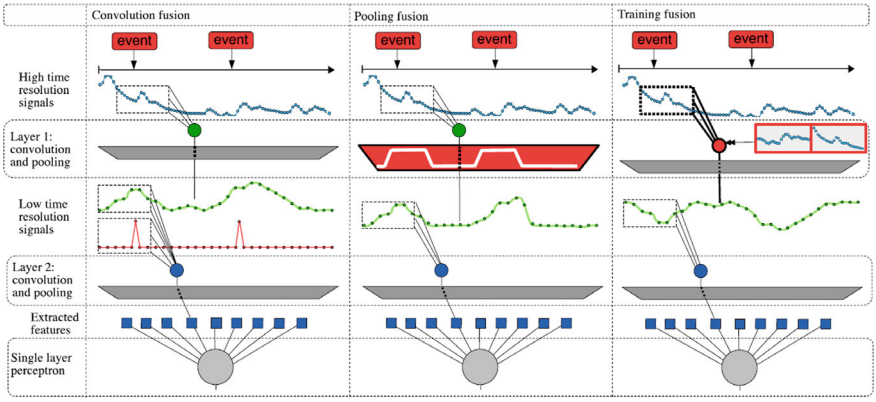


Fig. 12.9 Three approaches to deep multimodal fusion via convolutional neural networks. Gameplay events are fused with skin conductance in this example. The networks present two layers with one neuron each. The first convolutional layer receives as input a continuous signal at a high time resolution, which is further reduced by a pooling layer. The resulting signal (feature map) presents a lower time resolution. The second convolutional layer can combine this feature map with additional modalities at the same low resolution. In the convolution fusion network (left figure), the two events are introduced at this level as a pulse signal. In the pooling fusion network (middle figure), the events are introduced as part of the first pooling layer, resulting in a filtered feature map. Finally, in the training fusion network (right figure), the events affect the training process of the first convolutional layer, leading to an alternative feature map. Image adopted from [715]

As in the example with the game variant of *Super Mario Bros* (Nintendo 1985) the rank-based nature of the annotations requires the use of **preference learning** for the construction of the player experience model. Thus, the collected data is used to train neural networks that predict the player states, given a player's gameplay behavior and its physiological manifestations, using evolutionary preference learning. The architecture of the neural network can be either shallow (e.g., a small multi-layered perceptron) or deep (e.g., a CNN [710, 715]). Figure 12.9 shows the different ways information from gameplay and physiology can been fused on a deep **convolutional neural network** which is trained via preference learning to predict player experience in any game experience dataset that contains discrete in-game events and continuous signals (e.g., the player's skin conductance).

Predictors of the player experience states can reach accuracies that vary from 72% for *challenge* up to 86% for *frustration* using a shallow multi-layer perceptron player model [1285]. Significant improvements are observed in those accuracies when the input space of the model is augmented with frequent sequences of in-game and physiological events (i.e., fusion on the input space). As in [1027], Martínez et al. used the generalized sequential patterns algorithm to extract those frequent patterns that were used as inputs of the player model [714]. Further accuracy improvements can be observed when in game events are fused with physiological signals on deep convolutional neural networks [715]. Using various **fusion** methods (see Fig. 12.9) accuracies of predicting player experience may surpass 82% for all player experience

states considered. Further information about the results obtained in the *Maze-Ball* game can be found in the following studies: [710, 714–716, 1285].

Modeling Player Arousal from Pixels

The third set of studies we present in this section puts an emphasis on modeling player arousal. A series of recent studies [692, 693] attempted to reframe the traditional player-centric (or player-specific) needs of player experience models. One may argue that any video of player-game interaction can capture a players' behavior and therefore the audiovisual information contained in such in-game footage videos may hold information about the player's experience. The hypothesis is that a player's emotional state (such as arousal) can be predicted relying solely on the audiovisual information of in-game footage. This novel player-agnostic experience modeling approach is inspired by the general-purpose representations introduced for general gameplying [773]. Put in simple words, the working hypothesis is whether the gameplay screen can act as the mirror of a player's experiences.

Any player experience modeling task that considers the audio and video content of a player's interaction with the game is bimodal by nature. One can thus employ a two-stream model (e.g., a deep neural network) that considers audiovisual gameplay footage information and predicts the player's annotated arousal. For instance, a CNN may process visual information as pixels of video frame sequences and a fully connected MLP may process the game audio information of the considered sequence of video frames. The audiovisual information can then be fused via a fully connected network that predicts player experience.

Results reported in [693] test the methodology across four dissimilar games (see Fig. 12.10). The dataset contains 155 videos (in between 28 and 50 videos for each game) that are annotated by the players themselves (i.e., in a first-person manner) in terms of arousal using the RankTrace [664] continuous annotation tool of PAGAN [742]. Findings suggest that the CNN and MLP fused models may predict arousal for three of the games appearing in Fig. 12.10 with a classification accuracy that reaches between 82 and 83% on average via the demanding leave-one-video-out cross-validation scheme. It thus seems that we can build highly accurate models of player experience using general-purpose representations that rely solely on audiovisual information of the player-game interaction.

The low performance of the player experience model (around 60% accuracy) on the survival horror game in Fig. 12.10—suggests that there are certain game types whose video and audio encodings are more appropriate for affect prediction than others. It appears that arcade games with fast-paced interactions, top-down camera perspectives and rather distinct forms, colors and sounds help the model predict player experience from the audiovisual observations alone. In contrast, the survival horror game *Sonancia* [661] has none of the above design patterns; player actions are passive and long-term (e.g., choose a room to go into), the game background is dark, and the background music is dissonant compared to the other three games.

Figure 12.10 depicts the activation maps [1012] for a sample frame of each game. A striking, yet unsurprising, observation is that the vision-based player experience

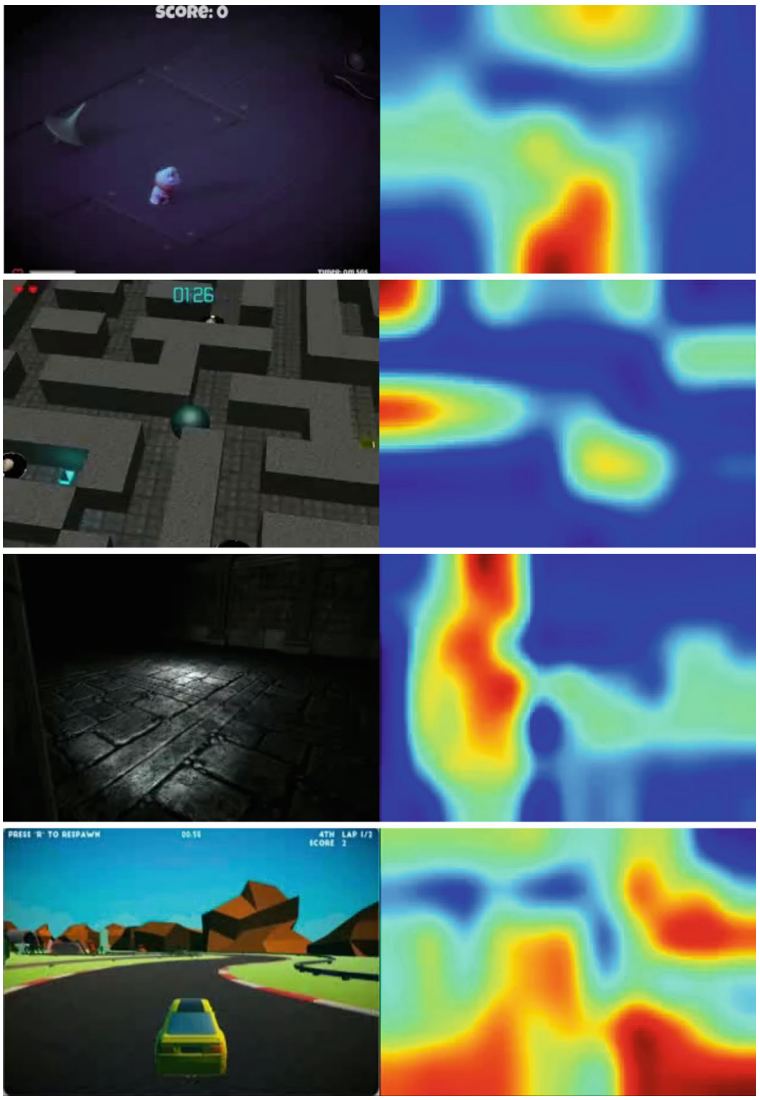


Fig. 12.10 The four games used as testbeds for predicting player arousal from audiovisual observations of the game. The games belong to different genres, with different mechanics, camera perspective, pace, visual and audio design. Specifically, *Survival Shooter* (top) is a fast-paced shooter game that requires accurate aiming and constant movement; *Maze Ball* (second from top) is the slow-paced physics game covered in earlier sections; *Sonancia* (second from bottom) is a survival horror game; and *Solid Rally* (bottom) is a fast-paced racing game. Indicative activation maps are generated via the Gradient-weighted Class Activation Mapping method [1012] and presented to the right of each game screenshot; red and blue areas indicate higher and lower, respectively, input activations for predicting the output. Image adapted from [693] with permission

model (CNN) considers and combines both gameplay aspects and some aspects of the heads-up display—such as the score, the health-bar and the timer—for its prediction. For the game appearing at the top left of Fig. 12.10, for instance, the score located at the top center of the screen increases each time the player kills an opponent or collects an item. The player model, thereby, associates the score (and time) progression with gameplay patterns in order to predict the annotated experience with higher accuracy.

Long-Term Player Engagement Modeling in *Tom Clancy's The Division II*

The player modeling studies presented so far rely on short in-game experiences; as short as seconds or a few minutes. We know, therefore, that it is possible to predict player experience quite accurately when a model considers such time-limited annotated games or video footage of games. What happens however when a model is tasked to predict player experience for extended periods of time, say hours?

To examine the degree to which one can build reliable long-term computational models of experience it is imperative to have access to corpora containing in-game stimuli and corresponding experience annotations over extended time periods. The first study to address this challenge was a collaborative effort between *Massive Entertainment* (A Ubisoft Studio) and the *Institute of Digital Games* in the online action roleplaying third-person shooter game *Tom Clancy's the Division II* (Ubisoft 2020). Players of this game must scavenge for resources to survive across over 30 missions available in the game and they must work their way through them, alone or in a group with other players.

The study in [888] introduces a game experience dataset from 20 participants who played one hour of *Tom Clancy's the Division II* (Ubisoft 2020) and annotated their own gameplay videos in terms of engagement via the PAGAN annotation tool [742]; see Fig. 12.11. Particularly, for the purposes of that dataset, participants played the first mission of the game (*Dawn's Early Light*), which offers a good balance between combat (intense periods) and in-game exploration (less intense periods).

A number of CNN architectures is trained on nearly 20 h of game footage and gamepad action data to predict long-term annotated engagement. Similarly to the pixel-based arousal modeling study presented earlier the assumption here is that rich and sufficient engagement information is existent (and interwoven) within the pixels and the actions of the gameplay. Supposedly a predictive model should be able to capture it effectively. Long-term player engagement can be represented as a binary classification task (i.e., high vs. low engagement) over time. To capture the long-term temporal aspect of engagement, a number of *time-conditioning* strategies can be adopted so that time of play can be considered by (or fused in) the player engagement model. Results reported in [888] suggest that CNN models that are time conditioned can predict high vs. low engagement states of 1h long game sessions with high accuracies of up to 72% on average (88% at best). Such findings validate the hypothesis that long-term player engagement can be predicted efficiently solely from pixels and gamepad actions in a game like *Tom Clancy's the Division II* (Ubisoft 2020).

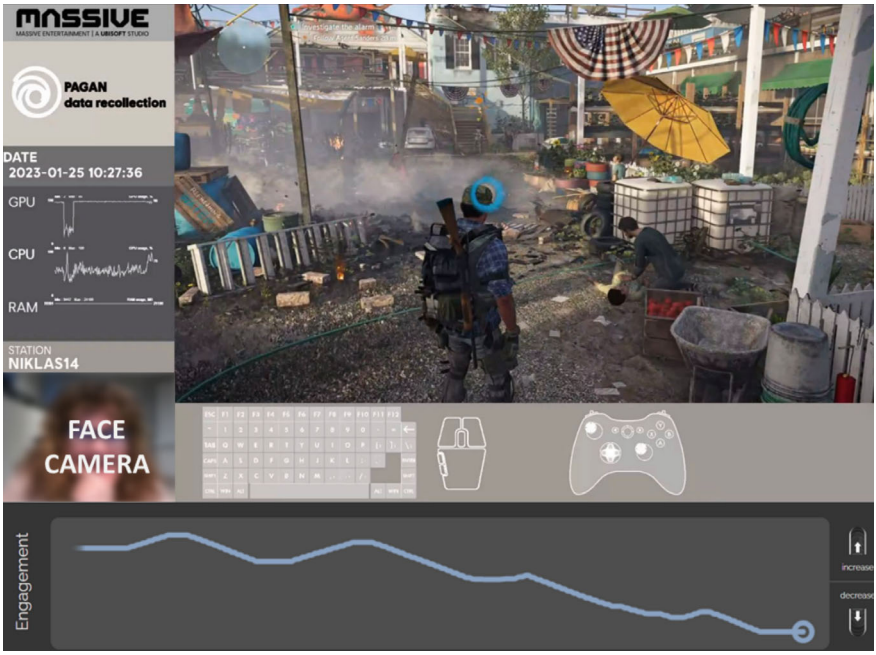


Fig. 12.11 Player engagement data collection in *Tom Clancy's the Division II* (Ubisoft 2020). Each participating annotator views their in-game video, timestamps for data synchronization, data regarding compute resources and their workstation, their gaze position (blue circle), their facial expression (blurred out here for privacy purposes), and the live input on their gamepad. The bottom of the layout visualizes the participant's engagement trace using PAGAN [742]. Image reproduced from [888] with permission

12.4 Further Reading

Readers interested in extensive examples of player behavior modeling are referred to the recent book dedicated to game data science [314]. Instead, readers interested in player experience modeling are referred to a recent and thorough review of the *affective game computing* field as a whole [1286].

12.5 Exercises

In this section we propose a set of exercises covering various aspects of player modeling. For that purpose, we outline only a few indicative datasets that can be used directly for analysis at the time of writing. Please note, however, that the book's website will remain up to date with more datasets and corresponding exercises beyond the ones covered here.

12.5.1 Player Behavior

A proposed semester-long project related to **game data mining**, game user research, game data science (or similar) class is as follows. You have to choose a dataset containing player behavioral attributes to apply the necessary preprocessing on the data such as extracting features and selecting features. (In the remainder of this section we provide some dataset options.) Then you must apply a relevant *semi-supervised*, *self-supervised* or *unsupervised* learning technique for compressing, analyzing, or reducing the dimensionality of your dataset. Based on the outcome of this learning process and the resulting player embedding you will need to implement a number of appropriate *supervised learning* techniques that learn to predict a data attribute (or a set of attributes) of the player. We leave the selection of algorithms to the course instructor. When we teach such a graduate course we normally offer a wide variety of options for students to select from (i.e., in terms of datasets, methods and tasks) and we assess them based on their selections and the performance of the algorithm on their selected downstream task. Next we discuss a number of example datasets one might wish to start from; the reader, however, may refer to the book's website for more options.

SteamSpy Dataset

SteamSpy (http://steamspy.com/) is a rich dataset of thousands of games released on Steam[1] containing several attributes each. While strictly not a dataset focused on player modeling, *SteamSpy* offers an accessible and large dataset for game analytics. The data attributes of each game include the game's name, the developer, the publisher, the score rank of the game based on user reviews, the number of owners of the game on **Steam**, the people that have played this game, the people that have played this game in the last two weeks, the average and median playtime, the game's price and the game's tags. The reader may use an API[2] to download all data attributes from all games contained in the dataset. Then one might wish to apply supervised learning to be able to predict an attribute (e.g., the game's playtime) based on other game features such as the game's score, release date and tags. Or alternatively, one might wish to construct a score predictor of a new game. The selection of the modeling task and the AI methods is left to the reader.

StarCraft: Brood War Repository

The *StarCraft: Brood War* repository contains a number of datasets that include thousands of professional *StarCraft* replays. The various data mining papers, datasets as well as replay websites, crawlers, packages and analyzers have been compiled by Alberto Uriarte at Drexel University.[3] In this exercise you are faced with the challenge of mining game replays with the aim to predict a player's strategy. For instance you are

[1] http://store.steampowered.com/.

[2] http://steamspy.com/api.php.

[3] Available at: http://nova.wolfwork.com/dataMining.html.

requested to predict the choices the Protoss player made using gameplay information of the Terran player by using the available datasets. Some results on the *StarCraft: Brood War* datasets are reported in [953, 1196, 1245]. Those can form the basis for further exploration with this repository.

Other Datasets

Beyond the specific datasets we mentioned previously, a growing community of researchers and practitioners of AI and games has put some effort soliciting datasets and resources for player behavior modeling. One of these notable efforts is the *Awesome Game Datasets*[4] repository that aims to serve as a guide for anyone who wishes to study AI and data mining methods as applied to games. The repository contains a series of datasets (i.e., from over 70 game titles, at the time of writing), but also tools and materials for the researcher to build their own dataset.

12.5.2 Player Experience

As a semester project on player experience modeling it is suggested you choose a game, one or more affective or cognitive states to model and one or more input modalities. You are expected to collect empirical data using your selected game and build models for the selected psychological state of the players that rely on the chosen input modalities.

As a smaller project that does not involve data collection you may opt to choose one of the following datasets and implement a number of AI methods that will derive accurate player experience models. The models should be compared in terms of a performance measure such as prediction accuracy. As with the player behavioral modeling exercises the selection of the AI methods is left to the reader. The datasets accompanying this book and outlined in this chapter are only indicative and not inclusive of the options available. The book's website is up to date with more datasets and exercises beyond the ones covered here.

AGAIN Dataset

The Arousal video Game AnnotatIoN (AGAIN) dataset [744] has been collected for the purposes of analyzing and modeling player experience in a general fashion: within different games of the same genres and across games of difference genres. At the time of writing, the AGAIN dataset is the largest—over 37 h of annotated video and game logs—and most diverse publicly available affective dataset based on games. The dataset features over 1100 in-game videos (with corresponding gameplay data) from 9 different games, which are annotated for arousal from 124 participants in a first-person continuous fashion via PAGAN [742].

[4] Available at: https://github.com/leomaurodesenv/game-datasets.

The AGAIN dataset is available online.[5] Students can test any of the AI methods covered in this chapter to analyze player arousal across each of the 9 games available, within the 3 games of each genre (shooters, racing and platformers) and even across the 3 game genres. Player arousal models can be trained on the manually extracted features of each game that are available in the dataset or even on the in-game video footage available (or both). More details about the dataset can be found in [744].

Platformer Experience Dataset

The Platformer Experience Dataset [545] is the first available game experience corpus that contains multiple modalities of data from players of *Infinite Mario Bros*, a variant of *Super Mario Bros* (Nintendo 1985). The database can be used to capture aspects of player experience based on *behavioral* and *visual* recordings of platform game players. In addition, the database contains aspects of the *game context*—such as level attributes—demographic data of the players and self-reported annotations of experience in two forms: *ratings* and *ranks*. The detailed description of the dataset is available online.[6] The book website contains further details and a set of exercises based on this dataset.

***GameVibe* Corpus**

The *GameVibe* corpus[7] is a multimodal affect dataset of viewer engagement for FPS game videos [887]. *GameVibe* consists of 2 h of high-quality audio and visual data from 30 different FPS games extracted and curated from publicly available "Let's Play" videos on YouTube. Engagement labels are provided by 20 annotators in the form of unbounded, time continuous signals via RankTrace [664]. The corpus includes the raw videos used and the latents extracted using pre-trained foundation models for visuals such as VideoMAE [1179] and audio such as BEATS [205]. The rich variety of the FPS game stimuli used encompasses multiple game modes, winning conditions, and art styles, and therefore it offers a unique opportunity for studying engagement models across a wide variety of contexts within a single genre.

12.6 Summary

This chapter presented a number of representative player modeling examples, for deriving player embeddings but also for modeling the behavior of players and their experience. Our hope is that these examples can help more thoroughly illustrate the taxonomies, methods, and considerations given in the two previous chapters, and perhaps also inspire further research. This was the final chapter dedicated to human players and the last chapter of Part IV. If you have read up to this point you have

[5] https://again.institutedigitalgames.com/.

[6] https://ped.institutedigitalgames.com/.

[7] https://osf.io/p4ngx/.

already been through introductory and background information on AI and games and a thorough presentation of the three core uses of AI in games. The next and final part of the book is dedicated to future trends in the field and ethical aspects of game AI. The first chapter of this part focuses specifically on the holistic synthesis of the various AI areas, the various methods and the various users of games under a common game AI framework.

Part V
The Road Ahead

Chapter 13
Panorama and Map of Synergies

Abstract While the other chapters in this book have focused on particular use cases, problem domains, and families of methods, this chapter instead focuses on how these various subfields of game AI relate to each other. What is the relation of player modeling to procedural content generation, and how is game adaptation informed by research into game-playing agents and human-like behavior? We try to map this web of interconnections, and suggest areas for further exploration. The chapter also contains alternative perspectives on the field, drawing on the perspectives of method, end-users, and player-game interaction.

This chapter attempts to give a high-level overview of the field of game AI, with particular reference to how the different core research areas within this field inform and interact with each other, both actually and potentially. In other words, we map the synergies, actual and potential, between the various research areas and topics within the field. For that purpose we first identify the main research areas and their sub-areas within the game AI field. We then view and analyze the areas from three key perspectives: (1) the dominant AI method(s) used under each area; (2) the relation of each area with respect to the human user; and (3) the placement of each area within a human-computer interaction perspective. In addition, for each of these areas we consider how it could inform or interact with each of the other areas; in those cases where we find that meaningful interaction either exists or is possible, we describe the type of that interaction and provide references to published studies, if any.

The main motivations for us writing this chapter is to help the reader understand how a particular area relates to other areas within this increasingly growing field, how the reader can benefit from knowledge created in other areas and how the reader can make their own research more relevant to other areas. To facilitate and foster synergies across active research areas we place all key studies into a taxonomy with the hope of developing a common understanding and vocabulary within the field of AI and games. The structure of this chapter is based on the first holistic overview of the game AI field presented in [1291]. The book takes an up-to-date perspective on the key game AI areas given its educational and research focus. We would like to point out that this chapter might be of less interest to those readers who are mainly interested in implementing existing methods, or who already have a well-defined

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_13

problem to solve. On the other hand, it could be much more interesting for those looking to define and explore new research problems.

The main game AI areas and core subareas already identified in this book and covered in this chapter are as follows:

- **Play Games** (see Part II) which includes the subareas of *Playing to Win* and *Playing for Experience*. Independently of the purpose (winning or experience) AI can control either the player character or the non-player character.
- **Generate Content** (see Part III) which includes the subareas of *autonomous (procedural) content generation* and *assisted content generation*. Note that the terms *assisted (procedural) content generation* and *mixed-initiative (procedural) content generation* (as defined in Chap. 7) are used interchangeably here.
- **Model Players** (see Part IV) which includes the subareas *player experience modeling* and *player behavior modeling*. The task of deriving *player embeddings* can serve the purposes of either behavioral or experience models and hence it is not defined as an independent area in this analysis.

The scope of this chapter is not to provide an inclusive survey of all game AI areas but rather a roadmap of interconnections between them via representative examples. As research progresses in this field, new research questions will pop up and new methods will be invented, while other questions and methods will recede in importance. We believe that all taxonomies of research fields are by necessity tentative. Consequently, the list of areas defined in this chapter should not be regarded as fixed and final.

The structure of the chapter is as follows: Sect. 13.1 analyzes holistically the areas within the AI and games field and we provide three alternative views: one with respect to the *methods* used, one with respect to the *end users* and one where we outline how each of the research areas fits within the *player-game interaction* loop of games. Then, Sect. 13.2 digs deeper into the research areas and describes each one of them in detail. With the subsection describing each area, there is a short description of the area and a paragraph on the possible interactions with each of the other areas for which we have been able to identify strong or weak influences. The chapter ends with section containing our key conclusions and vision for the future of the field (Sect. 13.3).

13.1 Panoramic Views of AI and Games

Analyzing any research field as a composition of various subareas with interconnections and interdependencies can be achieved in several ways. In this section we view AI and games research from three high-level perspectives that focus on the computer (i.e., the AI methods), the human (i.e., the potential end user) and the interaction between the key end user (i.e., player) and the game. Instead in Sect. 13.2 we outline each area and present the interconnections between them.

AI in games is composed of methods, processes and algorithms in artificial intelligence as those are applied to, or inform the development of, games. Naturally, game AI can be analyzed through the *method* used by identifying the dominant AI approaches under each game AI area (see Sect. 13.1.1). Alternatively, game AI can be viewed from the game domain perspective with a focus on the *end users* of each game AI area (see Sect. 13.1.2). Finally game AI is, by nature, realized through systems that entail rich human-computer interaction and, thus, the different areas can be mapped to the interaction framework between the player and the game (see Sect. 13.1.3).

13.1.1 Methods (Computer) Perspective

The first panoramic view of the field we present is centered around the AI methods used. As the basis of this analysis we first list the core AI methods used in the game AI field. Beyond ad-hoc behavior authoring the key areas identified in Chap. 2 include tree search, evolutionary computation, supervised learning, semi-supervised learning, self-supervised learning, reinforcement learning, and unsupervised learning. For each of the game AI areas investigated we have identified the AI methods that are **dominant** or **secondary** in the area. While the dominant methods represent the most popular techniques used in the literature, secondary methods represent techniques that have been considered from a substantial volume of studies but are not dominant.

We have chosen to group methods according to what we perceive as a taxonomy and following the structure of Chap. 2. While it would certainly be possible to classify the various methods differently, we argue that the proposed classification is compact (containing solely key methodology areas) and it follows standard method classifications in AI. While this taxonomy is commonly accepted, the lines can be blurred. For instance, evolutionary computation, being a very general optimization method, can be used to perform supervised, unsupervised or reinforcement learning task (more or less proficiently). The model-building aspect of reinforcement learning can be seen as a supervised learning problem (mapping from action sequences to rewards), and the commonly used tree search method Monte Carlo tree search can be seen as a form of temporal difference learning and can be used in combination with deep (supervised or reinforcement) learning methods. The result of any tree search algorithm can be seen as a plan, though it is often not guaranteed to lead to the desired end state. That the various methods have important similarities and some overlap does not detract from the fact that each of them is clearly defined.

Table 13.1 illustrates the relationship between game AI areas and corresponding methods. It is evident that evolutionary computation and variants of supervised learning appear to be of dominant or secondary use in most game AI areas. Evolutionary computation is a dominant method for generating content (in an assisted fashion or autonomously), and for modeling players; it has also been considered for agents that play to win and for the design of believable play (play for experience) research.

Table 13.1 Dominant (●) and secondary (○) AI methods for each of the core AI areas we cover in this book. The total number of methods used for each area appears at the bottom row of the table

	Play Games		Generate Content		Model Players	
	Winning	Experience	Autonomously	Assisted	Experience	Behavior
Behavior authoring	●	●				
Tree search	●	○	○	○		
Evolutionary computation	○	○	●	●	●	
Supervised learning	○	●			●	●
Semi-Supervised learning	●	○			○	○
Self-Supervised learning	●	●	●	○		○
Reinforcement learning	●	○	○		○	
Unsupervised learning				○	○	●
Total (Dominant)	7 (5)	7 (3)	4 (2)	4 (1)	5 (2)	4 (2)

Supervised learning and its variants (semi and self-supervised learning) are of substantial use across the game AI areas. Supervised learning appears to be dominant in player experience and behavioral modeling, as well as in the area of AI that plays for experience. Self-supervised learning is a dominant method primarily for playing games and generating content. Semi-supervised learning is currently at its early days when it comes to playing games and modeling players. Behavior authoring, on the other hand, is useful solely for gameplaying. Reinforcement learning and unsupervised learning find relatively limited use across the game AI areas, respectively, being dominant only on AI that plays to win and player behavior modeling. Finally, tree search is applied primarily in playing to win and it is also considered—as a form of planning—for controlling play for experience and in computational narrative (as part of autonomous or assisted PCG).

Viewing Table 13.1 from the game AI areas' perspective (columns) it seems that AI that plays games (either for winning or for experience) defines the game AI area with the most diverse and richest palette of AI methods. On the contrary, procedural content generation is dominated by evolutionary computation, self-supervised learning methods, and tree search, RL and unsupervised learning to a secondary degree. It is important to state that the popularity of any AI method within a particular area is closely tied to the task performed or the goal in mind. For example, evolutionary computation is largely regarded as a computationally heavy process which is mostly used in tasks associated with offline training. As PCG so far mainly relies on content that is generated offline, fine-tuned self-supervised learning methods such as LLMs

and evolutionary computation define good candidate methods. If online learning is a requirement for the task at hand, however, other methods (such as reinforcement learning or pruned tree-search) tend to be preferred.

Clearly the possibility space for future implementations of AI methods under particular game AI areas seems rather large. While certain methods have been traditionally dominant in specific areas for good reasons (e.g., planning in computational narrative) there are equally good reasons to believe that the research in a game AI area itself has been heavily influenced by (and limited to) its corresponding dominant AI methods. Not surprisingly, foundation models and LLMs already influence most of the game AI areas discussed and will likely find a role in even more game AI use cases in the future [365]. The empty cells of Table 13.1 indicate potential areas for exploration and offer us an alternative view of promising new intersections between game AI areas and methods.

We conclude this section with an interesting statistic showcasing how the interaction between methods and game AI areas has evolved over time. The table we presented in the first edition of the book in 2018 contained 6 method types (instead of 8) and a total of 20 dominant and secondary uses of AI methods across the six game AI areas (those have not changed!). Six years after, the total number of method uses summarized in Table 13.1 is over 30, indicating the increasing diversity of AI method applications in games.

13.1.2 End User (Human) Perspective

The second panoramic view of the game AI field puts an emphasis on the end user of the AI technology or general outcome. Towards that aim we investigate three core dimensions of the game AI field and classify all game AI areas with respect to the *process* AI follows, the game *context* under which algorithms operate and, finally, the **end user** that benefits most from the resulting outcome. The classes identified under these dimensions are used as the basis of the taxonomy we propose.

The first dimension (phrased as a question) refers to the AI process: *In general, what can AI do within games?* We identify two potential classes in this dimension: AI can *model* or *generate*. For instance, an artificial neural network can model a playing pattern, or a genetic algorithm can generate game assets. Given that AI can model or generate the second dimension refers to the context: *What can AI methods model or generate in a game?* The two possible classes here are *content* and *behavior*. For example, AI can model a players' affective state, or generate a level. Finally, the third dimension is the end user: *AI can model, or generate, either content or behavior; but, for whom?* The classes under the third dimension are the **designer**, the **player**, the **AI researcher**, and the **producer/publisher**.

Note that this taxonomy serves as a framework for classifying the game AI areas according to the end user and is, by no means, inclusive of all potential processes, contexts, and end users. For instance, one could claim that the producer's role should be distinct from the publisher's role and that a developer should also be included in

that class. One could even identify more fine grained roles say in between the designer and the player end user via the user-generated content process by which players act as designers. Moreover, game content could be further split into smaller subclasses such as narrative, levels, etc. Nevertheless, the proposed taxonomy provides distinct roles for the AI process (model vs. generate), clear-cut classification for the context (content vs. behavior) and a high-level classification of the available stakeholders in game research and development (designer vs. player vs. AI researcher vs. producer/publisher). The **taxonomy** presented here is a modified version of the one introduced in [1291] and it does not consider *evaluation* as a process for AI since it is out of the primary scope of this book.

Figure 13.1 depicts the relationship between the game AI core areas, the subareas and the end users in game research and development. Assisted content generation is useful for the designer and entails all possible combinations of processes and context as both content and behavior can be either modeled or generated. Compared to the other stakeholders the player benefits directly from more game AI research areas. In particular the player and their experience are affected by research on player modeling, which results from the modeling of experience and behavior; research on autonomous and assisted procedural content generation, as a result, respectively, of any underlying generative AI processes and user-generated content; and studies on AI agents playing (for winning or experience) resulting from the generation of behavior. Autonomous content generation and player character based game playing (for winning or experience) areas provide input to the AI researcher primarily. Finally, the game producer/publisher is primarily affected by results on behavioral player

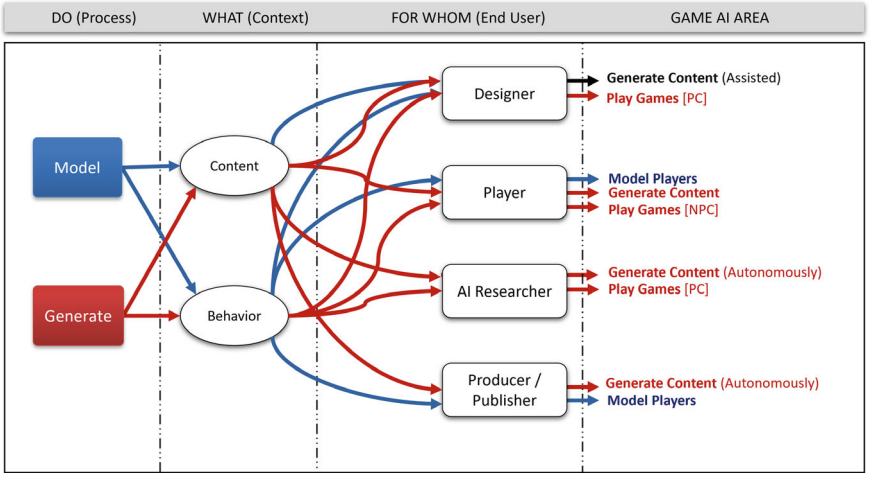


Fig. 13.1 The end user perspective of the identified game AI areas. Each AI area follows a **process** (model or generate) under a **context** (content or behavior) for a particular **end user** (designer, player, AI researcher or game producer/publisher). Blue and red arrows represent the processes of modeling and generation, respectively. Modified graph from [1291]

modeling, game analytics and game data mining as a result of behavior modeling; moreover, the development of their games can be assisted by content generation processes.

13.1.3 Player-Game Interaction Perspective

The third and final panoramic perspective of game AI presented in this section couples the computational processes with the end user within a game and views all game AI areas through a human-computer interaction lens. The analysis builds on the findings of Sect. 13.1.2 and places the five game AI areas that concern the player as an end user on a player-game interaction framework as depicted in Fig. 13.2. Putting an emphasis on player experience and behavior, player modeling directly focuses on the interaction between a player and the game context. Game content is influenced primarily by research on autonomous procedural content generation but also by research on user-generated content in an assistive manner. In addition to other types

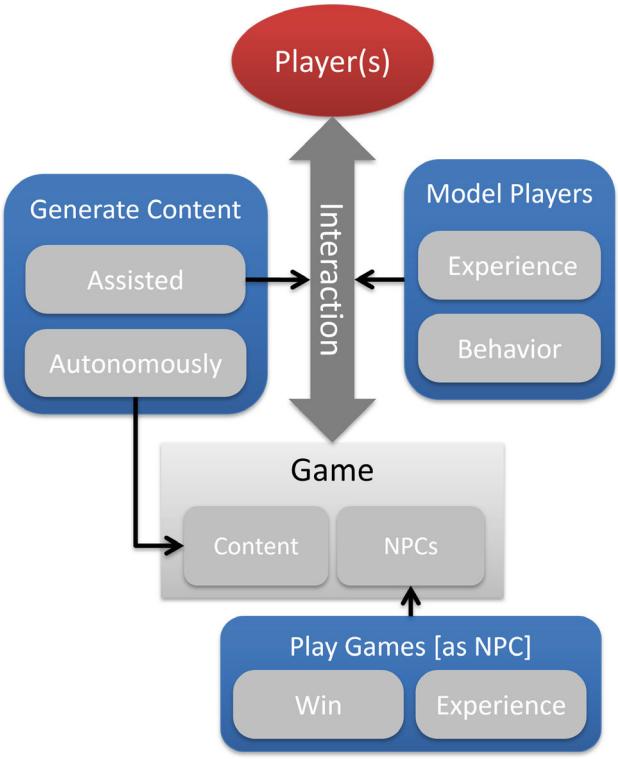


Fig. 13.2 Game AI areas and sub-areas viewed from a player-game interaction perspective

of content, most games feature NPCs, the behavior of which is controlled by some form of AI. NPC behavior is informed by research in agents that play the game to win or any other playing-experience purpose such as believability.

Looking at the player-game interaction perspective of game AI it is obvious that the player modeling area has the most immediate and direct impact on the player experience as it is the only area linked to the player-game interaction directly. From the remaining areas, PCG influences player experience the most as all games have some form of environment representation and mechanics that could be generated. Importantly some games such as *Spore* (Electronic Arts, 2008) and *AI Dungeon* (Latitude, 2019) interweave user-generated content (assisted) as part of play, directly or not. Finally, AI that plays as an NPC (either to win or for the experience of play) is constrained to games that include agents or non-player characters.

The areas not considered directly in this game AI perspective affect the player rather remotely. As we discussed earlier, players of some games may design parts of their own game in a *direct* manner. However, AI tools that assist designers creating content may improve the game's quality *indirectly* and in retrospect affect the player experience as designers tend to maintain a *second-order player model* [639] while designing. Finally, AI that plays the game as a player character can be offered for testing both the content and the NPC behaviors of a game, but also the interaction between the player and the game but is mainly directed to AI researchers (see Fig. 13.1).

13.2 How Game AI Areas Inform Each Other

In this section, we outline the core game AI areas and discuss how they inform or influence (the terms are used interchangeably) each other. All research areas could be seen as potentially influencing each other to some degree; however, making a list of all such influences would be impractical and the result would be uninteresting. Therefore we only describe *direct* influences. Direct influences can be either **strong** (represented by a • as the bullet point style next to the corresponding influence) or **weak** (represented by a ◦). We do not list influences we do not consider potentially important for the informed research area, or which only go through a third research area.

The sections in what follows list *outgoing* influence. Therefore, to know how area A influences area B you should look in the section describing area A. Some influences are mutual, some not. The notation $A \rightarrow B$ in the headings of this section denotes that "A influences B". In addition to the text description each section provides a figure representing all outgoing influences of the area as arrows. Dark and light gray colored areas represent, respectively, *strong* and *weak* influence. Areas with white background are not influenced by the area under consideration. The figures also depict the incoming influences from other areas. Incoming *strong* and *weak* influences are represented, respectively, with a red solid and dashed outline around the game AI

areas that influence the area under consideration. Note that the description of the incoming influence from an area is presented in the corresponding section of that area.

13.2.1 Play Games

The key area in which AI plays a game (as covered in Chap. 4) involves the sub-areas of **Playing to Win** and **Play for Experience**. As mentioned earlier in the chapter the AI can control either player or non-player characters of the game. We cover the influences to (and from) these subareas of game AI here.

Playing to Win

As already seen in Part II research in AI that learns to play (and win) a game focuses primarily on using reinforcement learning techniques such as deep reinforcement learning, temporal difference learning, or evolutionary algorithms to learn policies that play games well. From the very beginning of AI research, reinforcement learning techniques have been applied to learn how to play board games (see for example Samuel's Checkers player [987]). Basically, playing the game is seen as a reinforcement learning problem, with the reinforcement tied to some measure of success in the game (e.g., the score, or length of time survived). As with all reinforcement learning problems, different methods can be used to solve the problem (find a good policy) [1169] including TD learning [1134], evolutionary computation [674], competitive co-evolution [38, 896, 972, 985], simulated annealing [78], deep RL [773] transformer architectures [203, 562, 615] other optimization algorithms and any combinations of such algorithms [563, 773]. In recent years the application of various learning methods to different types of video games has appeared in the literature (including several overviews [674, 761, 785, 1029, 1050]). Finally, using games to develop artificial general intelligence [868] builds on the idea that games can be useful environments for algorithms to learn complex and useful generalist agent behaviors [930]; thus research in algorithms that learn to win is essential.

It is also worth noting that most game-based **benchmarks** introduced in the literature measure how well an agent plays a game. Methods for learning to play a game are vital for such benchmarks, as the benchmarks are only meaningful in the context of the algorithms. When algorithms are developed that "beat" existing benchmarks, new benchmarks need to be developed. For example, the success of an early planning agent in the first Mario AI competition necessitated that the software be augmented with a better level generator for the next competition [539]. For the Simulated Car Racing competition, the performance of the best agents on the original competition game spurred the change to a new more sophisticated racing game [658, 1164]. For the interested reader a recent survey paper reviews all game benchmarks currently available for AI research [482]; see for example [539, 672, 829].

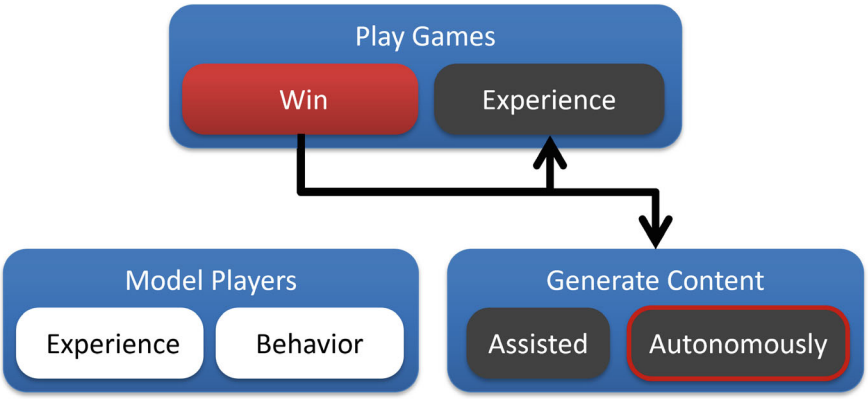


Fig. 13.3 Playing to Win: influence on (and from) other game AI research areas. **Outgoing** influence (represented by arrows): black and dark gray colored areas reached by arrows represent, respectively, **strong** and **weak** influence. **Incoming** influence is represented by red lines around the areas that influence the area under investigation (i.e., AI that plays to win in this figure): **strong** and **weak** influences are represented, respectively, by a solid and a dashed red outline

Research in this area impacts game AI at large as three game AI subareas are directly affected. In turn, the subarea of autonomous content generation is directly affecting AI that plays to win (see the red outline of the area in Fig. 13.3).

- **Playing to Win → Playing for Experience:** An agent cannot be believable or tasked to augment the game's experience if it is not proficient enough. Being able to play a game well is in several ways a precondition for playing games in a believable manner though well playing agents can be developed without learning (e.g., via top-down approaches). In early years of game AI for example, successful entries to competitions focused on believable agents, such as the 2K BotPrize and the Mario AI Championship Turing test track, have included a healthy dose of learning algorithms [1004, 1174]. Recent studies on the use of diffusion models of human-like FPS gameplay reveal interesting interrelations between play style and playing proficiency [856].
- **Playing to Win → Generating Content (Autonomously):** Having an agent that is capable of playing a game proficiently is useful for **simulation-based testing** in procedural content generation, i.e., the testing of newly generated game content by playing through that content with an agent. For example, in a program generating levels for the platform game *Super Mario Bros* (Nintendo, 1985), the levels can be tested by allowing a trained agent to play them; those that the agent cannot complete can be discarded [557]. Browne's *Ludi* system generates complete board games; those games are evaluated through simulated playthrough of AI agents that learn to adapt their strategy to each game [133]. Simulated play (to win) is key in recent investigations of generalist agents across diverse or increasingly complex game environments [930, 947].

- **Playing to Win → Generating Content (Assisted):** Just as with autonomous procedural content generation, many tools for AI-assisted game design rely on being able to simulate playthroughs of some aspect of the game. For instance, the *Sentient Sketchbook* tool for level design uses simple simulations of game-playing agents to evaluate aspects of levels as they are being edited by a human designer [640]. Another example is the automated playtesting framework named *Restricted Play* [503] which aims mostly at assisting designers on aspects of game balance during game design. A form of *Restricted Play* is featured in the *Ludocore* game engine [1057]. Some recent playtesting methods for evaluating PCG—such as those derived from the deep learning family—are detailed in [654].

Playing for Experience

Research on AI that plays a game for a purpose other than winning is central to studies where playing the game well is not the primary research aim. AI can play the game as a *player* character attempting to maximize the believability value of play as, for instance, in [60, 165, 856, 1026, 1174]. It can alternatively play the game in a role of an opponent player or an NPC for the same purpose [460]. Work under this research subarea involves the study of **believability**, interestingness or playing experience in games and the investigations of mechanisms for the construction of agent architectures that appear to have e.g., believable or human-like characteristics. The approaches for developing such architectures can be top-down behavior authoring such as the FAtiMA model used in *My Dream Theatre* [170] and the Mind Module model [319] used in *The Pataphysic Institute*. Alternatively, bottom-up methods can attempt to imitate believable gameplay from human players such as the early work of Thurau et al. [1144] in *Quake II* (Activision, 1997) bots, the human imitation attempts in *Super Mario Bros* (Nintendo, 1985) [837], the *Unreal Tournament 2004* (Epic Games, 2004) believable bots of Schrum et al. [1004], the crowdsourcing studies of the *Restaurant game* [834] and the recent player imitation via diffusion models in *Counter-Strike: Global Offensive* (Valve, 2012) [856]. Evidently, commercial games have for long benefited from agent believability research. Examples of this include popular games such as *The Sims* series (Electronic Arts, 2000–2024). The industry puts a strong emphasis on the design of believability in games as this contributes to more immersive game environments.

The study of AI that plays games not for winning, but for other purposes, affects research on three other game AI areas as illustrated in Fig. 13.4, whereas it is affected by four other game AI areas.

- **Playing for Experience → Modeling Players (Experience and Behavior):** There is a direct link between player modeling and believable agents as research carried out for the modeling of human, human-like, and supposedly believable playing behavior can inform the construction of more appropriate models for players. Examples of studies that can inform the construction of player models include the imitation of human play styles in *Super Mario Bros* (Nintendo, 1985) [837], *Quake II* (Activision, 1997) [1144] and *Counter-Strike: Global Offensive* (Valve, 2012) [856] and the imitation of both playing behavior and a player's affective

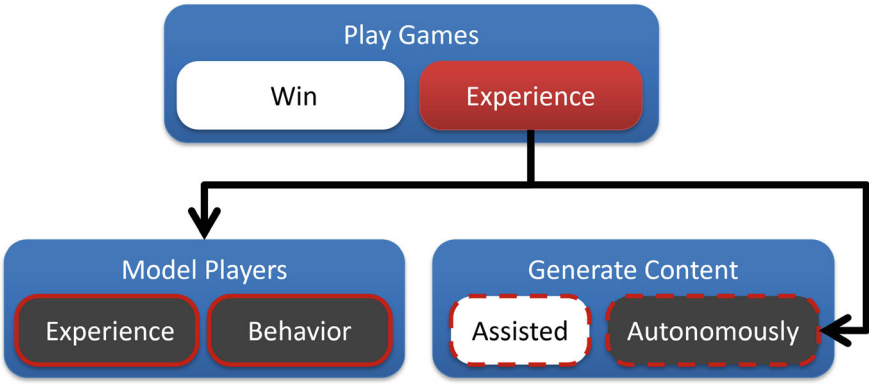


Fig. 13.4 Playing for Experience: influence on (and from) other game AI research areas

responses in racing games [60, 62]. Since player modeling focuses on the human player, the model may consider the behavior of AI controlled opponents or NPCs, as in *Left 4 Dead* (Valve Corporation, 2009). The intended outcome for such a learning algorithm is a player model that is informed by agents that play for experience [41, 837, 1205].

- **Playing for Experience → Generating Content (Autonomously):** Human-like or believable characters may contribute to better levels [165], more believable stories [669, 880, 1322] and, generally, better game representations [940]. An indicative example of a full integration of characters in the narrative are the early FAtiMa agents in *FearNot!* [848] and *My Dream Theater* [170]. Another example is the generation of *Super Mario Bros* (Nintendo, 1985) levels that maximize the believability of any Mario player [165]. Among more recent examples, the *AI people* prototype game discussed in Part III features conversational NPCs empowered by LLMs which allow them to be socially interactive and take actions that are relevant to the current game narrative [389].

13.2.2 Generate Content

As covered in detail in Part III AI can be used to design whole (or parts of) games in an autonomous or in an assisted fashion. This core game AI area includes the subareas of **autonomous content generation** and **assisted content generation**. We cover the interactions of these subareas with the remaining areas of game AI here.

Generating Content (Autonomously)

As stated in Chap. 7 procedural content generation has been included in limited roles in some commercial games since the early 1980s; however, the last decade has seen an expansion of research on more controllable PCG for multiple types of game

content [1268], using techniques such as evolutionary search [1176], constraint solving [1056] and machine learning [419]. There are several surveys of PCG available, including books [419, 1023], vision papers [1155], surveys of frameworks [1289], reviews of the field [632], sub-areas of PCG [921, 1202] and methods [365, 405, 559, 1038, 1056, 1176].

Autonomous content generation is one of the areas of recent academic research on AI in games which bears most promise for incorporation into **commercial games**. Indicative PCG games developed by game AI researchers that have been released as commercial games on online platforms like Steam include *Petalz* [944, 945] and *Galactic Arms Race* [431]. A number of games have been based heavily on PCG, including independent ("indie") game production successes such as *Spelunky* (Mossmouth, 2009) and *Minecraft* (Mojang, 2011), and mainstream AAA games such as *Diablo III* (Blizzard Entertainment, 2012) and *Civilization VI* (2K, 2016). Two notable recent examples, as detailed in Chapter 7, include *No Man's Sky* (Hello Games, 2016) with its quintillion different procedurally generated planets and *AI Dungeon* (Latitude, 2019) with its personalised narratives which are generated through the player's interaction with LLMs. Figure 13.5 depicts the three (and five) areas that are influenced by (and influence) autonomous PCG.

- **Generating Content (Autonomously) → Playing to Win:** If an agent is trained to perform well in only a single game environment, it is easy to overspecialize the training and arrive at a policy that will not generalize well to other levels. Therefore, it is important to have a large number of environments available for training. PCG can help with this, potentially providing an infinite supply of test environments. For example, when training players for a game like *Super Mario Bros* (Nintendo, 1985) it is common practice to test each agent on a

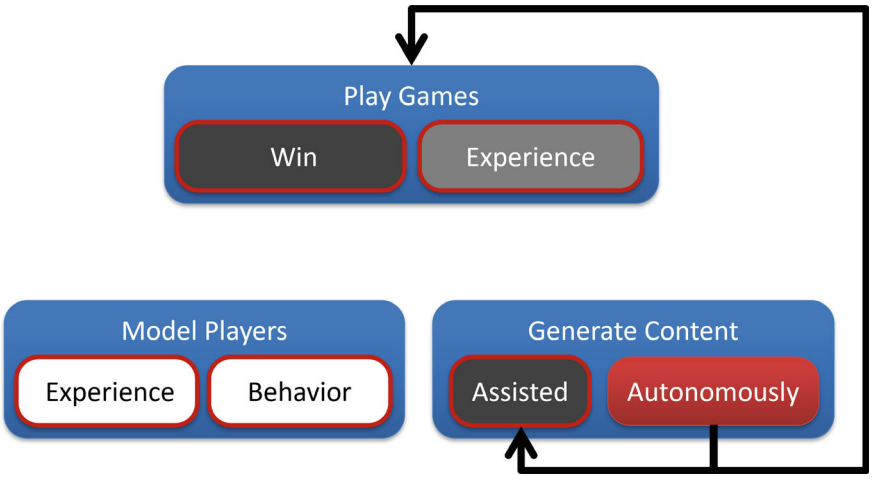


Fig. 13.5 Generating Content (Autonomously): influence on (and from) other game AI research areas

large set of freshly generated levels, to avoid overtraining [539]. There has also been research on adapting NPC behavior specifically to generated content [553]. One common approach to **artificial general intelligence** is to train agents to be good at playing games in general, and test them on a large variety of games drawn from some genre or distribution [615, 930]. To avoid overfitting, this requires games to be generated automatically as in [947, 998] for instance. The generation of new environments is very important for NPC behavior learning, and this extends to benchmarks that measure some aspect of NPC behavior. As an indicative example, AI competitions such as the Simulated Car Racing Championship used freshly generated tracks, unseen by the participants, to test submitted controllers [179].

- **Generating Content (Autonomously) → Playing for Experience:** Research on autonomous PCG naturally influences research on agent control for believability, interestingness, human-likeness, style, or other aims aside from winning given that these agents are performing in a particular environment and under a specific game context. This influence is still in its infancy and the only study we can point the reader to is the one by Camilleri et al. [165]; in that study authors test the impact of level design on player character believability in a clone of *Super Mario Bros* (Nintendo, 1985). Further, research on interactive narrative benefits from and influences the use of believable agents that interact with the player and are interwoven in the story plot. The narrative can yield more (or less) believability to agents and thus the relationship between the behavior of the agents and the emergent story is strong [669, 880, 1322]. In that sense, the computational narrative of a game may define the arena for believable agent design. Recent LLM-powered games and tools that fuse story generation with agent decision making as the ones detailed in [365] will likely make this influence—from autonomous PCG to agents that are playing for experience—an even stronger one over the coming years.

- **Generating Content (Autonomously) → Generating Content (Assisted):** Many AI-assisted design tools incorporate some form of assisted content design. Early examples of such tools include *Tanagra*, which helps designers create complete platform game levels which ensure playability through the use of constraint solvers [1059], and *SketchaWorld* [1052]. Another early example is *Sentient Sketchbook*, which assists humans in designing strategy game levels through giving immediate feedback on properties of levels and autonomously suggesting modifications [640]. As detailed in Part III many contemporary AI-assistive tools that are driven by large multimodal models will likely dominate this direct influence from autonomous PCG methods to AI-assisted tools for game design [365]. Mixed-initiative PCG tools presume that there is, at the very least, a simple and controllable generator in place in order to offer a *computational initiative* to the designer. In that sense, the autonomous and controlled generation of content is mandatory for assisted generation.

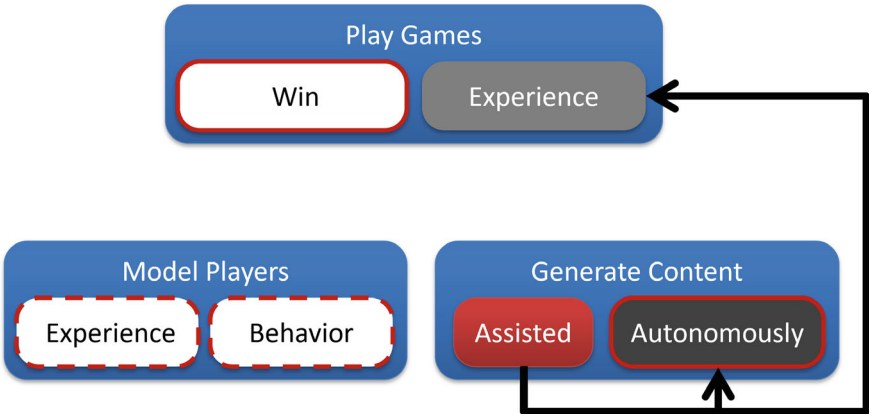


Fig. 13.6 Generating Content (Assisted): influence on (and from) other game AI research areas

Generating Content (Assisted)

Assisted content generation is perhaps the AI research area which is most promising for the development of better games [1268]. In particular, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives. The impact of AI-enabled authoring tools on design and development influences the study of AI that plays games for believability, interestingness or player experience, and research in autonomous procedural content generation (see Fig. 13.6). **AI-assisted game design** tools range from those designed to assist with generation of complete game rulesets such as *RuLearn* [1150] to those focused on more specific domains such as strategy game levels [640], platform game levels [1060], horror games [661], physics-based puzzles [1021] or adventure game levels [540]. Recent AI-assistive tools go beyond the use of constructive methods, constraint optimization or search, and realize mixed-initiative design via text-to-level models [365, 751] and reinforcement learning [272] among other methods.

It is worth noting that AI tools have been used extensively for supporting design and commercial game development. Early examples such as the *SpeedTree* (Interactive Data Visualization Inc., 2013) generator for trees and other plants [492] have seen uses in several game productions. Furthermore, there are tools designed for interactive modeling and analysis of game rules and mechanics, which are not focused on generating complete games but on prototyping and understanding aspects of complex games; such systems could be applied to existing commercial games [1057].

- **Generating Content (Assisted) → Playing for Experience:** Authoring tools in forms of open-world sandboxes could potentially be used for the creation of more believable behaviors. While this is largely still an unexplored area of research and development, notable early attempts include the NERO game AI platform where players can train game agents for efficient and believable first-person shooter bot behaviors [1079]. While strictly speaking NERO featured manually

crafted levels, level generators as the ones used in [947] could improve the generalizability of NERO bot behaviors. An open version of this platform focused on crowdsourcing behaviors has been released nearly a decade later [546]. A similar line of research is the generation of *Super Mario Bros* (Nintendo, 1985) players by means of interactive evolution [1070]. A notable recent example is the XLand multimodal generation framework [1130] that enables the generation of simple multiplayer games within small 3D worlds. While XLand is build for the purpose of training generalist gameplaying agents [930], frameworks as such can be used potentially for the study and the advancement of human-like, believable agents through assisted generation of levels.

- **Generating Content (Assisted) → Generating Content (Autonomously):** Research on methods of **mixed-initiative co-creation** [1279] and design can feed input to and spur discussion on central topics in procedural content generation. Given the importance of content design in the development process as a whole, any form of mixed-initiative AI assistance in the generation process can support and augment procedural content generation. Notable early examples of mixed-initiative PCG—already detailed in Part III—include the *Tanagra* platform game level design AI assistant [1059], and the *SketchaWorld* [1052], the *Sentient World* [641], the *Sentient Sketchbook* [640, 1279] and the *Sonancia* [661] tools. Such AI-assistive tools can generate game maps and worlds via a plethora of methods and varying degrees of human computation. As we covered earlier in Part III, recent methods and tools assisting the image and level creation process employ predominately large multimodal models featuring diffusion model variants or transformer architectures. Beyond images and levels, tools can assist the authoring of narrative in games; in particular, **drama management** tools have long been investigated within the game AI community. A popular example that emerged from the academic community is ABL which allowed the authoring of narrative in *Façade* (Procedural Arts, 2005) [721]. Among the first and most well-functioning authoring tools, the *Versu* [329] storytelling system was used in the game *Blood* and *Laurels* (Emily Short, 2014) and the *Inform 7* [800] software package that led to the design of *Mystery House Possessed* (Emily Short, 2005). Contemporary text and story generation systems rely primarily on large language models as covered in Part III and detailed in a recent survey paper [365]. Players of games such as *AI Dungeon* (Latitude, 2019) sit on the designer's seat and instead "play" with the underlying conversational LLMs which, in turn, assist them create their own personalised gameplay.

13.2.3 Model Players

As already explored in Part IV, modeling players involves the downstream tasks of modeling their **behavior** or their **experience**. Given the interwoven nature of these two tasks we present their influences to (and from) other game AI areas under one common section. Deriving a player **embedding** such as a latent or a data cluster may

serve its own purposes or even enrich the analysis of any player behavior or experience model. Research and development in player modeling can inform attempts for player experience in commercial-standard games. Player experience detection methods and algorithms can advance the study of user experience in commercial games. In addition, the appropriateness of sensor technology, the technical plausibility of biofeedback sensors, and the suitability of various modalities of human input can inform industrial developments. Quantitative testing may also improve through game metrics, varying from behavioral data mining to in-depth low scale user studies [295, 312, 1268]. By now, a considerable number of academic studies use directly datasets from commercial games to induce models of players that could inform further development of the game. For example, we refer the reader to the derived player archetypes of *Tomb Raider: Underworld* (Square Enix, 2008) via clustering [291] and the predictive models of player's late-game performance based on early-game behavior [686], or even the trained predictors of long-term player engagement in *Tom Clancy's the Division II* (Ubisoft, 2020) [888]. Examples of player modeling components within high-profile commercial games include the arousal-driven appearance of NPCs in *Left 4 Dead 2* (Valve Corporation, 2009), the fearful combat skills of the opponent NPCs in *F.E.A.R.* (Monolith, 2005), and the avatars' emotion expression in *The Sims* series (Electronic Arts, 2000–2024) and *Black and White* (Lionhead Studios, 2001). A notable example of a game that is based on player experience modeling is *Nevermind* (Flying Mollusk, 2016); the game adapts its content based on the stress of the player, which is manifested via a number of physiological sensors.

Player modeling is considered to be one of the core non-traditional uses of AI in games [1268] and affects research in AI-assisted game design, agent that play for experience, and autonomous procedural content generation (see Fig. 13.7).

- **Modeling Players → Playing for Experience:** Player models can inform and update believable agent models and architectures. Models of behavioral, affective and cognitive aspects of gameplay can improve the human-likeness and

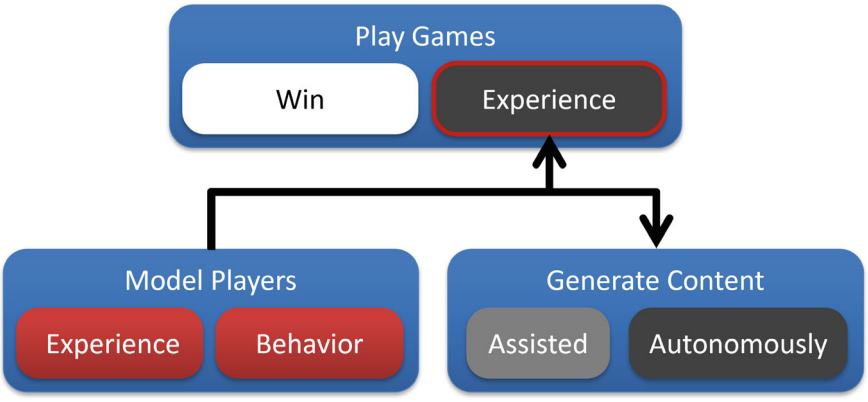


Fig. 13.7 Modeling Players: influence on (and from) other game AI research areas

believability of any agent controller—whether it is ad-hoc designed or built on data derived from gameplay. While the link between player modeling and believable agent design is obvious and direct, research efforts towards this integration within games are still sparse. However, the few efforts made on the capturing aspects of humans for the construction of believable gameplaying have resulted in successful outcomes. Approaches for developing *Unreal Tournament 2004* (Epic Games, 2004) bots (e.g., in [547]) managed to pass the Turing test in the 2K BotPrize competition over a decade ago. Notably, one of the two agents that passed the Turing test in 2K BotPrize managed to do so by mirroring aspects of human play [893]. A line of work that stands in between (primarily) playing games for the experience and (secondarily) player modeling is the study on **procedural personas** [60, 459–461]. As introduced in Part II procedural personas are NPCs that are trained to imitate the decision making process of humans during play. The design of procedural personas usually relies on player models that consider high level utilities of player behavior such as level completion times or various types of player embeddings. Their study both influences our understanding about the internal (cognitive) processes of playing behavior and advances our knowledge on how to build believable characters in games.

- **Modeling Players → Generating Content (Autonomously):** There is an obvious link between computational models of players and PCG as player models can drive the generation of new personalized content for the player. The **experience-driven** role of PCG [1289], as covered in Part III, views game content as an indirect building block of a player's affective, cognitive and behavioral state and proposes adaptive mechanisms for synthesizing personalized game experiences. The "core loop" of an experience-driven PCG solution involves learning a model that can predict player experience, and then using this model as part of an evaluation function for evolving (or otherwise optimizing) game content. Game content is evaluated based on how well it elicits a particular player experience, according to the model. Examples of PCG that are driven by player models include the generation of racing game levels [58, 1157], platformer game levels [1024, 1038, 1239], camera profiles [151, 1285], and class statistics for player vs. player (PVP) combats [876]. Most work that goes under the label **game adaptation** can be said to implement the experience-driven architecture; this includes work on adapting the game content to the player using reinforcement learning [47, 1239] or semantic constraint solving [665] rather than evolution. Player models may also inform the generation of computational narrative. Predictive models of playing experience can drive the generation of individualized scenarios in a game. Examples of the coupling between player modeling and computational narrative include the affect-driven narrative systems met in *Façade* (Procedural Arts, 2005) [721] and *FearNot!* [40], and the affect-centered game narratives such as the one of *Final Fantasy VII* (Square, 1997).
  - **Modeling Players → Generating Content (Assisted):** User models can enhance authoring tools that, in turn, can assist the design process. The research area that bridges user modeling and AI-assisted design—while still at its early

stages—has already offered a few important studies. Indicatively, **designer models** [18, 638–640, 1279] and player models [760, 875, 876] have been employed to personalize mixed-initiative design processes, varying from level and quest generation to player class adaptation.

13.3 What's Next?

This chapter has initially identified the currently most active areas and subareas within game AI and placed them on three holistic frameworks: an AI method mapping, a game stakeholder (end user) taxonomy and the player-game interaction loop. This analysis revealed dominant AI algorithms within particular areas as well as room for exploration of new methods within areas. In addition, it revealed the dissimilar impact of different areas on different end users such as the AI researcher and the designer and, finally, outlined the influence of the different game AI areas on the player, the game and their interaction. From the high-level analysis of the game AI field we moved on to the detailed analysis of the game AI areas that compose it and thoroughly surveyed the meaningful interconnections between the different areas.

The total number of strong and weak influences is rather small compared to all possible interconnections between the areas, which clearly signals the research capacity of the game AI field for further explorations. We can distinguish a number of connections which are currently very active, meaning that much work currently goes on in one area that draws on work in another area. Here we see, for example, the connection between AI that plays to win in a general fashion in conjunction with the use of tree search algorithms and deep reinforcement learning. The MCTS algorithm was invented in the context of board gameplaying, proved to be really useful in the general game playing competition, and has been investigated for use in games as different as *StarCraft* (Blizzard Entertainment, 1998) and *Super Mario Bros* (Nintendo, 1985). Improvements and modifications to the algorithm have been flowing back and forth between the various areas up to its historical performance in Go when combined with deep RL methods [1040]. Nearly 30 years ago an early version of deep RL (TD-Gammon [1134]) proved its efficiency in board game playing in Backgammon and almost a decade ago it shined on Atari gameplaying [773]. Another indicative connection that is alive and strong is between player modeling and procedural content generation, where it is now common for newly devised PCG algorithms and experimental studies to include player behavioral or player experience models.

But it is maybe even more interesting to look at all those connections that are unexploited or underexploited or potentially strong. For example, player modeling is potentially very important in the development of AI that controls believable, interesting or curious agents, but this has not been explored in enough depth yet; the same holds for the application of user (or else, designer) modeling principles towards the personalization of AI-assisted game design. Believable agents have, in turn, not been used enough in content generation (either autonomous or assisted). A grand vision

of game AI for the years to come is to let it identify its own role within game design and development as it sees fit. In the next chapter of this part we discuss frontier research topics as such and identify unexplored roles of AI in games.

13.4 Summary

We hope that with this chapter of this book, we have been able to give our readers a sense of how this—by now rather large and multifaceted—research field hangs together, and what could be done to integrate it further. We realize that this is only our view of its dynamics and interconnections, and that there are (or could be) many competing views. We look forward to seeing those in upcoming studies in the field.

Finally, it is important to note that it would have been impossible to provide a complete survey of all the areas as, first, the game AI field is growing rapidly and, second, it is not the core objective of the book. This means that the bibliography is indicative rather than exhaustive and serves as a general guideline for the reader. The website of the book, instead of the book per se, will be kept up to date regarding important new readings for each area.

The next chapter of the book is dedicated to a few long-standing, yet rather unexplored, research frontiers of game AI. We believe that any advances made in these directions will lead to scientific breakthroughs not merely within game AI but largely in both games (their design, technology and analysis) and AI per se.

Chapter 14
Frontiers of AI and Games Research

Abstract This chapter discusses a number of issues relating to the future of AI and games that do not fit squarely into the preceding chapters, and which are likely to become (or already are) important research topics in the field. The first half of this chapter discusses the challenge of generalization: developing general game players, general content generators, and general player models. A key argument is that the age of foundation models has not quite hit games yet, and that creating such general methods is a huge challenge with a potential huge payoff. There is also a discussion of multiple use cases for AI in games and AI research problems spurred by games that do not fit neatly into the classification in the rest of the book, such as virtual cinematography, chat monitoring, and hyper-formalist game studies. Finally, there is a short discussion of the limitations on game AI brought by existing game engines and the prospects for building AI-enabling game engines.

In this chapter we discuss a number of long-term visionary goals of game AI, putting an emphasis on the *generality* of AI and the *extensibility* of its roles within games. In particular, in Sect. 14.1 we discuss our vision for general behavior for each one of the three main uses of AI in games. Play needs to become general; generators are required to have general generative capacities across games, content types, designers and players; models of players also need to showcase general modeling abilities. In Sect. 14.2 we outline the current limitations of game engines for further AI development and propose ways forward. Finally, in Sect. 14.3 we also discuss uses and roles of AI that are still unexplored and certainly worth investigating in the future.

14.1 General General Game AI

As evidenced from the large volume of studies the game AI research area has been supported by an active and healthy research community for over two decades—at least since the start of the IEEE CIG and the AIIDE conference series in 2005. Before then, research had been conducted on AI in board games since the dawn of automatic computing. Initially, most of the work published at IEEE CIG or AIIDE was concerned with learning to play a particular game as well as possible, or using

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_14

search/planning algorithms to play a game as well as possible without learning. Gradually, a number of new applications for AI in games and for games in AI have come to complement the original focus on AI for playing games [1268]. Papers on procedural content generation, player modeling, game data mining, human-like playing behavior, automatic game testing and so on have become commonplace within the community. As we saw in the previous chapter there is also a recognition that all these research endeavors depend on each other [1291]. Throughout the years far more venues opened their doors to game AI research that vary from traditional AI, machine learning and computer vision conferences such as NeurIPS, IJCAI, ICML, ICCV to game research and human computer interaction venues such as FDG, DIGRA and CHIPlay (among many others). However, traditionally, the vast majority of research projects presented in such venues are very *specific*. Most published papers describe a particular method—or a comparison of two or more methods—for performing a single task (playing, modeling, generating, etc.) in a single game. This is problematic in several ways, both for the scientific value and for the practical applicability of the methods developed and studies performed in the field. If an AI approach is only tested on a single task for a single game, how can we argue that is an advance in the scientific study of artificial intelligence? And how can we argue that it is a useful method for a game designer or developer, who is likely working on a completely different game than the one it was tested on?

As discussed in several parts of this book general game playing is an area that has already been studied extensively and constitutes one of the key areas of game AI [1291]. The focus of generality solely on play, however, is *very narrow* as the possible roles of AI and general intelligence in games are many, including game design, content design and player experience design. The richness of the cognitive skills and affective processes required to successfully complete these tasks has so far been largely ignored by game AI research. We argue that while the focus on general AI needs to be retained, research on general game AI needs to expand beyond mere game playing. The scope for ***general* general game AI** [1172] beyond gameplaying broadens the applicability and capacity of AI algorithms and our understanding of intelligence as tested in a creative domain that interweaves problem solving, art, and engineering.

For general game AI to eventually be *truly* general, we argue that we need to extend the generality of general game playing to all other ways in which AI is (or can be) applied to games. More specifically we argue that the field should move towards methods, systems and studies that incorporate three different types of generality:

1. **Game generality**. We should develop AI methods that work with not just one game, but with any given game that the method is applied to.
2. **Task generality**. We should develop methods that can perform not only one downstream task (playing, modeling, testing, etc.) but a number of different, related tasks.
3. **User (player or designer) generality**. We should develop methods that can model, respond to and/or reproduce the very large variability among humans in design style, playing style, preferences and abilities.

We further argue that all of this generality can be embodied into the concept of **general game design**, which can be thought of as a final frontier of AI research within games. Further details about the notion of *general* general game AI can be found in the vision paper we co-authored about this frontier research area [1172]. It is important to note that we are not arguing that more focused investigations into methods for single tasks in single games are useless; these are often important as proofs-of-concept or industrial applications and they will continue to be important in the future, but there will be an increasing need to validate such case studies in a more general context. We are also not envisioning that everyone will suddenly start working on general methods. Rather, we are positing generalizations as a long-term goal for our entire research community. Finally, the general systems of game AI that we envision ought to have a real-world use. There is a risk that by making systems too general we might end up not finding applications of these general systems to any specific real-world problem. Thus, the system's applicability (or usefulness) sets our core constraint towards this vision of general game AI. More specifically, we envision *general* general game AI systems that are nevertheless integrated successfully within *specific* game platforms or game engines.

14.1.1 General Play

The problem of playing games in a general fashion is the one that has been studied the most so far. There already exist at least three serious **benchmarks** or **competitions** attempting to pose the problem of playing games *in general*, each in its own imperfect way. The General Game Playing Competition, often abbreviated GGP [374], the Arcade Learning Environment [75] and the General Video Game AI competition [870]; all three have been discussed in various places in this book. The results from these competitions so far indicate that general purpose search and learning algorithms by far outperform more domain-specific solutions and "clever hacks". Somewhat simplified, we can say that variations of Monte Carlo tree search perform best on GVGAI and GGP [337], and for ALE (where no forward model is available so learning a policy for each game is necessary) reinforcement learning with deep networks [773] and search-based iterative width [511, 652, 653] perform best. This is a very marked difference from the results of the game-specific competitions, indicating the lack of domain-independent solutions.

While these are laudable initiatives and currently the focus of much research, in the future we will need to expand the scope of these competitions considerably, including expanding the range of games available to play and the conditions under which gameplay happens. We need game playing benchmarks and competitions capable of expressing any kind of game, including puzzle games, 2D arcade games, 3D action-adventures and so on; this is the best way to test general AI capacities and reasoning skills. We also need a number of different ways of interfacing with these games—there is room for both benchmarks that give agents no information beyond the raw screen data but give them hours to learn how to play the game, and

those that give agents access to a forward model and perhaps the game code itself, but expect them to play any game presented to them with no time to learn. These different modes test different AI capabilities and tend to privilege different types of algorithms.

We also need to differentiate away from just measuring how to play games optimally. In the past, several competitions have focused on agents that play games in a human-like manner; these competitions have been organized similarly to the classic **Turing test** [450, 1026]. Playing games in a human-like manner is important for a number of reasons, such as being able to test levels and other game content as part of search-based generation, and to demonstrate new content to players. So far, the question of how to play games in a **human-like** manner *in general* is mostly unexplored with a few exceptions [560, 930]. In [560] it was found that standard MCTS agents can be made to perform in a much more human-like manner across different games through modifying the distributions of actions taken so as to resemble statistics of humans playing.

In the more recent study by Reed et al., [930] the generalist agent, named Gato, learns a multi-modal, multi-task, and multi-embodiment policy that enables the same transformer architecture to play several play Atari games, but even chat or caption images. Making further progress here will likely involve modeling how humans play games in general, including characteristics such as short-term memory, reaction time and perceptual capabilities, and then translating these characteristics to playing style in individual games.

14.1.2 General Game Generation and Orchestration

The study of PCG [1023] for the design of game levels has reached a certain maturity and is, by far, the most popular domain for the application of PCG algorithms and approaches (e.g., see [654, 1176, 1289, 1291] among many). What is common in most of the content generation studies covered in this book, however, is their *specificity* and strong dependency of the representation chosen on the game genre examined. For the Mario AI Framework, for instance, the focus on a single level generation problem has been very much a mixed blessing: it has allowed for the proliferation and simple comparison of multiple approaches to solving the same problem, but has also led to a clear overfitting of methods. Even though some limited generalization is expected within game levels of the same genre, the level generators that have been explored so far clearly do not have the capacity of *general* level design. We argue that there needs to be a shift in how level generation is viewed. The obvious change of perspective is to create **general level generators**—level generators with general intelligence that can generate levels for any game (within a specified range). That would mean that levels are generated successfully across game genres and players and that the output of the generation process is meaningful and playable as well as entertaining for the player. Further, a general level generator should be able

to coordinate the generative process with the other computational game designers who are responsible for the other parts of the game design.

To achieve general level design intelligence, algorithms are required to capture as much of the level design space as possible at different representation resolutions. We can think of representation learning approaches such as deep autoencoders [1218] or vision transformers [290] capturing core elements of the level design space and fusing various game genres within a sole representation—as already showcased by a few methods, such as the Deep Learning Novelty Explorer [633]. The first attempt to create a benchmark for general level generation was launched in the form of the Level Generation Track of the GVGAI competition [868]. In this competition track, competitors could submit level generators capable of generating levels for unseen games. The generators were then supplied with the description of several games, and produced levels which were judged by human judges [561]. Overall results obtained from that competition suggest that constructing competent level generators that can produce levels for any game is much more challenging than constructing competent level generators for a single game. A related effort is the Video Game Level Corpus [1105] which aims to provide a set of game levels across multiple games and genres which can be used for training level generators for data-driven procedural content generation.

While level generation, as discussed earlier, is one of the main examples of procedural content generation, there are many other aspects (or **facets**) of games that can be generated. These include visuals, such as textures and sprites; narrative, such as quests and backstories; audio, such as sound effects and music; and of course all kinds of things that go into game levels, such as items, weapons, enemies and personalities [643, 1023]. The creative facets of games are covered extensively in Chap. 9. However, an even greater challenge is the generation of complete games, including some or all of these facets together with the rules of the game. While, as covered in Chap. 9, there have been several attempts to generate games (including their rules) there have been limited efforts to build generators that attempt to generate more than two game facets [636]. An indicative example is the recent *Genie* [138] playable video generator that uses a large LLM-like transformer model to generate and play snippets of platformer levels in a general fashion when prompted with text and images. *Genie 2* was announced at the time of writing these lines. The new generator claims to offer improved visuals of 3D generated game worlds across various genres; no further details are available on that system, however. Even though systems like *Genie* seem to have some path to cover towards complete game generation they point to a future of game development by which game engines are prompted trough text, images, sounds, or some combination of those.

Multi-faceted generation systems like *Sonancia* [661, 662] co-generate horror game levels with corresponding soundscapes but do not cater to the generation of rules. A more recent multi-faceted generation system is the XLand framework [1130] that can generate simple multiplayer games within small 3D worlds. While XLand attempts to generate game rules and levels in a generalized manner, it is build for the purpose of training generalist gameplaying agents—as in [930]—and is limited to small 3D generated worlds with a generated set of rules. It is clear that the very

domain-limited and facet-limited aspects of current game generation systems result from intentionally limiting design choices in order to make the very difficult problem of generating complete games tractable. Yet, in order to move beyond what could be argued to be toy domains and start to fulfill the promise of game generation, we need systems that can generate multiple facets of games at the same time, and that can generate games of different kinds.

This process has been defined as facet **orchestration** in games [541, 630, 636]. Evidently, orchestration is a necessary process when we consider the output of *two or more* content type generators—such as visuals and audio—for the generation of a complete game. Drawing inspiration from music, orchestration may vary from a *top-down*, conductor-driven process to a *bottom-up*, free-from generation process [636]. More than a decade ago, something very much like general game generation and orchestration was outlined as the challenges of "multi-content, multi-domain PCG" and "generating complete games" [1155]. It is interesting to note that there has seemingly been limited attempts to create more general game generators since then, perhaps due to the complexity of the task. An indicative study by Karavolos et al. [541] moves towards the orchestration direction as it fuses level and game design parameters in first-person shooters via deep convolutional neural networks. The trained networks can be used to generate balanced games. A number of more recent examples including *Mechanic Miner* [238] and *Puck* [231] are detailed Section 9.6. Currently the only genre for which generators have been built that can generate high-quality (complete) games is abstract board games [133, 881]. Once more genres have been "conquered", we hope that the task of building more general level generators can begin. As we covered in Chap. 7 already, game rules and game levels are necessary content types that are largely interwoven. We thus argue that they define the building blocks of any general game generation method.

Linked to the tasks of orchestration and general game generation there are important questions with respect to the creative capacity of the generation process that remain largely unanswered. For example, how creative can a generator be and how can we assess it? Is it, for instance, deemed to have appreciation, skill, and imagination [225]? When it comes to the evaluation of the creative capacity of current generative AI algorithms a case can be made that most of them possess only skill. Does the creator manage to explore novel combinations within a constrained space, thereby resulting in **exploratory** game design creativity [98]; or, is it trying to break existing boundaries and constraints within game design to come up with entirely new designs, demonstrating **transformational** creativity [98]? If used in a mixed-initiative fashion, does it enhance the designer's creativity by boosting the possibility space for them? Arguably, the appropriateness of various evaluation methods for autonomous PCG creation or mixed-initiative co-creation [1279] remains largely unexplored within both human and computational creativity research.

14.1.3 General Affective Game Loop

It stands to reason that general intelligence implies (and is tightly coupled with) general emotional intelligence [724]. The ability to recognize human behavior and emotion is a complex yet critical task for human communication that acts as a facilitator of general intelligence [261]. Throughout our evolution, we have developed particular forms of advanced cognitive, emotive and social skills to address this challenge. Beyond these skills, we also have the capacity to detect affective patterns across people with different moods, cultural backgrounds and personalities. This generalization ability also extends, to a degree, across contexts and social settings. Despite their importance, the characteristics of social intelligence have not yet been transferred to AI in the form of general emotive, cognitive or behavioral models. While research in affective computing [877] has reached important milestones such as the capacity for real-time emotion recognition [1308]—which can be faster than humans under particular conditions—all key findings suggest that any success of affective computing is heavily dependent on the domain, the task at hand, and the context in general. This *specificity* limitation is particularly evident in the domain of games [1287] as most work in modeling player experience focuses on particular games, under well-controlled conditions with particular, small sets of players (see [715, 1017, 1018, 1289] among many). In this section we identify and discuss two core unexplored and interwoven aspects of modeling players that are both important and necessary steps towards the long-term aim of game AI to realize truly adaptive games. The first aspect is the closure of the affective loop in games; the second aspect is the construction of general models capable of capturing experience across players and games.

As stated at the start of this book, affective computing is best realized within games in what we name the **affective game loop** [1286]. While the phases of emotion elicitation, affect modeling and affect expression have offered some robust solutions by now, the very loop of affective-based interaction has been realised successfully in limited instances. Aside from a few early studies demonstrating some affect-enabled adaptation of the game [1024, 1277] the area remains largely unexplored [1286]. It is not only the complexity of modeling players and their experience that is the main hurdle against any advancement. What is also far from trivial is the appropriate and meaningful integration of any of these models in a game. The questions of how often the system should adapt, what it should alter and by what degree are not easy to answer. As most of the questions are still open to the research community the only way to move forward is to do more research in adaptive games involving affective aspects of the experience. Existing commercial-standard games that already realize the affective loop such as *Nevermind* (Flying Mollusk, 2016) are the ambassadors for further work in this area.

Assuming that the game affective loop can be successfully realized within a particular game, the next long-term and ambitious goal for game AI is the *generality* of affect-based interaction across games. The game affective loop should not only be operational; it should ideally be general too. For AI in games to be general

beyond gameplaying it needs to be able to recognize general emotional and cognitive-behavioral patterns. This is essentially AI that can detect context-free emotive and cognitive reactions and expressions across contexts and can build computational models which are grounded in a general gold standard of human behavior. So far we have only seen a few proof-of-concept studies in this direction. Early work within the game AI field focused on the ad-hoc design of general metrics of player interest that were tested across different prey-predator games [1272, 1273]. In more recent studies predictors of player experience were tested for their ability to capture player experience across dissimilar games [166, 711, 715, 1020]. Recent work in the area is based on datasets featuring human demonstrations across several games such as the AGAIN corpus [744]. A number of promising studies have been performed to test for the generality of player arousal [694, 743] and engagement [59, 886, 887] models across games. Discovering entirely new representations of player behavior and emotive manifestations across games, modalities of data, and player types is a first step towards achieving general player modeling. Such representations can, in turn, be used as the basis for approximating the *ground truth* of user experience in games. Methods from representation learning, causal modeling, and transfer learning could be of great assistance to this cause and a number of studies is already in place [694, 885, 886].

14.2 Game Engines for Game AI

Most games are developed with the help of game engines, which oversee the execution of the game code and handle a myriad things, most prominently graphics rendering and resource management, but also sound, animation, path finding, cross-platform compilation and so on. Two widely used third-party game engines are Unreal Engine and Unity 3D, though there is competition from open-source engines such as Godot, and many larger game studios have their own in-house engines. Like any product, game engines were designed at a particular time for a particular purpose, and the basic architecture bears witness to this even after a decade or two.

There are many ways in which current game engines are not well-suited to various AI methods applicable to games. Some of them are easy to rectify and have indeed been rectified to some extent. For example, there are now built-in features for logging playtraces. However, the real roadblock when it comes to implementing many types of AI methods for gameplaying is that existing game engines do not provide forward models. In other words, is not generally possible to accurately simulate the game running significantly faster than real-time. This is obviously a blocker for any type of planning algorithm. While some game engines, such as Unity, can be sped up by some multiple, the physics engine is not guaranteed to act consistently when sped up. This issue stemming from the game engines themselves is compounded by standard game software engineering practices, where animation is sometimes part of the core game loop, making it very hard to simulate the games mechanics without also simulating animation, which is costly.

There is a real opportunity for building a new type of game engine, which obeys to a Model-View-Controller (MVC) framework so that the game state is clearly separable from the presentation layer, and there are distinct interfaces for controlling the player character as well as NPCs. In such a game engine, it would be trivial to forward-simulate the game state at a much higher speed than standard gameplay, without having to animate or render anything. This would allow for real breakthroughs in game AI as planning algorithms—potentially supported by machine learning—could be used for all the different use cases for gameplaying AI discussed in Chap. 4.

14.3 Other Uses and Roles of AI in Games

The structure of this book reflects our belief that playing games, generating content and modeling players are the central applications of AI methods in games. However, there are many variants and use cases of game playing, player modeling or content generation that we have not had time to explore properly in the book, and which in some cases have not been explored in the literature at all. Further, there are some applications of AI in games that cannot be really classified as special cases of our "big three" AI applications in games, despite our best efforts. This section briefly sketches some of these applications, which may be important future research directions.

Game Representation Learning

Representation learning and computer vision can offer methods with great potential for the future of AI and games. While not a direct use of AI in games, such methods will likely help the field to move from *specific* game applications to *general* findings. As we saw throughout the chapters of this book, a number of studies have shown that it is possible to rely primarily on the pixels of gameplay footage for predicting player behavior [773], generating new playable games [138] and even for detecting player emotion with high levels of accuracy [691–693]. What is fascinating about pixel-based representation models is that they are general and can be potentially game-, task- and user-agnostic. Computer vision models are general as they can eventually represent any contextual information and any behavioral and emotional patterns by observing the game footage pixels. Such methods are also user agnostic as they do not rely on any other personal information about players beyond their gameplay (such as manifestations of emotion via the player's physiology of facial expressions). These properties make pre-trained pixel-based representation models of games ideal for learning generalized representations of game states across games with varying artistic styles [1187]. Additional modalities of the game including audio, and possibly text, can further augment the representation capacity of such models [693, 925, 1189].

Complementary to the above, self-supervised learning methods such as contrastive learning define a relatively recent machine learning paradigm which has been widely and successfully employed for learning general representations of data [609]. As

covered briefly in Part I such methods attempt to learn by processing different views of the same input that have similar representations. For example think of different views of a basketball court showing the same slam dunk competition. The views may be different in terms of camera angle, light, crowd, stadium type, and even graphical style (e.g., pixel-art vs. photorealistic) but the game state remains the same. It is only recently that contrastive learning methods started gaining interest and gradually found applications in game AI [1186, 1187] and player modeling [885, 1032]. Such modeling methods assume that information about the player is an inherent property of the game's context and thus can be fused and learned in a contrastive learning manner. We foresee a bright future of such methods for learning generalizable game representations [925, 1188] that can be used for any downstream task such as playing, generating content and modeling players.

Text-to-Game

Given the recent successes of contemporary large scale language (or media) models such as GPT-3 [342] and GPT-4 [831] one could imagine and represent games as forms of text that AI could consider and process accordingly. LLMs appear to offer promising, yet largely unexplored, methods for anything that could potentially be generated in games via text [365]. Some early attempts have demonstrated that text-to-game (i.e., text-to-levels primarily) is possible but lacks certain degrees of controllability. Often, the power of the LLM can be amplified (and its shortcomings mitigated) by dividing up the generation process in multiple steps, and using other algorithms to check and repair its outputs, as is done in e.g. the Word2World game generator [797].

Beyond the obvious use of LLMs as a PCG method that we detail in Section 8.5.4 we envision the use of LLMs for learning and interweaving game context and human feedback. In principle, an LLM that is hosted in a game engine could predict any player state transition including states where "the game is more engaging" and thereby change the surrounding environment or spawn a number of enemies to get the player to a supposedly more engaging state. Learning to generate affect transitions via foundation models builds on the experience-driven procedural content generation paradigm [1289] but with a contemporary touch. Generative affect-based AI methods including GPT variants and diffusion models [961] will likely expand beyond text-to-(affect)-image applications [363], to text-to-video, text-to-3D models and finally text-to-games with prescribed affect patterns [365]; *text* can be replaced (or complemented) by other modalities including images and sound.

Compared to the very specific game state transitions that an LLM could use to play a game, the relationship between the game context and player experience appears to be more general [744]. One can thus argue that such relationships could potentially be learned from many games rather than one. We could then imagine the future development (or fine tuning) of large foundation models that are capable of representing and inferring affect transitions based on in-game observations and affect

demonstrations. Referring to our earlier discussion about the general affective game loop, it remains to be seen to which degree we can build a generalized computational game experience designer, and how.

Critiquing Games

Can AI methods meaningfully judge and critique games? Game criticism is hard and generally depends on deep understanding of not only games but also the surrounding cultural context. Still, there might be automated metrics that are useful for game criticism, and can provide information to help reviewers, game curators and others in selecting which games to consider for reviewing in app stores. The ANGELINA game generation system is one of the few examples towards this direction [233] in which AI generates an overview of the game to be played. You could also imagine using VLMs or LLMs to go through and comment on playthroughs of a game (maybe generated by AI players), and try to relate them to current events or cultural streams. It is hard to image that we would be able to create a fully automated game critic worth listening to, but you could definitely imagine systems that could provide useful input to game critics.

Hyper-Formalist Game Studies

Game studies scholars study games much like literature scholars study literature, that is, in many different ways. But, as games are computational artifacts, we can use algorithms to help us understand them. AI methods can be applied to corpora of games in order to understand distributions of game characteristics. For example, decision trees can be used to visualize patterns of resource systems in games [526].

There are many other ways of using game AI for game studies that are still to be discovered. You could imagine using a large variety of game-playing agents to characterize games or game levels from various perspectives. Alternatively, you could let agents explore games and attempt to validate (or simply invalidate) a theoretical model of games and player experience. You could also argue that building a generator for a game, or levels in a game, is a way of manually modeling the design of that game. And you could further argue—though not everyone would agree—that you have not really understood game design until you have built algorithms that can design games, in the same way you have not really understood sorting until you have studied sorting algorithms.

Game Directing

The outstanding feature of *Left 4 Dead* (Valve Corporation, 2008) was its AI director, which adjusted the onslaught of zombies to provide a dramatic curve of challenge for players. While simple and literally one-dimensional (only a single dimension of player experience was tracked), the AI director proved highly effective. There is much room for creating more sophisticated AI directors; the experience-driven PCG framework [1289] is one potential way within which to work towards this. Another

metaphor for what a successful game director could be is the Dungeon Master of a pen-and-paper role-playing game, who knows his players intimately and constantly readjusts the narrative to keep the game session interesting.

At some point, the vision of AI directors and general PCG overlap. You could imagine future AI directors to be based on much more complete models of both performance, preferences, and experience of players, which could then be modified or prompted with the behavior (and possibly objective and profile information) of a particular player. That director may then choose to serve or adapt content from the game that the player did not even know they wanted. It may also be able to generate such content on the fly, to suit particular players.

Virtual Cinematography

Virtual cinematography refers to algorithmic solutions for selecting what views of the game to present to the player as they are playing, or alternatively to spectators [23, 150, 321, 435, 508, 969]. This task is strongly related to AI game directors (as cinematography is strongly related to direction in movies), but differs in that virtual cinematography does generally not directly affect the game world, only how we perceive it. The interplay between interactive narrative, game narration, and virtual cinematography is obvious but not as well explored yet; their relationship is discussed briefly in Part III and in particular in the section dedicated to narrative generation. Once foundation models of the future (say from the LVM family) are able to synthesize 3D playable demos with interactive narrative capacities, virtual cinematography will be tasked to optimize shots for particular moments of the game story under specific experience curves that are tailored to the player.

Creative Inspiration

While designing a complete game that actually works likely requires a very complex generator, it can be simpler to generate an idea for new games, that are then designed by humans. AI methods can play the role of a design assistant on a conceptual level where the algorithm provides high-level guidance. The provided guidance is not a complete game; instead, it is intended to *inspire* the designer to adapt and curate the AI output—e.g., concept idea in text, concept visual art, or musical theme—so that it fits the vision and the intent of the designer. Creative ideation tools range from simple word-recombination-based tools implemented as card games or Twitter bots, to elaborated computational creativity systems such as the *What If-Machine* [657] and contemporary LLMs [365].

We argue that any interaction with existing LLMs or Large Multimodal Models may act as a conceptual inspiration to a design process according to which a designer may prompt certain specifications and receive a number of suggestions as a response. It is no surprise that LLMs are used for brainstorming and concept development in many creative processes [1216] including game design. A recent research effort in this area [198] realizes game design assistance through a tool that suggests high-level conceptual game features to a designer when prompted with a specific game

description. MeLITA [1304] is another recent LLM-based mixed-initiative tool that creates game titles, short descriptions, and concept art "banner" for imaginary games that could define concepts for game designers. Given the conversational nature of many generative AI algorithms nowadays, we expect a number of future research studies to focus on the development of efficient and controllable mixed-initiative tools for inspiration and concept design.

At the time of writing LLMs are predominately used as conceptual assistants or as content production assistants [365]. In other words, the assistance is offered either in early design phases or directly as complete and finalized outcomes. Another largely underexplored role of AI as a design assistant, however, is that of a colleague that iteratively refines a game with a human throughout a co-creative design process [1279]. We argue that the conversational capacities of LLMs are particularly well-suited for assistance throughout the design process and extend far beyond the generation of a concept or a final output.

Chat Monitoring

In-game chats are important in many online multi-player games, as they allow people to collaborate within games and socialize through them. Unfortunately, such chats can also be used to threaten or abuse other players. Given the very large volume of chat messages sent through a successful online game, it becomes impossible for the game developers to curate chats manually. In the efforts to combat toxic behavior, some game developers have therefore turned to machine learning. Notably, Riot Games have trained algorithms to recognize and remove toxic behavior in the MOBA *League of Legends* (Riot Games, 2009) [685] and more recently, Ubisoft run some similar studies in the MOBA *For Honor* (Ubisoft, 2017) [178]. There have also been attempts to use machine learning to detect sexual predators in game chats [411].

Game Narrator (Commentator or Reteller)

We envision AI entities that are able to narrate a sequence of story plot points or events and serve games as commentators or retellers. Such in-game agents could act, for instance, as sports *commentators* in the *FIFA* (EA Sports, 1993–2022) game series by considering critical micro in-game events (e.g., passes, goals), and even observe macro events such as player defense strategies and offensive tactics. Similar agents, can, alternatively, act as *retellers* [317] that exclusively narrate past events such as a game session or an entire quest. Some early versions of AI entities in the role of game commentators are already in place [418]. Indicatively, LSTM models have been trained to generate commentary scripts in racing games [496] and in side-scrolling games such as *Getting Over It With Bennett Foddy* (Bennett Foddy, 2017) [625]. More recent approaches, instead, tend to rely on LLMs [933], for instance, to automatically comment upon *League of Legends* (Riot Games, 2009) games. Beyond the few aforementioned studies, however, research on AI as a game narrator remains limited. We expect algorithmic innovations to yield methods (most

likely based on LLMs) that would be able to consider contextual in-game features and generate meaningful retellings or commentaries highlighting critical game events [365].

AI-Based Game Design

Throughout most of the book, we have assumed the existence of a game or at least a game design, and discussed how AI can be used to play that game, generate content for it or model its players. However, one could also start from some AI method or capability and try to design a game that builds on that method or capability. This could be seen as an opportunity to showcase AI methods in the context of games, but it could also be seen as a way of advancing game design. Most classic game designs originate in an era where there were few effective AI algorithms, there was little knowledge among game designers about those AI algorithms that existed, and CPU and memory capacity of home computers was too limited to allow anything beyond simple heuristic AI and some best-first search to be used. One could even say that many classic video game designs are an attempt to design around the lack of AI—for example, the lack of good dialog AI for NPCs led to the use of dialog trees, the lack of AIs that could play FPS games believably and competently led to FPS game designs where most enemies are only on-screen for a few seconds so that you do not notice their lack of smarts, and the lack of level generation methods that guaranteed balance and playability led to game designs where levels did not need to be completable. The persistence of such design patterns may be responsible for the relatively low use of interesting AI methods within commercial game development. By starting with the AI and designing a game around it, new design patterns that actually exploit some of the recent AI advances can be found.

Several games have been developed within the game AI research community specifically to showcase AI capabilities, some of which have been discussed in this book. Three of the more prominent early examples are based on Stanley et al.'s work on **neuroevolution** and the NEAT [1081] algorithm: *NERO*, which is an RTS-like game where the player trains an army through building a training environment rather than controlling it directly [1079]; *Galactic Arms Race*, in which weapons controlled through neural networks are collectively evolved by thousands of players [431, 432]; and *Petalz*, which is a Facebook game about collecting flowers based on a similar idea of selection-based collective neuroevolution [944, 945]. Another more recent game example designed to primarily demonstrate evolutionary principles is *Darwin's Demons* (Polymorphic Games, 2017), an arcade style space shooter in which the player battles a population of evolving aliens as in *Space Invaders* (Atari, Inc., 1978). The aliens' shape, fire projectiles, and behaviors are represented as chromosomes. Those alien NPCs are assessed through their in-game performance against a player and the best of those evolve so as to reproduce the next generation of aliens to combat the player [1072]. Beyond evolution, a number of other games have been built to demonstrate various types of adaptation mechanisms, such as *Infinite Tower Defense* [39] and *Maze-Ball* [1285].

Within **interactive narrative** it is relatively common to build games that showcase specific theories and methods; a famous example is *Façade* (Procedural Arts, 2005) [721] and another prominent example is *Prom Week* [729]. The former is an interactive fiction game that demonstrates the potential use of NLP in games via the interpersonal relationships developed among the game characters and the player. The latter is a social simulation game demonstrating the capacities of the *Comme il Faut* [731] social AI tool; *Prom Week* is a game that simulates social interactions like those of *The Sims* (Electronic Arts, 2000–2024) series but with more elaborated game characters and detailed dialogues. Treanor et al. [1184] have attempted to identify AI-based game design patterns, and found a diverse array of roles in which AI can be or has been used in games, and a number of avenues for future AI-based game design. In recent years we witness a gradual adoption of LLMs and foundation models at the core of game design. Indicative examples we discussed thoroughly in Part III include but are not limited to the *AI Dungeon* (Latitude, 2019) text-based adventure game, the *AI people* sandbox simulation game currently developed by *GoodAI* [389], the game *1001 nights*—in which a player co-creates a story with an LLM [1106]—and the game *Gandalf*[1] in which players are required to trick an LLM to reveal a password for them.

14.4 Summary

In this chapter we went through directions that we view as critical and important for the advancement of the AI and games field. Initially we have argued that the **general intelligence** capacity of machines needs to be both explored and exploited to its full potential (1) across the different *tasks* that exist within the game design and development process, including but absolutely no longer limited to game playing; (2) across different *games* within the game design space; and (3) across different *users* (players or designers) of AI. We claim that, thus far, we have underestimated the potential for general AI within games. We also claim that the currently dominant practice of only designing AI for a specific task within a specific domain will eventually be detrimental to game AI research as algorithms, methods and epistemological procedures will remain specific to the task at hand. As a result, we will not manage to push the boundaries of AI and exploit its full capacity for game design. We are inspired by the general gameplaying paradigm and the recent successes of AI algorithms in that domain and suggest that we become less specific about all subareas of game AI including player modeling and content generation. Doing so would allow us to detect and mimic different general cognitive and emotive skills of humans when designing games. It is worth noting, again, that we are not advocating that all research within the game AI field focuses on generality right now; studies on particular games and particular tasks are still valuable, given how little we still understand and can do. But over time, we predict that more and more research will

[1] https://gandalf.lakera.ai/.

focus on generality across tasks, games and users, because it is in the general problems that the interesting research questions of the future lie. It seems that we are not alone in seeing this need as a growing number of researchers have argued for the use of various game-related tasks (not just game playing) to be used in artificial general intelligence research [1320].

The path towards achieving general game artificial intelligence is still largely unexplored. For AI to become less specific—yet remain relevant and useful for game design—we envision a number of immediate steps that could be taken: first and foremost, the game AI community needs to gradually adopt an **open-source** accessible strategy so that methods and algorithms developed across the different tasks are shared among researchers for the advancement of this research area. For the algorithms and methods to be of direct use particular technical specifications need to be established—such as those established within game-based AI benchmarks—which will maximize the interoperability among the various tools and elements submitted. Early examples of benchmarked specifications for the purpose of general game AI research include the **general video game description language** and the puzzle game engine PuzzleScript.[2] Finally, following the GVGAI competition paradigm, we envision a new set of competitions rewarding general player models, AI-assisted tools and game generation techniques. These competitions would further motivate researchers to work in this exciting research area and enrich the database of open-access interoperable methods and algorithms, directly contributing to the state of the art in computational general game design.

Beyond generality we put a focus on the importance of game engine development for the advancement of game AI in the future. Importantly, in this chapter we also looked at the extensibility of AI roles within games. In that regard, we outlined a number of AI uses and roles that are underrepresented currently but nevertheless define very promising research frontiers for game AI. These include the uses of AI as a game representation learning process, a language-based foundation model, but also the roles of a game critic, a game studies formalist, a director, a virtual camera operator, a creative designer, and game narrator. Further, we view the placement of AI at the very center of the design process (AI-based game design) as another critical research frontier. In the next and final chapter of this part, and the book as a whole, we discuss the numerous ethical considerations of the various uses of AI in and for games.

[2] http://www.puzzlescript.net/.

Chapter 15
Ethics of AI and Games

Abstract This chapter outlines some of the ethical concerns and challenges in AI applied to games. There are many potential ethical challenges, including the ownership of data and models, induction of artificial emotions, the potential to create and reinforce addictive playing behavior, and the computational reinforcement of stereotypes. The chapter is structured around the affective loop, discussing ethical issues within affect elicitation, sensing, detection, and adaptation. There is also a brief discussion about the role of games in creating artificial general intelligence and potential concerns associated with this.

Like all technologies, artificial intelligence, including game AI, can be used for many purposes, some of them nefarious. Perhaps even more importantly, technology can have ethically negative or at least questionable effects even when there is no malicious intent. The ethical effects of using AI with and in games are not always obvious, and the topic is perhaps not receiving the attention it should. The final chapter of this book is dedicated to general **ethical considerations** of AI and games and the ways in which game AI intersects with ethical questions. The chapter relies heavily on a recent article on the topic [745] and is co-authored by its first author, David Melhart. For general AI research issues, ethics and values refer to the Asilomar AI Principles[1] developed in conjunction with the 2017 Asilomar conference, the UNESCO Global AI Ethics and Governance Observatory,[2] and the recent AI act by the European Commission.[3]

15.1 Overview

After going through the earlier chapters of this book it should be obvious to the reader that video games are key to our understanding of human behavior. Games are vastly popular, they offer multi-modal ways players can interact with them, and they

[1] https://futureoflife.org/ai-principles/.

[2] https://www.unesco.org/ethics-ai/en.

[3] https://artificialintelligenceact.eu/.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2_15

can elicit various emotional patterns according to a player's style and preferences. While ethical values such as transparency, trustworthiness and responsibility are core aspects (and requirements) of AI systems in other domains [513], video games present unique challenges in terms of ethics. A number of challenges may hinder transparency [762] and raise several ethical concerns including the dark patterns often interwoven in game design [1301], the predatory monetization strategies [566], and the black-box nature of game development.

Player modeling is perhaps the part of game AI where the ethical questions are most direct, and perhaps most urgent. There is an ongoing and vigorous debate about the mass collection of data about us both by government entities—such as the US National Security Agency, the United Kingdom's GCHQ or the EU's GDPR and AI act initiatives—and private entities such as Google, Amazon, Facebook and Microsoft [116]. With methodological advances in deep learning and large-scale models, it is becoming possible to learn more and more about individual people from their digital traces, including inferring sensitive information and predicting behavior. Given that player modeling involves large-scale data collection and mining, many of the same ethical challenges exist in player modeling as in the mining of data about humans in general [762].

Inspired by [745], in this chapter we attempt to address the ethical considerations of game AI tools and methods through the lens of *affective computing* with a particular focus on *player modeling* [1288]; the reader is also referred to Part IV for further details on player modeling. We structure the discussion of AI ethics in games around the notion of the **affective game loop** [1286, 1287] (see Fig. 15.1) which studies the relationships between player experience elicitation, player emotion detection and prediction, and finally in-game adaptation and personalization. The *affective game loop* can support AI methods to generate personalized aspects of games that are tailored to one's desired experience or playstyle. As we have seen already throughout the chapters of this book, such aspects may include agent behavior but also game content such as levels, audio and visuals [636, 1289]. It is important to note that there are several ethical issues that do not relate to player modeling directly. These may include aspects of generative AI that are not necessarily associated with players such as copyright issues in LLM-generated content [327]. While we highlight the relationship between content generation and human play throughout this chapter, the focus of the chapter remains a player-centered one.

Following the core phases of the affective game loop, the chapter is structured as follows. Section 15.2 covers aspects of *elicitation* and the ways dark patterns are used for manipulating and reducing the players' emotional agency in harmful or exploitative ways. Then, Sect. 15.3 focuses on *sensing* and potential challenges of malicious action in games as derived from the fine line between privacy and control. Moving on, Sect. 15.4 discusses player experience *detection* and the arising complexity of transparency in limited information systems such as games. Finally, Sect. 15.5 attempts to address the potential challenges of data and model ownership during in-game *adaptation*. The chapter ends with a discussion on several other issues related to AI ethics in games—such as AI algorithmic biases, fairness, and in-game toxicity—that cannot fit within the structure of the affective game loop. Throughout the chapter

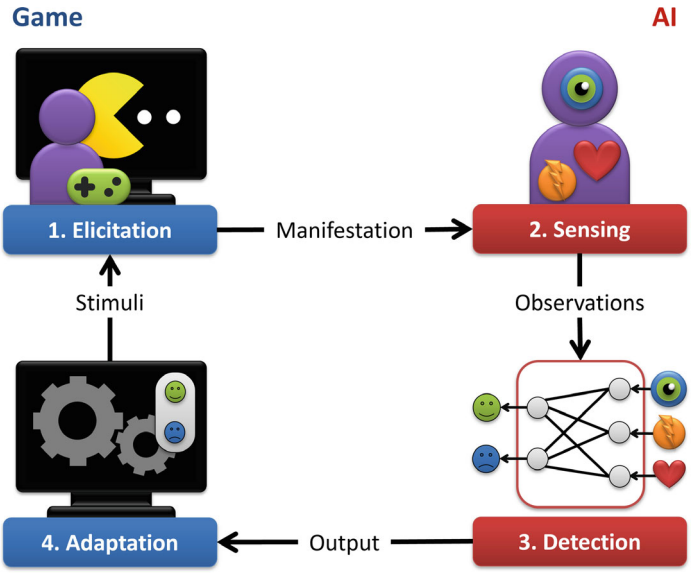


Fig. 15.1 The *Affective Game Loop* [1286, 1287]. The loop relies on the game's parameter space (stimuli) to *elicit* an emotional response. This response is *sensed* by an AI model that *detects* change(s) in the player's emotional state. The output of the affect model can be used to *adapt* the game' parameter space (agents and content) and generate a new set of stimuli for the player. Image inspired by [1286]

we make references to the *AI virtues* of *Responsibility*, *Transparency*, *Auditability*, *Incorruptibility*, and *Predictability* introduced by Bostrom and Yudkowsky [110].

15.2 Elicitation—Boundaries of Artificially Induced Emotions

The core challenges we will address in this section are the ownership and autonomy over a player's own emotions. The *Elicitation* phase of the affective game loop raises a number of ethical concerns associated primarily with the boundaries of artificially induced emotions. The game design practice coined as *dark design patterns* [1301] is an indicative example of using games to manipulate players' emotional patterns. Such design patterns usually take the form of a user interface carefully crafted to deceive players into harmful decisions such as buying overpriced items in a game. While the use of dark design principles nowadays is still rather common, one could argue that the future does not look promising either: the advent of large-scale player data analytics and AI processes, will likely increase the risk for even more elaborate and dangerous dark design patterns [566].

A few years ago, the dominant monetization strategy adopted by many games was based on *loot boxes* [297, 381, 457]. *Loot boxes* are virtual items that can be redeemed for other random items that, in turn, provide some value to the players of a game [1306]. Recent years have seen a diversification of monetization strategies with the most popular of them all currently being the *battle pass* (or *season pass*). In contrast to earlier strategies (e.g., premium subscription) the *battle pass* concept does not offer any immediate or even tangible reward to its owners. Instead, players buy time-limited access to content updates [874]. As a result the focus of monetisation has shifted from a commodity to a service [518]. The early loot box strategy was designed to trigger similar cognitive and affective patterns as those elicited in gambling [297, 381]. The recent battle pass systems build on societal pressure principles [567] and elicit feelings of "missing out" [874]. Many recent online games including *Fortnite* (Epic Games, 2017), *Apex Legends* (Respawn Entertainment, 2019), *Fall Guys* (Mediatonic, 2020) and *Overwatch 2* (Blizzard Entertainment, 2022) have incorporated such monetization practices by which one can spend real money to buy in-game currencies that can, in turn, be exchanged on content upgrades and battle passes.

What makes things particularly concerning is when AI algorithms exploit any given monetization strategy, such as loot boxes and battle passes, to target people that are more prone to spending. Predictive models such as those used for the estimation of churn [871] can, in principle, be used to detect and target potential excessive spenders—also know as *whales* [204]. Such player modeling methods, however, form the foundations of more responsible systems that detect potential risks and ensure, to the degree possible, a responsible behavior against overspending. Affective game computing [1286] can be particularly useful to detect certain psychosocial aspects of addiction—such as mood modification and tolerance [408]—and, at the very least, flag users at-risk.

Beyond the potential risk of overspending (money, time or even attention) large-scale computational models of AI can also encode biases—such as gender biases or offensive stereotypes—which, in turn, might yield harmful generated content for players [677, 1033] that are not intended by the game designer. A recent indicative example of the complex relationship between large scale generative AI systems, privacy and content moderation is the case of Latitude, developer of *AI Dungeon* (Latitude, 2019), a text-based adventure game based on GPT variants [147, 342]. Latitude banned generated stories containing sexual content and child pornography [1046] even though much of the questionable content was generated deliberately by players themselves. A number of players raised concerns about decisions that enforce strict moderation and monitoring players' privately generated stories. On the one hand, such a decision gave players the full *responsibility* of the generated content even though it was co-created with algorithms featured in *AI Dungeon*. On the other hand, any human moderation of private content raises privacy concerns. This controversial example shows that despite the best intentions of a game developer, a certain lack of *transparency* and *responsibility* [110] might yield undesired outcomes for both the company and their players. What makes this example particularly interesting is that Latitude is not the developer of any of the underlying LLMs so it was far from trivial

how *AI Dungeon* can be *audited* in an effective and fair manner. Given that there was no established *responsibility* for *AI Dungeon* as a whole, the blame fell primarily to the players that co-created offensive or inappropriate content with the model.

In summary, AI-assisted systems within games may potentially impact the player in harmful ways. While video game developers may rely on player-adaptive methods to build their monetization strategy they may (intentionally or unintentionally) facilitate forms of addiction by eliciting certain affective patterns. Moreover, generative AI systems within games are still rather unreliable and may yield unwanted content without the player's (or even the designer's) consent since the black-box nature of modern large-scale media models limits their *auditability* [103]. In our view such systems should clearly set the bounds of responsibility for anything the model outputs and importantly, offer tools to players to tag, report, and mitigate unwanted content.

We wish to end this section with a positive note and call for action. Leveraging predatory monetization strategies has been a dominant pattern in the games industry; not all developers, however, have adopted such strategies. Most notably, *Six to Start*—the developer of the popular *Zombies, Run!* (Six to Start, 2012) mobile—actively demonstrated that games can be successful without putting any psychological pressure on their players. Beyond their ethically compliant game design ideas, Six to Start took the extra mile and made a firm public stance against dark design pattern practices [467].

15.3 Sensing—Privacy and Control

The next phase of the in-game experience loop we focus on is *sensing*. Sensing is the process of observing and capturing any manifested emotion. Unsurprisingly, the main ethical issue associated with any sensing method is that of *privacy*—i.e., when, how, and what kind of data is being captured. Most, if not all, users involved in any kind of affective interaction wish to be informed about any underlying data collection occurring and the potential control they have over the sensors that monitor them [935]. Up until recently several affect-based HCI applications failed to address such user needs as the data and data pipeline would be kept secret from users [433]. Game companies are no exception to this; a recent study surveying the data collection pipelines of the video games industry reveals that data is often collected without the user's consent [591]. With or without the consent or even knowledge of the subject, some of the information collected can be of a private and sensitive nature. The recent AI act of the EU, however. Will likely put an end to the practice—in part at least—as it requires video games to inform their players when data is collected to infer their emotional state.

A critical question to be asked here is: what is sensed ultimately? The majority of collected data seems to be of in-game behavioral nature. Such data can be used to predict content consumption and even spending habits which can clearly fuel harmful policies. Instead such data can be used to harmlessly predict player skill and in-game preferences. But are such predictions truly harmless? While player modeling tasks

such as matchmaking or dynamic difficulty adjustment are there to act beneficially for the player experience, they have led to certain concerns in the past. In a recent lawsuit, Electronic Arts was accused to use difficulty adjustment to influence player spending on loot boxes [1198]. While the lawsuit was dismissed [67] such instances highlight the public distrust towards systems that collect player data.

Beyond in-game behavioral data, game companies are nowadays embedding player data derived from sensors at an increasing rate [591]. The multiple modalities of player data one can track during play include gaze tracking and voice data [590], GPS positional information, and peripheral signals from accessories such smartwatches and phones. The more our world becomes interconnected, the easier it becomes to triangulate information among linked data and infer the identity of players by using a wealth of information. Such information includes in-game data, preferences and distinct play patterns [696, 787], but also a player's username and GPS location. All this data processing capacity we have available nowadays raises the question of whether the data we leave as players can be truly anonymized after all [591].

It should be clear by now that it is becoming increasingly possible (and even practicable) to infer various real-life traits and properties of people from their in-game behavior. For example, Yee and colleagues investigated how player choices in *World of Warcraft* (Blizzard Entertainment, 2004) correlated with the personalities of players. They used data about players' characters from the Armory database of *World of Warcraft* (Blizzard Entertainment, 2004) and correlated this information with personality tests administered to players; multiple strong correlations were found [1296]. In a similar vein, a study investigated how players' life motives correlated with their *Minecraft* (Mojang, 2011) log files [177]. That research used the life motivation questionnaires of Reiss [932], and found that players' self-reported life motives (independence, family, etc.) were expressed in a multitude of ways inside constructed *Minecraft* (Mojang, 2011) worlds. Using a very different type of game strong correlations have been found between playing style in the first-person shooter *Battlefield 3* (Electronic Arts, 2011) and player characteristics such as **personality** [1132], age [1131] and nationality [88]. As a result, one may build player profiles to infer the gender, age, socioeconomic status, and interests of players, which can, in turn, feed harmful models exacerbating behavior such as gambling and excessive spending. It is also entirely plausible that similar methods could be used to infer sexual preferences, political views, health status and religious beliefs. Such information could be used by advertising networks to serve targeted ads, by criminals looking to blackmail the player, by insurance companies looking to differentiate premiums, or by malevolent political regimes for various forms of suppression. We do not know yet what can be predicted and with what accuracy, but it is imperative that more research is needed on this within the publicly available literature; it is clear that this kind of research will also be carried out behind locked doors.

Let us end this section with a positive tone though. Note that any model incorporating behavioral or emotional data of players can be used to enhance player experience directly but also indirectly through the moderation of content and interaction aspects. As detailed in the study of Sect. 12.2, Canossa et al. [178] presented a robust method

to flag the occurrence and severity of toxic and emotionally abusive behavior in *For Honor* (Ubisoft, 2017). As toxic players often try to find new ways to circumvent regulations, upcoming automatic flagging systems could potentially make games a safer place for their players. The aforementioned study is a good indicative example of how player modeling applications can be used to enhance the *predictability* of the system towards its users and how we can use AI in a *transparent* fashion to both deliver clear value to players and improve the game experience.

15.4 Detection—Transparency in Limited Information Systems

The third phase of the affective game loop [1286] is *Affect Detection* which refers to the computational processing and prediction of certain aspects of affect [164] in games. We argue that the core ethical challenge associated with affect detection in games is that of the *transparency* of deployed models towards users [433, 762]. In an ideal world, players should be informed in a clear and comprehensive way about the output and the goal of any algorithm that processes their affective and behavioral patterns, to preserve the system's *transparency* and *predictability*. The sad truth, however, is that the majority of video game companies do not disclose to their players that they are being modeled, let alone informing users about their system's predictions. The recent AI act will likely force a change to this lack of transparency as it requires emotion-processing systems to inform their users about the underlying process.

Another truth about games is that they often withhold information from players intentionally. This strategy allows games to create uncertainty, decrease cognitive load, construct challenges for their players [4, 581, 1233]. And ultimately yield greater player satisfaction [4]. Because uncertainty is so deeply interwoven in game design processes, it is not trivial to mitigate any emergent issues of *transparency*.

When it comes to modeling player behavior and affect, game developers must also cater for potential inherent bias present in the AI algorithms their game uses. Anytime a model is trained on a dataset, we run the risk of reproducing stereotypes within that dataset. Arguably the most common cause attributed to algorithmic bias is a biased dataset [372, 679, 826, 1294] that is solicited from a skewed (or non-representative) population or it features biased historical data [679, 826]. For example, it has been shown that word embeddings trained on standard datasets of the English language reproduce gender-based stereotypes [161]. An infamous instance of algorithmic bias is *Google Photos'* image processing algorithm that mislabeled pictures featuring black people as "gorillas" [1217]. The algorithm was obviously not created to be racist; instead, the issue stems from the lack of diversity in the dataset that was used to train the model. Models used in the video game industry can fall in similar pitfalls, some of which are particularly tied to games. As with any other dataset, game datasets can be skewed towards atypical players, yielding an

unfair distribution of the gamers' population as a whole. The same effects could be present when modeling player preferences and behavior, and the model might learn to reproduce prejudiced conceptions regarding gender, race, etc. Such problems can be exacerbated or ameliorated by the tools made available to players for expressing themselves in-game. For example, Lim and Harrell [649] have developed quantitative methods for measuring and addressing bias in character creation tools. Another example of in-game bias would be discrimination towards a particular emotional response. Systems monitoring toxic behavior, for instance, are often based on high-level aggregated data on in-game actions and chat interactions [178]. These toxicity modeling approaches might mislabel good-faith players who do not conform to certain behavior or communication patterns. This mislabeling occurs when models are trained on data from a Western, neuro-typical audience, and therefore expect behavioral and emotional responses based on this population. Another infamous example of mislabeling within games is Julias Jackson—an autistic boy on the *Xbox Live* ecosystem [330]—whose supreme (outlier) performance was labelled as cheating.

While it is entirely possible, and advisable, to use AI methods to promote ethical behavior and uphold ethical values there are important ethical considerations in how they should be deployed. The aforementioned examples reveal the fragility levels of many AI-based automated labeling systems when they are tasked to predict outside of their trained distributions. For example, a model that has been trained to recognize hate speech might also react to normal in-game jargon; setting the right decision threshold might involve a delicate tradeoff between ensuring a welcoming game environment and not restricting communications unduly. It therefore becomes critical that we develop new labeling tools and we advance our know-how on the best practices of organizing and labeling our data. Even though it is relatively easy to rely on user reporting, human feedback comes with a number of limitations (see Sect. 2.4). Unfortunately human feedback can often augment other underlying biases and result in incidents where men, for instance, accuse women of cheating just because they manage to outperform their peers [32].

Once again we would like to end this section dedicated to ethical considerations of emotion detection with a positive note. A growing number of industry leaders are nowadays well-invested in creating fulfilling experiences that promote their player's well-being. Indicatively, companies such as Nintendo [514], Ubisoft [740] and Riot Games [775] collaborate with academia with the aim to study players' well-being, to combat toxicity, and even evaluate content moderation. Importantly, these studies also expose some of the issues related to algorithmic and data biases lending *transparency* and *auditability* to the industry. Arguably, such industry-academia collaborations offer good examples of how data transparency can bring value to industry players and, ultimately, the games they make.

15.5 Adaptation—Ownership of Data and Models

Adaptation is the final phase of affect-based interaction that essentially closes the affective game loop. This phase is responsible for adjusting the game's parameters to certain player experience requirements as specified by a player experience model [1289]. Much of the research within game AI is concerned with adaptation of games, with the experience-driven PCG framework being perhaps the most complete account on how to combine player modeling with procedural content generation to create personalized game experiences [1289, 1290]. It is not clear, however, that it is always a good thing to adapt games to players. The "filter bubble" is a concept within discussion of social networks which refers to the phenomenon where **collaborative filtering** ensures that users are only provided with content that is already in line with their political, ethical, or aesthetic preferences, leading to a lack of healthy engagement with other perspectives. Excessive adaptation and personalization might have a similar effect, where players are funneled into a narrow set of game experiences.

The core ethical question that arises when we are faced with any human-centered adaptive system—such as the affective game loop—is the question of *ownership*. Particularly within games, it is unclear how much ownership players have over any models built on their data. Even though players, in principle, should have rights to control their own data [1215], in reality, information inferred from large datasets is often exempt from legal protection afforded to first-hand personal data [1227]. It is also currently not clear what game developers can (or are allowed to) do with player data as some of this data can be used to recreate aspects of the player's behavior; this is the case for e.g., the *Drivatar*s in Microsoft's *Forza Motorsport* series, and more generally for agents created according to the **procedural persona** concept [459]. Will the game allow other people to play against a model of you, i.e., how you would have played the game? If so, can it identify you to other players as the origin of this data? Does it have to be faithful to the behavioral model of you, or can it add or distort aspects of your playing behavior?

Additional issues emerge when it comes to the ownership of the player models themselves. At the moment of writing this book, a key challenge with the existing IP protection framework is that the training data and the trained model are not as well separated or modular as the guidelines suggest [1227]. Importantly, in many instances it is entirely possible to reverse engineer the model and extract information about the training set per se, thereby, extracting personal information about the subjects included in the training set [1182]. Fortunately, there are direct ways to address this issue in an algorithmic fashion. For instance, current research in the field of *machine unlearning* will likely offer methods that will be able to remove knowledge from a trained model as if a given datapoint was never part of the training set [114, 402, 415, 1011]. For *machine unlearning* to be realized, however, the owner of the model most likely has to retain the raw training data. While not all unlearning methods require dataset storage [1011], removing a particular datapoint has still to be verified to preserve the *predictability* of the model. Despite the fact that unlearning a model would most certainly add a computational overhead to the

model owner, it is still more efficient compared to training a model from scratch. Importantly, the preference toward unlearning methods becomes particularly more appealing (cost- and resource-wise) the larger the model grows.

While the ethical challenges of small-scale models can potentially be addressed through methods such as machine unlearning, far more complex ethical questions arise through large-scale world models or **foundation models** [103]. An increasing number of games nowadays use various foundation models as their AI backbone but the game developer has neither access to the source code nor the original data that trained these models. As a result the lines of *responsibility* are somewhat blurred as in the case of *AI Dungeon* (Latitude, 2019) discussed earlier. Moreover the ethics of constructing training datasets for foundation models is highly questionable, as training data is often scraped directly from the internet. When it comes to the players and users of these models, they normally don't have a way to revoke their participation for training of fine tuning these models and their data is normally discarded or kept secret.

A general ethical concern associated with foundation models is **scaling maximalism**, i.e., the larger the model is, the better it performs [1173]. Larger models, however, require more data and more compute to be trained upon. The very fact that large-scale models, in the billions or trillions of parameters, can only be constructed using an immense amount of computational and human resources makes the market of training such models a monopoly (or oligopoly at best). This trend can already be seen in language-based models, where the current dominance of the closed-source GPT-3 and—increasingly more popular—GPT-4 [831] models imply that any new game AI applications have to subscribe to the black-box rules of such systems.

Even though the games industry has been generally slow to react to growing concerns about ownership and AI trustworthiness, the landscape is gradually changing. A recent inspiring example of such initiatives is Microsoft's *Xbox Transparency Report* [757]. In that report, Microsoft provides data analyses and reflections over statistics about their content moderation policies. Such policies have been introduced in an ongoing effort to increase both the *reliability* and *transparency* of their Xbox ecosystem. It is our hope that initiatives from large industry players such as Microsoft will inspire the games industry at large to actively participate and collectively improve the ethical standards of making and playing games.

15.6 Other Issues and Concerns

There are a number of ethical issues that do not fit comfortably into the structure of this chapter. In this last section of the chapter we want to, at least, mention a few of these concerns and provide some pointers for further reading.

All games train you, in some way, to perform the actions required to win the game. Well-designed games are built on a set of learning experiences that introduce gradually harder versions of the same challenge to the player. Arguably, this learning process is a core element of what makes games fun [581]. While many games

feature in-game challenges as representations of peaceful real-life activities such as gardening in *FarmVille* (Zynga, 2009) or warehouse stocking in *Sokoban* (Thinking Rabbit, 1982), more games represent some kind of real-life violence. Fighting games of any type, for instance, feature violent action by definition. Many video games, far beyond the fighting genre, include "hit", "fire", or "shoot" as one of the possible player actions. One might argue that combat and violence is existent and pervasive in games since the era of *Pac-Man* (Namco, 1980) and its ghost-eating skills. A critical research question that emerges from this game design strategy is whether playing video games inspires, teaches or even encourages violent behavior. To address this question, researchers have been involved on a debate around the effects of video games on violent behavior for at least three decades now. Even though video games teach skills that can potentially be transferred to the real-world, thorough and well-performed studies have largely failed to find any causal link between violent games and violent real-life behavior [122].

A potential ethical concern is that we one day develop **artificial general intelligence**, that is as capable (as we are) across a large range of areas and tasks. Games could potentially play a critical role in that development. If developed such AI systems might become very influential in human affairs, and may also gain the ability to improve themselves, potentially leading to what has been termed an *intelligence explosion* [109]. Given that video games are used predominantly in AI research, it is worth pondering what impact training on video games might have on the ethics of a potential superintelligence. Games are frequently used to train and test AI algorithms—this is the main aim of, for example, the MineRL Competition [416] and the Arcade Learning Environment [75]. Many games are focused on violent competition, however; does this mean that we focus unduly on the development of violence in artificial intelligence? What effects could this have on AI that is trained on games but employed in other domains, such as transport or health care?

15.7 Summary

This chapter, and the book itself, concluded with a discussion on the ethical implications of whatever we do in AI and games research, building on a recent article by Melhart et al. [745]. In particular, we discussed thoroughly the various ethical aspects of artificial intelligence *in* and *for* games and opted to view the most critical of those under the affective game loop concept [1286, 1287]. We thus raised a number of ethical dimensions and concerns and the current (lack of) measures and tools available to address them. With this final chapter of the book we aim to further facilitate a dialogue among all stakeholders involved—AI researchers and practitioners, game developers, and ultimately players—with the hope that ethical awareness is increased and that necessary action is taken for the mutual benefit of players and their games.

References

1. Espen Aarseth. Genre trouble. *Electronic Book Review*, 3, 2004.
2. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint* arXiv:1603.04467, 2016.
3. Ryan Abela, Antonios Liapis, and Georgios N. Yannakakis. A constructive approach for the generation of underwater environments. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2015.
4. Sami Abuhamdeh, Mihaly Csikszentmihalyi, and Baland Jalal. Enjoying the possibility of defeat: Outcome uncertainty, suspense, and intrinsic motivation. *Motivation and Emotion*, 39(1), 2015.
5. Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint* arXiv:2303.08774, 2023.
6. David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.
7. Alexandros Agapitos, Julian Togelius, Simon M. Lucas, Jürgen Schmidhuber, and Andreas Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 135–142. IEEE, 2008.
8. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, pages 207–216. ACM, 1993.
9. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, 1994.
10. David W. Aha. Lazy learning. In *Lazy learning*, pages 7–10. Springer, 1997.
11. William Ahlberg, Alessandro Sestini, Konrad Tollmar, and Linus Gisslén. Generating personas for games with multimodal adversarial imitation learning. *arXiv preprint* arXiv:2308.07598, 2023.
12. John B. Ahlquist and Jeannie Novak. *Game development essentials: Game artificial intelligence*. Delmar Pub, 2008.
13. Zach Aikman. Galak-Z: Forever: Building Space-Dungeons Organically. In *Game Developers Conference*, 2015.
14. Shaad Alaka and Rafael Bidarra. Mixed-initiative generation of virtual worlds-a comparative study on the cognitive load of WFC and HSWFC. In *Proceedings of the 19th International Conference on the Foundations of Digital Games*, 2024.

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2

15. Bob Alexander. The beauty of response curves. *AI Game Programming Wisdom*, page 78, 2002.
16. Krishna Aluru, Stefanie Tellex, John Oberlin, and James MacGlashan. Minecraft as an experimental world for AI in robotics. In *AAAI Fall Symposium*, 2015.
17. Alberto Alvarez and Jose Font. Tropetwist: trope-based narrative structure generation. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022.
18. Alberto Alvarez, Jose Font, and Julian Togelius. Toward designer modeling through design style clustering. *IEEE Transactions on Games*, 14(4):676–686, 2022.
19. Alberto Alvarez, Eric Grevillius, Elin Olsson, and Jose Font. Questgram [qg]: Toward a mixed-initiative quest generation tool. In *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 2021.
20. Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *IEEE Conference on Computational Intelligence and Games*. IEEE, 2017.
21. Omar Alzoubi, Rafael A. Calvo, and Ronald H. Stevens. Classification of EEG for Affect Recognition: An Adaptive Approach. In *AI 2009: Advances in Artificial Intelligence*, pages 52–61. Springer, 2009.
22. Mike Ambinder. Biofeedback in gameplay: How Valve measures physiology to enhance gaming experience. In *Game Developers Conference*, San Francisco, California, US, 2011.
23. Dan Amerson, Shaun Kime, and R. Michael Young. Real-time cinematic camera control for interactive narratives. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 369–369. ACM, 2005.
24. Prithviraj Ammanabrolu, Wesley Cheung, Dan Tu, William Broniec, and Mark Riedl. Bringing stories alive: Generating interactive fiction worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 3–9, 2020.
25. Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 7–12, August 2005.
26. Elisabeth André, Martin Klesen, Patrick Gebhard, Steve Allen, and Thomas Rist. Integrating models of personality and emotions into lifelike characters. In *Affective interactions*, pages 150–165. Springer, 2000.
27. John L. Andreassi. *Psychophysiology: Human Behavior and Physiological Response*. Psychology Press, 2000.
28. Ana Antunes, Joana Campos, Manuel Guimarães, João Dias, and Pedro A. Santos. Prompting for socially intelligent agents with chatgpt. In *Proceedings of the 23rd ACM International Conference on Intelligent Virtual Agents*, 2023.
29. Rudolf Arnheim. *Art and visual perception: A psychology of the creative eye*. University of California Press, 1956.
30. Ivon Arroyo, David G. Cooper, Winslow Burleson, Beverly Park Woolf, Kasia Muldner, and Robert Christopherson. Emotion sensors go to school. In *Proceedings of Conference on Artificial Intelligence in Education (AIED)*, pages 17–24. IOS Press, 2009.
31. W. Ross Ashby. Principles of the self-organizing system. In *Facets of Systems Science*, pages 521–536. Springer, 1991.
32. Brian Ashcraft. Korean woman kicks ass at overwatch, gets accused of cheating [update], 2016.
33. Daniel Ashlock. *Evolutionary computation for modeling and optimization*. Springer, 2006.
34. Andrea Asperti, Daniele Cortesi, Carlo De Pieri, Gianmaria Pedrini, and Francesco Sovrano. Crawling in rogue's dungeons with deep reinforcement techniques. *IEEE Transactions on Games*, 12(2):177–186, 2019.
35. Stylianos Asteriadis, Kostas Karpouzis, Noor Shaker, and Georgios N. Yannakakis. Does your profile say it all? Using demographics to predict expressive head movement during gameplay. In *Proceedings of UMAP Workshops*, 2012.

36. Stylianos Asteriadis, Paraskevi Tzouveli, Kostas Karpouzis, and Stefanos Kollias. Estimation of behavioral user state based on eye gaze and head pose—application in an e-learning environment. *Multimedia Tools and Applications*, 41(3):469–493, 2009.
37. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.
38. Phillipa Avery, Sushil Louis, and Benjamin Avery. Evolving coordinated spatial tactics for autonomous entities using influence maps. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 341–348. IEEE, 2009.
39. Phillipa Avery, Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen. Computational intelligence and tower defence games. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1084–1091. IEEE, 2011.
40. Ruth Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. FearNot!–an experiment in emergent narrative. In *Intelligent Virtual Agents*, pages 305–316. Springer, 2005.
41. Batu Aytemiz, Mikhail Jacob, and Sam hofman. Acting with style: Towards designer-centred reinforcement learning for the video games industry. In *CHI Workshop on Reinforcement Learning for Humans, Computer, and Interaction (RL4HCI)*, 2021.
42. Sasha Azad and Chris Martens. Little computer people: A survey and taxonomy of simulated models of social interaction. *Proceedings of the ACM on Human-Computer Interaction*, 5(CHI PLAY):1–30, 2021.
43. Ahmad Azadvar and Alessandro Canossa. UPEQ: ubisoft perceived experience questionnaire: a self-determination evaluation tool for video games. In *Proceedings of the 13th international conference on the foundations of digital games*, pages 1–7, 2018.
44. Simon E. Ortiz B., Koichi Moriyama, Ken-ichi Fukui, Satoshi Kurihara, and Masayuki Numao. Three-subagent adapting architecture for fighting videogames. In *Pacific Rim International Conference on Artificial Intelligence*, pages 649–654. Springer, 2010.
45. Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, pages 507–517. PMLR, 2020.
46. Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.
47. Sander Bakkes, Pieter Spronck, and Jaap van den Herik. Rapid and reliable adaptation of video game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104, 2009.
48. Sander Bakkes, Shimon Whiteson, Guangliang Li, George Viorel Vişniuc, Efstathios Charitos, Norbert Heijne, and Arjen Swellengrebel. Challenge balancing for personalised game spaces. In *Games Media Entertainment (GEM), 2014 IEEE*. IEEE, 2014.
49. Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023.
50. Martin Balla, George EM Long, James Goodman, Raluca D. Gaina, and Diego Perez-Liebana. Pytag: Tabletop games for multi-agent reinforcement learning. *IEEE Transactions on Games*, 2024.
51. Tadas Baltrusaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency. Openface 2.0: Facial behavior analysis toolkit. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, pages 59–66. IEEE, 2018.
52. Chris Bamford and Simon M Lucas. Neural game engine: Accurate learning of generalizable forward models from pixels. In *2020 IEEE Conference on Games (CoG)*, pages 81–88. IEEE, 2020.
53. Christopher Bamford. Griddly: A platform for AI research in games. *Software Impacts*, 8:100066, 2021.

54. Rainer Banse and Klaus R. Scherer. Acoustic profiles in vocal emotion expression. *Journal of Personality and Social Psychology*, 70(3):614, 1996.
55. Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The Hanabi Challenge: A new frontier for AI research. *Artificial Intelligence*, 280:103216, 2020.
56. Ray Barrera, Aung Sithu Kyaw, Clifford Peters, and Thet Naing Swe. *Unity AI Game Programming*. Packt Publishing Ltd, 2015.
57. Gabriella AB Barros, Michael Cerny Green, Antonios Liapis, and Julian Togelius. Data-driven design: A case for maximalist game design paper type: Position paper. In *9th International Conference on Computational Creativity, ICCC 2018*, pages 169–176. Association for Computational Creativity (ACC), 2018.
58. Matthew Barthet, Diogo Branco, Roberto Gallotta, Ahmed Khalifa, and Georgios N. Yannakakis. Closing the affective loop via experience-driven reinforcement learning designers. In *International Conference on Affective Computing and Intelligent Interaction*, 2024.
59. Matthew Barthet, Maria Kaselimi, Kosmas Pinitas, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. GameVibe: A Multimodal Affective Game Corpus. *Nature Scientific Data*, 2024.
60. Matthew Barthet, Ahmed Khalifa, Antonios Liapis, and Georgios N. Yannakakis. Generative personas that behave and experience like humans. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022.
61. Matthew Barthet, Ahmed Khalifa, Antonios Liapis, and Georgios N. Yannakakis. Play with emotion: Affect-driven reinforcement learning. In *2022 10th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2022.
62. Matthew Barthet, Antonios Liapis, and Georgios N. Yannakakis. Go-blend behavior and affect. In *9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 2021.
63. Matthew Barthet, Antonios Liapis, and Georgios N. Yannakakis. Open-ended evolution for Minecraft building generation. *IEEE Transactions on Games*, 2022.
64. Matthew Barthet, Chintan Trivedi, Kosmas Pinitas, Emmanouil Xylakis, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Knowing your annotator: Rapidly testing the reliability of affect annotation. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, 2023.
65. Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.
66. Richard A. Bartle. *Designing virtual worlds*. New Riders, 2004.
67. James Batchelor. 'dynamic difficulty' loot box lawsuit against ea dropped, 2021.
68. Chris Bateman and Richard Boon. *21st Century Game Design (Game Development Series)*. Charles River Media, Inc., 2005.
69. Chris Bateman and Lennart E. Nacke. The neurobiology of play. In *Proceedings of the International Academic Conference on the Future of Game Design and Technology*. ACM, 2010.
70. Christian Bauckhage, Anders Drachen, and Rafet Sifa. Clustering game behavior data. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3):266–278, 2015.
71. Yoann Baveye, Jean-Noël Bettinelli, Emmanuel Dellandrea, Liming Chen, and Christel Chamaret. A large video database for computational models of induced emotion. In *Proceedings of Affective Computing and Intelligent Interaction*, pages 13–18, 2013.
72. Jessica D. Bayliss. Teaching game AI through Minecraft mods. In *2012 IEEE International Games Innovation Conference (IGIC)*, pages 1–4. IEEE, 2012.
73. BBC. Minecraft becomes first video game to hit 300m sales. BBC News website, 16 October 2023. https://www.bbc.com/news/technology-67105983.
74. Farès Belhadj. Terrain modeling: a constrained fractal model. In *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 197–204. ACM, 2007.

75. Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint* arXiv:1207.4708, 2012.
76. Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.
77. Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint* arXiv:1912.06680, 2019.
78. José Luis Bernier, C. Ilia Herráiz, J. J. Merelo, S. Olmeda, and Alberto Prieto. Solving Mastermind using GAs and simulated annealing: a case of dynamic constraint optimization. In *Parallel Problem Solving from Nature (PPSN) IV*, pages 553–563. Springer, 1996.
79. Kent C. Berridge. Pleasures of the brain. *Brain and Cognition*, 52(1):106–128, 2003.
80. Timo Bertram, Johannes Fürnkranz, and Martin Müller. Predicting human card selection in Magic: The Gathering with contextual preference ranking. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.
81. Timo Bertram, Johannes Fürnkranz, and Martin Müller. Learning With Generalised Card Representations for "Magic: The Gathering". In *2024 IEEE Conference on Games (CoG)*. IEEE, 2024.
82. Dimitri Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.
83. Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995.
84. James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. https://cdn.openai.com/papers/dall-e-3.pdf, 2(3):8, 2023.
85. Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1:3–52, 2002.
86. Aditya Bhatt, Scott Lee, Fernando de Mesentier Silva, Connor W Watson, Julian Togelius, and Amy K Hoover. Exploring the hearthstone deck space. In *Proceedings of the 13th international conference on the foundations of digital games*, 2018.
87. Nadav Bhonker, Shai Rozenberg, and Itay Hubara. Playing SNES in the Retro Learning Environment. *arXiv preprint* arXiv:1611.02205, 2016.
88. Mateusz Bialas, Shoshannah Tekofsky, and Pieter Spronck. Cultural influences on play style. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–7. IEEE, 2014.
89. Nadia Bianchi-Berthouze and Christine L. Lisetti. Modeling multimodal expression of user's affective subjective experience. *User Modeling and User-Adapted Interaction*, 12(1):49–84, 2002.
90. Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI/IAAI*, pages 493–499, 1998.
91. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2006.
92. Christopher M Bishop and Hugh Bishop. *Deep learning: Foundations and concepts*. Springer Nature, 2023.
93. Staffan Björk and Jesper Juul. Zero-player games. In *Philosophy of Computer Games Conference, Madrid*, 2012.
94. Sverre Johann Bjørke and Knut Aron Fludal. Deckbuilding in Magic: The Gathering Using a Genetic Algorithm. Master's thesis, NTNU, 2017.
95. Vikki Blake. Minecraft Has 55 Million Monthly Players, 122 Million Sales. *Imagine Games Network*, February 2017.
96. Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint* arXiv:2311.15127, 2023.
97. Paris Mavromoustakos Blom, Sander Bakkes, Chek Tien Tan, Shimon Whiteson, Diederik M. Roijers, Roberto Valenti, and Theo Gevers. Towards Personalised Gaming via Facial Expression Recognition. In *Proceedings of AIIDE*, 2014.

98. Margaret A. Boden. What is creativity. *Dimensions of creativity*, pages 75–117, 1994.
99. Margaret A. Boden. Creativity and artificial intelligence. *Artificial Intelligence*, 103(1):347–356, 1998.
100. Margaret A. Boden. *The creative mind: Myths and mechanisms*. Psychology Press, 2004.
101. Slawomir Bojarski and Clare Bates Congdon. REALM: A rule-based evolutionary computation agent that learns to play Mario. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 83–90. IEEE, 2010.
102. Luuk Bom, Ruud Henken, and Marco Wiering. Reinforcement learning to train Ms. Pac-Man using higher-order action-relative inputs. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 156–163. IEEE, 2013.
103. Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint* arXiv:2108.07258, 2021.
104. Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.
105. Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
106. Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–9. IEEE, 2018.
107. Michael Booth. The AI systems of Left 4 Dead. In *Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (Keynote)*, 2009.
108. Felix Born, Linda Graf, and Maic Masuch. Exergaming: The impact of virtual reality on cognitive performance and player experience. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.
109. Nick Bostrom. How long before superintelligence? *International Journal of Futures Studies*, 2, 1998.
110. Nick Bostrom and Eliezer Yudkowsky. The ethics of artificial intelligence. *The Cambridge handbook of artificial intelligence*, 1:316–334, 2014.
111. Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28, 2004.
112. Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.
113. David M. Bourg and Glenn Seemann. *AI for game developers*. O'Reilly Media, Inc., 2004.
114. Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.
115. Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.
116. Danah Boyd and Kate Crawford. Six provocations for big data. In *A decade in internet time: Symposium on the dynamics of the internet and society*. Oxford Internet Institute, Oxford, 2011.
117. S. R. K. Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a Monte-Carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012.
118. Michael Bratman. Intention, plans, and practical reason. 1987.
119. Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355, 1988.
120. Ivan Bravi, Diego Perez-Liebana, Simon M Lucas, and Jialin Liu. Rinascimento: Optimising statistical forward planning agents for playing splendor. In *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.
121. Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. CRC Press, 1984.

122. Johannes Breuer, Jens Vogelgesang, Thorsten Quandt, and Ruth Festl. *Violent video games and physical aggression: Evidence for a selection effect among adolescents.*, volume 4. Educational Publishing Foundation, 2015.
123. Daniel Brewer. Tactical pathfinding on a navmesh. *Game AI Pro: Collected Wisdom of Game AI Professionals*, page 361, 2013.
124. Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
125. Jean-Pierre Briot and François Pachet. Deep learning for music generation: challenges and directions. *Neural Computing and Applications*, 32(4):981–993, 2020.
126. Greg Brockman, Vicki Cheung, Jonas Schneider Ludwig Pettersson, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint* arXiv:1606.01540, 2016.
127. Joost Broekens and Willem-Paul Brinkman. Affectbutton: A method for reliable and valid affective self-report. *International Journal of Human-Computer Studies*, 71(6):641–667, 2013.
128. Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986.
129. David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *Royals Signals & Radar Establishment*, 1988.
130. Anna Brown and Alberto Maydeu-Olivares. How IRT can solve problems of ipsative data in forced-choice questionnaires. *Psychological Methods*, 18(1):36, 2013.
131. Daniel Lankford Brown. Mezzo: An adaptive, real-time composition program for game soundtracks. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.
132. Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.
133. Cameron Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.
134. Cameron Browne. Yavalath. In *Evolutionary Game Design*, pages 75–85. Springer, 2011.
135. Cameron Browne. Modern techniques for ancient games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.
136. Cameron Browne and Frederic Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.
137. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.
138. Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. *arXiv preprint* arXiv:2402.15391, 2024.
139. Florian Brühlmann, Philipp Baumgartner, Günter Wallner, Simone Kriglstein, and Elisa D Mekler. Motivational profiling of league of legends players. *Frontiers in Psychology*, 11:1307, 2020.
140. Nicholas J. Bryan, Gautham J. Mysore, and Ge Wang. ISSE: An Interactive Source Separation Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–266, 2014.
141. Bobby D. Bryant and Risto Miikkulainen. Evolving stochastic controller networks for intelligent game agents. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1007–1014. IEEE, 2006.
142. Joanna Bryson and Brendan McGonigle. Agent architecture as object oriented design. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 15–29. Springer, 1997.
143. Joanna Bryson and Lynn Andrea Stein. Architectures and idioms: Making progress in agent design. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 73–88. Springer, 2000.

144. Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint* arXiv:2303.12712, 2023.
145. Mat Buckland. *Programming Game AI by example*. Jones & Bartlett Learning, 2005.
146. Mat Buckland and Mark Collins. *AI techniques for game programming*. Premier Press, 2002.
147. Paweł Budzianowski and Ivan Vulić. Hello, it's GPT-2–how can I help you? towards the use of pretrained language models for task-oriented dialogue systems. *arXiv preprint* arXiv:1907.05774, 2019.
148. Vadim Bulitko, Yngvi Björnsson, Nathan R. Sturtevant, and Ramon Lawrence. Real-time heuristic search for pathfinding in video games. In *Artificial Intelligence for Computer Games*, pages 1–30. Springer, 2011.
149. Vadim Bulitko, Greg Lee, Sergio Poo Hernandez, Alejandro Ramirez, and David Thue. Techniques for AI-Driven Experience Management in Interactive Narratives. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, pages 523–534. AK Peters/CRC Press, 2015.
150. Paolo Burelli. Virtual cinematography in games: investigating the impact on player experience. *Foundations of Digital Games*, 2013.
151. Paolo Burelli and Georgios N. Yannakakis. Combining Local and Global Optimisation for Virtual Camera Control. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Copenhagen, Denmark, August 2010. IEEE.
152. Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and Knowledge Discovery*, 2(2):121–167, 1998.
153. Michael Buro and David Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106, 2012.
154. Carlos Busso, Zhigang Deng, Serdar Yildirim, Murtaza Bulut, Chul Min Lee, Abe Kazemzadeh, Sungbok Lee, Ulrich Neumann, and Shrikanth Narayanan. Analysis of emotion recognition using facial expressions, speech and multimodal information. In *Proceedings of the International Conference on Multimodal Interfaces (ICMI)*, pages 205–211. ACM, 2004.
155. Eric Butler, Adam M. Smith, Yun-En Liu, and Zoran Popovic. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, pages 377–386. ACM, 2013.
156. F. Butterfield. Video game specialists come to Harvard to praise Pac-Man, not to bury him. *The New York Times*, 24:22, 1983.
157. Martin V. Butz and Thies D. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. In *IEEE Symposium on Computational Intelligence and Games*, pages 317–324. IEEE, 2009.
158. William Cachia, Antonios Liapis, and Georgios N. Yannakakis. Multi-level evolution of shooter levels. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
159. John T. Cacioppo, Gary G. Berntson, Jeff T. Larsen, Kirsten M. Poehlmann, and Tiffany A. Ito. The psychophysiology of emotion. *Handbook of emotions*, 2:173–191, 2000.
160. Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-HEX: an artificial player for Angry Birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):128–139, 2016.
161. Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.
162. Gordon Calleja. *In-game: from immersion to incorporation*. MIT Press, 2011.
163. Rafael Calvo, Iain Brown, and Steve Scheding. Effect of experimental factors on the recognition of affective mental states through physiological measures. In *AI 2009: Advances in Artificial Intelligence*, pages 62–70. Springer, 2009.
164. Rafael A Calvo and Sidney D'Mello. Affect detection: An interdisciplinary review of models, methods, and their applications. *Affective Computing, IEEE Transactions on*, 1(1):18–37, 2010.

165. Elizabeth Camilleri, Georgios N. Yannakakis, and Alexiei Dingli. Platformer Level Design for Player Believability. In *IEEE Computational Intelligence and Games Conference*. IEEE, 2016.
166. Elizabeth Camilleri, Georgios N. Yannakakis, and Antonios Liapis. Towards General Models of Player Affect. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
167. Jonathan Campbell and Clark Verbrugge. Procedural generation of rollercoasters. *IEEE Transactions on Games*, 2024.
168. Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1):57–83, 2002.
169. Henrique Campos, Joana Campos, João Cabral, Carlos Martinho, Jeppe Herlev Nielsen, and Ana Paiva. My Dream Theatre. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1357–1358. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
170. Joana Campos, Carlos Martinho, Gordon Ingram, Asimina Vasalou, and Ana Paiva. My dream theatre: Putting conflict on center stage. In *FDG*, pages 283–290, 2013.
171. Rodrigo Canaan, Xianbo Gao, Youjin Chung, Julian Togelius, Andy Nealen, and Stefan Menzel. Evaluating RL agents in Hanabi with unseen partners. In *AAAI'20 Reinforcement Learning in Games Workshop*, 2020.
172. Rodrigo Canaan, Xianbo Gao, Julian Togelius, Andy Nealen, and Stefan Menzel. Generating and adapting to diverse ad hoc partners in Hanabi. *IEEE Transactions on Games*, 15(2):228–241, 2022.
173. Rodrigo Canaan, Christoph Salge, Julian Togelius, and Andy Nealen. Leveling the playing field: Fairness in AI versus human game benchmarks. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019.
174. Rodrigo Canaan, Haotian Shen, Ruben Torrado, Julian Togelius, Andy Nealen, and Stefan Menzel. Evolving agents for the Hanabi 2018 cig competition. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018.
175. Rodrigo Canaan, Julian Togelius, Andy Nealen, and Stefan Menzel. Diverse agents for ad-hoc cooperation in Hanabi. In *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.
176. Alessandro Canossa and Anders Drachen. Patterns of play: Play-personas in user-centred game development. In *Proceedings of DiGRA 2009 Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*, 2009.
177. Alessandro Canossa, Josep B. Martinez, and Julian Togelius. Give me a reason to dig Minecraft and psychology of motivation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
178. Alessandro Canossa, Dmitry Salimov, Ahmad Azadvar, Casper Harteveld, and Georgios Yannakakis. For honor, for toxicity: Detecting toxic behavior through gameplay. *Proceedings of the ACM on Human-Computer Interaction*, 5(CHI PLAY):1–29, 2021.
179. Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 395–402. ACM, 2011.
180. Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*, pages 63–72. Springer, 2011.
181. Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR, 2023.
182. Justine Cassell. *Embodied conversational agents*. MIT Press, 2000.
183. Justine Cassell, Timothy Bickmore, Mark Billinghurst, Lee Campbell, Kenny Chang, Hannes Vilhjálmsson, and Hao Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 520–527. ACM, 1999.

184. Marc Cavazza, Fred Charles, and Steven J. Mead. Character-based interactive storytelling. *IEEE Intelligent Systems*, 17(4):17–24, 2002.
185. Marc Cavazza, Fred Charles, and Steven J. Mead. Interacting with virtual characters in interactive storytelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 1*, pages 318–325. ACM, 2002.
186. Tristan Cazenave. Generalized rapid action value estimation. In *24th International conference on artificial intelligence*, pages 754–760, 2015.
187. Tristan Cazenave, Swann Legras, and Véronique Ventos. Optimizing $\alpha\mu$. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.
188. Michal Čertickỳ, David Churchill, Kyung-Joong Kim, Martin Čertickỳ, and Richard Kelly. Starcraft AI competitions, bots, and tournament manager software. *IEEE Transactions on Games*, 11(3):227–237, 2018.
189. Kranti Chalamalasetti, Jana Götze, Sherzod Hakimov, Brielen Madureira, Philipp Sadler, and David Schlangen. clembench: Using game play to evaluate chat-optimized language models as conversational agents. *arXiv preprint* arXiv:2305.13455, 2023.
190. Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. Computational aspects of cooperative game theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(6):1–168, 2011.
191. Alex J. Champandard. *AI game development: Synthetic creatures with learning and reactive behaviors*. New Riders, 2003.
192. Alex J. Champandard. Behavior trees for next-gen game AI. In *Game Developers Conference, Audio Lecture*, 2007.
193. Alex J. Champandard. Understanding Behavior Trees. *AiGameDev. com*, 2007.
194. Alex J. Champandard. Getting started with decision making and control systems. *AI Game Programming Wisdom*, 4:257–264, 2008.
195. Jason C. Chan. Response-order effects in Likert-type scales. *Educational and Psychological Measurement*, 51(3):531–540, 1991.
196. Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech 2011*, pages 1025–1028, 2011.
197. Devendra Singh Chaplot and Guillaume Lample. Arnold: An autonomous agent to play FPS games. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
198. M. Charity, Yash Bhartia, Daniel Zhang, Ahmed Khalifa, and Julian Togelius. A preliminary study on a conceptual game feature generation and recommendation system. *arXiv preprint* arXiv:2308.13538, 2023.
199. Megan Charity, Ahmed Khalifa, and Julian Togelius. Baba is y'all: Collaborative mixed-initiative level design. In *2020 IEEE Conference on Games (CoG)*, pages 542–549. IEEE, 2020.
200. Darryl Charles and Michaela Black. Dynamic player modelling: A framework for player-centric digital games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.
201. Fred Charles, Miguel Lozano, Steven J. Mead, Alicia Fornes Bisquerra, and Marc Cavazza. Planning formalisms and authoring in interactive storytelling. In *Proceedings of TIDSE*, 2003.
202. Guillaume M. J. B. Chaslot, Mark H. M. Winands, H. Jaap van Den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.
203. Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.
204. Pei Pei Chen, Anna Guitart, Ana Fernández del Río, and Africa Periánez. Customer lifetime value in video games using deep learning and parametric models. In *2018 IEEE international conference on big data (big data)*, pages 2134–2140. IEEE, 2018.

205. Sanyuan Chen, Yu Wu, Chengyi Wang, Shujie Liu, Daniel Tompkins, Zhuo Chen, and Furu Wei. Beats: Audio pre-training with acoustic tokenizers. *arXiv preprint* arXiv:2212.09058, 2022.
206. Xiang 'Anthony' Chen, Tovi Grossman, Daniel J. Wigdor, and George Fitzmaurice. Duet: Exploring joint interactions on a smart phone and a smart watch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 159–168, 2014.
207. Zhengxing Chen, Magy Seif El-Nasr, Alessandro Canossa, Jeremy Badler, Stefanie Tignor, and Randy Colvin. Modeling individual differences through frequent pattern mining on role-playing game actions. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*, 2015.
208. Sonia Chernova, Jeff Orkin, and Cynthia Breazeal. Crowdsourcing HRI through online multiplayer games. In *AAAI Fall Symposium: Dialog with Robots*, pages 14–19, 2010.
209. Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. Back to basics: benchmarking canonical evolution strategies for playing Atari. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 1419–1426, 2018.
210. Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
211. Wei Chu and Zoubin Ghahramani. Preference learning with Gaussian processes. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 137–144. ACM, 2005.
212. David Churchill. AIIDE StarCraft AI Competition. http://www.cs.mun.ca/~dchurchill/starcraftaicomp/, 2016. Accessed 6 October 2024.
213. David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
214. David Churchill, Mike Preuss, Florian Richoux, Gabriel Synnaeve, Alberto Uriarte, Santiago Ontañón, and Michal Certickỳ. StarCraft Bots and Competitions. In *Encyclopedia of Computer Graphics and Games*. Springer, 2016.
215. J. Clement. Number of video game users worldwide from 2017 to 2027, 2022.
216. Andrea Clerico, Cindy Chamberland, Mark Parent, Pierre-Emmanuel Michon, Sebastien Tremblay, Tiago H. Falk, Jean-Christophe Gagnon, and Philip Jackson. Biometrics and classifier fusion to predict the fun-factor in video gaming. In *IEEE Computational Intelligence and Games Conference*. IEEE, 2016.
217. Jess Cliffe. CS V1.0 Released! [Archived Online] https://web.archive.org/web/20001201214200/http://counter-strike.net/ , 2000. Accessed 29 October 2024.
218. Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pages 1282–1289. PMLR, 2019.
219. Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12):1245–1287, 2002.
220. Carlos A. Coello Coello, Gary B. Lamont, and David A. van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.
221. Nicholas Cole, Sushil J. Louis, and Chris Miles. Using a genetic algorithm to tune first-person shooter bots. In *Congress on Evolutionary Computation (CEC)*, pages 139–145. IEEE, 2004.
222. Allan M Collins and M Ross Quillian. Retrieval time from semantic memory. *Journal of verbal learning and verbal behavior*, 8(2):240–247, 1969.
223. Karen Collins. An introduction to procedural music in video games. *Contemporary Music Review*, 28(1):5–15, 2009.
224. Karen Collins. *Playing with sound: a theory of interacting with sound and music in video games*. MIT Press, 2013.
225. Simon Colton. Creativity versus the perception of creativity in computational systems. In *AAAI Spring Symposium: Creative Intelligent Systems*, 2008.

226. Kate Compton, Adam Smith, and Michael Mateas. Anza island: Novel gameplay using asp. In *Proceedings of the third workshop on Procedural Content Generation in Games*, pages 1–4, 2012.
227. Cristina Conati. Intelligent tutoring systems: New challenges and directions. In *IJCAI*, pages 2–7, 2009.
228. Cristina Conati, Abigail Gertner, and Kurt VanLehn. Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417, 2002.
229. Cristina Conati and Heather Maclaren. Modeling user affect from causes and effects. *User Modeling, Adaptation, and Personalization*, pages 4–15, 2009.
230. John Conway. The game of life. *Scientific American*, 223(4):4, 1970.
231. Michael Cook. Puck: a slow and personal automated game designer. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 232–239, 2022.
232. Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *IEEE Computational Intelligence and Games*, pages 289–296, 2011.
233. Michael Cook and Simon Colton. Ludus ex machina: Building a 3D game designer that competes alongside humans. In *Proceedings of the 5th International Conference on Computational Creativity*, 2014.
234. Michael Cook and Simon Colton. A rogue dream: Automatically generating meaningful content for games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, pages 2–7, 2014.
235. Michael Cook, Simon Colton, and Jeremy Gow. The Angelina videogame design system-Part I. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):192–203, 2016.
236. Michael Cook, Simon Colton, and Jeremy Gow. The Angelina videogame design system-Part II. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):254–266, 2016.
237. Michael Cook, Simon Colton, and Alison Pease. Aesthetic Considerations for Automated Platformer Design. In *AIIDE*, 2012.
238. Michael Cook, Simon Colton, Azalea Raad, and Jeremy Gow. Mechanic miner: Reflection-driven game mechanic discovery and level design. *Applications of Evolutionary Computation*, page 284, 2013.
239. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
240. Paul T. Costa and Robert R. MacCrae. *Revised NEO personality inventory (NEO PI-R) and NEO five-factor inventory (NEO-FFI): Professional manual*. Psychological Assessment Resources, Incorporated, 1992.
241. Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.
242. Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. In *Computer Games Workshop*, 2007.
243. Roddy Cowie and Randolph R. Cornelius. Describing the emotional states that are expressed in speech. *Speech Communication*, 40(1):5–32, 2003.
244. Roddy Cowie, Ellen Douglas-Cowie, Susie Savvidou, Edelle McMahon, Martin Sawey, and Marc Schröder. 'FEELTRACE': An instrument for recording perceived emotion in real time. In *ISCA Tutorial and Research Workshop (ITRW) on Speech and Emotion*, 2000.
245. Roddy Cowie and Martin Sawey. GTrace-General trace program from Queen's University, Belfast, 2011.
246. Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.
247. Koby Crammer and Yoram Singer. Pranking with ranking. *Advances in Neural Information Processing Systems*, 14:641–647, 2002.

248. Chris Crawford. *Chris Crawford on interactive storytelling*. New Riders, 2012.
249. Mihaly Csikszentmihalyi. *Creativity: Flow and the psychology of discovery and invention*. New York: Harper Collins, 1996.
250. Mihaly Csikszentmihalyi. *Beyond boredom and anxiety.* Jossey-Bass, 2000.
251. Mihaly Csikszentmihalyi. *Toward a psychology of optimal experience*. Springer, 2014.
252. Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. Playing Atari with six neurons. In *Proceedings of AAMAS*, 2019.
253. Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. Playing Atari with few neurons: Improving the efficacy of reinforcement learning by decoupling feature extraction and decision making. *Autonomous Agents and Multi-Agent Systems*, 35, 2021.
254. George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
255. Ryan S. J. d. Baker, Gregory R. Moore, Angela Z. Wagner, Jessica Kalka, Aatish Salvi, Michael Karabinos, Colin A. Ashe, and David Yaron. The Dynamics between Student Affect and Behavior Occurring Outside of Educational Software. In *Affective Computing and Intelligent Interaction*, pages 14–24. Springer, 2011.
256. Oscar Dadfar, Lingdong Huang, and Hizal Çelik. GAN Theft Auto: Autonomous texturing of procedurally generated interactive cities. In *Proceedings of the 4th International Conference on Computer Science and Software Engineering*, pages 362–366, 2021.
257. Anders Dahlbom and Lars Niklasson. Goal-Directed Hierarchical Dynamic Scripting for RTS Games. In *AIIDE*, pages 21–28, 2006.
258. Steve Dahlskog and Julian Togelius. Patterns as objectives for level generation. In *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2013.
259. Steve Dahlskog, Julian Togelius, and Mark J. Nelson. Linear levels through n-grams. In *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, pages 200–206. ACM, 2014.
260. Gillian Dale, Augustin Joessel, Daphne Bavelier, and C Shawn Green. A new look at the cognitive neuroscience of video game play. *Annals of the New York Academy of Sciences*, 1464(1):192–203, 2020.
261. Antonio R. Damasio, Barry J. Everitt, and Dorothy Bishop. The somatic marker hypothesis and the possible functions of the prefrontal cortex [and discussion]. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 351(1346):1413–1420, 1996.
262. Sahith Dambekodi, Spencer Frazier, Prithviraj Ammanabrolu, and Mark Riedl. Playing text-based games with common sense. In *Proceedings of the NeurIPS Workshop on Wordplay: When Language Meets Games*, 2020.
263. Gustavo Danzi, Andrade Hugo Pimentel Santana, André Wilson Brotto Furtado, André Roberto Gouveia, Amaral Leitao, and Geber Lisboa Ramalho. Online adaptation of computer games agents: A reinforcement learning approach. In *II Workshop de Jogos e Entretenimento Digital*, pages 105–112, 2003.
264. Isaac M Dart, Gabriele De Rossi, and Julian Togelius. Speedrock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, pages 1–4, 2011.
265. Charles Darwin. On the origin of species. 1859.
266. Michael Dawe, Steve Gargolinski, Luke Dicken, Troy Humphreys, and Dave Mark. Behavior selection algorithms: an overview. *Game AI Pro 360: Guide to Architecture*, pages 1–14, 2019.
267. Dave De Jonge and Carles Sierra. D-Brane: a diplomacy playing agent for automated negotiations research. *Applied Intelligence*, 47(1):158–177, 2017.
268. Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. AI-based Playtesting of Contemporary Board Games. In *Proceedings of Foundations of Digital Games (FDG)*, 2017.
269. Maarten de Waard, Diederik M. Roijers, and Sander Bakkes. Monte Carlo tree search with options for general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.

270. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
271. Edward L. Deci and Richard M. Ryan. *Intrinsic motivation*. Wiley Online Library, 1975.
272. Omar Delarosa, Hang Dong, Mindy Ruan, Ahmed Khalifa, and Julian Togelius. Mixed-initiative level design with RL brush. In *Artificial Intelligence in Music, Sound, Art and Design: 10th International Conference, EvoMUSART 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 10*, pages 412–426. Springer, 2021.
273. Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is Harder/Easier than We Thought. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN 2016)*, pages 13:1–13:14, La Maddalena, Italy, June 8–10 2016.
274. Jack Dennerlein, Theodore Becker, Peter Johnson, Carson Reynolds, and Rosalind W. Picard. Frustrating computer users increases exposure to physical factors. In *Proceedings of the International Ergonomics Association (IEA)*, 2003.
275. Jörg Denzinger, Kevin Loose, Darryl Gates, and John W. Buchanan. Dealing with Parameterized Actions in Behavior Testing of Commercial Computer Games. In *IEEE Symposium on Computational Intelligence and Games*, 2005.
276. L. Devillers, R. Cowie, J. C. Martin, E. Douglas-Cowie, S. Abrilian, and M. McRorie. Real life emotions in French and English TV video clips: an integrated annotation protocol combining continuous and discrete approaches. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy*, page 22, 2006.
277. Hannah Devlin. Is AI lying to me? Scientists warn of growing capacity for deception. *The Guardian*, May 2024.
278. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint* arXiv:1810.04805, 2018.
279. Ravi Dhar and Itamar Simonson. The effect of forced choice on choice. *Journal of Marketing Research*, 40(2):146–160, 2003.
280. Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint* arXiv:2005.00341, 2020.
281. Joao Dias, Samuel Mascarenhas, and Ana Paiva. Fatima modular: Towards an agent architecture with a generic appraisal framework. In *Emotion Modeling*, pages 44–56. Springer, 2014.
282. Kevin Dill. A pattern-based approach to modular AI for Games. *Game Programming Gems*, 8:232–243, 2010.
283. Kevin Dill. *What is game AI?* CRC Press, Boca Raton, FL, 2011.
284. Kevin Dill. Introducing GAIA: A Reusable, Extensible architecture for AI behavior. In *Proceedings of the 2012 Spring Simulation Interoperability Workshop*, 2012.
285. Kevin Dill and L. Martin. A game AI approach to autonomous control of virtual characters. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2011.
286. Sidney D'Mello and Art Graesser. Automatic detection of learner's affect from gross body language. *Applied Artificial Intelligence*, 23(2):123–150, 2009.
287. Mihai Dobre and Alex Lascarides. POMCP with human preferences in Settlers of Catan. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14, pages 17–23, 2018.
288. Joris Dormans. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
289. Joris Dormans and Sander Bakkes. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):216–228, 2011.
290. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly,

et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint* arXiv:2010.11929, 2020.
291. Anders Drachen, Alessandro Canossa, and Georgios N. Yannakakis. Player modeling using self-organization in Tomb Raider: Underworld. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, pages 1–8. IEEE, 2009.
292. Anders Drachen, Pejman Mirza-Babaei, and Lennart E. Nacke. *Games user research*. Oxford University Press, 2018.
293. Anders Drachen, Lennart Nacke, Georgios N. Yannakakis, and Anja Lee Pedersen. Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In *Proceedings of the SIGGRAPH Symposium on Video Games*. ACM-SIGGRAPH Publishers, 2010.
294. Anders Drachen and Matthias Schubert. Spatial game analytics. In *Game Analytics*, pages 365–402. Springer, 2013.
295. Anders Drachen, Christian Thurau, Julian Togelius, Georgios N. Yannakakis, and Christian Bauckhage. Game Data Mining. In *Game Analytics*, pages 205–253. Springer, 2013.
296. H. Drucker, C.J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 155–161. Morgan Kaufmann Publishers, 1997.
297. Aaron Drummond, James D. Sauer, Lauren C. Hall, David Zendle, and Malcolm R. Loudon. Why loot boxes could be regulated as gambling. *Nature Human Behaviour*, 4(10):986–988, 2020.
298. Guanglong Du, Wenpei Zhou, Chunquan Li, Di Li, and Peter Xiaoping Liu. An emotion recognition method for game evaluation based on electroencephalogram. *IEEE Transactions on Affective Computing*, 2020.
299. Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint* arXiv:2407.21783, 2024.
300. Sam Earle, Maria Edwards, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Learning controllable content generators. In *2021 IEEE Conference on Games (CoG)*, pages 1–9. IEEE, 2021.
301. Sam Earle, Zehua Jiang, and Julian Togelius. Scaling, control and generalization in reinforcement learning level generators. In *2024 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2024.
302. David S. Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.
303. Marc Ebner, John Levine, Simon M. Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.
304. Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint* arXiv:1901.10995, 2019.
305. Arthur S. Eddington. The Constants of Nature. In *The World of Mathematics 2*, pages 1074–1093. Simon & Schuster, 1956.
306. Stefan Edelkamp. *Algorithmic Intelligence: Towards an Algorithmic Foundation for Artificial Intelligence*. Springer, 2023.
307. Arjan Egges, Sumedha Kshirsagar, and Nadia Magnenat-Thalmann. Generic personality and emotion simulation for conversational agents. *Computer animation and virtual worlds*, 15(1):1–13, 2004.
308. Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
309. P. Ekman and W.V. Friesen. Nonverbal leakage and clues to deception. Technical report, Defense Technical Information Center, 1969.
310. Paul Ekman and Karl G. Heider. The universality of a contempt expression: A replication. *Motivation and emotion*, 12(3):303–308, 1988.
311. Magy Seif El-Nasr. Intelligent lighting for game environments. *Journal of Game Development*, 2005.

312. Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game analytics: Maximizing the value of player data*. Springer, 2013.
313. Magy Seif El-Nasr, Shree Durga, Mariya Shiyko, and Carmen Sceppa. Data-driven retrospective interviewing (DDRI): a proposed methodology for formative evaluation of pervasive games. *Entertainment Computing*, 11:1–19, 2015.
314. Magy Seif El-Nasr, Truong-Huy D Nguyen, Alessandro Canossa, and Anders Drachen. *Game data science*. Oxford University Press, 2021.
315. Magy Seif El-Nasr, Athanasios Vasilakos, Chinmay Rao, and Joseph Zupko. Dynamic intelligent lighting for directing visual attention in interactive 3-D scenes. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(2):145–153, 2009.
316. Magy Seif El-Nasr, John Yen, and Thomas R. Ioerger. Flame—fuzzy logic adaptive model of emotions. *Autonomous Agents and Multi-Agent Systems*, 3(3):219–257, 2000.
317. Mirjam Palosaari Eladhari. Re-tellings: the fourth layer of narrative as an instrument for critique. In *Interactive Storytelling: 11th International Conference on Interactive Digital Storytelling, ICIDS 2018, Dublin, Ireland, December 5–8, 2018, Proceedings 11*, pages 65–78. Springer, 2018.
318. Mirjam Palosaari Eladhari and Michael Mateas. Semi-autonomous avatars in World of Minds: A case study of AI-based game design. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 201–208. ACM, 2008.
319. Mirjam Palosaari Eladhari and Michael Sellers. Good moods: outlook, affect and mood in dynemotion and the mind module. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 1–8. ACM, 2008.
320. George Skaff Elias, Richard Garfield, K. Robert Gutschera, and Peter Whitley. *Characteristics of games*. MIT Press, 2012.
321. David K. Elson and Mark O. Riedl. A lightweight intelligent virtual cinematography system for machinima production. In *AIIDE*, pages 8–13, 2007.
322. Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint* arXiv:1902.08710, 2019.
323. Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable digital signal processing. *arXiv preprint* arXiv:2001.04643, 2020.
324. Nathan Ensmenger. Is Chess the Drosophila of AI? A Social History of an Algorithm. *Social Studies of Science*, 42(1):5–30, 2012.
325. Ido Erev and Alvin E. Roth. Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *American Economic Review*, pages 848–881, 1998.
326. Susanne Erk, Markus Kiefer, Jo Grothe, Arthur P Wunderlich, Manfred Spitzer, and Henrik Walter. Emotional context modulates subsequent memory effect. *Neuroimage*, 18(2):439–447, 2003.
327. Jason K Eshraghian. Human ownership of artificial creativity. *Nature Machine Intelligence*, 2(3):157–160, 2020.
328. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.
329. Richard Evans and Emily Short. Versu—a simulationist storytelling system. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2):113–130, 2014.
330. Mike Fahey. Autistic boy branded a cheater by xbox live [update], 2013.
331. Meta Fundamental AI Research Diplomacy Team (FAIR), Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

332. Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in DQN. *arXiv preprint* arXiv:1810.00123, 2018.
333. Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *arXiv preprint* arXiv:1708.00524, 2017.
334. Lisa A. Feldman. Valence focus and arousal focus: Individual differences in the structure of affective experience. *Journal of personality and social psychology*, 69(1):153, 1995.
335. David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.
336. Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, volume 8, pages 259–264, 2008.
337. Hilmar Finnsson and Yngvi Björnsson. Learning simulation control in general game-playing agents. In *AAAI*, pages 954–959, 2010.
338. Jacob Fischer, Nikolaj Falsted, Mathias Vielwerth, Julian Togelius, and Sebastian Risi. Monte-Carlo Tree Search for Simulated Car Racing. In *Proceedings of FDG*, 2015.
339. John H. Flavell. *The developmental psychology of Jean Piaget*. Ardent Media, 1963.
340. Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
341. Dario Floreano, Toshifumi Kato, Davide Marocco, and Eric Sauser. Coevolution of active vision and feature selection. *Biological Cybernetics*, 90(3):218–228, 2004.
342. Luciano Floridi and Massimo Chiriatti. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.
343. Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951. PMLR, 2019.
344. David B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, 2001.
345. David B. Fogel, Timothy J. Hays, Sarah L. Hahn, and James Quon. The Blondie25 chess program competes against Fritz 8.0 and a human chess master. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 230–235. IEEE, 2006.
346. Matthew C Fontaine, Scott Lee, Lisa B. Soros, Fernando de Mesentier Silva, Julian Togelius, and Amy K. Hoover. Mapping hearthstone deck spaces through map-elites with sliding boundaries. In *Proceedings of The Genetic and Evolutionary Computation Conference*, pages 161–169, 2019.
347. Matthew C Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K Hoover, and Stefanos Nikolaidis. Illuminating Mario scenes in the latent space of a generative adversarial network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5922–5930, 2021.
348. Tom Forsyth. Cellular automata for physical modelling. *Game Programming Gems*, 3:200–214, 2002.
349. Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.
350. Billy Joe Franks, Benjamin Dinkelmann, Marius Kloft, and Sophie Fellenz. Ordinal regression for difficulty prediction of stepmania levels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 497–512. Springer, 2023.
351. Michael Freed, Travis Bear, Herrick Goldman, Geoffrey Hyatt, Paul Reber, A. Sylvan, and Joshua Tauber. Towards more human-like computer opponents. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 22–26, 2000.
352. Jonas Freiknecht and Wolfgang Effelsberg. Procedural generation of interactive stories using language models. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020.

353. Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.
354. Marie Gustafsson Friberger and Julian Togelius. Generating interesting Monopoly boards from open data. In *2012 IEEE conference on computational intelligence and games (CIG)*, pages 288–295. IEEE, 2012.
355. Marie Gustafsson Friberger, Julian Togelius, Andrew Borg Cardona, Michele Ermacora, Anders Mousten, Martin Møller Jensen, V Tanase, and Ulrik Brøndsted. Data games. In *4th Workshop on Procedural Content Generation*, pages 1–8. ACM, 2013.
356. Nico Frijda. *The Emotions*. Cambridge University Press, Englewood Cliffs, NJ, 1986.
357. Frederik Frydenberg, Kasper R. Andersen, Sebastian Risi, and Julian Togelius. Investigating MCTS modifications in general video game playing. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 107–113. IEEE, 2015.
358. Drew Fudenberg and David K. Levine. *The theory of learning in games*. MIT Press, 1998.
359. Johannes Fürnkranz and Eyke Hüllermeier. Preference learning and ranking by pairwise comparison. In *Preference learning*, pages 65–82. Springer, 2010.
360. Raluca D. Gaina, Martin Balla, Alexander Dockhorn, Raul Montoliu, and Diego Perez-Liebana. Design and implementation of TAG: a tabletop games framework. *arXiv preprint* arXiv:2009.12065, 2020.
361. Raluca D. Gaina, Sam Devlin, Simon M Lucas, and Diego Perez-Liebana. Rolling horizon evolutionary algorithms for general video game playing. *IEEE Transactions on Games*, 14(2):232–242, 2021.
362. Raluca D. Gaina, Diego Perez-Liebana, Simon M Lucas, Chiara F. Sironi, and Mark H. M. Winands. Self-adaptive rolling horizon evolutionary algorithms for general video game playing. In *2020 IEEE Conference on Games (CoG)*, pages 367–374. IEEE, 2020.
363. Theodoros Galanos, Antonios Liapis, and Georgios N. Yannakakis. AffectGAN: affect-based generative art driven by semantics. In *2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, pages 01–07. IEEE, 2021.
364. Theodoros Galanos, Antonios Liapis, and Georgios N. Yannakakis. Architext: Language-driven generative architecture design. *arXiv preprint* arXiv:2303.07519, 2023.
365. Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Large language models and games: A survey and roadmap. *IEEE Transactions on Games*, 2024.
366. Zhongye Gao, Mengjun Yi, Ying Jin, Hanwen Zhang, Yun Hao, Ming Yin, Ziwen Cai, and Furao Shen. A survey of research on several problems in the RoboCup3D simulation environment. *Autonomous Agents and Multi-Agent Systems*, 38(1):13, 2024.
367. Maurizio Garbarino, Simone Tognetti, Matteo Matteucci, and Andrea Bonarini. Learning general preference models from physiological responses in video games: How complex is it? In *Affective Computing and Intelligent Interaction*, pages 517–526. Springer, 2011.
368. Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J. Merelo. Evolutionary deckbuilding in Hearthstone. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
369. Tom A. Garner. From Sinewaves to Physiologically-Adaptive Soundscapes: The Evolving Relationship Between Sound and Emotion in Video Games. In *Emotion in Games: Theory and Praxis*, pages 197–214. Springer, 2016.
370. Tom A. Garner and Mark Grimshaw. Sonic virtuality: Understanding audio in a virtual world. *The Oxford Handbook of Virtuality*, 2014.
371. H. P. Gasselseder. Re-scoring the games score: Dynamic music and immersion in the ludonarrative. In *Proceedings of the Intelligent Human Computer Interaction conference*, 2014.
372. Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.
373. Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, Tomáš Plch, and Cyril Brom.

Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In *Agents for games and simulations*, pages 1–15. Springer, 2009.
374. Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62, 2005.
375. Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1998.
376. Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *Interspeech*, pages 893–896, 2005.
377. Panayiotis G. Georgiou, Matthew P. Black, Adam C. Lammert, Brian R. Baucom, and Shrikanth S. Narayanan. "That's Aggravating, Very Aggravating": Is It Possible to Classify Behaviors in Couple Interactions Using Automatically Derived Lexical Features? In *Affective Computing and Intelligent Interaction*, pages 87–96. Springer, 2011.
378. Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
379. Michail Giannakos, Iro Voulgari, Sofia Papavlasopoulou, Zacharoula Papamitsiou, and Georgios N. Yannakakis. Games for artificial intelligence and machine learning education: Review and perspectives. *Non-formal and informal science learning in the ICT era*, pages 117–133, 2020.
380. Spyridon Giannatos, Yun-Gyung Cheong, Mark J. Nelson, and Georgios N. Yannakakis. Generating narrative action schemas for suspense. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.
381. Erin Gibson, Mark Griffiths, Filipa Calado, and Andrew Harris. The relationship between videogame micro-transactions and problem gaming and gambling: A systematic review. *Computers in Human Behavior*, page 107219, 2022.
382. Arthur Gill. *Introduction to the theory of Finite-State Machines*. McGraw-Hill, 1962.
383. Jeffrey M. Girard. Carma: Software for continuous affect rating and media annotation. *Journal of open research software*, 2(1), 2014.
384. Jeffrey M. Girard and Aidan G. C. Wright. Darma: Software for dual axis rating and media annotation. *Behavior research methods*, 50:902–909, 2018.
385. Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar. Adversarial reinforcement learning for procedural content generation. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.
386. Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*. Springer, 2007.
387. Carlos A. Gomez-Uribe and Neil Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.
388. Johor Jara Gonzalez, Seth Cooper, and Matthew Guzdial. Mechanic maker 2.0: reinforcement learning for evaluating generated rules. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 266–275, 2023.
389. GoodAI. Introducing our work on general-purpose LLM Agents. https://www.goodai.com/introducing-general-purpose-llm-agents/, 2023. Accessed 15 March 2024.
390. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
391. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.
392. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
393. Scott Goodfriend. A competition winning deep reinforcement learning agent in microRTS. *arXiv preprint* arXiv:2402.08112, 2024.
394. Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

395. Nitesh Goyal, Gilly Leshed, Dan Cosley, and Susan R. Fussell. Effects of implicit sharing in collaborative analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 129–138, 2014.
396. Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When Will AI Exceed Human Performance? Evidence from AI Experts. *arXiv preprint* arXiv:1705.08807, 2017.
397. Thore Graepel, Ralf Herbrich, and Julian Gold. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 193–200, 2004.
398. Joseph F. Grafsgaard, Kristy Elizabeth Boyer, and James C. Lester. Predicting facial indicators of confusion with hidden Markov models. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 97–106. Springer, 2011.
399. Jonathan Gratch. Emile: Marshalling passions in training and education. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 325–332. ACM, 2000.
400. Jonathan Gratch and Stacy Marsella. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306, 2004.
401. Jonathan Gratch and Stacy Marsella. Evaluating a computational model of emotion. *Autonomous Agents and Multi-Agent Systems*, 11(1):23–43, 2005.
402. Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. *arXiv preprint* arXiv:2010.10981, 2020.
403. Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.
404. Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. Constrained surprise search for content generation. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
405. Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. Quality diversity through surprise. *IEEE Transactions on Evolutionary Computation*, 23(4):603–616, 2018.
406. Michael Cerny Green, Gabriella AB Barros, Antonios Liapis, and Julian Togelius. Data agent. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–10, 2018.
407. Simon Greipl, Katharina Bernecker, and Manuel Ninaus. Facial and bodily expressions of emotional engagement: How dynamic measures reflect the use of game elements and subjective experience of emotions and effort. *Proceedings of the ACM on Human-Computer Interaction*, 5(CHI PLAY):1–25, 2021.
408. Mark Griffiths. A 'components' model of addiction within a biopsychosocial framework. *Journal of Substance use*, 10(4):191–197, 2005.
409. Xinjie Gu, Gang Liu, Xiangbo Zhang, Lili Tang, Xihong Zhou, and Weifang Qiu. Infrared-visible synthetic data from game engine for image fusion improvement. *IEEE Transactions on Games*, 2023.
410. Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen, Bartlomiej Kozakowski, Richard Meurling, and Lele Cao. Human-like playtesting with deep learning. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018.
411. Elin Rut Gudnadottir, Alaina K. Jensen, Yun-Gyung Cheong, Julian Togelius, Byung Chull Bae, and Christoffer Holmgård Pedersen. Detecting predatory behaviour in online game chats. In *The 2nd Workshop on Games and NLP*, 2014.
412. Markus Guhe and Alex Lascarides. Game strategies for the Settlers of Catan. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.
413. Manuel Guimaraes, Pedro Santos, and Arnav Jhala. CiF-CK: An architecture for social NPCs in commercial games. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 126–133. IEEE, 2017.
414. Maxim Gumin. Wave Function Collapse Algorithm. Personal repository: https://github.com/mxgmn/WaveFunctionCollapse, 2016. Accessed 6 October 2024.

415. Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint* arXiv:1911.03030, 2019.
416. William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. *Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.
417. Pedro Antonio Gutiérrez, Maria Perez-Ortiz, Javier Sanchez-Monedero, Francisco Fernandez-Navarro, and Cesar Hervas-Martinez. Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):127–146, 2015.
418. Matthew Guzdial, Shukan Shah, and Mark Riedl. Towards automated let's play commentary. *arXiv preprint* arXiv:1809.09424, 2018.
419. Matthew Guzdial, Sam Snodgrass, and Adam J. Summerville. *Procedural Content Generation Via Machine Learning: An Overview*. Springer, 2022.
420. David Ha and Jürgen Schmidhuber. World models. *arXiv preprint* arXiv:1803.10122, 2018.
421. Reza Habibi, Johannes Pfau, and Magy Seif El-Nasr. Modeling player personality factors from in-game behavior and affective expression. In *Proceedings of ACII*, 2023.
422. Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint* arXiv:2301.04104, 2023.
423. Johan Hagelbäck. Potential-field based navigation in StarCraft. In *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2012.
424. Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, et al. Insights from the NeurIPS 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 41–52. PMLR, 2022.
425. Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.
426. Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1):1–18, 2003.
427. Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
428. Daniel Damir Harabor and Alban Grastien. Online Graph Pruning for Pathfinding on Grid Maps. In *AAAI*, 2011.
429. Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, (37):28–29, 1972.
430. Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 297–304. IEEE, 2011.
431. Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic content generation in the Galactic Arms Race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.
432. Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Evolving content in the Galactic Arms Race video game. In *IEEE Symposium on Computational Intelligence and Games*, pages 241–248. IEEE, 2009.
433. Andreas Häuselmann. Fit for purpose? Affective computing meets EU data protection law. *International Data Privacy Law*, 2021.
434. Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general Atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.
435. Brian Hawkins. *Real-Time Cinematography for Games (Game Development Series)*. Charles River Media, Inc., 2004.
436. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillian College Publishing Company Inc., Upper Saddle River, NJ, USA, 1998.

437. Richard L. Hazlett. Measuring emotional valence during interactive experiences: boys at video game play. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 1023–1026. ACM, 2006.
438. Jennifer Healey. Recording affect in the field: Towards methods and metrics for improving ground truth labels. In *Affective Computing and Intelligent Interaction*, pages 107–116. Springer, 2011.
439. D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
440. Christine Hegedues, Joao Pedro Dias Constantino, Laurits Dixen, and Paolo Burelli. Investigating perceived and mechanical challenge in games through cognitive activity. In *IEEE Conference on Games*, 2023.
441. Norbert Heijne and Sander Bakkes. Procedural Zelda: A PCG Environment for Player Experience Research. In *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2017.
442. Alexander Heimerl, Tobias Baur, Florian Lingenfelser, Johannes Wagner, and Elisabeth André. Nova-a tool for explainable cooperative machine learning. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 109–115. IEEE, 2019.
443. Harry Helson. *Adaptation-level theory*. Harper & Row, 1964.
444. Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: a bayesian skill rating system. *Advances in neural information processing systems*, 19, 2006.
445. Ralf Herbrich, Michael E. Tipping, and Mark Hatton. Personalized behavior of computer controlled avatars in a virtual reality environment, August 15 2006. US Patent 7,090,576.
446. Javier Hernandez, Rob R. Morris, and Rosalind W. Picard. Call center stress recognition with person-specific models. In *Affective Computing and Intelligent Interaction*, pages 125–134. Springer, 2011.
447. Laura Herrewijn, Karolien Poels, and Gordon Calleja. The relationship between player involvement and immersion: An experimental investigation. In *FDG*, pages 364–367, 2013.
448. David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.
449. Philip Hingston. A Turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):169–186, 2009.
450. Philip Hingston. A new design for a Turing test for bots. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 345–350. IEEE, 2010.
451. Philip Hingston. *Believable Bots: Can Computers Play Like People?* Springer, 2012.
452. Philip Hingston, Clare Bates Congdon, and Graham Kendall. Mobile games with intelligence: A killer application? In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–7. IEEE, 2013.
453. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint* arXiv:1207.0580, 2012.
454. Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.
455. Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
456. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
457. Sarah E Hodge, Max Vykoukal, John McAlaney, Reece D Bush-Evans, Ruijie Wang, and Raian Ali. What's in the box? Exploring UK players' experiences of loot boxes in games; the conceptualisation and parallels with gambling. *PloS one*, 17(2):e0263567, 2022.
458. Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. Automated playtesting with procedural personas through MCTS with evolved heuristics. *IEEE Transactions on Games*, 11(4):352–362, 2018.
459. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Evolving personas for player decision modeling. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.

460. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Generative agents for player decision modeling in games. In *FDG*, 2014.
461. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Personas versus clones for player decision modeling. In *International Conference on Entertainment Computing*, pages 159–166. Springer, 2014.
462. Christoffer Holmgård, Georgios N. Yannakakis, Karen-Inge Karstoft, and Henrik Steen Andersen. Stress detection for PTSD via the Startlemart game. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 523–528. IEEE, 2013.
463. Christoffer Holmgård, Georgios N. Yannakakis, Héctor P. Martínez, and Karen-Inge Karstoft. To rank or to classify? Annotating stress for reliable PTSD profiling. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 719–725. IEEE, 2015.
464. Christoffer Holmgård, Georgios N. Yannakakis, Héctor P. Martínez, Karen-Inge Karstoft, and Henrik Steen Andersen. Multimodal PTSD characterization via the Startlemart game. *Journal on Multimodal User Interfaces*, 9(1):3–15, 2015.
465. Nils Iver Holtar, Mark J. Nelson, and Julian Togelius. Audioverdrive: Exploring bidirectional communication between music and gameplay. In *Proceedings of the 2013 International Computer Music Conference*, pages 124–131, 2013.
466. Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 25–30, 2007.
467. Adrian Hon. *You've Been Played: How Corporations, Governments and Schools Use Games to Control Us All*. Swift Press, 2022.
468. Amy K. Hoover, William Cachia, Antonios Liapis, and Georgios N. Yannakakis. AudioInSpace: Exploring the Creative Fusion of Generative Audio, Visuals and Gameplay. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 101–112. Springer, 2015.
469. Amy K. Hoover, Paul A. Szerlip, and Kenneth O. Stanley. Functional scaffolding for composing additional musical voices. *Computer Music Journal*, 2014.
470. Amy K. Hoover, Julian Togelius, Scott Lee, and Fernando de Mesentier Silva. The many AI challenges of Hearthstone. *KI-Künstliche Intelligenz*, 34:33–43, 2020.
471. Amy K. Hoover, Julian Togelius, and Georgios N. Yannakakis. Composing video game levels with music metaphors through functional scaffolding. In *First Computational Creativity and Games Workshop, ICCC*, 2015.
472. John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
473. Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
474. Khondoker Murad Hossain, Md Ariful Islam, Shahera Hossain, Anton Nijholt, and Md Atiqur Rahman Ahad. Status of deep learning for EEG-based brain–computer interface applications. *Frontiers in computational neuroscience*, 16:1006763, 2023.
475. Ben Houge. Cell-based music organization in Tom Clancy's EndWar. In *Demo at the AIIDE 2012 Workshop on Musical Metacreation*, 2012.
476. Ryan Houlette. *Player Modeling for Adaptive Games. AI Game Programming Wisdom II*, pages 557–566. Charles River Media, Inc., 2004.
477. Andrew Howlett, Simon Colton, and Cameron Browne. Evolving pixel shaders for the prototype video game Subversion. In *The Thirty Sixth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB'10), De Montfort University, Leicester, UK, 30th March*, 2010.
478. Johanna Höysniemi, Perttu Hämäläinen, Laura Turkki, and Teppo Rouvi. Children's intuitive gestures in vision-based action games. *Communications of the ACM*, 48(1):44–50, 2005.
479. Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

480. Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002.
481. Chengpeng Hu, Yunlong Zhao, and Jialin Liu. Game generation via large language models. In *IEEE Conference on Games*, 2024.
482. Chengpeng Hu, Yunlong Zhao, Ziqi Wang, Haocheng Du, and Jialin Liu. Games for artificial intelligence research: A review and perspectives. *IEEE Transactions on Artificial Intelligence*, pages 1–20, 2024.
483. Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint* arXiv:2106.09685, 2021.
484. Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning. *arXiv preprint* arXiv:1912.02288, 2019.
485. Shiying Hu, Zengrong Huang, Chengpeng Hu, and Jialin Liu. 3D Building Generation in Minecraft via Large Language Models. In *IEEE Conference on Games*, 2024.
486. Minh Hua and Rita Raley. Playing with unicorns: AI dungeon and citizen NLP. *Digital Humanities Quarterly*, 14(4), 2020.
487. Dan Huang and Scott Lee. A self-play policy optimization approach to battling pokémon. In *2019 IEEE conference on games (CoG)*, pages 1–4. IEEE, 2019.
488. Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.
489. E. Hudlicka. Affective game engines: motivation and requirements. In *Proceedings of International Conference on Foundations of Digital Games (FDG)*, pages 299–306. ACM, 2009.
490. M. Hussain, O. Alzoubi, R. Calvo, and S. D'Mello. Affect detection from multichannel physiology during learning sessions with autotutor. In *Proceedings of International Conference in Artificial Intelligence in Education (AIED)*, pages 131–138. Springer, 2011.
491. Wijnand IJsselsteijn, Karolien Poels, and Y. A. W. De Kort. The game experience questionnaire: Development of a self-report measure to assess player experiences of digital games. *TU Eindhoven, Eindhoven, The Netherlands*, 2008.
492. Interactive Data Visualization. SpeedTree, 2010. http://www.speedtree.com/.
493. Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. Discovering unique game variants. In *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, 2015.
494. Aaron Isaksen, Daniel Gopstein, and Andrew Nealen. Exploring Game Space Using Survival Analysis. In *Proceedings of Foundations of Digital Games (FDG)*, 2015.
495. Katherine Isbister and Noah Schaffer. *Game usability: Advancing the player experience*. CRC Press, 2015.
496. Tatsuya Ishigaki, Goran Topić, Yumi Hamazono, Hiroshi Noji, Ichiro Kobayashi, Yusuke Miyao, and Hiroya Takamura. Generating racing game commentary from vision, language, and structured data. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 103–113, 2021.
497. Takaaki Ishii, Hiroki Komiyama, Takahiro Shinozaki, Yasuo Horiuchi, and Shingo Kuroiwa. Reverberant speech recognition based on denoising autoencoder. In *Interspeech*, pages 3512–3516, 2013.
498. Damian Isla. Handling complexity in the Halo 2 AI. In *Game Developers Conference*, 2005.
499. Damian Isla and Bruce Blumberg. New challenges for character-based AI for games. In *Proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment*, pages 41–45. AAAI Press, 2002.
500. Toru Iwatani, Mirei Takahashi, and Youichiro Miyake. Toru Iwatani, the PAC-MAN Creator, Interview about the Origin of Game AI "PAC-MAN". *Japanese Society of Artificial Intelligence*, 34(1):86–99, 2019.
501. Susan A. Jackson and Robert C. Eklund. Assessing flow in physical activity: the flow state scale-2 and dispositional flow scale-2. *Journal of Sport & Exercise Psychology*, 24(2), 2002.

502. Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte Mario: platforming with MCTS. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 293–300. ACM, 2014.
503. Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. In *AIIDE*, 2012.
504. Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Autoencoders for level generation, repair, and recognition. In *ICCC Workshop on Computational Creativity and Games*, 2016.
505. Daniel Jallov, Sebastian Risi, and Julian Togelius. EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):181–191, 2017.
506. Susan Jamieson. Likert scales: how to (ab) use them. *Medical Education*, 38(12):1217–1218, 2004.
507. Aki Järvinen. Gran stylissimo: The audiovisual elements and styles in computer and video games. In *Proceedings of Computer Games and Digital Cultures Conference*, 2002.
508. Arnav Jhala and R. Michael Young. Cinematic visual discourse: Representation, generation, and evaluation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):69–81, 2010.
509. Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint* arXiv:2310.06825, 2023.
510. Zehua Jiang, Sam Earle, Michael Green, and Julian Togelius. Learning controllable 3D level generators. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, pages 1–9, 2022.
511. Yuu Jinnai and Alex S. Fukunaga. Learning to prune dominated action sequences in online black-box planning. In *AAAI*, pages 839–845, 2017.
512. Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142, 1998.
513. Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of AI ethics guidelines. *Nature machine intelligence*, 1(9):389–399, 2019.
514. Niklas Johannes, Matti Vuorre, and Andrew K Przybylski. Video game play is positively correlated with well-being. *Royal Society Open Science*, 8(2):202049, 2021.
515. Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
516. Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, pages 4246–4247, 2016.
517. Tom Johnstone and Klaus R. Scherer. Vocal communication of emotion. In *Handbook of emotions*, pages 220–235. Guilford Press, New York, 2000.
518. Daniel Joseph. Battle pass capitalism. *Journal of Consumer Culture*, 21(1):68–83, 2021.
519. Patrik N. Juslin and Klaus R. Scherer. *Vocal expression of affect*. Oxford University Press, Oxford, UK, 2005.
520. Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20, 2019.
521. Niels Justesen, Maria Kaselimi, Sam Snodgrass, Miruna Vozaru, Matthew Schlegel, Jonas Wingren, Gabriella A. B. Barros, Tobias Mahlmann, Shyam Sudhakaran, Wesley Kerr, Albert Wang, Christoffer Holmgård, Georgios N. Yannakakis, Sebastian Risi, and Julian Togelius. Human-like Bots for Tactical Shooters Using Compute-Efficient Sensors. Technical report, modl.ai, 2024.
522. Niels Justesen, Tobias Mahlmann, and Julian Togelius. Online evolution for multi-action adversarial games. In *European Conference on the Applications of Evolutionary Computation*, pages 590–603. Springer, 2016.
523. Niels Justesen and Sebastian Risi. Continual Online Evolutionary Planning for In-Game Build Order Adaptation in StarCraft. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, 2017.

524. Niels Justesen, Bálint Tillman, Julian Togelius, and Sebastian Risi. Script-and cluster-based UCT for StarCraft. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.
525. Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint* arXiv:1806.10729, 2018.
526. Tróndur Justinussen, Peter Hald Rasmussen, Alessandro Canossa, and Julian Togelius. Resource systems in games: An analytical approach. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 171–178. IEEE, 2012.
527. Jesper Juul. Games telling stories. *Game Studies*, 1(1):45, 2001.
528. Jesper Juul. Swap adjacent gems to make sets of three: A history of matching tile games. *Artifact: Journal of Design Practice*, 1(4):205–217, 2007.
529. Jesper Juul. *A casual revolution: Reinventing video games and their players*. MIT Press, 2010.
530. Jesper Juul. The independent mode: A functionalist account of independent games and game history. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, pages 1–4, 2020.
531. Souhila Kaci. *Working with preferences: Less is more*. Springer, 2011.
532. Daniel Kahneman. A perspective on judgment and choice: mapping bounded rationality. *American psychologist*, 58(9):697, 2003.
533. Daniel Kahneman and Jason Riis. Living, and thinking about it: Two perspectives on life. *The science of well-being*, pages 285–304, 2005.
534. Daniel Kahneman, Paul Slovic, and Amos Tversky. *Judgment under uncertainty: Heuristics and biases*. Cambridge university press, 1982.
535. Anssi Kanervisto, Tomi Kinnunen, and Ville Hautamaki. GAN-Aimbots: Using Machine Learning for Cheating in First Person Shooters. *IEEE Transactions on Games*, 2022.
536. Theofanis Kannetis and Alexandros Potamianos. Towards adapting fantasy, curiosity and challenge in multimodal dialogue systems for preschoolers. In *Proceedings of International Conference on Multimodal Interfaces (ICMI)*, pages 39–46. ACM, 2009.
537. Theofanis Kannetis, Alexandros Potamianos, and Georgios N. Yannakakis. Fantasy, curiosity and challenge as adaptation indicators in multimodal dialogue systems for preschoolers. In *Proceedings of the 2nd Workshop on Child, Computer and Interaction*. ACM, 2009.
538. Ashish Kapoor, Winslow Burleson, and Rosalind W. Picard. Automatic prediction of frustration. *International Journal of Human-Computer Studies*, 65(8):724–736, 2007.
539. Sergey Karakovskiy and Julian Togelius. The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.
540. Daniël Karavolos, Anders Bouwer, and Rafael Bidarra. Mixed-initiative design of game levels: Integrating mission and space into level generation. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*, 2015.
541. Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. Learning the patterns of balance in a multi-player shooter game. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2017.
542. Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. Pairing character classes in a deathmatch shooter game via a deep-learning surrogate model. In *Proceedings of the 13th international conference on the Foundations of digital games*, 2018.
543. Sara Karimi, Sahar Asadi, Francesco Lorenzo, and Amir H. Payberah. CandyRL: A hybrid reinforcement learning model for gameplay. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 567–572. IEEE, 2022.
544. Kostas Karpouzis and Georgios N. Yannakakis. *Emotion in Games: Theory and Praxis*. Springer, 2016.
545. Kostas Karpouzis, Georgios N. Yannakakis, Noor Shaker, and Stylianos Asteriadis. The Platformer Experience Dataset. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 712–718. IEEE, 2015.

546. Igor V. Karpov, Leif Johnson, and Risto Miikkulainen. Evaluation methods for active human-guided neuroevolution in games. In *2012 AAAI Fall Symposium on Robots Learning Interactively from Human Teachers (RLIHT)*, 2012.
547. Igor V. Karpov, Jacob Schrum, and Risto Miikkulainen. Believable bot navigation via playback of human traces. In Philip Hingston, editor, *Believable Bots: Can Computers Play Like People?* Springer, 2012.
548. Isaac Karth and Adam M. Smith. WaveFunctionCollapse: content generation via constraint solving and machine learning. *IEEE Transactions on Games*, 14(3):364–376, 2021.
549. Leonard Kaufman and Peter J. Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.
550. Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.
551. Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A survey of reinforcement learning from human feedback. *arXiv preprint* arXiv:2312.14925, 2023.
552. Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.
553. Markus Kemmerling and Mike Preuss. Automatic adaptation to generated content via car setup optimization in TORCS. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 131–138. IEEE, 2010.
554. Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *arXiv preprint* arXiv:1605.02097, 2016.
555. Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A Survey of NP-Complete Puzzles. *ICGA Journal*, 31(1):13–34, 2008.
556. Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
557. Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios N. Yannakakis. A procedural procedural level generator generator. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 335–341. IEEE, 2012.
558. Rilla Khaled and Georgios N. Yannakakis. Village voices: An adaptive game for conflict resolution. In *Proceedings of FDG*, pages 425–426, 2013.
559. Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. PCGRL: Procedural content generation via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 95–101, 2020.
560. Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying MCTS for Human-like General Video Game Playing. In *Proceedings of IJCAI*, 2016.
561. Ahmed Khalifa, Diego Perez-Liebana, Simon M. Lucas, and Julian Togelius. General video game level generation. In *Proceedings of IJCAI*, 2016.
562. Changyeon Kim, Jongjin Park, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Preference transformer: Modeling human preferences using transformers for RL. *arXiv preprint* arXiv:2303.00957, 2023.
563. K-J Kim, Heejin Choi, and Sung-Bae Cho. Hybrid of evolution and reinforcement learning for Othello players. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 203–209. IEEE, 2007.
564. Kyung-Min Kim, Chang-Jun Nan, Jung-Woo Ha, Yu-Jung Heo, and Byoung-Tak Zhang. Pororobot: A deep learning robot that plays video Q&A games. In *AAAI 2015 Fall Symposium on AI for Human-Robot Interaction (AI-HRI 2015)*, 2015.
565. Steven Orla Kimbrough, Gary J. Koehler, Ming Lu, and David Harlan Wood. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2):310–327, 2008.
566. Daniel L King, Paul H Delfabbro, Sally M Gainsbury, Michael Dreier, Nancy Greer, and Joël Billieux. Unfair play? video games as exploitative monetized services: An examination of game patents from a consumer protection perspective. *Computers in Human Behavior*, 101:131–143, 2019.

567. Daniel L. King, Alex M. T. Russell, Paul H. Delfabbro, and Dean Polisena. Fortnite microtransaction spending was associated with peers' purchasing behaviors but not gaming disorder symptoms. *Addictive Behaviors*, 104:106311, 2020.
568. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint* arXiv:1412.6980, 2014.
569. Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint* arXiv:1312.6114, 2013.
570. Michael Kipp. Anvil-a generic annotation tool for multimodal dialogue. In *Seventh European conference on speech communication and technology*. Citeseer, 2001.
571. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for AI. *AI magazine*, 18(1):73–73, 1997.
572. A. Kleinsmith and N. Bianchi-Berthouze. Affective body expression perception and recognition: A survey. *IEEE Transactions on Affective Computing*, 2012.
573. Andrea Kleinsmith and Nadia Bianchi-Berthouze. Form as a cue in the automatic recognition of non-acted affective body expressions. In *Affective Computing and Intelligent Interaction*, pages 155–164. Springer, 2011.
574. Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint* arXiv:2310.00166, 2023.
575. Yana Knight, Héctor Perez Martínez, and Georgios N. Yannakakis. Space maze: Experience-driven game camera control. In *FDG*, pages 427–428, 2013.
576. Matthias J. Koepp, Roger N. Gunn, Andrew D. Lawrence, Vincent J. Cunningham, Alain Dagher, Tasmin Jones, David J. Brooks, C. J. Bench, and P. M. Grasby. Evidence for striatal dopamine release during a video game. *Nature*, 393(6682):266–268, 1998.
577. Teuvo Kohonen. *Self-Organizing Maps*. Springer, Secaucus, NJ, USA, 3rd edition, 2001.
578. Andrey N. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Russian, American Mathematical Society Translation 28 (1963) 55-59. Doklady Akademiia Nauk SSR*, 14(5):953–956, 1957.
579. Richard Konečnỳ. Modeling of fighting game players. Master's thesis, Institute of Digital Games, University of Malta, 2016.
580. Michael Kosfeld, Markus Heinrichs, Paul J. Zak, Urs Fischbacher, and Ernst Fehr. Oxytocin increases trust in humans. *Nature*, 435(7042):673–676, 2005.
581. Raph Koster. *Theory of fun for game design*. O'Reilly Media, Inc., 2013.
582. Bartosz Kostka, Jaroslaw Kwiecien, Jakub Kowalski, and Pawel Rychlikowski. Text-based Adventures of the Golovin AI Agent. *arXiv preprint* arXiv:1705.05637, 2017.
583. Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 1061–1068. ACM, 2013.
584. Jakub Kowalski and Andrzej Kisielewicz. Towards a Real-time Game Description Language. In *ICAART (2)*, pages 494–499, 2016.
585. Jakub Kowalski, Radosław Miernik, Katarzyna Polak, Dominik Budzki, and Damian Kowalik. Introducing Tales of Tribute AI Competition. In *2024 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2024.
586. Jakub Kowalski and Radostaw Miernik. Evolutionary approach to collectible arena deckbuilding using active card game genes. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020.
587. Jakub Kowalski and Marek Szykuła. Evolving chess-like games using relative algorithm performance profiles. In *European Conference on the Applications of Evolutionary Computation*, pages 574–589. Springer, 2016.
588. John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.

589. Teofebano Kristo and Nur Ulfa Maulidevi. Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies. In *Data and Software Engineering (ICoDSE), 2016 International Conference on*. IEEE, 2016.
590. Jacob Leon Kröger, Otto Hans-Martin Lutz, and Philip Raschke. Privacy implications of voice and speech analysis–information disclosure by inference. In *IFIP International Summer School on Privacy and Identity Management*, pages 242–258. Springer, 2019.
591. Jacob Leon Kröger, Philip Raschke, Jessica Percy Campbell, and Stefan Ullrich. Surveilling the gamers: Privacy impacts of the video game industry. *Available at SSRN 3881279*, 2021.
592. Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
593. Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline Q-learning on Diverse Multi-Task Data Both Scales And Generalizes. In *The Eleventh International Conference on Learning Representations*, 2023.
594. Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.
595. Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4501–4510, 2020.
596. Haewoon Kwak, Jeremy Blackburn, and Seungyeop Han. Exploring cyberbullying and other toxic behavior in team competition online games. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3739–3748, 2015.
597. Ben Kybartas and Rafael Bidarra. A semantic foundation for mixed-initiative computational storytelling. In *Interactive Storytelling*, pages 162–169. Springer, 2015.
598. Alexandros Labrinidis and Hosagrahar V. Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.
599. Gorm Lai, Frederic Fol Leymarie, and William Latham. On mixed-initiative content creation for video games. *IEEE Transactions on Games*, 14(4):543–557, 2022.
600. John Laird and Michael van Lent. Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15, 2001.
601. Thijmen Stefanus Leendert Langendam and Rafael Bidarra. miWFC-Designer empowerment through mixed-initiative Wave Function Collapse. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022.
602. G. B. Langley and H. Sheppeard. The visual analogue scale: its use in pain measurement. *Rheumatology International*, 5(4):145–148, 1985.
603. Frank Lantz. Bridge. *Donkeyspace (Blog)*, 2024.
604. Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. Depth in strategic games. In *Proceedings of the AAAI WNAIG Workshop*, 2017.
605. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. *Learning classifier systems: from foundations to applications*. Springer, 2003.
606. Steven M. LaValle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics*, pages 303–307, 2001.
607. Richard S. Lazarus. *Emotion and adaptation*. Oxford University Press, 1991.
608. Nicole Lazzaro. Why we play games: Four keys to more emotion without story. Technical report, XEO Design Inc., 2004.
609. Phuc H. Le-Khac, Graham Healy, and Alan F. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 2020.
610. Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*, volume 3361. Cambridge, MA: MIT Press, 1995.
611. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

612. David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
613. Eunjo Lee, Yoonjae Jang, Du-Mim Yoon, JiHoon Jeon, Seong-il Yang, Sang-Kwang Lee, Dae-Wook Kim, Pei Pei Chen, Anna Guitart, Paul Bertens, et al. Game data mining competition on churn prediction and survival analysis using commercial game log data. *IEEE Transactions on Games*, 11(3):215–226, 2018.
614. Jae Joong Lee, Bosheng Li, and Bedrich Benes. Latent l-systems: Transformer-based tree generator. *ACM Transactions on Graphics*, 43(1):1–16, 2023.
615. Kuang-Huei Lee, Ofir Nachum, Mengjiao Sherry Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35:27921–27936, 2022.
616. Scott Lee and Julian Togelius. Showdown AI competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 191–198. IEEE, 2017.
617. Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.
618. Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218, 2011.
619. Wendy G Lehnert and Elaine W Vine. The role of affect in narrative structure. *Cognition and emotion*, 1(3):299–322, 1987.
620. James C Lester, Sharolyn A Converse, Susan E Kahler, S Todd Barlow, Brian A Stone, and Ravinder S Bhogal. The persona effect: affective impact of animated pedagogical agents. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 359–366, 1997.
621. Alan Levinovitz. The mystery of Go, the ancient game that computers still can't win. *Wired Magazine*, 2014.
622. M Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint* arXiv:1910.13461, 2019.
623. Mike Lewis and Kevin Dill. Game AI appreciation, revisited. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, pages 3–18. AK Peters/CRC Press, 2015.
624. Boyang Li, Stephen Lee-Urban, Darren Scott Appling, and Mark O. Riedl. Crowdsourcing narrative intelligence. *Advances in Cognitive Systems*, 2(1), 2012.
625. Chengxi Li, Sagar Gandhi, and Brent Harrison. End-to-end let's play commentary generation using multi-modal video representations. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–7, 2019.
626. Wenzhe Li, Zihan Ding, Seth Karten, and Chi Jin. Fightladder: A benchmark for competitive multi-agent reinforcement learning. *arXiv preprint* arXiv:2406.02081, 2024.
627. Yuchen Li, Ziqi Wang, Qingquan Zhang, and Jialin Liu. Measuring diversity of game scenarios. *arXiv preprint* arXiv:2404.15192, 2024.
628. Hai Liang and Jiaqi Li. A study on the agent in fighting games based on deep reinforcement learning. *Mobile Information Systems*, 2022(1):9984617, 2022.
629. Antonios Liapis. The Discourse on Computational Creativity in Indie Tabletop Role-Playing Games. In preparation.
630. Antonios Liapis. Creativity facet orchestration: the whys and the hows. *Artificial and Computational Intelligence in Games: Integration; Dagstuhl Follow-Ups*, page 217, 2015.
631. Antonios Liapis. Mixed-initiative Creative Drawing with webIconoscope. In *Proceedings of the 6th International Conference on Computational Intelligence in Music, Sound, Art and Design. (EvoMusArt)*. Springer, 2017.
632. Antonios Liapis. 10 Years of the PCG workshop: Past and Future Trends. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020.
633. Antonios Liapis, Héctor P. Martınez, Julian Togelius, and Georgios N. Yannakakis. Transforming exploratory creativity with DeLeNoX. In *Proceedings of the Fourth International Conference on Computational Creativity*, pages 56–63, 2013.

634. Antonios Liapis, Gillian Smith, and Noor Shaker. Mixed-initiative content creation. *Procedural content generation in games*, pages 195–214, 2016.
635. Antonios Liapis and Georgios N. Yannakakis. Boosting computational creativity with human interaction in mixed-initiative co-creation tasks. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.
636. Antonios Liapis, Georgios N. Yannakakis, Mark J Nelson, Mike Preuss, and Rafael Bidarra. Orchestrating game generation. *IEEE Transactions on Games*, 11(1):48–68, 2018.
637. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Neuroevolutionary constrained optimization for content creation. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 71–78. IEEE, 2011.
638. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.
639. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Designer modeling for personalized game content creation tools. In *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013.
640. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient Sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, pages 213–220, 2013.
641. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient World: Human-Based Procedural Cartography. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 180–191. Springer, 2013.
642. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Towards a generic method of evaluating game levels. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 9, pages 30–36, 2013.
643. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Computational Game Creativity. In *Proceedings of the Fifth International Conference on Computational Creativity*, pages 285–292, 2014.
644. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Constrained novelty search: A study on game content generation. *Evolutionary Computation*, 23(1):101–129, 2015.
645. Diego Pérez Liébana, Simon M. Lucas, Raluca D. Gaina, Julian Togelius, Ahmed Khalifa, and Jialin Liu. *General video game artificial intelligence*. Springer, 2020.
646. Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.
647. Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.
648. Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving behaviour trees for the commercial game DEFCON. In *European Conference on the Applications of Evolutionary Computation*, pages 100–110. Springer, 2010.
649. Chong-U Lim and D. Fox Harrell. Revealing social identity phenomena in videogames with archetypal analysis. In *Proceedings of the 6th International AISB Symposium on AI and Games*, 2015.
650. Aristid Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.
651. R. L. Linn and N. E. Gronlund. *Measurement and assessment in teaching*. Prentice-Hall, 2000.
652. Nir Lipovetzky and Hector Geffner. Width-based algorithms for classical planning: New results. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 1059–1060. IOS Press, 2014.
653. Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical Planning with Simulators: Results on the Atari Video Games. In *Proceedings of IJCAI*, pages 1610–1616, 2015.

654. Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37, 2021.
655. Xi Liu, Muhe Xie, Xidao Wen, Rui Chen, Yong Ge, Nick Duffield, and Na Wang. A semi-supervised and inductive embedding model for churn prediction of large-scale mobile games. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 277–286. IEEE, 2018.
656. Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on knowledge and data engineering*, 35(1):857–876, 2021.
657. Maria Teresa Llano, Michael Cook, Christian Guckelsberger, Simon Colton, and Rose Hepworth. Towards the automatic generation of fictional ideas for games. In *Experimental AI in Games (EXAG'14), a Workshop collocated with the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'14). AAAI Publications*, 2014.
658. Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lönneker, Luigi Cardamone, Diego Perez, Yago Sáez, Mike Preuss, and Jan Quadflieg. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131–147, 2010.
659. Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard Kinnaird-Heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G. Reynolds, and Yago Saez. The WCCI 2008 simulated car racing competition. In *IEEE Symposium on Computational Intelligence and Games*, pages 119–126. IEEE, 2008.
660. Matthew Lombard and Theresa Ditton. At the heart of it all: The concept of presence. *Journal of computer-mediated communication*, 3(2):JCMC321, 1997.
661. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Sonancia: Sonification of procedurally generated game levels. In *Proceedings of the ICCC workshop on Computational Creativity & Games*, 2015.
662. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Framing tension for game generation. In *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.
663. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Modelling affect for horror soundscapes. *IEEE Transactions on Affective Computing*, 2017.
664. Phil Lopes, Georgios N. Yannakakis, and Antonios Liapis. RankTrace: Relative and Unbounded Affect Annotation. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
665. Ricardo Lopes and Rafael Bidarra. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):85–99, 2011.
666. Sandy Louchart, Ruth Aylett, Joao Dias, and Ana Paiva. Unscripted narrative for affectively driven characters. In *AIIDE*, pages 81–86, 2005.
667. Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group, Computer Science Department, Stanford University, 2008.
668. Peter Lovei, Iryna Nazarchuk, Suhaib Aslam, Bin Yu, Carl Megens, and Natalia Sidorova. Designing micro-intelligences for situated affective computing. In *CHIIoT@ EWSN/EICS*, 2021.
669. A. Bryan Loyall and Joseph Bates. Personality-rich believable agents that use language. In *Proceedings of the First International Conference on Autonomous Agents*, pages 106–113. ACM, 1997.
670. Feiyu Lu, Kaito Yamamoto, Luis H. Nomura, Syunsuke Mizuno, YoungMin Lee, and Ruck Thawonmas. Fighting game artificial intelligence competition platform. In *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pages 320–323. IEEE, 2013.

671. Simon M. Lucas. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 203–210, 2005.
672. Simon M. Lucas. Ms Pac-Man competition. *ACM SIGEVOlution*, 2(4):37–38, 2007.
673. Simon M. Lucas. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing*, 5(1):45–57, 2008.
674. Simon M. Lucas and Graham Kendall. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE*, 1(1):10–18, 2006.
675. Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). *Dagstuhl Reports*, 2(5):43–70, 2012.
676. Simon M. Lucas and T. Jeff Reynolds. Learning finite-state transducers: Evolution versus heuristic state merging. *IEEE Transactions on Evolutionary Computation*, 11(3):308–325, 2007.
677. Li Lucy and David Bamman. Gender and representation bias in GPT-3 generated stories. In *Proceedings of the Third Workshop on Narrative Understanding*, pages 48–55, 2021.
678. Jeremy Ludwig and Art Farley. A learning infrastructure for improving agent performance and game balance. In Georgios N. Yannakakis and John Hallam, editors, *Proceedings of the AIIDE'07 Workshop on Optimizing Player Satisfaction, Technical Report WS-07-01*, pages 7–12. AAAI Press, 2007.
679. Kristian Lum and William Isaac. To predict and serve? *Significance*, 13(5):14–19, 2016.
680. Kevin Lynch. *The Image of the City*. MIT Press, 1960.
681. Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.
682. Tiago Machado, Dan Gopstein, Andy Nealen, and Julian Togelius. Pitako-recommending game design elements in cicero. In *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.
683. James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, number 14, pages 281–297. Oakland, CA, USA, 1967.
684. Brian Magerko. Story representation and interactive drama. In *AIIDE*, pages 87–92, 2005.
685. Brendan Maher. Can a video game company tame toxic behaviour? *Nature*, 531(7596):568–571, 2016.
686. Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N. Yannakakis. Predicting player behavior in Tomb Raider: Underworld. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 178–185. IEEE, 2010.
687. Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. Modelling and evaluation of complex scenarios with the strategy game description language. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 174–181. IEEE, 2011.
688. Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. Evolving card sets towards balancing Dominion. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012.
689. Emanuel Maiberg. 'No Mans Sky'is like 18 quintillion bowls of oatmeal. *Vice, Aug*, 2016.
690. Kevin Majchrzak, Jan Quadflieg, and Günter Rudolph. Advanced dynamic scripting for fighting game AI. In *International Conference on Entertainment Computing*, pages 86–99. Springer, 2015.
691. Konstantinos Makantasis. Affranknet+: ranking affect using privileged information. In *2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 2021.
692. Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. From pixels to affect: A study on games and player experience. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 1–7. IEEE, 2019.
693. Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. The pixels and sounds of emotion: General-purpose representations of arousal in games. *IEEE Transactions on Affective Computing*, 2021.

694. Konstantinos Makantasis, Kosmas Pinitas, Antonios Liapis, and Georgios N. Yannakakis. The invariant ground truth of affect. In *2022 10th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 2022.
695. Konstantinos Makantasis, Kosmas Pinitas, Antonios Liapis, and Georgios N. Yannakakis. From the lab to the wild: Affect modeling via privileged information. *IEEE Transactions on Affective Computing*, (01):1–13, 2023.
696. Sasha Makarovych, Alessandro Canossa, Julian Togelius, and Anders Drachen. Like a DNA string: Sequence-based Player Profiling in Tom Clancy's The Division. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*. York, 2018.
697. Nikos Malandrakis, Alexandros Potamianos, Georgios Evangelopoulos, and Athanasia Zlatintsi. A supervised approach to movie emotion tracking. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2376–2379. IEEE, 2011.
698. Thomas W. Malone. What makes computer games fun? *Byte*, 6:258–277, 1981.
699. Regan L. Mandryk and M. Stella Atkins. A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies. *International Journal of Human-Computer Studies*, 65(4):329–347, 2007.
700. Regan L. Mandryk, Kori M. Inkpen, and Thomas W. Calvert. Using psychophysiological techniques to measure user experience with entertainment technologies. *Behaviour & Information Technology*, 25(2):141–158, 2006.
701. Jacek Mandziuk. Computational intelligence in mind games. *Challenges for Computational Intelligence*, 63:407–442, 2007.
702. Jacek Mandziuk. *Knowledge-free and learning-based methods in intelligent game playing*. Springer, 2010.
703. Mehak Maniktala, Chris Miller, Aaron Margolese-Malin, Arnav Jhala, and Chris Martens. Minuet: Procedural musical accompaniment for textual narratives. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, pages 1–7, 2020.
704. Benjamin Mark, Tudor Berechet, Tobias Mahlmann, and Julian Togelius. Procedural Generation of 3D Caves for Games on the GPU. In *Proceedings of the Conference on the Foundations of Digital Games (FDG)*, 2015.
705. Dave Mark. *Behavioral Mathematics for game AI*. Charles River Media, 2009.
706. Dave Mark and Kevin Dill. Improving AI decision modeling through utility theory. In *Game Developers Conference*, 2010.
707. Gloria Mark, Yiran Wang, and Melissa Niiya. Stress and multitasking in everyday college life: An empirical study of online activity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 41–50, 2014.
708. Stacy Marsella, Jonathan Gratch, and Paolo Petta. Computational models of emotion. *A Blueprint for Affective Computing—A sourcebook and manual*, 11(1):21–46, 2010.
709. Chris Martens. Ceptre: A Language for Modeling Generative Interactive Systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
710. Héctor P. Martínez, Yoshua Bengio, and Georgios N. Yannakakis. Learning deep physiological models of affect. *Computational Intelligence Magazine, IEEE*, 9(1):20–33, 2013.
711. Héctor P. Martínez, Maurizio Garbarino, and Georgios N. Yannakakis. Generic physiological features as predictors of player experience. In *Affective Computing and Intelligent Interaction*, pages 267–276. Springer, 2011.
712. Héctor P. Martínez, Kenneth Hullett, and Georgios N. Yannakakis. Extending neuroevolutionary preference learning through player modeling. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 313–320. IEEE, 2010.
713. Héctor P. Martínez and Georgios N. Yannakakis. Genetic search feature selection for affective modeling: a case study on reported preferences. In *Proceedings of the 3rd International Workshop on Affective Interaction in Natural Environments*, pages 15–20. ACM, 2010.
714. Héctor P. Martínez and Georgios N. Yannakakis. Mining multimodal sequential patterns: a case study on affect detection. In *Proceedings of International Conference on Multimodal Interfaces (ICMI)*, pages 3–10. ACM, 2011.

715. Héctor P. Martínez and Georgios N. Yannakakis. Deep multimodal fusion: Combining discrete events and continuous signals. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 34–41. ACM, 2014.
716. Héctor P. Martínez, Georgios N. Yannakakis, and John Hallam. Don't Classify Ratings of Affect; Rank them! *IEEE Transactions on Affective Computing*, 5(3):314–326, 2014.
717. Giovanna Martinez-Arellano, Richard Cant, and David Woods. Creating AI Characters for Fighting Games using Genetic Programming. *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.
718. Michael Mateas. *Interactive Drama, Art and Artificial Intelligence*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.
719. Michael Mateas. Expressive AI: Games and Artificial Intelligence. In *DIGRA Conference*, 2003.
720. Michael Mateas and Andrew Stern. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4):39–47, 2002.
721. Michael Mateas and Andrew Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference*, 2003.
722. Michael Mauderer, Simone Conte, Miguel A. Nacenta, and Dhanraj Vishwanath. Depth perception with gaze-contingent depth of field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 217–226, 2014.
723. Paris Mavromoustakos-Blom, David Melhart, Antonios Liapis, Georgios N. Yannakakis, Sander Bakkes, and Pieter Spronck. Multiplayer tension in the wild: A hearthstone case. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023.
724. John D. Mayer and Peter Salovey. The intelligence of emotional intelligence. *Intelligence*, 17(4):433–442, 1993.
725. Allan Mazur, Elizabeth J. Susman, and Sandy Edelbrock. Sex difference in testosterone response to a video game contest. *Evolution and Human Behavior*, 18(5):317–326, 1997.
726. Andrew McAfee, Erik Brynjolfsson, Thomas H. Davenport, D. J. Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012.
727. John McCarthy. Chess as the Drosophila of AI. In *Computers, chess, and cognition*, pages 227–237. Springer, 1990.
728. John McCarthy. Partial formalizations and the Lemmings game. Technical report, Stanford University, 1998.
729. Josh McCoy, Mike Treanor, Ben Samuel, Michael Mateas, and Noah Wardrip-Fruin. Prom Week: social physics as gameplay. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 319–321. ACM, 2011.
730. Josh McCoy, Mike Treanor, Ben Samuel, Aaron A. Reed, Noah Wardrip-Fruin, and Michael Mateas. Prom Week. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 235–237. ACM, 2012.
731. Joshua McCoy, Mike Treanor, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. Comme il faut: A system for authoring playable social models. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 7, pages 158–163, 2011.
732. R Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L Griffiths. Embers of autoregression: Understanding large language models through the problem they are trained to solve. *arXiv preprint* arXiv:2309.13638, 2023.
733. Robert R. McCrae and Paul T. Costa Jr. A five-factor theory of personality. *Handbook of personality: Theory and research*, 2:139–153, 1999.
734. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
735. Daniel McDuff, Abdelrahman Mahmoud, Mohammad Mavadati, May Amr, Jay Turcot, and Rana el Kaliouby. AFFDEX SDK: a cross-platform real-time multi-face expression recognition toolkit. In *Proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems*, pages 3723–3726, 2016.

736. Scott W. McQuiggan, Sunyoung Lee, and James C. Lester. Early prediction of student frustration. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction*, pages 698–709. Springer, 2007.
737. Scott W. McQuiggan, Bradford W. Mott, and James C. Lester. Modeling self-efficacy in intelligent tutoring systems: An inductive approach. *User Modeling and User-Adapted Interaction*, 18(1):81–123, 2008.
738. Albert Mehrabian. An approach to environmental psychology. *Massachusetts Institute of Technology*, 1974.
739. Elisa D. Mekler, Julia Ayumi Bopp, Alexandre N. Tuch, and Klaus Opwis. A systematic review of quantitative studies on the enjoyment of digital entertainment games. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 927–936, 2014.
740. David Melhart, Ahmad Azadvar, Alessandro Canossa, Antonios Liapis, and Georgios N. Yannakakis. Your gameplay says it all: Modelling motivation in tom clancy's the division. In *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.
741. David Melhart, Daniele Gravina, and Georgios N. Yannakakis. Moment-to-moment engagement prediction through the eyes of the observer: PUBG streaming on Twitch. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020.
742. David Melhart, Antonios Liapis, and Georgios N. Yannakakis. PAGAN: Video affect annotation made easy. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 130–136. IEEE, 2019.
743. David Melhart, Antonios Liapis, and Georgios N. Yannakakis. Towards general models of player experience: A study within genres. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE, 2021.
744. David Melhart, Antonios Liapis, and Georgios N. Yannakakis. The arousal video game annotation (AGAIN) dataset. *IEEE Transactions on Affective Computing*, 13(4):2171–2184, 2022.
745. David Melhart, Julian Togelius, Benedikte Mikkelsen, Christoffer Holmgård, and Georgios N. Yannakakis. The Ethics of AI in Games. *IEEE Transactions on Affective Computing*, 2023.
746. David Melhart, Georgios N. Yannakakis, and Antonios Liapis. I feel I feel you: A theory of mind experiment in games. *KI-Künstliche Intelligenz*, 34:45–55, 2020.
747. Andre Mendes, Julian Togelius, and Andy Nealen. Hyper-heuristic general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
748. Hongying Meng, Andrea Kleinsmith, and Nadia Bianchi-Berthouze. Multi-score learning for affect recognition: The case of body postures. In *Affective Computing and Intelligent Interaction*, pages 225–234. Springer, 2011.
749. Antoine Mercier, Ruan Erasmus, Yashesh Savani, Manik Dhingra, Fatih Porikli, and Guillaume Berger. Efficient neural supersampling on a novel gaming dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 296–306, 2023.
750. Timothy Merino, Megan Charity, and Julian Togelius. Interactive latent variable evolution for the generation of Minecraft structures. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8, 2023.
751. Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. The five-dollar model: generating game maps and sprites from sentence embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 107–115, 2023.
752. Paul Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 105–112, 2007.
753. Paul Merrell and Dinesh Manocha. Model synthesis: A general procedural modeling algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):715–728, 2010.
754. Daniel S. Messinger, Tricia D. Cassel, Susan I. Acosta, Zara Ambadar, and Jeffrey F. Cohn. Infant smiling dynamics and perceived positive emotion. *Journal of Nonverbal Behavior*, 32(3):133–155, 2008.

755. Angeliki Metallinou and Shrikanth Narayanan. Annotation and processing of continuous emotional attributes: Challenges and opportunities. In *10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. IEEE, 2013.
756. Zbigniew Michalewicz. Do not kill unfeasible individuals. In *Proceedings of the Fourth Intelligent Information Systems Workshop*, pages 110–123, 1995.
757. Microsoft. Xbox Transparency Report. [Online] https://www.xbox.com/en-GB/legal/xbox-transparency-report, 2023. Accessed 19 October 2024.
758. Midjourney. Midjourney.com, 2022. Accessed March 2024.
759. Radosław Miernik and Jakub Kowalski. Evolving Evaluation Functions for Collectible Card Game AI, 2021.
760. Panagiotis Migkotzidis and Antonios Liapis. Susketch: Surrogate models of gameplay as a design assistant. *IEEE Transactions on Games*, 14(2):273–283, 2021.
761. Risto Miikkulainen, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. Computational intelligence in games. *Computational Intelligence: Principles and Practice*, pages 155–191, 2006.
762. Benedikte Mikkelsen, Christoffer Holmgård, and Julian Togelius. Ethical Considerations for Player Modeling. In *Proceedings of the AAAI WNAIG workshop*, 2017.
763. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint* arXiv:1301.3781, 2013.
764. Stephanie Milani, Arthur Juliani, Ida Momennejad, Raluca Georgescu, Jaroslaw Rzepecki, Alison Shaw, Gavin Costello, Fei Fang, Sam Devlin, and Katja Hofmann. Navigates Like Me: Understanding How People Evaluate Human-Like AI in Video Games. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2023.
765. George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81, 1956.
766. Ian Millington. *AI for Games*. CRC Press, 2019.
767. Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2009.
768. Tom Minka, Ryan Cleven, and Josh Menke. Machine learning for optimal matchmaking. GDC 2020 talk, 2020.
769. Tom Minka, Ryan Cleven, and Yordan Zaykov. Trueskill 2: An improved bayesian skill rating system. *Technical Report*, 2018.
770. Talya Miron-Shatz, Arthur Stone, and Daniel Kahneman. Memories of yesterday's emotions: Does the valence of experience affect the memory-experience gap? *Emotion*, 9(6):885, 2009.
771. Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
772. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.
773. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
774. Mathew Monfort, Matthew Johnson, Aude Oliva, and Katja Hofmann. Asynchronous data aggregation for training end to end visual control networks. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pages 530–537. International Foundation for Autonomous Agents and Multiagent Systems, May 2017.
775. CK Monge and TC O'Brien. Effects of individual toxic behavior on team performance in league of legends. *Media Psychology*, 25(1):82–105, 2022.
776. Daniel Monroe and The Leela Chess Zero Team. Mastering chess with a transformer model. *arXiv preprint* arXiv:2409.12272, 2024.
777. Nick Montfort and Ian Bogost. *Racing the beam: The Atari video computer system*. MIT Press, 2009.

778. Luís Morais, João Dias, and Pedro A. Santos. From caveman to gentleman: a CiF-based social interaction model applied to conan exiles. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–11, 2019.
779. Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *arXiv preprint* arXiv:1701.01724, 2017.
780. Jon D. Morris. Observations: SAM: The self-assessment manikin—An efficient cross-cultural measurement of emotional response. *Journal of Advertising Research*, 35(6):63–68, 1995.
781. Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint* arXiv:1504.04909, 2015.
782. Luvneesh Mugrai, Fernando Silva, Christoffer Holmgård, and Julian Togelius. Automated playtesting of matching tile games. In *2019 IEEE Conference on Games (CoG)*, pages 1–7. IEEE, 2019.
783. Matthias Müller-Brockhausen, Giulio Barbero, and Mike Preuss. Chatter generation through language models. In *IEEE Conference on Games, CoG 2023, Boston, MA, USA, August 21-24, 2023*. IEEE, 2023.
784. Jorge Munoz, Georgios N. Yannakakis, Fiona Mulvey, Dan Witzner Hansen, German Gutierrez, and Araceli Sanchis. Towards gaze-controlled platform games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 47–54. IEEE, 2011.
785. Hector Munoz-Avila, Christian Bauckhage, Michal Bida, Clare Bates Congdon, and Graham Kendall. Learning and Game AI. *Dagstuhl Follow-Ups*, 6, 2013.
786. Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.
787. Chelsea M. Myers, Luis Fernando Laris Pardo, Ana Acosta-Ruiz, Alessandro Canossa, and Jichen Zhu. "Try, Try, Try Again:" Sequence Analysis of User Interaction Data with a Voice User Interface. In *CUI 2021-3rd Conference on Conversational User Interfaces*, 2021.
788. Roger B. Myerson. Game theory: analysis of conflict. 1991. *Cambridge: Mass, Harvard University*.
789. Roger B. Myerson. *Game theory*. Harvard University Press, 2013.
790. Lennart Nacke and Craig A. Lindley. Flow and immersion in first-person shooters: measuring the player's gameplay experience. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 81–88. ACM, 2008.
791. Frederik Nagel, Reinhard Kopiez, Oliver Grewe, and Eckart Altenmüller. Emujoy: Software for continuous measurement of perceived emotions in music. *Behavior Research Methods*, 39(2):283–290, 2007.
792. Sang-Gyu Nam, Chu-Hsuan Hsueh, Pavinee Rerkjirattikal, and Kokolo Ikeda. Using reinforcement learning to generate levels of super mario bros. with quality and diversity. *IEEE Transactions on Games*, 2024.
793. Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint* arXiv:1506.08941, 2015.
794. Alexander Nareyek. Intelligent agents for computer games. In T.A. Marsland and I. Frank, editors, *Computers and Games, Second International Conference, CG 2002*, pages 414–422. Springer, 2002.
795. Alexander Nareyek. Game AI is dead. Long live game AI! *IEEE Intelligent Systems*, (1):9–11, 2007.
796. John F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, number 1, pages 48–49, 1950.
797. Muhammad U Nasir, Steven James, and Julian Togelius. Word2world: Generating stories and worlds through large language models. *arXiv preprint* arXiv:2405.06686, 2024.
798. Muhammad Umair Nasir, Steven James, and Julian Togelius. GameTaversalBenchmark: Evaluating planning abilities of large language models through traversing 2D game maps. *arXiv preprint* arXiv:2410.07765, 2024.

799. Steve Nebel, Sascha Schneider, and Günter Daniel Rey. Mining learning and crafting scientific experiments: a literature review on the use of Minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355, 2016.
800. Graham Nelson. Natural language, semantic analysis, and interactive fiction. *IF Theory Reader*, 141, 2006.
801. Mark J. Nelson. Game Metrics Without Players: Strategies for Understanding Game Artifacts. In *AIIDE Workshop on Artificial Intelligence in the Game Design Process*, 2011.
802. Mark J. Nelson, Simon Colton, Edward J. Powley, Swen E. Gaudl, Peter Ivey, Rob Saunders, Blanca Pérez Ferrer, and Michael Cook. Mixed-initiative approaches to on-device mobile game design. In *Proceedings of the CHI Workshop on Mixed-Initiative Creative Interfaces*, 2017.
803. Mark J. Nelson and Michael Mateas. Search-Based Drama Management in the Interactive Fiction Anchorhead. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 99–104, 2005.
804. Mark J. Nelson and Michael Mateas. An interactive game-design assistant. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 90–98, 2008.
805. Mark J. Nelson and Adam M. Smith. ASP with applications to mazes and levels. In *Procedural Content Generation in Games*, pages 143–157. Springer, 2016.
806. Mark J. Nelson, Julian Togelius, Cameron Browne, and Michael Cook. Rules and mechanics. In *Procedural Content Generation in Games*, pages 99–121. Springer, 2016.
807. John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.
808. Newzoo. Games Market Reports and Forecasts. https://newzoo.com/games-market-reports-forecasts, 2024. Accessed 1 November 2024.
809. Truong-Huy D. Nguyen, Shree Subramanian, Magy Seif El-Nasr, and Alessandro Canossa. Strategy Detection in Wuzzit: A Decision Theoretic Approach. In *International Conference on Learning Science—Workshop on Learning Analytics for Learning and Becoming a Practice*, 2014.
810. Mihalis A. Nicolaou, Hatice Gunes, and Maja Pantic. Continuous prediction of spontaneous affect from multiple cues and modalities in valence-arousal space. *Affective Computing, IEEE Transactions on*, 2(2):92–105, 2011.
811. Jakob Nielsen. Usability 101: Introduction to usability, 2003. Available at http://www.useit.com/alertbox/20030825.html.
812. Jon Lau Nielsen, Benjamin Fedder Jensen, Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. AI for General Strategy Game Playing. *Handbook of Digital Games*, pages 274–304, 2014.
813. Thorbjørn S. Nielsen, Gabriella A. B. Barros, Julian Togelius, and Mark J. Nelson. General Video Game Evaluation Using Relative Algorithm Performance Profiles. In *Applications of Evolutionary Computation*, pages 369–380. Springer, 2015.
814. Thorbjørn S. Nielsen, Gabriella A. B. Barros, Julian Togelius, and Mark J. Nelson. Towards generating arcade game rules with VGDL. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games*, 2015.
815. Anton Nijholt. BCI for games: A 'state of the art' survey. In *Entertainment Computing-ICEC 2008*, pages 225–228. Springer, 2009.
816. Ekaterina Nikonova and Jakub Gemrot. Deep Q-Network for Angry Birds. *arXiv preprint* arXiv:1910.01806, 2019.
817. Nils J. Nilsson. Shakey the robot. Technical report, DTIC Document, 1984.
818. Kai Ninomiya, Mubbasir Kapadia, Alexander Shoulson, Francisco Garcia, and Norman Badler. Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds*, 26(2):119–139, 2015.
819. David Noever, Matt Ciolino, and Josh Kalin. The chess transformer: Mastering play using generative language models. *arXiv preprint* arXiv:2008.04057, 2020.
820. Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, 2000.

821. David G. Novick and Stephen Sutton. What is mixed-initiative interaction. In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, pages 114–116, 1997.
822. Gabriela Ochoa. On genetic algorithms and Lindenmayer systems. In *Parallel Problem Solving from Nature—PPSN V*, pages 335–344. Springer, 1998.
823. Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 87–94. IEEE, 2008.
824. Jacob Olsen. Realtime procedural terrain generation. 2004.
825. Peter Thorup Ølsted, Benjamin Ma, and Sebastian Risi. Interactive evolution of levels for a competitive multiplayer FPS. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1527–1534. IEEE, 2015.
826. Cathy O'Neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.
827. Santiago Ontañón. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.
828. Santiago Ontañón, Nicolas A Barriga, Cleyton R Silva, Rubens O Moraes, and Levi HS Lelis. The first microRTS artificial intelligence competition. *AI Magazine*, 39(1):75–83, 2018.
829. Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311, 2013.
830. Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. RTS AI: Problems and Techniques. In *Encyclopedia of Computer Graphics and Games*. Springer, 2015.
831. OpenAI. GPT-4 technical report. *arXiv preprint* arXiv:2303.08774, 2023.
832. Jeff Orkin. Applying goal-oriented action planning to games. *AI game programming wisdom*, 2:217–228, 2003.
833. Jeff Orkin. Three states and a plan: the AI of F.E.A.R. In *Game Developers Conference*, 2006.
834. Jeff Orkin and Deb Roy. The restaurant game: Learning social behavior and language from thousands of players online. *Journal of Game Development*, 3(1):39–60, 2007.
835. Mauricio Orozco, Juan Silva, Abdulmotaleb El Saddik, and Emil Petriu. The role of haptics in games. In *Haptics Rendering and Applications*. InTech, 2012.
836. Brian O'Neill and Mark Riedl. Emotion-driven narrative generation. In *Emotion in Games: Theory and Praxis*, pages 167–180. Springer, 2016.
837. Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. Imitating human playing styles in Super Mario Bros. *Entertainment Computing*, 4(2):93–104, 2013.
838. Andrew Ortony, Gerald L. Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge University Press, 1990.
839. Martin J. Osborne. *An introduction to game theory*. Oxford University Press, 2004.
840. Alexander Osherenko. *Opinion Mining and Lexical Affect Sensing. Computer-aided analysis of opinions and emotions in texts*. PhD thesis, University of Augsburg, 2010.
841. Abdessamed Ouessai, Mohammed Salem, and Antonio M Mora. Online adversarial planning in μrts: A survey. In *2019 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS)*, volume 1, pages 1–8. IEEE, 2019.
842. Abdessamed Ouessai, Mohammed Salem, and Antonio M Mora. Improving the performance of MCTS-based μRTS agents through move pruning. In *2020 IEEE Conference on Games (CoG)*, pages 708–715. IEEE, 2020.
843. Nicolas Oulianov, Pierre-Louis Biojout, PL Venard, Stan Girard, and Sam Pink. LLM-Colosseum, 2024.
844. Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

845. Seth Ovadia. Ratings and rankings: Reconsidering the structure of values and their measurement. *International Journal of Social Research Methodology*, 7(5):403–414, 2004.
846. François Pachet. A joyful ode to automatic orchestration. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–13, 2016.
847. Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint* arXiv:1810.12282, 2018.
848. Ana Paiva, Joao Dias, Daniel Sobral, Ruth Aylett, Polly Sobreperez, Sarah Woods, Carsten Zoll, and Lynne Hall. Caring for agents and agents that care: Building empathic relations with synthetic agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 194–201. IEEE Computer Society, 2004.
849. Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1–2):1–135, 2008.
850. Giovanni Paolini, Lorenzo Moreschini, Francesco Veneziano, and Alessandro Iraci. Learning to play 7 wonders duel without human supervision. In *IEEE Conference on Games*, 2024.
851. Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.
852. Matt Parker and Bobby D. Bryant. Visual control in Quake II with a cyclic controller. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 151–158. IEEE, 2008.
853. Matt Parker and Bobby D. Bryant. Neurovisual control in the Quake II environment. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):44–54, 2012.
854. David Parlett. What's a ludeme. *Game & Puzzle Design*, 2(2):81–84, 2016.
855. Thomas D. Parsons, Timothy McMahan, and Ian Parberry. Classification of video game player experience using consumer-grade electroencephalography. *IEEE Transactions on Affective Computing*, 13(1):3–15, 2020.
856. Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, et al. Imitating human behaviour with diffusion models. *arXiv preprint* arXiv:2301.10677, 2023.
857. Tim Pearce and Jun Zhu. Counter-strike deathmatch with large-scale behavioural cloning. In *2022 IEEE Conference on Games (CoG)*, pages 104–111. IEEE, 2022.
858. Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience in Super Mario Bros. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 132–139. IEEE, 2009.
859. Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.
860. Barney Pell. *Strategy generation and evaluation for meta-game playing*. PhD thesis, University of Cambridge, 1993.
861. Barney Pell. A strategic metagame player for general chess-like games. *Computational intelligence*, 12(1):177–198, 1996.
862. Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games. *arXiv preprint* arXiv:1703.10069, 2017.
863. Tom Pepels, Mark H. M. Winands, and Marc Lanctot. Real-time Monte Carlo tree search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.
864. Diego Perez, Edward J. Powley, Daniel Whitehouse, Philipp Rohlfshagen, Spyridon Samothrakis, Peter I. Cowling, and Simon M. Lucas. Solving the physical traveling salesman problem: Tree search and macro actions. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):31–45, 2014.

865. Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen. Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 351–358. ACM, 2013.
866. Diego Perez-Liebana. Abstract Forward Models for Modern Games. [Online] https://gaigresearch.github.io/afm/. Accessed 6 October 2024.
867. Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M Lucas. General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214, 2019.
868. Diego Perez-Liebana, Simon M. Lucas, Raluca D. Gaina, Julian Togelius, Ahmed Khalifa, and Jialin Liu. *General Video Game Artificial Intelligence*, volume 3. Morgan & Claypool Publishers, 2019. https://gaigresearch.github.io/gvgaibook/.
869. Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M. Lucas. General video game AI: Competition, challenges and opportunities. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
870. Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.
871. África Periáñez, Alain Saas, Anna Guitart, and Colin Magne. Churn prediction in mobile social games: towards a complete assessment using survival ensembles. In *Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA)*, pages 564–573, 2016.
872. Ana Perišić, Dubravka Šišak Jung, and Marko Pahor. Churn in the mobile gaming field: Establishing churn definitions and measuring classification similarities. *Expert Systems with Applications*, 191:116277, 2022.
873. Ken Perlin. An image synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3):287–296, 1985.
874. Elena Petrovskaya and David Zendle. The battle pass: A mixed-methods investigation into a growing type of video game monetisation. *OSF Preprints, Sep*, 2020.
875. Johannes Pfau, Antonios Liapis, Georg Volkmar, Georgios N. Yannakakis, and Rainer Malaka. Dungeons & replicants: automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)*, pages 431–438. IEEE, 2020.
876. Johannes Pfau, Antonios Liapis, Georgios N. Yannakakis, and Rainer Malaka. Dungeons & replicants II: automated game balancing across multiple difficulty dimensions via deep player behavior modeling. *IEEE Transactions on Games*, 15(2):217–227, 2022.
877. Rosalind W. Picard. *Affective Computing*. MIT Press, Cambridge, MA, 1997.
878. R.W. Picard. Affective computing. Technical report, MIT Media Laboratory Perceptual Computing, 1995.
879. Martin Pichlmair and Mads Johansen. Designing game feel: A survey. *IEEE Transactions on Games*, 14(2):138–152, 2021.
880. Grant Pickett, Foaad Khosmood, and Allan Fowler. Automated generation of conversational non player characters. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
881. Éric Piette, Dennis J. N. J. Soemers, Matthew Stephenson, Chiara F. Sironi, Mark H. M. Winands, and Cameron Browne. Ludii–the ludemic general game system. *arXiv e-prints*, pages arXiv–1905, 2019.
882. Éric Piette, Dennis JNJ Soemers, Matthew Stephenson, and Cameron Browne. The 2022 Ludii AI competition. *ICGA Journal*, (Preprint):1–12, 2023.
883. Eric Piette, Matthew Stephenson, Dennis JNJ Soemers, and Cameron Browne. General board game concepts. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE, 2021.
884. Kosmas Pinitas, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. RankNEAT: outperforming stochastic gradient search in preference learning tasks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1084–1092, 2022.

885. Kosmas Pinitas, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Supervised contrastive learning for affect modelling. In *Proceedings of the 2022 International Conference on Multimodal Interaction*, pages 531–539, 2022.
886. Kosmas Pinitas, Konstantinos Makantasis, and Georgios N. Yannakakis. Across-game engagement modelling via few-shot learning. *ECCV Workshops*, 2024.
887. Kosmas Pinitas, Nemanja Rasajski, Matthew Barthet, Maria Kaselimi, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Varying the Context to Advance Affect Modelling: A Study on Game Engagement Prediction. In *International Conference on Affective Computing and Intelligent Interaction*. IEEE, 2024.
888. Kosmas Pinitas, David Renaudie, Mike Thomsen, Matthew Barthet, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Predicting Player Engagement in Tom Clancy's The Division 2: A Multimodal Approach via Pixels and Gamepad Actions. In *Proceedings of the 25th International Conference on Multimodal Interaction*, pages 488–497, 2023.
889. Michele Pirovano. The use of Fuzzy Logic for Artificial Intelligence in Games. Technical report, University of Milano, Milano, 2012.
890. Jacques Pitrat. Realization of a general game-playing program. In *IFIP congress (2)*, pages 1570–1574, 1968.
891. David Plans and Davide Morelli. Experience-driven procedural music generation for games. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):192–198, 2012.
892. Isabella Poggi, Catherine Pelachaud, Fiorella de Rosis, Valeria Carofiglio, and Berardina De Carolis. GRETA. A believable embodied conversational agent. In *Multimodal intelligent information presentation*, pages 3–25. Springer, 2005.
893. Mihai Polceanu. Mirrorbot: Using human-inspired mirroring behavior to pass a Turing test. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
894. Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A field guide to genetic programming*. 2008. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (With contributions by J. R. Koza).
895. Jordan B. Pollack and Alan D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine learning*, 32(3):225–240, 1998.
896. Jordan B. Pollack, Alan D. Blair, and Mark Land. Coevolution of a backgammon player. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. Cambridge, MA: The MIT Press, 1997.
897. Alexandru Popescu, Joost Broekens, and Maarten Van Someren. Gamygdala: An emotion engine for games. *IEEE Transactions on Affective Computing*, 5(1):32–44, 2013.
898. David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(04):515–526, 1978.
899. Mike Preuss, Daniel Kozakowski, Johan Hagelbäck, and Heike Trautmann. Reactive strategy choice in StarCraft by means of Fuzzy Control. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
900. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer, 1990.
901. Andrew K. Przybylski, C. Scott Rigby, and Richard M. Ryan. A motivational model of video game engagement. *Review of general psychology*, 14(2):154–166, 2010.
902. Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.
903. Jan Quadflieg, Mike Preuss, and Günter Rudolph. Driving as a human: a track learning based adaptable architecture for a car racing controller. *Genetic Programming and Evolvable Machines*, 15(4):433–476, 2014.
904. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
905. J. Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.
906. Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. *arXiv preprint* arXiv:2404.10179, 2024.

907. Steve Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., 2002.
908. Steve Rabin. *AI Game Programming Wisdom 2*. Charles River Media, Inc., 2003.
909. Steve Rabin. *AI Game Programming Wisdom 3*. Charles River Media, Inc., 2006.
910. Steve Rabin. *AI Game Programming Wisdom 4*. Nelson Education, 2014.
911. Steve Rabin. *Game AI Pro 360: Guide to Architecture*. CRC Press, 2019.
912. Steve Rabin. *Game AI Pro 360: Guide to Character Behavior*. CRC Press, 2019.
913. Steve Rabin. *Game AI Pro 360: Guide to Movement and Pathfinding*. CRC Press, 2019.
914. Steve Rabin. *Game AI Pro 360: Guide to Tactics and Strategy*. CRC Press, 2019.
915. Steve Rabin and Nathan Sturtevant. Pathfinding Architecture Optimizations. In *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2013.
916. Steve Rabin and Nathan Sturtevant. Combining Bounding Boxes and JPS to Prune Grid Pathfinding. In *AAAI Conference on Artificial Intelligence*, 2016.
917. Steven Rabin. *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2013.
918. Steven Rabin. *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. CRC Press, 2015.
919. Steven Rabin. *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. CRC Press, 2017.
920. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
921. William L. Raffe, Fabio Zambetta, and Xiaodong Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012.
922. Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint* arXiv:2204.06125, 1(2):3, 2022.
923. Judith Ramey, Ted Boren, Elisabeth Cuddihy, Joe Dumas, Zhiwei Guan, Maaike J. van den Haak, and Menno D. T. De Jong. Does think aloud work? How do we know? In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 45–48. ACM, 2006.
924. Pramila Rani, Nilanjan Sarkar, and Changchun Liu. Maintaining optimal challenge in computer games through real-time physiological feedback. In *Proceedings of the 11th International Conference on Human Computer Interaction*, pages 184–192, 2005.
925. Nemanja Rašajski, Chintan Trivedi, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. BehAVE: Behaviour Alignment of Video Game Encodings. In *Proceedings of the ECCV Workshop on Computer Vision For Videogames*, 2024.
926. Jakob Rasmussen. *Are Behavior Trees a Thing of the Past?* Gamasutra, 2016.
927. Niklas Ravaja, Timo Saari, Mikko Salminen, Jari Laarni, and Kari Kallinen. Phasic emotional reactions to video game events: A psychophysiological investigation. *Media Psychology*, 8(4):343–367, 2006.
928. Genaro Rebolledo-Mendez, Ian Dunwell, Erika Martínez-Mirón, Maria Dolores Vargas-Cerdán, Sara De Freitas, Fotis Liarokapis, and Alma R. García-Gaona. Assessing Neurosky's usability to detect attention levels in an assessment exercise. *Human-Computer Interaction. New Trends*, pages 149–158, 2009.
929. J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
930. Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint* arXiv:2205.06175, 2022.
931. Simão Reis, Rita Novais, Luís Paulo Reis, and Nuno Lau. An adversarial approach for automated pokémon team building and meta-game balance. *IEEE Transactions on Games*, 2023.
932. Steven Reiss. Multifaceted nature of intrinsic motivation: The theory of 16 basic desires. *Review of general psychology*, 8(3):179–193, 2004.
933. Noah Renella and Markus Eger. Towards automated video game commentary using generative AI. In *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2023.

934. Jochen Renz, Xiaoyu Ge, Stephen Gould, and Peng Zhang. The Angry Birds AI Competition. *AI Magazine*, 36(2):85–87, 2015.
935. Carson Reynolds and Rosalind Picard. Affective sensors, privacy, and ethical contracts. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1103–1106, 2004.
936. Antonio Ricciardi and Patrick Thill. Adaptive AI for Fighting Games. Technical report, Stanford University, 2008.
937. Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1700–1715, 2022.
938. Mark O. Riedl. Computational narrative intelligence: A human-centered goal for artificial intelligence. *arXiv preprint* arXiv:1602.06484, 2016.
939. Mark O. Riedl and Vadim Bulitko. Interactive narrative: An intelligent systems approach. *AI Magazine*, 34(1):67, 2012.
940. Mark O. Riedl and Andrew Stern. Believable agents and intelligent story adaptation for interactive storytelling. *Technologies for Interactive Digital Storytelling and Entertainment*, pages 1–12, 2006.
941. Mark O. Riedl and Alexander Zook. AI for game production. In *IEEE Conference on Computational Intelligence in Games (CIG)*. IEEE, 2013.
942. Scott Rigby and Richard M Ryan. *Glued to games: How video games draw us in and hold us spellbound*. Bloomsbury Publishing USA, 2011.
943. Fabien Ringeval, Andreas Sonderegger, Juergen Sauer, and Denis Lalanne. Introducing the RECOLA multimodal corpus of remote collaborative and affective interactions. In *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. IEEE, 2013.
944. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, and Kenneth O. Stanley. Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game. In *Proceedings of AIIDE*, 2012.
945. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, and Kenneth O. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):244–255, 2016.
946. Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.
947. Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.
948. David L. Roberts, Harikrishna Narayanan, and Charles L. Isbell. Learning to influence emotional responses for interactive storytelling. In *Proceedings of the 2009 AAAI Symposium on Intelligent Narrative Technologies II*, 2009.
949. Paul Roberts. *Artificial Intelligence in Games*. CRC Press, 2022.
950. Paul Roberts. *Game AI Uncovered: Volume One*. CRC Press, 2024.
951. Paul Roberts. *Game AI Uncovered: Volume Two*. CRC Press, 2024.
952. Glen Robertson and Ian D. Watson. A review of real-time strategy game AI. *AI Magazine*, 35(4):75–104, 2014.
953. Glen Robertson and Ian D. Watson. An Improved Dataset and Extraction Process for StarCraft AI. In *FLAIRS Conference*, 2014.
954. Michael D. Robinson and Gerald L. Clore. Belief and feeling: evidence for an accessibility model of emotional self-report. *Psychological Bulletin*, 128(6):934, 2002.
955. Raquel Robinson, Katelyn Wiley, Amir Rezaeivahdati, Madison Klarkowski, and Regan L. Mandryk. "Let's Get Physiological, Physiological!" A Systematic Review of Affective Gaming. In *Proceedings of the annual symposium on computer-human interaction in play*, pages 132–147, 2020.
956. Jennifer Robison, Scott McQuiggan, and James Lester. Evaluating the consequences of affective feedback in intelligent tutoring systems. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2009.

957. Katja Rogers, Vincent Le Claire, Julian Frommel, Regan Mandryk, and Lennart E. Nacke. Using evolutionary algorithms to target complexity levels in game economies. *IEEE Transactions on Games*, 15(1):56–66, 2023.
958. Philipp Rohlfshagen, Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas. Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
959. Philipp Rohlfshagen and Simon M. Lucas. Ms Pac-Man versus Ghost team CEC 2011 competition. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 70–77. IEEE, 2011.
960. Edmund T. Rolls. The orbitofrontal cortex and reward. *Cerebral Cortex*, 10(3):284–294, 2000.
961. Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
962. Shaghayegh Roohi, Christian Guckelsberger, Asko Relas, Henri Heiskanen, Jari Takatalo, and Perttu Hämäläinen. Predicting game difficulty and engagement using AI players. *Proceedings of the ACM on Human-Computer Interaction*, 5(CHI PLAY):1–17, 2021.
963. Shaghayegh Roohi, Asko Relas, Jari Takatalo, Henri Heiskanen, and Perttu Hämäläinen. Predicting game difficulty and churn without players. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 585–593, 2020.
964. Shaghayegh Roohi, Jari Takatalo, Christian Guckelsberger, and Perttu Hämäläinen. Review of intrinsic motivation in simulation-based game testing. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–13, 2018.
965. Shaghayegh Roohi, Jari Takatalo, J. Matias Kivikangas, and Perttu Hämäläinen. Neural network based facial expression analysis of gameevents: a cautionary tale. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 429–437, 2018.
966. Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.
967. Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
968. Jonathan Rowe, Bradford Mott, Scott McQuiggan, Jennifer Robison, Sunyoung Lee, and James Lester. Crystal Island: A narrative-centered learning environment for eighth grade microbiology. In *Workshop on Intelligent Educational Games at the 14th International Conference on Artificial Intelligence in Education, Brighton, UK*, pages 11–20, 2009.
969. Jonathan P. Rowe, Lucy R. Shores, Bradford W. Mott, and James C. Lester. Integrating learning, problem solving, and engagement in narrative-centered learning environments. *International Journal of Artificial Intelligence in Education*, 21(1-2):115–133, 2011.
970. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
971. Gavin A. Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
972. Thomas Philip Runarsson and Simon M. Lucas. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go. *IEEE Transactions on Evolutionary Computation*, 9(6):628–640, 2005.
973. Julian Runge, Peng Gao, Florent Garcin, and Boi Faltings. Churn prediction for high-value players in casual social games. In *2014 IEEE conference on Computational Intelligence and Games*. IEEE, 2014.
974. James A. Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161, 1980.

975. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.
976. Richard M. Ryan, C. Scott Rigby, and Andrew Przybylski. The motivational pull of video games: A self-determination theory approach. *Motivation and emotion*, 30:344–360, 2006.
977. Jennifer L. Sabourin and James C. Lester. Affect and engagement in Game-Based Learning environments. *IEEE Transactions on Affective Computing*, 5(1):45–56, 2014.
978. Owen Sacco, Antonios Liapis, and Georgios N. Yannakakis. A holistic approach for semantic-based game generation. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
979. Frantisek Sailer, Michael Buro, and Marc Lanctot. Adversarial planning through strategy simulation. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 80–87. IEEE, 2007.
980. Vardan Saini and Matthew Guzdial. A demonstration of mechanic maker: An AI for mechanics co-creation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 325–327, 2020.
981. Katie Salen and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT Press, 2004.
982. Christoph Salge, Christian Lipski, Tobias Mahlmann, and Brigitte Mathiak. Using genetically optimized artificial intelligence to improve gameplaying fun for strategical games. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 7–14, New York, NY, USA, 2008. ACM.
983. Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint* arXiv:1703.03864, 2017.
984. Gonçalo Filipe Duarte Salvador, Patrícia J Bota, Vinoba Vinayagamoorthy, Hugo Plácido da Silva, and Ana Fred. Smartphone-based content annotation for ground truth collection in affective computing. In *Proceedings of the 2021 ACM International Conference on Interactive Media Experiences*, pages 199–204, 2021.
985. Spyridon Samothrakis, Simon M. Lucas, Thomas Philip Runarsson, and David Robles. Coevolving game-playing agents: Measuring performance and intransitivities. *Evolutionary Computation, IEEE Transactions on*, 17(2):213–226, 2013.
986. Spyridon Samothrakis, David Robles, and Simon M. Lucas. Fast approximate max-n Monte Carlo tree search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154, 2011.
987. Arthur L. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.
988. André Santos, Pedro A. Santos, and Francisco S. Melo. Monte carlo tree search experiments in hearthstone. In *2017 IEEE conference on computational intelligence and games (CIG)*, pages 272–279. IEEE, 2017.
989. Naomi Saphra, Eve Fleisig, Kyunghyun Cho, and Adam Lopez. First Tragedy, then Parse: History Repeats Itself in the New Era of Large Language Models. *arXiv preprint* arXiv:2311.05020, 2023.
990. Anurag Sarkar and Seth Cooper. tile2tile: Learning game filters for platformer style transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 53–60, 2022.
991. Frederik Schadd, Sander Bakkes, and Pieter Spronck. Opponent modeling in real-time strategy games. In *GAMEON*, pages 61–70, 2007.
992. Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.
993. Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.
994. Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook: the world man-machine Checkers champion. *AI Magazine*, 17(1):21, 1996.
995. Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 21(2):97–126, 2006.

996. Tom Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
997. Tom Schaul. An extensible description language for video games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):325–331, 2014.
998. Tom Schaul, Julian Togelius, and Jürgen Schmidhuber. Measuring intelligence through games. *arXiv preprint* arXiv:1109.1314, 2011.
999. Jesse Schell. *The Art of Game Design: A book of lenses*. CRC Press, 2014.
1000. Klaus R. Scherer. What are emotions? and how can they be measured? *Social Science Information*, 44(4):695–729, 2005.
1001. Klaus R. Scherer, Angela Schorr, and Tom Johnstone. *Appraisal processes in emotion: Theory, methods, research*. Oxford University Press, 2001.
1002. Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.
1003. Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.
1004. Jacob Schrum, Igor V. Karpov, and Risto Miikkulainen. UT^2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 329–336. IEEE, 2011.
1005. John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.
1006. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint* arXiv:1707.06347, 2017.
1007. Brian Schwab. *AI game engine programming*. Nelson Education, 2009.
1008. Brian Schwab, Dave Mark, Kevin Dill, Mike Lewis, and Richard Evans. GDC: Turing tantrums: AI developers rant, 2011.
1009. Marco Scirea, Yun-Gyung Cheong, Mark J. Nelson, and Byung-Chull Bae. Evaluating musical foreshadowing of videogame narrative experiences. In *Proceedings of the 9th Audio Mostly: A Conference on Interaction With Sound*. ACM, 2014.
1010. Marco Scirea, Peter Eklund, Julian Togelius, and Sebastian Risi. Evolving in-game mood-expressive music with metacompose. In *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion*. 2018.
1011. Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34, 2021.
1012. Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.
1013. Ben Seymour and Samuel M. McClure. Anchors, scales and the relative coding of value in the brain. *Current Opinion in Neurobiology*, 18(2):173–178, 2008.
1014. Konstantinos Sfikas and Antonios Liapis. Playing against the board: Rolling horizon evolutionary algorithms against pandemic. *IEEE Transactions on Games*, 14(3):339–349, 2021.
1015. Konstantinos Sfikas, Antonios Liapis, and Georgios N. Yannakakis. Monte Carlo Elites: Quality-diversity selection as a multi-armed bandit problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 180–188, 2021.
1016. Mohammad Shaker, Mhd Hasan Sarhan, Ola Al Naameh, Noor Shaker, and Julian Togelius. Automatic generation and analysis of physics-based puzzle games. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
1017. Noor Shaker, Stylianos Asteriadis, Georgios N. Yannakakis, and Kostas Karpouzis. A game-based corpus for analysing the interplay between game context and player experience. In *Affective Computing and Intelligent Interaction*, pages 547–556. Springer, 2011.

1018. Noor Shaker, Stylianos Asteriadis, Georgios N. Yannakakis, and Kostas Karpouzis. Fusing visual and behavioral cues for modeling user experience in games. *Cybernetics, IEEE Transactions on*, 43(6):1519–1531, 2013.
1019. Noor Shaker, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neil. Evolving levels for Super Mario Bros using grammatical evolution. In *IEEE Conference on Computational Intelligence and Games*, pages 304–311. IEEE, 2012.
1020. Noor Shaker, Mohammad Shaker, and Mohamed Abou-Zleikha. Towards generic models of player experience. In *Proceedings, the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2015.
1021. Noor Shaker, Mohammad Shaker, and Julian Togelius. Evolving Playable Content for Cut the Rope through a Simulation-Based Approach. In *AIIDE*, 2013.
1022. Noor Shaker, Mohammad Shaker, and Julian Togelius. Ropossum: An Authoring Tool for Designing, Optimizing and Solving Cut the Rope Levels. In *AIIDE*, 2013.
1023. Noor Shaker, Julian Togelius, and Mark J. Nelson, editors. *Procedural Content Generation in Games*. Springer, 2016.
1024. Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, October 2010.
1025. Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. The experience-driven perspective. In *Procedural Content Generation in Games*, pages 181–194. Springer, 2016.
1026. Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Likith Poovanna, Vinay S. Ethiraj, Stefan J. Johansson, Robert G. Reynolds, Leonard K. Heether, Tom Schumann, and Marcus Gallagher. The Turing test track of the 2012 Mario AI championship: entries and evaluation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
1027. Noor Shaker, Georgios N. Yannakakis, and Julian Togelius. Crowdsourcing the aesthetics of platform games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(3):276–290, 2013.
1028. Amirhosein Shantia, Eric Begue, and Marco Wiering. Connectionist reinforcement learning for intelligent unit micro management in StarCraft. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1794–1801. IEEE, 2011.
1029. Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint* arXiv:1912.10944, 2019.
1030. Manu Sharma, Manish Mehta, Santiago Ontañón, and Ashwin Ram. Player modeling evaluation for interactive fiction. In *Proceedings of the AIIDE 2007 Workshop on Optimizing Player Satisfaction*, pages 19–24, 2007.
1031. Nandita Sharma and Tom Gedeon. Objective measures, sensors and computational techniques for stress recognition and classification: A survey. *Computer methods and programs in biomedicine*, 108(3):1287–1301, 2012.
1032. Xinke Shen, Xianggen Liu, Xin Hu, Dan Zhang, and Sen Song. Contrastive learning of subject-invariant EEG representations for cross-subject emotion recognition. *IEEE Transactions on Affective Computing*, 2022.
1033. Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. The woman worked as a babysitter: On biases in language generation. *arXiv preprint* arXiv:1909.01326, 2019.
1034. Peter Shizgal and Andreas Arvanitogiannis. Gambling on dopamine. *Science*, 299(5614):1856–1858, 2003.
1035. Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
1036. Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
1037. Alexander Shoulson, Francisco M. Garcia, Matthew Jones, Robert Mead, and Norman I. Badler. Parameterizing behavior trees. In *International Conference on Motion in Games*, pages 144–155. Springer, 2011.
1038. Tianye Shu, Jialin Liu, and Georgios N. Yannakakis. Experience-driven PCG via reinforcement learning: A Super Mario Bros study. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.

1039. Nikolaos Sidorakis, George Alex Koulieris, and Katerina Mania. Binocular eye-tracking for the control of a 3D immersive multimedia user interface. In *Everyday Virtual Reality (WEVR), 2015 IEEE 1st Workshop on*, pages 15–18. IEEE, 2015.
1040. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
1041. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.
1042. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *nature*, 550(7676):354–359, 2017.
1043. David Simoes, Simao Reis, Nuno Lau, and Luis Paulo Reis. Competitive deep reinforcement learning over a pokémon battling simulator. In *2020 IEEE international conference on autonomous robot systems and competitions (ICARSC)*, pages 40–45. IEEE, 2020.
1044. Herbert A. Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.
1045. Herbert A Simon. Bounded rationality. *Utility and probability*, pages 15–18, 1990.
1046. Tom Simonite. It began as an AI-fueled dungeon game. it got much darker, 2021.
1047. Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds*, 22(2-3):151–158, 2011.
1048. Matthew Siper, Sam Earle, Zehua Jiang, Ahmed Khalifa, and Julian Togelius. Controllable path of destruction. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.
1049. Matthew Siper, Ahmed Khalifa, and Julian Togelius. Path of destruction: Learning an iterative level generator using a small dataset. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 337–343. IEEE, 2022.
1050. Moshe Sipper. *Evolved to Win*. Lulu.com, 2011.
1051. Burrhus Frederic Skinner. *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.
1052. Ruben M. Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. Interactive creation of virtual worlds using procedural sketching. In *Proceedings of Eurographics*, 2010.
1053. Adam M. Smith, Erik Andersen, Michael Mateas, and Zoran Popović. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 156–163. ACM, 2012.
1054. Adam M. Smith, Chris Lewis, Kenneth Hullett, Gillian Smith, and Anne Sullivan. An inclusive taxonomy of player modeling. Technical Report UCSC-SOE-11-13, University of California, Santa Cruz, 2011.
1055. Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 273–280. IEEE, 2010.
1056. Adam M. Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187–200, 2011.
1057. Adam M. Smith, Mark J. Nelson, and Michael Mateas. Ludocore: A logical game engine for modeling videogames. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 91–98. IEEE, 2010.
1058. Gillian Smith and Jim Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
1059. Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

1060. Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):201–215, 2011.
1061. Ian Sneddon, Gary McKeown, Margaret McRorie, and Tijana Vukicevic. Cross-cultural patterns in dynamic ratings of positive and negative natural emotional behaviour. *PloS ONE*, 6(2), 2011.
1062. Sam Snodgrass and Santiago Ontañón. A Hierarchical MdMC Approach to 2D Video Game Map Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
1063. Dennis Soemers. Tactical planning using MCTS in the game of StarCraft, 2014. Bachelor Thesis, Department of Knowledge Engineering, Maastricht University.
1064. Dennis JNJ Soemers, Éric Piette, Matthew Stephenson, and Cameron Browne. Spatial state-action features for general games. *Artificial Intelligence*, 321:103937, 2023.
1065. Dennis JNJ Soemers, Éric Piette, Matthew Stephenson, and Cameron Browne. The Ludii game description language is universal. In *2024 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2024.
1066. Dennis JNJ Soemers, Spyridon Samothrakis, Éric Piette, and Matthew Stephenson. Extracting tactics learned from self-play in general games. *Information Sciences*, 624:277–298, 2023.
1067. Andreas Sonderegger, Andreas Uebelbacher, Manuela Pugliese, and Juergen Sauer. The influence of aesthetics in usability testing: the case of dual-domain products. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 21–30, 2014.
1068. Yan Song, He Jiang, Haifeng Zhang, Zheng Tian, Weinan Zhang, and Jun Wang. Boosting studies of multi-agent reinforcement learning on google research football environment: the past, present, and future. *arXiv preprint* arXiv:2309.12951, 2023.
1069. Bhuman Soni and Philip Hingston. Bots trained to play like a human are more fun. In *IEEE International Joint Conference on Neural Networks (IJCNN); IEEE World Congress on Computational Intelligence*, pages 363–369. IEEE, 2008.
1070. Patrikk D. Sørensen, Jeppeh M. Olsen, and Sebastian Risi. Interactive Super Mario Bros Evolution. In *Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, pages 41–42. ACM, 2016.
1071. Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. *Applications of Evolutionary Computation*, pages 131–140, 2010.
1072. Terence Soule, Samantha Heck, Thomas E. Haynes, Nicholas Wood, and Barrie D. Robison. Darwin's demons: Does evolution improve the game? In *Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part I 20*, pages 435–451. Springer, 2017.
1073. Laura Spinney. Artificial intelligence beats eight world champions at bridge. *The Guardian*, March 2022.
1074. Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.
1075. Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Difficulty scaling of game AI. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.
1076. Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.
1077. Stability.ai. Stable diffusion public release. https://stability.ai/news/stable-diffusion-public-release, 2023. Accessed 6 October 2024.
1078. Kenneth O. Stanley. Compositional Pattern Producing Networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.
1079. Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the NERO video game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653–668, 2005.

1080. Kenneth O. Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.
1081. Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
1082. Kenneth O. Stanley and Risto Miikkulainen. Evolving a roving eye for Go. In *Genetic and Evolutionary Computation Conference*, pages 1226–1238. Springer, 2004.
1083. Statista. Video Games – Worldwide. https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide, 2024. Accessed 1 November 2024.
1084. Matthew Stephenson and Jochen Renz. Deceptive Angry Birds: Towards smarter game-playing agents. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 2018.
1085. Matthew Stephenson, Dennis JNJ Soemers, Éric Piette, and Cameron Browne. General game heuristic prediction based on ludeme descriptions. In *2021 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2021.
1086. Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.
1087. Stanley Smith Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.
1088. Neil Stewart, Gordon D. A. Brown, and Nick Chater. Absolute identification by relative judgment. *Psychological Review*, 112(4):881, 2005.
1089. Andreas Stiegler, Keshav Dahal, Johannes Maucher, and Daniel Livingstone. Symbolic Reasoning for Hearthstone. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
1090. Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.
1091. Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *arXiv preprint cs/0512049*, 2005.
1092. Nathan Sturtevant. Memory-Efficient Pathfinding Abstractions. In *AI Programming Wisdom 4*. Charles River Media, 2008.
1093. Nathan Sturtevant and Matheus Ota. Exhaustive and semi-exhaustive procedural content generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14, pages 109–115, 2018.
1094. Nathan Sturtevant and Steve Rabin. Canonical orderings on grids. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 683–689, 2016.
1095. Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.
1096. Nathan R. Sturtevant and Richard E. Korf. On pruning techniques for multi-player games. *Proceedings of The National Conference on Artificial Intelligence (AAAI)*, pages 201–208, 2000.
1097. Nathan R. Sturtevant, Jason Traish, James Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin. The Grid-Based Path Planning Competition: 2014 Entries and Results. In *Eighth Annual Symposium on Combinatorial Search*, pages 241–251, 2015.
1098. Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint* arXiv:1712.06567, 2017.
1099. Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. MarioGPT: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

1100. Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. Prompt-guided level generation. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 179–182, 2023.
1101. Sugiyanto Sugiyanto, I. Ketut Eddy Purnama, Eko Mulyanto Yuniarno, Hanny Haryanto, Nur Rokhman, and Mauridhi Hery Purnomo. Acquiface Interface as a Device for Acquisition of User's Facial Expressions in Game Expression Hunter. In *2020 IEEE Conference on Games (CoG)*, pages 291–296. IEEE, 2020.
1102. Adam Summerville and Michael Mateas. Super Mario as a String: Platformer Level Generation Via LSTMs. *arXiv preprint* arXiv:1603.00930, 2016.
1103. Adam James Summerville, Shweta Philip, and Michael Mateas. MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
1104. Adam James Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural Content Generation via Machine Learning (PCGML). *arXiv preprint* arXiv:1702.00539, 2017.
1105. Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón Villar. The VGLC: The Video Game Level Corpus. *arXiv preprint* arXiv:1606.07487, 2016.
1106. Yuqian Sun, Zhouyi Li, Ke Fang, Chang Hee Lee, and Ali Asadipour. Language as reality: a co-creative storytelling game experience in 1001 nights using generative AI. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 425–434, 2023.
1107. Petra Sundström. *Exploring the affective loop*. PhD thesis, Stockholm University, 2005.
1108. Ben Sunshine-Hill, Michael Robbins, and Chris Jurney. Off the Beaten Path: Non-Traditional Uses of AI. In *Game Developers Conference, AI Summit*, 2012.
1109. Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.
1110. Reid Swanson and Andrew S. Gordon. Say anything: Using textual case-based reasoning to enable open-domain interactive storytelling. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(3):16, 2012.
1111. William R. Swartout, Jonathan Gratch, Randall W. Hill Jr, Eduard Hovy, Stacy Marsella, Jeff Rickel, and David Traum. Toward virtual humans. *AI Magazine*, 27(2):96, 2006.
1112. Penelope Sweetser, Daniel M. Johnson, and Peta Wyeth. Revisiting the GameFlow model with detailed heuristics. *Journal: Creative Technologies*, 2012(3), 2012.
1113. Penelope Sweetser and Janet Wiles. Scripting versus emergence: issues for game developers and players in game environment design. *International Journal of Intelligent Games and Simulations*, 4(1):1–9, 2005.
1114. Penelope Sweetser and Janet Wiles. Using cellular automata to facilitate emergence in game environments. In *Proceedings of the 4th International Conference on Entertainment Computing (ICEC'05)*, 2005.
1115. Penelope Sweetser and Peta Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.
1116. Maciej Świechowski and Jacek Mańdziuk. Self-adaptation of playing strategies in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):367–381, 2014.
1117. Steve Swink. *Game feel: a game designer's guide to virtual sensation*. CRC press, 2008.
1118. Gabriel Synnaeve and Pierre Bessière. Multiscale Bayesian Modeling for RTS Games: An Application to StarCraft AI. *IEEE Transactions on Computational intelligence and AI in Games*, 8(4):338–350, 2016.
1119. Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint* arXiv:1611.00625, 2016.
1120. Nicolas Szilas. IDtension: a narrative engine for Interactive Drama. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, pages 1–11, 2003.

1121. István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in Settlers of Catan. In *Advances in Computer Games: 12th International Conference, ACG 2009, Pamplona Spain, May 11-13, 2009. Revised Papers 12*, pages 21–32. Springer, 2010.
1122. Niels A. Taatgen, Marcia van Oploo, Jos Braaksma, and Jelle Niemantsverdriet. How to construct a believable opponent using cognitive modeling in the game of set. In *Proceedings of the Fifth International Conference on Cognitive Modeling*, pages 201–206, 2003.
1123. Mohammad Reza Taesiri, Finlay Macklon, and Cor-Paul Bezemer. Clip meets gamephysics: Towards bug identification in gameplay videos using zero-shot transfer learning. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 270–281, 2022.
1124. Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, pages 113–120. ACM, 2007.
1125. Ala N. Tak and Jonathan Gratch. Is GPT a computational model of emotion. In *11th International Conference on Affective Computing and Intelligent Interaction, Boston, MA*, 2023.
1126. Bulent Tastan and Gita Reese Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.
1127. Michelle Taub, Nicholas V. Mudrick, Roger Azevedo, Garrett C. Millar, Jonathan Rowe, and James Lester. Using multi-channel data with multi-level modeling to assess in-game performance during gameplay with Crystal Island. *Computers in Human Behavior*, 76:641–655, 2017.
1128. Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint* arXiv:2301.07608, 2023.
1129. Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint* arXiv:2312.11805, 2023.
1130. Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint* arXiv:2107.12808, 2021.
1131. Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap van Den Herik, and Jan Broersen. Play style: Showing your age. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
1132. Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap van den Herik, and Jan Broersen. Psyops: Personality assessment through gaming behavior. In *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft, The Netherlands, November 7-8, 2013*, 2013.
1133. Gerald Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3-4):257–277, 1992.
1134. Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.
1135. Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan-Ta Chen. Detection of MMORPG bots based on behavior analysis. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 91–94. ACM, 2008.
1136. TheGammonPress. A short history of modern backgammon. https://thegammonpress.com/a-short-history-of-modern-backgammon/, 2016. Accessed 6 October 2024.
1137. Alexios Theodoridis and Georgios Chalkiadakis. Monte Carlo Tree Search for the game of Diplomacy. In *11th Hellenic Conference on Artificial Intelligence*, pages 16–25, 2020.
1138. Michael Thielscher. A General Game Description Language for Incomplete Information Games. In *AAAI*, pages 994–999, 2010.

1139. Tommy Thompson. How Townscaper Works: A Story Four Games in the Making. https://www.gamedeveloper.com/game-platforms/how-townscaper-works-a-story-four-games-in-the-making, 2022. Accessed 14 november 2024.
1140. William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.
1141. David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen. Interactive Storytelling: A Player Modelling Approach. In *AIIDE*, pages 43–48, 2007.
1142. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like opponent behavior for interactive computer games. *Pattern Recognition, Lecture Notes in Computer Science 2781*, pages 148–155, 2003.
1143. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Imitation learning at all levels of game AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education*, 2004.
1144. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like Movement Behavior for Computer Games. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB-04)*, pages 315–323, Santa Monica, CA, July 2004. The MIT Press.
1145. Tim J. W. Tijs, Dirk Brokken, and Wijnand A. Ijsselsteijn. Dynamic game balancing by recognizing affect. In *Proceedings of International Conference on Fun and Games*, pages 88–93. Springer, 2008.
1146. Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023.
1147. Graham Todd, Tim Merino, Sam Earle, and Julian Togelius. Missed connections: Lateral thinking puzzles for large language models. In *IEEE Conference on Games*, 2024.
1148. Graham Todd, Alexander Padula, Matthew Stephenson, Éric Piette, Dennis JNJ Soemers, and Julian Togelius. Gavel: Generating games via evolution and language models. In *Neural Information Processing Systems*, 2024.
1149. Julian Togelius. Evolution of a subsumption architecture neurocontroller. *Journal of Intelligent & Fuzzy Systems*, 15(1):15–20, 2004.
1150. Julian Togelius. A procedural critique of deontological reasoning. In *Proceedings of DiGRA*, 2011.
1151. Julian Togelius. AI researchers, Video Games are your friends! In *Computational Intelligence*, pages 3–18. Springer, 2015.
1152. Julian Togelius. How to run a successful game-based AI competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(1):95–100, 2016.
1153. Julian Togelius. *Playing smart: On games, intelligence, and artificial intelligence*. MIT Press, 2019.
1154. Julian Togelius. *Artificial General Intelligence*. MIT Press, 2024.
1155. Julian Togelius, Alex J. Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O. Stanley. Procedural Content Generation in Games: Goals, Challenges and Actionable Steps. *Dagstuhl Follow-Ups*, 6, 2013.
1156. Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Making racing fun through player modeling and track evolution. In *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.
1157. Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.
1158. Julian Togelius and Marie Gustafsson Friberger. Bar chart ball, a data game. In *Foundations of Digital Games (FDG), Chania, Crete, Greece (2013)*, pages 451–452. Society for the Advancement of the Science of Digital Games (SASDG), 2013.
1159. Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 Mario AI competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010.

1160. Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 156–161. IEEE, 2009.
1161. Julian Togelius and Simon M. Lucas. Evolving controllers for simulated car racing. In *IEEE Congress on Evolutionary Computation*, pages 1906–1913. IEEE, 2005.
1162. Julian Togelius and Simon M. Lucas. Arms races and car races. In *Parallel Problem Solving from Nature-PPSN IX*, pages 613–622. Springer, 2006.
1163. Julian Togelius and Simon M. Lucas. Evolving robust and specialized car racing skills. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1187–1194. IEEE, 2006.
1164. Julian Togelius, Simon M. Lucas, Ho Duc Thang, Jonathan M. Garibaldi, Tomoharu Nakashima, Chin Hiong Tan, Itamar Elhanany, Shay Berant, Philip Hingston, Robert M. MacCallum, Thomas Haferlach, Aravind Gowrisankar, and Pete Burrow. The 2007 IEEE CEC Simulated Car Racing Competition. *Genetic Programming and Evolvable Machines*, 9(4):295–329, 2008.
1165. Julian Togelius, Mark J. Nelson, and Antonios Liapis. Characteristics of generatable games. In *Proceedings of the Fifth Workshop on Procedural Content Generation in Games*, 2014.
1166. Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the StarCraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 265–272. IEEE, 2010.
1167. Julian Togelius, Mike Preuss, and Georgios N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
1168. Julian Togelius, Tom Schaul, Jürgen Schmidhuber, and Faustino Gomez. Countering poisonous inputs with memetic neuroevolution. In *International Conference on Parallel Problem Solving from Nature*, pages 610–619. Springer, 2008.
1169. Julian Togelius, Tom Schaul, Daan Wierstra, Christian Igel, Faustino Gomez, and Jürgen Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, 23(3):30–33, 2009.
1170. Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 111–118. IEEE, 2008.
1171. Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N. Yannakakis. The Mario AI championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.
1172. Julian Togelius and Georgios N. Yannakakis. General General Game AI. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.
1173. Julian Togelius and Georgios N. Yannakakis. Choose Your Weapon: Survival Strategies for Depressed AI Academics [Point of View]. *Proceedings of the IEEE*, 112(1):4–11, 2024.
1174. Julian Togelius, Georgios N. Yannakakis, Sergey Karakovskiy, and Noor Shaker. Assessing believability. In Philip Hingston, editor, *Believable bots*, pages 215–230. Springer, 2012.
1175. Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation. In *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, pages 141–150. Springer, 2010.
1176. Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, 2011.
1177. Simone Tognetti, Maurizio Garbarino, Andrea Bonarini, and Matteo Matteucci. Modeling enjoyment preference from physiological responses in a car racing game. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 321–328. IEEE, 2010.
1178. Gustavo F. Tondello, Rina R. Wehbe, Rita Orji, Giovanni Ribeiro, and Lennart E. Nacke. A framework and taxonomy of videogame playing preferences. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 329–340, 2017.

1179. Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.
1180. Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint* arXiv:2307.09288, 2023.
1181. Paul Tozour and I. S. Austin. Building a near-optimal navigation mesh. *AI Game Programming Wisdom*, 1:298–304, 2002.
1182. Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.
1183. Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-O-Matic: Generating Videogames that Represent Ideas. In *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*. ACM, 2012.
1184. Mike Treanor, Alexander Zook, Mirjam P. Eladhari, Julian Togelius, Gillian Smith, Michael Cook, Tommy Thompson, Brian Magerko, John Levine, and Adam Smith. AI-based game design patterns. 2015.
1185. Chintan Trivedi. Turning Fortnite into PUBG with deep learning (cycleGAN). *Towards Data Science, available at: https://towardsdatascience.com/turning-fortnite-into-pubg-with-deep-learning-cyclegan-2f9d339dcdb0 (Accessed Jun. 18, 2018)*, 2018.
1186. Chintan Trivedi, Antonios Liapis, and Georgios N. Yannakakis. Contrastive learning of generalized game representations. In *2021 IEEE Conference on Games (CoG)*. IEEE, 2021.
1187. Chintan Trivedi, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Learning task-independent game state representations from unlabeled images. In *2022 IEEE Conference on Games (CoG)*, pages 88–95. IEEE, 2022.
1188. Chintan Trivedi, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Towards general game representations: Decomposing games pixels into content and style. In *Computer Vision and Games Workshop, British Machine Vision Conference (BMVC)*, 2023.
1189. Chintan Trivedi, Nemanja Rašajski, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Simulator-free visual domain randomization via video games. *arXiv preprint* arXiv:2402.01335, 2024.
1190. Chen Feng Tsai, Xiaochen Zhou, Sierra S Liu, Jing Li, Mo Yu, and Hongyuan Mei. Can large language models play text games well? Current state-of-the-art and open questions. *arXiv preprint* arXiv:2304.02868, 2023.
1191. Alan M. Turing. Digital computers applied to games. *Faster than thought*, 101, 1953.
1192. Anders Tychsen and Alessandro Canossa. Defining personas in games using metrics. In *Proceedings of the 2008 conference on future play: Research, play, share*, pages 73–80, 2008.
1193. Hiroto Udagawa, Tarun Narasimhan, and Shim-Young Lee. Fighting Zombies in Minecraft With Deep Reinforcement Learning. Technical report, Stanford University, 2016.
1194. Alfred Ultsch. Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. *Kohonen Maps*, 46:33–46, 1999.
1195. Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. Learning to speak and act in a fantasy text adventure game. *arXiv preprint* arXiv:1903.03094, 2019.
1196. Alberto Uriarte and Santiago Ontañón. Automatic learning of combat models for RTS games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
1197. Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint* arXiv:1609.02993, 2016.
1198. Rebekah Valentine. EA faces yet another class-action lawsuit connected to loot boxes. GamesIndustry.biz, 2020. Accessed 31 October 2024.
1199. Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. *arXiv preprint* arXiv:2408.14837, 2024.

1200. Josep Valls-Vargas, Santiago Ontañón, and Jichen Zhu. Towards story-based content generation: From plot-points to maps. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
1201. Wouter van den Hoogen, Wijnand A. IJsselsteijn, and Yvonne de Kort. Exploring behavioral expressions of player experience in digital games. In *Proceedings of the Workshop on Facial and Bodily Expression for Control and Adaptation of Games (ECAG)*, pages 11–19, 2008.
1202. Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(1):78–89, 2014.
1203. Pascal van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press, Cambridge, 1989.
1204. Niels van Hoorn, Julian Togelius, and Jürgen Schmidhuber. Hierarchical controller learning in a first-person shooter. In *Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 294–301. IEEE, 2009.
1205. Niels van Hoorn, Julian Togelius, Daan Wierstra, and Jürgen Schmidhuber. Robust player imitation using multiobjective evolution. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 652–659. IEEE, 2009.
1206. Giel van Lankveld, Sonny Schreurs, Pieter Spronck, and Jaap van Den Herik. Extraversion in games. In *International Conference on Computers and Games*, pages 263–275. Springer, 2010.
1207. Giel van Lankveld, Pieter Spronck, Jaap van den Herik, and Arnoud Arntz. Games as personality profiling tools. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 197–202. IEEE, 2011.
1208. Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid Reward Architecture for Reinforcement Learning. *arXiv preprint* arXiv:1706.04208, 2017.
1209. Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.
1210. Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.
1211. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
1212. Kanav Vats, William McNally, Pascale Walters, David A. Clausi, and John S. Zelek. Ice hockey player identification via transformers and weakly supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3451–3460, June 2022.
1213. Véronique Ventos, Daniel Braun, Colin Deheeger, Jean-Pierre Desmoulins, Jean-Baptiste Fantun, Swann Legras, Alexis Rimbaud, Céline Rouveirol, Henry Soldano, and Solène Thépaut. Construction and elicitation of a black box model in the game of bridge. In *Advances in Knowledge Discovery and Management: Volume 10*, pages 29–53. Springer, 2024.
1214. Ronaldo Vieira, Anderson Rocha Tavares, and Luiz Chaimowicz. Drafting in Collectible Card Games via Reinforcement Learning. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 54–61, 2020.
1215. Eduard Fosch Villaronga, Peter Kieseberg, and Tiffany Li. Humans forget, machines remember: Artificial intelligence and the right to be forgotten. *Computer Law & Security Review*, 34(2):304–313, 2018.
1216. Veera Vimpari, Annakaisa Kultima, Perttu Hämäläinen, and Christian Guckelsberger. “An Adapt-or-Die Type of Situation": Perception, Adoption, and Use of Text-to-Image-Generation AI by Game Industry Professionals. *Proceedings of the ACM on Human-Computer Interaction*, 7(CHI PLAY):131–164, 2023.
1217. James Vincent. Google ’fixed’ its racist algorithm by removing gorillas from its image-labeling tech. https://www.theverge.com/2018/1/12/16882408/google-racist-gorillas-photo-recognition-algorithm-ai, 2018. Accessed 31 October 2024.

1218. Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1096–1103. ACM, 2008.
1219. Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
1220. Madhubalan Viswanathan. Measurement of individual differences in preference for numerical information. *Journal of Applied Psychology*, 78(5):741–752, 1993.
1221. Thurid Vogt and Elisabeth André. Comparing feature sets for acted and spontaneous speech in view of automatic emotion recognition. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, pages 474–477. IEEE, 2005.
1222. Vanessa Volz, Günter Rudolph, and Boris Naujoks. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 269–276, 2016.
1223. Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference: GECCO'18*, pages 221–228. Association for Computing Machinery, 2018.
1224. John Von Neumann. The general and logical theory of automata. *Cerebral Mechanisms in Behavior*, 1(41):1–2, 1951.
1225. John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.
1226. Iro Voulgari, Marvin Zammit, Elias Stouraitis, Antonios Liapis, and Georgios N. Yannakakis. Learn to machine learn: designing a game based approach for teaching machine learning to primary and secondary education students. In *Proceedings of the 20th Annual ACM Interaction Design and Children Conference*, pages 593–598, 2021.
1227. Sandra Wachter and Brent Mittelstadt. A right to reasonable inferences: re-thinking data protection law in the age of big data and ai. *Colum. Bus. L. Rev.*, page 494, 2019.
1228. Gary L. Wagener, André Schulz, and André Melzer. Games, hormones, and "dark" personalities: Dark Tetrad and the effects of violent gaming on aggression, cortisol, and testosterone. *Physiology & Behavior*, page 114421, 2023.
1229. Karol Walędzik and Jacek Mańdziuk. An automatically generated evaluation function in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):258–270, 2014.
1230. Joseph Walton-Rivers, Piers R. Williams, Richard Bartle, Diego Perez-Liebana, and Simon M. Lucas. Evaluating and modelling hanabi-playing agents. In *2017 IEEE congress on evolutionary computation (CEC)*, pages 1382–1389. IEEE, 2017.
1231. Che Wang, Pan Chen, Yuanda Li, Christoffer Holmgård, and Julian Togelius. Portfolio Online Evolution in StarCraft. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
1232. Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint* arXiv:2305.16291, 2023.
1233. Hao Wang and Chuen-Tsai Sun. Game reward systems: Gaming experiences and social meanings. In *DiGRA conference*, volume 114, 2011.
1234. Hao Wang and Hao Wang. Games, logic and computers. *Computation, Logic, Philosophy: A Collection of Essays*, pages 195–217, 1990.
1235. Ruoyao Wang and Peter Jansen. Self-supervised behavior cloned transformers are path crawlers for text games. *arXiv e-prints*, pages arXiv–2312, 2023.
1236. Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. Can language models serve as text-based world simulators? *arXiv preprint* arXiv:2406.06485, 2024.

1237. Ziqi Wang, Chengpeng Hu, Jialin Liu, and Xin Yao. Negatively correlated ensemble reinforcement learning for online diverse game level generation. In *The Twelfth International Conference on Learning Representations*, 2024.
1238. Ziqi Wang, Yuchen Li, Haocheng Du, Jialin Liu, and Georgios N Yannakakis. Fun as Moderate Divergence: Evaluating Experience-Driven PCG via RL. *IEEE Transactions on Games*, 2024.
1239. Ziqi Wang, Jialin Liu, and Georgios N. Yannakakis. The fun facets of Mario: Multifaceted Experience-driven PCG via Reinforcement Learning. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022.
1240. Colin D. Ward and Peter I. Cowling. Monte Carlo search applied to card selection in Magic: The Gathering. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 9–16. IEEE, 2009.
1241. Henry N. Ward, Bobby Mills, Daniel J. Brooks, Dan Troha, and Arseny S. Khakhalin. AI solutions for drafting in Magic: The Gathering. In *2021 IEEE Conference on Games (CoG)*, 2021.
1242. Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.
1243. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
1244. Ben G. Weber. ABL versus Behavior Trees. *Gamasutra*, 2012.
1245. Ben G. Weber and Michael Mateas. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. IEEE, 2009.
1246. Joseph Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
1247. Paul John Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.
1248. Jim Whitehead. Toward proccedural decorative ornamentation in games. In *Proceedings of the 2010 workshop on procedural content generation in games*, pages 1–4, 2010.
1249. Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation (CEC) 2008. (IEEE World Congress on Computational Intelligence).*, pages 3381–3387. IEEE, 2008.
1250. Geraint A. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458, 2006.
1251. Minecraft Wiki. Minecraft. *Mojang AB, Stockholm, Sweden*, 2013.
1252. Christopher Winship and Robert D. Mare. Regression models with ordinal variables. *American sociological review*, pages 512–525, 1984.
1253. Peter Wittenburg, Hennie Brugman, Albert Russel, Alex Klassmann, and Han Sloetjes. Elan: A professional framework for multimodality research. In *5th international conference on language resources and evaluation (LREC 2006)*, pages 1556–1559, 2006.
1254. Robert F. Woodbury. Searching for designs: Paradigm and practice. *Building and Environment*, 26(1):61–73, 1991.
1255. Steven Woodcock. Game AI: The State of the Industry 2000-2001: It's not Just Art, It's Engineering. *Game Developer Magazine*, 2001.
1256. Stephen J Wright. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34, 2015.
1257. Xiaoshi Wu, Keqiang Sun, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score: Better aligning text-to-image models with human preference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2096–2105, 2023.
1258. Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.
1259. Wei Xi, Yongxin Zhang, Changnan Xiao, Xuefeng Huang, Shihong Deng, Haowei Liang, Jie Chen, and Peng Sun. Mastering strategy card game (Legends of Code and Magic) via

end-to-end policy and optimistic smooth fictitious play. *arXiv preprint* arXiv:2303.04096, 2023.
1260. Changnan Xiao, Yongxin Zhang, Xuefeng Huang, Qinhan Huang, Jie Chen, and c Sun. Mastering Strategy Card Game (Hearthstone) with Improved Techniques. In *2023 IEEE Conference on Games (CoG)*, 2023.
1261. Tong Xue, Abdallah El Ali, Tianyi Zhang, Gangyi Ding, and Pablo Cesar. RCEA-360VR: Real-time, continuous emotion annotation in 360 VR videos for collecting precise viewport-dependent ground truth labels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.
1262. Emmanouil Xylakis, Antonios Liapis, and Georgios N. Yannakakis. The Scream Stream: Multimodal Affect Analysis of Horror Game Spaces. In *Proceedings of the International Conference on Affective Computing and Intelligent Interaction*, 2024.
1263. Kaito Yamamoto, Syunsuke Mizuno, Chun Yin Chu, and Ruck Thawonmas. Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.
1264. Ya-Ju Yang, Tsung-Su Yeh, and Tsung-Che Chiang. Deck Building in Collectible Card Games using Genetic Algorithms: A Case Study of Legends of Code and Magic. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–07, 2021.
1265. Yi-Hsuan Yang and Homer H. Chen. Ranking-based emotion recognition for music organization and retrieval. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(4):762–774, 2011.
1266. Georgios N. Yannakakis. *AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation*. PhD thesis, University of Edinburgh, November 2005.
1267. Georgios N. Yannakakis. Preference learning for affective modeling. In *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*, pages 1–6. IEEE, 2009.
1268. Georgios N. Yannakakis. Game AI revisited. In *Proceedings of the 9th conference on Computing Frontiers*, pages 285–292. ACM, 2012.
1269. Georgios N. Yannakakis, Roddy Cowie, and Carlos Busso. The Ordinal Nature of Emotions. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
1270. Georgios N. Yannakakis, Roddy Cowie, and Carlos Busso. The ordinal nature of emotions: An emerging approach. *IEEE Transactions on Affective Computing*, 12(1):16–35, 2018.
1271. Georgios N. Yannakakis and John Hallam. Evolving Opponents for Interesting Interactive Computer Games. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8: Proceedings of the* 8^{th} *International Conference on Simulation of Adaptive Behavior (SAB-04)*, pages 499–508, Santa Monica, CA, July 2004. The MIT Press.
1272. Georgios N. Yannakakis and John Hallam. A Generic Approach for Generating Interesting Interactive Pac-Man Opponents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005.
1273. Georgios N. Yannakakis and John Hallam. A generic approach for obtaining higher entertainment in predator/prey computer games. *Journal of Game Development*, 1(3):23–50, 2005. December
1274. Georgios N. Yannakakis and John Hallam. Modeling and augmenting game entertainment through challenge and curiosity. *International Journal on Artificial Intelligence Tools*, 16(06):981–999, 2007.
1275. Georgios N. Yannakakis and John Hallam. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–971, 2007.
1276. Georgios N. Yannakakis and John Hallam. Entertainment modeling through physiology in physical play. *International Journal of Human-Computer Studies*, 66(10):741–755, 2008.
1277. Georgios N. Yannakakis and John Hallam. Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133, 2009.

1278. Georgios N. Yannakakis and John Hallam. Rating vs. preference: A comparative study of self-reporting. In *Affective Computing and Intelligent Interaction*, pages 437–446. Springer, 2011.
1279. Georgios N. Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.
1280. Georgios N. Yannakakis, Henrik Hautop Lund, and John Hallam. Modeling children's entertainment in the playware playground. In *2006 IEEE Symposium on Computational Intelligence and Games*, pages 134–141. IEEE, 2006.
1281. Georgios N. Yannakakis and Manolis Maragoudakis. Player modeling impact on player's entertainment in computer games. In *Proceedings of International Conference on User Modeling (UM)*. Springer, 2005.
1282. Georgios N. Yannakakis and Héctor P. Martínez. Grounding truth via ordinal annotation. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 574–580. IEEE, 2015.
1283. Georgios N. Yannakakis and Héctor P. Martínez. Ratings are Overrated! *Frontiers in ICT*, 2:13, 2015.
1284. Georgios N. Yannakakis, Héctor P. Martínez, and Maurizio Garbarino. Psychophysiology in games. In *Emotion in Games: Theory and Praxis*, pages 119–137. Springer, 2016.
1285. Georgios N. Yannakakis, Héctor P. Martínez, and Arnav Jhala. Towards affective camera control in games. *User Modeling and User-Adapted Interaction*, 20(4):313–340, 2010.
1286. Georgios N. Yannakakis and David Melhart. Affective game computing: A survey. *Proceedings of the IEEE*, 2023.
1287. Georgios N. Yannakakis and Ana Paiva. Emotion in games. *Handbook on Affective Computing*, pages 459–471, 2014.
1288. Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. *Dagstuhl Follow-Ups*, 6, 2013.
1289. Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161, 2011.
1290. Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 519–525. IEEE, 2015.
1291. Georgios N. Yannakakis and Julian Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.
1292. Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer Nature, 2018.
1293. Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
1294. Adrienne Yapo and Joseph Weiss. Ethical implications of bias in machine learning. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.
1295. Nick Yee. The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. *Presence: Teleoperators and virtual environments*, 15(3):309–329, 2006.
1296. Nick Yee, Nicolas Ducheneaut, Les Nelson, and Peter Likarish. Introverted elves & conscientious gnomes: the expression of personality in World of WarCraft. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 753–762. ACM, 2011.
1297. Serdar Yildirim, Shrikanth Narayanan, and Alexandros Potamianos. Detecting emotional state of a child in a conversational computer game. *Computer Speech & Language*, 25(1):29–44, 2011.
1298. Shubu Yoshida, Makoto Ishihara, Taichi Miyazaki, Yuto Nakagawa, Tomohiro Harada, and Ruck Thawonmas. Application of Monte-Carlo tree search in a fighting game AI. In *Consumer Electronics, 2016 IEEE 5th Global Conference on*. IEEE, 2016.
1299. David Young. *Learning game AI programming with Lua*. Packt Publishing Ltd, 2014.

1300. R. Michael Young, Mark O. Riedl, Mark Branly, Arnav Jhala, R. J. Martin, and C. J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1):51–70, 2004.
1301. José P Zagal, Staffan Björk, and Chris Lewis. Dark patterns in the design of games. In *Foundations of Digital Games 2013*, 2013.
1302. Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.
1303. Marvin Zammit, Antonios Liapis, and Georgios N. Yannakakis. CrawLLM: Theming games with Large Language Models. In *Proceedings of the IEEE Conference on Games*, 2024.
1304. Marvin Zammit, Antonios Liapis, and Georgios N. Yannakakis. MAP-elites with transverse assessment for multimodal problems in creative domains. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 401–417. Springer, 2024.
1305. Mar Zamorano, Carlos Cetina, and Federica Sarro. The quest for content: A survey of search-based procedural content generation for video games. *arXiv preprint* arXiv:2311.04710, 2023.
1306. David Zendle and Paul Cairns. Video game loot boxes are linked to problem gambling: Results of a large-scale survey. *PloS one*, 13(11):e0206767, 2018.
1307. Nianyin Zeng, Hong Zhang, Baoye Song, Weibo Liu, Yurong Li, and Abdullah M. Dobaie. Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing*, 273:643–649, 2018.
1308. Zhihong Zeng, Maja Pantic, Glenn I. Roisman, and Thomas S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):39–58, 2009.
1309. Jenny Zhang, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Omni: Open-endedness via models of human notions of interestingness. *arXiv preprint* arXiv:2306.01711, 2023.
1310. Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint* arXiv:1605.06450, 2016.
1311. Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. PDDLEGO: Iterative Planning in Textual Environments. *arXiv preprint* arXiv:2405.19793, 2024.
1312. Peng Zhang and Jochen Renz. Qualitative Spatial Representation and Reasoning in Angry Birds: The Extended Rectangle Algebra. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2014.
1313. Ruohan Zhang, Calen Walshe, Zhuode Liu, Lin Guan, Karl Muller, Jake Whritner, Luxin Zhang, Mary Hayhoe, and Dana Ballard. Atari-HEAD: Atari human eye-tracking and demonstration dataset. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 6811–6820, 2020.
1314. Tianyi Zhang, Abdallah El Ali, Chen Wang, Alan Hanjalic, and Pablo Cesar. RCEA: Real-time, continuous emotion annotation for collecting precise mobile video ground truth labels. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2020.
1315. Yulun Zhang, Matthew C. Fontaine, Amy K. Hoover, and Stefanos Nikolaidis. Deep surrogate assisted map-elites for automated hearthstone deckbuilding. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 158–167, 2022.
1316. Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, et al. Sotopia: Interactive evaluation for social intelligence in language agents. *arXiv preprint* arXiv:2310.11667, 2023.
1317. Jichen Zhu, Antonios Liapis, Sebastian Risi, Rafael Bidarra, and G. Michael Youngblood. Explainable AI for designers: A human-centered perspective on mixed-initiative co-creation. In *2018 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2018.
1318. Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2008.

1319. Albert L. Zobrist. *Feature extraction and representation for pattern recognition and the game of Go*. PhD thesis, The University of Wisconsin, Madison, 1970.
1320. Alexander Zook. Game AGI beyond Characters. In *Integrating Cognitive Architectures into Virtual Character Design*, pages 266–293. IGI Global, 2016.
1321. Alexander Zook and Mark O. Riedl. A Temporal Data-Driven Player Model for Dynamic Difficulty Adjustment. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2012.
1322. Robert Zubek and Ian Horswill. Hierarchical Parallel Markov Models of Interaction. In *AIIDE*, pages 141–146, 2005.
1323. Evelyn Zuniga, Stephanie Milani, Guy Leroy, Jaroslaw Rzepecki, Raluca Georgescu, Ida Momennejad, Dave Bignell, Mingfei Sun, Alison Shaw, Gavin Costello, et al. How humans perceive human-like behavior in video game navigation. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–11, 2022.

Index

A
AAA games, 236, 343, 415
Activation function, 107
  Leaky ReLU, 107
  ReLU, 100, 107
  Softmax, 100, 107
  Softplus, 100
A Behavior Language, 65
Adam Optimizer, 110, 113
Adaptive games, 237, 429
Adversarial games, 152, 168
Adversarial learning, 48, 278
Adversarial NPCs, 149
Adversarial planning, 182, 204
Adversarial player, 318
Adversarial search, 74
Affect expression, 429
Affective computing, 23, 58, 226, 316, 429
  Context, 429
Affective game loop, 429, 440
Affective interaction, 23, 343
Affective loop, 19, 24, 429
Affective sciences, 325
Affect modeling, 429
AGAIN dataset, 397
Agent architecture, 227, 419
AI, *see* Artificial intelligence
  Fundamentals, 35
AI and Games, **2**
  Co-revolutionary loop, 28
  Ethics, 439
  Panorama, 403
  Synergies, 403
AI-assisted game design, 244, 417
AI-assistive tools, 236
AI-based game design, 436
AI-complete, 268
AI controller, **160**
  Planner, 160
  Reactive, 160
AI designer, 161
AI Director, 12, 363
AI researcher, xv, 407, 410
ALE, *see* Arcade Learning Environment
AlphaStar, 6, 221
AlphaZero, 176
Analytics, 323
Angry Birds, 216
  AI competition, 216
Angry Birds AI competition, 216
ANN, *see* Artificial neural networks
ANN function approximator, 126, 129, 171
Annotation, 310, 350
  Absolute, **58**
  Categorical, **355**
  Dimensions, **355**
  During-experience, **55**
  First-person, **54**
  Forced, 53
  Free-response, 53
  In the wild, 359
  Lab, 359
  Ordinal, 60, 367
  Post-experience, **55**
  Pre-experience, **55**
  Relative, **58**
  Third-person, **54**
  Time-discrete, **358**
  Trace, **358**

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*,
https://doi.org/10.1007/978-3-031-83347-2

Answer set programming, 216, 245, 274, 302, 307
Anytime algorithm, 75, 273
Approximate forward model, 159
Approximate replay, 146
Arcade games, 25, 191, 392
Arcade Learning Environment, 154, **197**, 425
Architecture, 266, 288
Arousal, 327
Artificial General Intelligence, 19, 25, 411, 416, 438, 449
Artificial Intelligence, **1**
Artificial neural networks, **99**, 105, 277
  Activation function, 99, 100
  Backpropagation, 62, 100, 103
  Deep architectures, 106
  Forward operation, 101
  Hidden layer, 101
  Input layer, 101
  Learning rate, 103
  Multi-layer perceptron, 100
  Neuron, 99
  Output layer, 101
  ReLU, 100
  Shallow, 276
  Softmax, 100
  Vector, 36
Artificial potential fields, 221
ASP, *see* Answer set programming
Atari 2600, 6, 154, 158, 171, 192, 197
Audio designer, 293
Autoencoders, 105, **111**, 111, 208, 247, 427
  Variational autoencoders, 275
Automatic game testing, 424
Automatic labeling, 359
Autoregression, 118
Avatar, 316

B
Backgammon, 75, 127, 171, 183
Backpropagation
  Batch mode, 104
  Dynamic learning rate, 104
  Momentum, 104
  Non-batch mode, 104
  Random restarts, 104
Balance, 299, 305
Balancing, 218, 324, 341
Behavioral economics, 59, 332
Behavior analytics, 27
Behavior authoring, 62, 163
Behavior cloning, 174, 209
Behavior trees, 36, **64**
  Decorator, 65
  Selector, 64
  Sequence, 64
Belief desire intention model, 329
Believability, 297, 413
Believable agents, 225, 297, 323
Believable NPC, 226
Benchmarks, 142, 154, 411, 425
Best-first search, **70**
  A*, 70, 152, 164, 195
  Jump point search, 71
  Uniform-cost grids, 71
Biofeedback, 270, 343, 419
Board games, 1, 5, 8, 156, 159, **182**, 300, 411, 421, 423
Bodily expressions, 343
Branching factor, 74, 155
Bridge, 186
  Nook, 187
BT, *see* Behavior trees

C
C4.5, 97
Card games, **185**
Catastrophic forgetting, 171
Cellular automata, 241, 256
  Neighborhood, 256
  States, 256
ChatGPT, 123, 227, 300
Cheating, 143, 159, 160
Checkers, 5, 72, 182
  Chinook, 5
Chess, 5, 72, 74, 159, 182
  Deep Blue, 5, 74, 141
CI, *see* Computational intelligence
Cicero, 219
Classes, 56
  Class splitting criteria, 58
Classification, 44, 365, 368, 383
CLIP, 290
Clustering, **89**, 91, 323, 349, 380
  Hierarchical, **90**
  k-means, **89**
Co-creation, 244
Co-creativity, 237
Cognitive appraisal theory, 326
Collaborative filtering, 447
Commentator, 435
Commercial games, 415

Commodore 64, 192
Competitions, 425
Compositional pattern producing networks, 100, 132
Computational cost, 110
Computational creativity, 23, 24, 247, 434
  Exploratory, 428
  Transformational, 428
Computational intelligence, xvi
Computational narrative, 30, 226, 294
Concept map, 303
ConceptNet, 303
Constraints, 306
Constraint satisfaction, 307
Constraint solvers, 273
  SAT, 273
Content, 21, 235
  Audio, **292**
  Decorative, 239, 287
  Functional, 239, 287
  Levels, **287**
  Maps, **287**
  Mechanics, **299**
  Narrative, **296**
  Necessary, 238, 287
  Non-Spatial, 239
  Optional, 238, 292
  Representation, 266
  Rules, **299**
  Spatial, 239
  Visuals, **289**
Content evaluation, 147
  Aesthetics, 306
  AI agent, 307
  Designer, 307
  Difficulty, 306
  Function, 306
  Human player, 307
  Stochasticity, 306
  Subjective, 306
Content-intensive, 21
Content representation, 267
Contrastive autoencoder, 111
Contrastive learning, 48, 224, 431
Convolutional neural network, **112**, 305, 391, 428
Cooperative NPCs, 150
Cost function, 37, 102
Creative facets, 244, 287, 427
Creative path, 245
Creativity, 236, 299
Crowdsourcing, 294, 310, 387, 418
Culture, 348

D
DALL-E, 290
Darwinian evolution, 82
Data augmentation, 110, 351
Data compression, 308
Data-driven, 332, 337
Data mining, 333
Data ownership, 447
Deathmatch, 305
Decision transformer, **120**, 122, 198
Decision tree learning, 62, **96**
  C4.5, 97
  Gain ratio, 97
  ID3, 97, 384
  Information gain, 97
Decision trees, 36, **96**, 384
Deep Blue, 182
Deep convolutional neural network, 183, 390
Deep learning, 62, 93, 106, 128
  Autoencoders, 111
  Convolutional neural networks, 112
  Diffusion model, 116
  Generative Adversarial Networks, 114
  Large Language Models, 122
  Neural networks, 106
  Transformer, 118
  Vision transformer, 119, 120
Deep Neural Networks, 106
Deep Q-learning, 129, 173, 198
Deep Q network, 127, 128, 158, 172, 225
Deep reinforcement learning, 5, 171, 221
Demographics, 348
Demonstration mode, 144
Dendrogram, 90, 380
Deployment, 140
Depth, 300
Designer, 54, 244, 246, 247, 268, 299, 306, 407, 416
Designer modeling, 421
Deterministic, 75, 153, 167
Diamond-square algorithm, 261
Diffusion model, **116**, 117, 275
  Forward process, 116
  Reverse diffusion process, 116
Dominance, 327
Dominion, 190
Doom, 158, 206, 208
Drama management, 418

Dreamer, 179
Drivatar, 12, 145, 204, 447
Dropout, 110
Dungeon crawlers, **213**
  Nethack Challenge, 214
Dynamic programming, 52, 126, 170
Dynamic scripting, 176

E
Economics, 37, 67
Educator, xv
Electrocardiography, 343
Electrodermal activity, 342
Electroencephalography, 343
Electromyography, 343
Eliza, 225
Embedding, 36, 118
Embodied conversational agents, 224, 227
Emotion elicitation, 429
Emotion recognition, 429
Emotions, 67, 316
End user, 404, 407
Episodic memory, 56, 367
ϵ-greedy, 50, 76
Error function, 37, 102
Ethics, 29, **439**
Evaluation function, 268
Evolutionary adversarial planning, 169
Evolutionary algorithms, 39, 82, 148, 246
Evolutionary computation, **39**, 62, 411
  Constrained optimization, 85
  Covariance matrix adaptation evolution strategy, 85
  Crossover, 82
  Evolution strategies, 84
  Feasible-infeasible 2-population, 86, 307
  Fitness function, 37, 39, 268
  Generation, 84
  Genetic algorithms, 85
  Genetic programming, 36
  Inversion, 80
  Multiobjective, 86
  Mutation, 80
  Natural evolution strategy, 85
  Quality-diversity algorithms, 87
  Representation, 39
  Rolling horizon evolution, 168
Evolutionary planning, 168
Evolutionary reinforcement learning, 173
Evolutionary robotics, 173
Experience-driven PCG and PCG via Reinforcement Learning, 284
Experience-driven PCG via RL (EDRL), 249

Experience replay, 172, 198
Expert systems, 37
Exploration vs. exploitation, 156
Expressive NPC, 226
Expressive range, 268, 307
Expressivity, 307
Expressivity analysis, 272
Eye gaze, 197

F
Façade, 298, 418, 437
Facial expression, 342
F.E.A.R., 170
Few-shot learning, 360, 372
Fighting game AI competition, 201
Fighting games, 201
Finite state machines, 36, **62**
  Actions, 63
  Hierarchical, 62
  State, 63
  Transitions, 63
First-person shooter, 138, **206**, 305
  Deathmatch, 305
  2K BotPrize, 207
  VizDoom, 207
Flow channel, 329
Fog of war, 154
For Honor
  Toxicity, 384
For Honor, 384
Forward model, **159**, 426
  Approximate, 159
  Fast, 159
  None, 159
Foundation models, 448
Fractals, 241
Frequent itemset mining, 41
Frequent pattern mining, **41**, 62, 92
  Apriori, 62, **91**
  Data sequence, 92
  Frequent sequence, 92
  Generalized sequential patterns, 62, 92
  Itemset, 91
  Support, 91
  Support count, 92
Frequent sequence mining, 41
FSM, *see* Finite state machines

Function approximator, 174
Fusion, 391
Fuzzy logic, 221
Fuzzy set, 67

G
Galvanic skin response, 343
Game adaptation, 420
Game AI, 2
  Academia, 8
  Areas, 404
  Deployment, 157
  Dominant methods, 405
  Frontier research, 29, 423, 438
  General, 423
  Industry, 9
  Panorama, 29
  Secondary methods, 405
Game AI deployment, **157**
Game analytics, 22, 27, 323
Game balancing, 341
Game characteristics, 140, 150
  Observability, 151
  Stochasticity, 151
  Time granularity, 151
Game context, 409
  AI agent, 347
  Content, 347
Game curator, 433
Game data, 21
Game data mining, 22, 27, 396
Game data science, 333
Game design, 266, 303, 323, 331, 369, 424, 436
Game directing, 433
Game engine, 430
Game experience questionnaire, 54
Game facets, 427
Game generation, 237, **302**, 428
Game industry, 67
Game metrics, 323, 346, 419
Gameplaying AI, 6
  Methods, 180
Game profile, 347
Game representation learning, 431
Game stakeholder, 421
Game studies, 294, 325, 331
Game testing, 146, 323
Game theory, 30, 37, 48
Game tree, 69
Game user study, 310
Games, xi
GameVibe corpus, 398
GANs, *see* Generative Adversarial Networks

Gaussian process, 366
Gaze tracking, 197, 345
Gender, 348
General affective game loop, 429
General emotional intelligence, 429
General game AI, 423
General game design, 425
General game generation, 428
General game playing, 70, 142, 147, 161, 197, 198, 424, 425
  Ludii, 183
General game playing competition, 25, 161, 183, 425
General game playing system, 183
General general game AI, 424
General intelligence, 23, 25, 426, 429, 437
General level design, 426
General video game AI competition, 168, **199**, 425
General video game description language, 438
Generative Adversarial Imitation Learning, 174
Generative Adversarial Networks, 48, **114**, 275
  Discriminator, 114
  Generator, 114
Generative AI, 13, 27, 253, 275
Genetic programming, 173, 193
Geneva wheel model, 56
Gesture, 343
Ghost agents, 145
Global optimization, 82
Global search, 62, **79**
Go, 5, 17, 74, 141, 159, 167, 182
  AlphaGo, 5, 18, 167
Goal-oriented action planning, 30, 69
Gradient descent, 103
Grammars, 36, 254, 300
  Grammatical evolution, 36
Graphics, 290
Graphs, 36
Grid-based path-planning, 71
Ground truth, 54, 353, 368, 386, 430
GVGAI, *see* General video game AI competition

H
Hanabi, 190
Haptics, 343

HCI, *see* Human-computer interaction
Head pose, 342
Hearthstone, 188
Heart rate variability, 390
Heatmap, 307, 340
Hebbian learning, 40, 99
Heightmap, 260
Heuristic, 37
Hidden information, 154
Hierarchical task networks, 30, 69
Hill climber, 79
  Gradient-based, 80
  Randomized, 80
Horror games, 343
Human-computer interaction, 316, 371, 409
  Rich, 19
Human-like, 144, 197, 226, 323, 370, 413, 426
Human-like agent, 143
Human feedback, 23, **52**, 365
  First vs. third person, 54
  Free vs. forced response, 53
  Objective vs. subjective, 53
  Ordinal, **58**
  Pre- vs. during-vs. post-experience, 55
  Rating vs. class vs. rank, 56
Human player, 319
Hybrid algorithms, 62, 126, 176, 325
HyperNEAT, 198

I
ID3, 97
Ideation tools, 434
Imitation learning, 209
Imperfect information, 75
Indie game development, 292, 415
Input representation, **157**
  Extracted data, 157
  Pixels, 157
Intelligent tutoring systems, 227
Intensity map, 260
Interactive evolution, 245, 291
Interactive fiction, **224**, 224, 299, 345
Interactive narrative, 224, 416, 437
Interactive storytelling, 295
Intrinsic motivation, 364, 370
Inverse Dynamics Model, 46
Iterative refinement, 291
Iterative width search, 70

K
Killer application, 8, 24
Knowledge distillation, 110
Knowledge representation, 22

L
Large Language Models, 13, 40, **122**, 418
Large Multimodal Models, 13
Latent variable evolution, 115, 270, 278
Lazy learning, 204
Learnability, 300, 369
Learning, 37
Learning classifier systems, 176
Learning objective, 228
Learning to rank, 45
Level design, 244, 287, 427
Likert, 56
Linear discriminant analysis, 366
LLMs, *see* Large Language Models
Locality, 268
Local optimum, 81
Local search, 62, **79**
Logic programming, 273
LoRa, 291
Loss function, 37, 109
  Cross entropy, 109
  Mean squared error, 109
  Reconstruction, 111
LSTM, 105
L-Systems, 255
Ludii, 183

M
Machine learning, 11, 22, 37, **40**, **93**, 160, 212, 220, 275, 333, 338, 368, 435
Macro-actions, 157
Magic: the Gathering, 187
Manual game testing, 146
MAP-Elites, 87, 189, 271, 279
Mario AI competition, 71, 144, 165, 194, 411

Marketing, 58
Markov decision process, 37, 49
Markov models, **276**, 276
Markov property, 49, 126, 152
Match-3 games, 216
Matchmaking, 324
MCTS, *see* Monte Carlo tree search
Memory-experience gap, 54
Metaverse, 28
Midpoint displacement algorithm, 261

Minecraft, 138, 178, **210**, **212**, 260, 288, 444
  Project Malmo, 212
  Video Pretraining, 213
  Voyager, 175, 212
MiniDungeons, 214
Minimax, 5, 62, **72**, 156, 164
  α-β pruning, 74, 152
Mixed-initiative co-creation, 418
Mixed-initiative co-creativity, 251
Mod, 15
Modalities of input, 343, 348
Modding, 15
Monetization, 323, 365, 383
Monte Carlo tree search, 62, **74**, 167, 200, 276, 421, 425
  Backpropagation, 75, 77
  Expansion, 75, 76
  Rollout, 75, 167
  Selection, 75, 76
  Simulation, 75, 77
  UCB1, 76
Moore neighborhood, 195, 257
Ms Pac-Man, **61**, 64, 66, 67, 70, 71, 74, 77, 81, 88, 91, 92, 95, 98, 105, 110, 111, 113, 115, 117, 120, 122, 123, 126, 129, 131, 133, **192**
  Competition, 230
Multiagent systems, 30
Multimodal, 19
Multimodal fusion, 391
Multiobjective Evolutionary Algorithms, 86

Multi-player, 152
Multiplayer Team Bots, 144
Muscle activation, 343

N
Narrator, 435
Natural language processing, 22, 122, 224, 329
Navigation, 23, 71
Navigation mesh, 71
NEAT, *see* Neuroevolution of augmenting topologies
Nethack Challenge, 214
Neural game engines, 179
Neuroevolution, 62, **131**, 133, 173, 197, 207, 436
Neuroevolution of augmenting topologies, 132, 198, 218, 436
Neuroscience, 325
n-grams, **276**, 276
Nintendo NES, 192
Nintendo Wii, 144
NLP, *see* Natural language processing
Noise, 260
  Perlin noise, 262, 290
Non-deterministic, 74, 75, 153
Non-player character, 4, 152
  Behavior, 165
NPC, *see* Non-player character
NP-hard, 17
Number of games, **161**
  Many, 161
  One, 161
Number of players
  Many-player, 152
  Single-player, 152
  Two-player, 152

O
Objective function, 37
Objective input
  Body, 343
  Language, 343
  Physiology, 342
Observability, **154**
  Hidden information, 154
  Partial observability, 154
  Perfect information, 154
One-and-a-half-player, 152
Ontology, 189
OpenAI Five, 6
OpenAI Universe, 203
Open-source, 438
Opponent modeling, 318
Optimization, 39, 95, 168
Orchestration, 305
Ordinal regression, 45, 366, 367
Overfitting, 110

P
Pac-Man, 61, **192**, 333
Pairwise preference, 58, 368
Pareto front, 86
Pathfinding, 30, 71, 165
  Hierarchical, 165
Path-planning, 164
PCG, *see* Procedural content generation
PCGRL, 311
PCG Taxonomy, 237
  Content, 237

Methods, 237
Roles, 237
Peak-end rule, 56
Perfect information, 154
Perlin noise, 290
Personality, 348, 444
Personality traits, 444
Personalization, 370
Personas, 145
Phenomenological debugging, 382
Photoplethysmography, 343
Photorealism, 290
Physical games, 330
Physiological signal, 345
Physiology, 326, 342, 419
Planning, 23, 30, 71, 164, 168, 216, 218, 295, 406, 411
Hierarchical Task Network, 295
Platformer experience dataset, 388, 398
Platformers, 156, 194, 311
Play as non-player, 139
Play as player, 139
Player archetypes, 330
Player behavior, 321, 323
Player behavior modeling, **352**, 404, 418
Player character, 316
Player embedding, 372, 373
Player embedding modeling, 404
Player experience, 27, 247, 321, 324, 386, 419
Player experience modeling, 248, 250, **353**, 368, 386, 404, 418
Player expertise, 349
Player metrics, 323, 340, 346
Player model
Input, 337, **338**
Method, 337
Output, 337, **350**
Player modeling, 7, 250, 315, 409, 413
Authoring, 362
Behavior-adaptive games, 323
Behavior analysis, 323
Behavior design, 323
Classification, 364
Dynamic observation, 334
Ethics, 440
Experience-adaptive games, 324
Experience analysis, 324
Experience design, 324
Fusion, 348
Hybrid algorithms, 334
Input, 332
Methods, **337**
Model-based, **325**, 325, 362
Model-free, 325, **332**
Objective labels, 334, 364
Offline, 370
Player State, 332
Preference learning, 364
Regression, 364
Runtime, 370
Static observation, 334
Subjective labels, 334
Taxonomy, 316
Player model input
Context, **346**
Game context, **338**
Gameplay, **338**, **340**
Objective, **338**, **342**
Player profile, **338**
Player profile, **348**
Player profiling, 319
Play for experience, 138, 139, 413
Playing games
Uses, 139
Playtesting, 141, 216, 218, 309, 369, 413
Playthroughs, 299, 413
Play to win, 138, 139, 411
Playtrace, 222, 369
Pokemon, 190
Poker, 75, 154, 186, 318
Libratus, 186
Policy gradient, 129
Posture, 342, 343
PPO, *see* Proximal Policy Optimization
Practical wisdom, 36
Preference, 366
Preference elicitation, 214
Preference learning, 44, 214, **365**, 368, 387, 391
Neuroevolution, 249, 387
Premature convergence, 104
Preschooler games, 330
Presence, 330
Prey-predator games, 330, 390
Probabilistic methods, 30
Probabilistic models, 36
Procedural architecture, 288
Procedural audio, 293
Procedural content generation, 6, 235, 418
Assisted, 417
Authored, 241
Autonomous, 242, **247**, 414
Constructive, 241, 253
Controllable, 240
Deterministic, 240

Diversity, 271
Evaluation, **305**
Experience-agnostic, **250**
Experience-driven, 242, **247**, 387, 420, 433
Fractal, 260
Generate-and-test, 241, 254
Grammar-based, 254
Iterative, 242
LLMs, 280
Machine learning, 275
Methods, 253
Mixed-initiative, 242, **244**, 273, 417
Noise, 260, 265
Non-controllable, 240
Offline, 242
Orchestration, 428
Quality, 271
Quality Diversity, 270, 271
Runtime, 242
Search-based, 266
Single pass, 242
Solver-based, 273
Stochastic, 240
Trained, 241
Wave Function Collapse, 265
Procedural personas, 31, 145, 214, 309, 370, 420, 447
Programmer, xv
Protocol design, 310
Proximal Policy Optimization, **129**, 172
Psychology, 58, 316, 325
Adaptation level theory, 59
Psychometrics, 368, 369
Psychophysiology, 327, 390
Pupil dilation, 342
Pupillometry, 345
Puzzle games, **215**

Q
QD score, 308
Q-learning, 36, 50, **124**, 126, 171
Discount factor, 125
Learning rate, 125
Quality diversity, 308
Archive, 87
MAP-Elites, 87
Quantization error, 89

R
Racing games, 158, **203**
Drivatar, 204
Simulated car racing championship, 205
Sophy, 206
TORCS, 158, 205
Ranks, 56
Ratings, 56, 367
Inter-personal biases, 368
Inter-personal differences, 56
RBF networks, 100
Real-time games, **156**
Real-time strategy, 157, 217
Reconstruction loss, 111
Recurrent neural networks, 105
Regression, 43, 364, 367
Regression testing, 146
Reinforcement learning, 5, **48**, 62, **124**, 127, 142, 153, 156, 160, 170, 369, 411
Backup, 51
Benchmarks, 142, 158
Bootstrapping, 50
Deep Q network, 124
Episodic vs. incremental, 50
Exploration vs. exploitation, 50
Hybrid reward architecture, 194
Neuroevolution, 124
Off-policy vs. on-policy, 50
Policy, 49
Proximal policy optimization, 124
Q-learning, 124
Reward, 37, 48
Taxonomy, 50
World model, 49
Reinforcement learning benchmarking, 142

Relative labels, 365
ReLU, 107, 129
Repair, 85
Representation, 35, 157, 158
Representation learning, 111, 177
Respiration, 345
Reteller, 435
Retention, 323
Retro Learning Environment, 202
Reward, 370
RL, *see* Reinforcement learning
RLHF, 54, 60, 123, 371
Robocup, 223
Rogue, 6, 10, 235, 288
Role-playing game, 138, 156
Rolling horizon evolution, 168
RPG, *see* Role-playing game
Rubber band AI, 143
Runtime, 285

S
Sandbox games, 210
SARSA, 201
Scaling maximalism, 36, 448
Search-based PCG
  Direct evaluation, 268
  Interactive, 269
  Simulation-based evaluation, 269
Search tree, 69, 74
Self-assessment manikin, 56
Self-Determination Theory, 329, 362
Self-organization, 40
Self-organizing maps, 105, **380**
Self-reporting, 365
Self-supervised learning, **46**, 372
Semantic memory, 56
Semi-supervised learning, **45**, 371
Sentient sketchbook, 70, 244, 413
Sentiment analysis, 329
Shallow architectures, 106
Shallow learning, 93, 106
Showdown AI Competition, 190
Signal processing, 22
Simulated annealing, 81
Simulated car racing championship, 205, 411
Simulation-based testing, 141, 143, 412
Single-player, 152
Single-player games
  Adversarial NPCs, 149
  Cooperative NPCs, 150
Skin conductance, 390
Social intelligence, 23
Social simulation games, **226**
Sound
  Diegetic, 294
  Non-diegetic, 294
SparCraft, 221
Speech, 342, 343
Stable Diffusion, 290
StarCraft, 18, 23, 154, **219**, 421
  AlphaStar, 221
  TorchCraft, 18
Steam, 396
Stochastic gradient descent, 104, 108
  Adam, 110
  Batch, 108
  Optimizers, 109
Stochasticity, **153**
Stochastic tree search, 167
  Deterministic, 153
  Non-deterministic, 153
Story generation, 295, 418
Strategy games, **217**
  microRTS, 218
  Real-time, 217
  Turn-based, 217
Street Fighter, 201
STRIPS, 170
Super Mario Bros, 17, 77, 144, **194**, 387
Supervised learning, **42**, 62, 160, 174, 364
  Features, 42
  Generalization, 42
  Labeled data, 42
  Testing, 43
  Training, 42
Support vector machines, 62, **93**
  Hard margin, 95
  Kernel, 95
  Maximum margin, 94
  Soft margin, 95
  Support vectors, 95
Surprise search, 291
Suspension of disbelief, 24
Symbolic representation, 169

T
Tabular representation, 36
Taxonomy, 237, 324, 408
  Game characteristics, 150
TD-Gammon, 5, 127, 171
TD learning, 36, 52, 62, 129, 411
TD learning with ANN function approximators, 126
Team sports games, **223**
  Robocup, 223
Telemetry, 22
Temporal difference learning, *see* TD learning
Text adventures, 224
Text-based adventure games, 345
Text-based games, 224
Text-to-game, 432
Theory of flow, 329
Theory of fun, 331
Think-aloud protocol, 53
Tic-Tac-Toe, 4, 74
Token, 118
Tom Clancy's The Division, 394
Toxicity, 384
Training signal, 37, 373
Transductive SVM, 46
Transformer, 19, **118**, 118, 275
  Attention, 118
  Autoregression, 118

Embedding, 118
GPT, 118
Large Language Models, 118
Self-Attention, 118
Token, 118
Traveling Salesman Problem, 168
Tree
Behavior, 36
Decision, 36
Tree search, 22, **38**, 62, **69**, 153, 156, 159, 160, 164, 182
Classic tree search, 164
State evaluation, 74
Turing test, 144, 197, 207, 412, 420, 426
Turn-based games, **156**
Turn-based strategy, 217
Turtle graphics, 255
2K BotPrize, 144, **207**, 207, 412, 420
Pogamut, 207
Two-player, 152

U
U-matrix, 382
Uncertainty, 299
Uninformed search, **69**
Breadth-first, 70
Depth-first, 69
Unity, 69, 311
Unity engine, 159
Unreal engine, 69, 159, 295
Unsupervised learning, **40**, 62, **89**, 350, 373
Association mining, 350
Clustering, **40**, 89, 323, 350, 373
Frequent pattern mining, **41**, 89, 373, 388
Hierarchical clustering, 62, 380
k-means, 62, 380
Self-organizing maps, 41, 62, 99, **380**
Usability, 329
User experience, 325, 419, 430
User experience research, 316
User modeling, 58, 316
User retention, 365, 383
User testing, 373
Utility, 35, 146, 267, 309
Utility-based AI, 62, **66**
Utility systems, 66

V
Valence, 327
Variational autoencoder, 111
Video game description language, 199, 300
Video generation, 291
Video Pretraining, 213
Virtual agents, 226, 296
Virtual camera, 295
Virtual Cinematography, 434
Virtual humans, 227
Virtual reality, 345
Vision Transformer, 119, 120
Visual cues, 345, 388
Visualization, 307
Visuals
Neural style transfer, 292
Upscaling, 292
VizDoom, 207, **208**
Von Neumann neighborhood, 257
Voxel, 260, 288

W
Wave Function Collapse, 13, 265, 289, 291
Weak supervision, 45
Weapon generation, 305
Weapon selection, 67
Weight vector, 36
WordNet, 303
World models, 179

Z
Zero-shot, 36, 360
Zero-sum, 152
Z-machine, 224
ZX Spectrum, 192