UNIVERSITY OF MANNHEIM

DOCTORAL THESIS

Combining Procedural Content Generation and Heuristic Simulations for Automated Game Balancing

Author:
Florian Alexander RUPP

Supervisors:
Prof. Dr. Heiner STUCKENSCHMIDT
Prof. Dr. Kai ECKERT

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Science at the University of Mannheim

September 15, 2025

Dekan: Prof. Dr. Claus Hertling, Universität Mannheim
Referent: Prof. Dr. Heiner Stuckenschmidt, Universität Mannheim
Koreferent: Prof. Dr. Kai Eckert, Technische Hochschule Mannheim

Tag der mündlichen Prüfung: 11. Dezember 2025

UNIVERSITÄT MANNHEIM

DOKTORARBEIT

Kombination von prozeduraler Inhaltsgenerierung und heuristischen Simulationen für automatisiertes Game Balancing

Autor:
Florian Alexander RUPP

Betreuer:
Prof. Dr. Heiner STUCKENSCHMIDT
Prof. Dr. Kai ECKERT

Inauguraldissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften der Universität Mannheim

15. September 2025

Abstract

This thesis introduces, explores, and evaluates algorithmic methods for automated game balancing with the goal of general applicability, independent of a specific game environment. It contributes to the academic fields of deep reinforcement learning (RL) and search-based optimization, with specific focus on the application for automated game balancing – a niche research area distinct from the practice-driven domain of game balancing.

Games play a substantial role in contemporary society and culture, functioning not only as entertainment but also as tools for education, social interaction, and cognitive development. To be engaging, competitive games must be balanced so that all players have equal initial chances of winning. While achieving balance typically requires extensive manual work and human playtesting during development, existing research proposes to support this process using search-based optimization combined with game simulations to determine balance. These approaches, however, are tailored to specific environments and, therefore, are difficult to transfer to other games. In addition, the simulation of games is computationally intense. The goal of this thesis is to develop methods for automatically generating balanced content for games using procedural content generation (PCG). We focus on two key facets that significantly impact a game's overall balance: game levels and game economies. To investigate automated balancing, we make the following six contributions (C 1–6):

(C 1) To enable automated balance estimation, we first define what balance means in the context of two-player games. To express the balance quantitatively and domain-independently based on simulation data, we derive a metric from the field of algorithmic fairness research. **(C 2)** Since estimating balance through multiple simulations is computationally expensive, we propose a novel approach to accelerate the automated balancing of tile-based levels in combination with agent-based simulations. To reduce the number of required simulations, we frame level balancing as (1) a PCG problem and (2) a trajectory optimization task modeled as a Markov decision process, enabling the use of RL. To this end, we extend the PCG via RL (PCGRL) framework with a new definition of the action space and demonstrate that our method generates balanced levels more efficiently than PCGRL or model-free search-based baselines. **(C 3)** While balance is evaluated using artificial agents, games are made for human players. To assess human perception of balance, we conduct an empirical study with human playtesters. Statistically, we prove that players perceive levels that were balanced using our method as more balanced in most cases than those that had not undergone this process. In addition to game levels, we study

the generation and balancing of game economies, building on an existing graph-based representation from related work. **(C 4)** We explore how graphs defined by sets of constraints can be generated by manipulating their adjacency matrices and introduce G-PCGRL (Graph-PCGRL), an adaptation of PCGRL that learns to controllably generate graphs using RL. While we demonstrate the feasibility of this approach and improved inference performance compared to baselines, we observe limited scalability. **(C 5)** To address this shortcoming, we propose GEEvo (Game Economy Evolution), a framework for generating and balancing abstract graph-based game economies using evolutionary algorithms. GEEvo offers greater flexibility and supports the generation of larger graphs, though at the cost of slower computational performance compared to G-PCGRL. **(C 6)** In addition, we contribute *Feast & Forage*, a flexible, easily adaptable, and extensible tile-based game environment to study automated game balance via PCG.

Finally, probabilistic elements are essential in games to promote replayability and strategic decision-making under uncertainty. In both facets, levels and economies, we observe that automated approaches tend to bypass randomness to stabilize balance. As a general conclusion, we argue that maintaining meaningful randomness requires incorporating mechanisms that prevent its elimination by optimization.

Kurzfassung

In dieser Dissertation werden algorithmische Methoden zur automatisierten Balancierung von Spielen vorgestellt, untersucht und evaluiert, mit dem Ziel einer allgemeinen Anwendbarkeit, unabhängig von einer spezifischen Spielumgebung. Die Arbeit leistet damit einen Beitrag zu den wissenschaftlichen Feldern des Deep Reinforcement Learning (RL) und der suchbasierten Optimierung, mit dem besonderen Fokus auf automatisiertem Game Balancing – einer Nische, die sich von der praxisorientierten Anwendung des Game Balancing unterscheidet.

In der heutigen Gesellschaft und Kultur kommt Spielen eine wichtige Bedeutung zu. Sie dienen nicht nur der Unterhaltung, sondern werden auch im Kontext von Bildung, sozialer Interaktion oder kognitiver Entwicklung eingesetzt. Um ein unterhaltsames Erlebnis bieten zu können, müssen insbesondere kompetitive Spiele ausbalanciert sein, sodass alle Spieler*innen zu Beginn die gleichen Gewinnchancen haben. In der Praxis erfordert das Ausbalancieren eines Spiels einen umfangreichen manuellen Aufwand, der unter anderem mit Spieltests von menschlichen Spieler*innen während der Entwicklung verbunden ist. Die Forschung schlägt daher vor, diesen Prozess durch suchbasierte Optimierung in Kombination mit Spielsimulationen zu unterstützen. Diese Ansätze sind jedoch auf bestimmte Umgebungen zugeschnitten und daher nur schwer auf andere Spiele übertragbar. Außerdem ist die Agenten-basierte Simulation eines Spiels rechenintensiv. Spiele, die zusätzlich automatisiert Inhalte erstellen, bieten ein abwechslungsreicheres Spielerlebnis und erhöhen den Wiederspielwert. Ziel dieser Arbeit ist es, Methoden der prozeduralen Inhaltsgenerierung (PCG) zu entwickeln, um balancierte Inhalte für Spiele automatisiert erstellen zu können. Der Fokus liegt dabei auf zwei Schlüsselaspekten, die die Gesamtbalance eines Spiels maßgeblich beeinflussen: Levels und Spielökonomien. Zur Untersuchung der automatisierten Ausbalancierung von Spielen leistet diese Dissertation die folgenden sechs Beiträge (C 1–6):

(C 1) Um eine automatisierte Schätzung der Balance eines Spiels zu ermöglichen, wird zunächst definiert, was Balance im Kontext eines kompetitiven Spiels für zwei Spieler*innen bedeutet. Um die Balance basierend auf Simulationsdaten quantitativ und domänenunabhängig auszudrücken, wird eine Metrik aus dem Bereich der algorithmischen Fairnessforschung abgeleitet. **(C 2)** Da die Schätzung der Balance aus mehreren Simulationsdurchläufen rechenintensiv ist, schlägt diese Dissertation einen neuartigen Ansatz zur automatischen Balancierung von kachelbasierten Levels in Kombination mit agentenbasierten Simulationen vor. Um die Anzahl der notwendigen Simulationen zu reduzieren, wird

der Prozess der Ausbalancierung von Levels als (1) ein Problem der prozeduralen Inhaltsgenerierung (PCG) und (2) der Trajektorienoptimierung betrachtet. Letzteres wird als Markov-Entscheidungsprozess modelliert, was den Einsatz von RL ermöglicht. Zu diesem Zweck wird das PCG via RL (PCGRL) Framework um neue Definitionen des Aktionsraums erweitert und es wird gezeigt, dass die vorgestellte Methode balancierte Levels effizienter erzeugt als PCGRL oder modellfreie, suchbasierte Ansätze. **(C 3)** In dieser Dissertation evaluieren wir die Balance mit künstlichen Agenten, Spiele werden jedoch für menschliche Spieler*innen entwickelt. Um die menschliche Wahrnehmung der simulierten Balance zu bewerten, wird eine empirische Studie mit menschlichen Spieler*innen durchgeführt. Statistisch gesehen werden die mit dieser Methode balancierten Levels von den Spieler*innen in den meisten Fällen als ausgewogener empfunden als die Levels vor der Balancierung.

Zusätzlich zu den Spiellevels wird die Erzeugung und die Balancierung von Spielökonomien untersucht. Zu diesem Zweck wird auf eine existierende graphbasierte Modellierung aus verwandten Arbeiten aufgebaut. **(C 4)** Zunächst wird untersucht, wie Graphen, die durch Sets aus Bedingungen definiert sind, durch Manipulation ihrer Adjazenzmatrizen erzeugt werden können. Dazu wird G-PCGRL (Graph-PCGRL) eingeführt, eine Erweiterung von PCGRL, die lernt, Graphen durch RL kontrollierbar zu generieren. Während die Machbarkeit dieses Ansatzes und die verbesserte Inferenzleistung im Vergleich zu suchbasierten Ansätzen aufgezeigt wird, wird jedoch eine begrenzte Skalierbarkeit beobachtet. **(C 5)** Um dieses Manko zu beheben, stellt diese Dissertation GEEvo (Game Economy Evolution) vor, ein Framework für die Generierung und die Balancierung von abstrakten graphbasierten Spielökonomien unter Verwendung evolutionärer Algorithmen. GEEvo bietet eine größere Flexibilität und unterstützt die Generierung größerer Graphen, allerdings auf Kosten einer höheren erforderlichen Rechenleistung im Vergleich zu G-PCGRL. **(C 6)** Außerdem stellt diese Arbeit *Feast & Forage* vor, eine flexible, leicht anpassbare und erweiterbare kachelbasierte Spielumgebung zur Untersuchung der automatischen Spielbalance durch PCG.

Probabilistische Elemente in Spielen sind unerlässlich, um die Wiederspielbarkeit und die strategische Entscheidungsfindung der Spieler unter Unsicherheit zu motivieren. Für beide Aspekte, Levels und Ökonomien, wird festgestellt, dass automatisierte Ansätze dazu neigen probabilistische Spielmechaniken zu umgehen, um die Balance zu stabilisieren. Als allgemeine Schlussfolgerung kommt diese Dissertation daher zu dem Schluss, dass die Aufrechterhaltung einer sinnvollen Zufälligkeit die Integration von Mechanismen erfordert, die ihre Eliminierung durch Optimierung verhindern.

Acknowledgments

I have long looked forward to the moment of writing the acknowledgments for this thesis -— and that moment has finally arrived, so here it is.

First and foremost, I would like to thank my supervisors, Heiner Stuckenschmidt and Kai Eckert, for their continuous support and valuable feedback over the years, both in terms of content and organization. Most of all, I am grateful for the freedom they gave me to follow my own research interests.

I consider the fall of 2022 to be the actual starting point for this dissertation. After a challenging phase of searching for a suitable research direction, the joint trip with a delegation from HdM Stuttgart to Tokyo Tech in Japan played an important role in shaping my ideas and setting the course for this work. I will always remember how I pitched the basic ideas for the first paper of this work to Kai on the bus to Shin-Yokohama station – directly after a 12-hour flight to Japan with hardly any sleep. I am very grateful to everyone who made this trip what it was: Kai Eckert, Magnus Pfeffer, Manuel Eberhardinger, Zoltán Kacsuk, and especially Patrick Takenaka for successfully securing the funding and organizing everything – どうもありがとう.

I would also like to thank the IAAI at HdM Stuttgart, especially the Professors Johannes Maucher, Christian Becker-Asano and Magnus Pfeffer, who have enabled me to participate in activities such as the annual research retreat, the journal club meetings, and the opportunity to use the institute's deep learn cluster. I am particularly grateful that I continued to be welcome in the HdM Stuttgart research landscape even after my official move to TH Mannheim. At this point I would like to take the opportunity to thank Kai for approving this and making it possible – which is not a matter of course.

And of course, a big thank you to every single member of the one-and-only AI Lab 143 and Humanoid Lab at HdM Stuttgart, in particular Manuel Eberhardinger, Patrick Takenaka, Johannes Theodoridis, Marcel Heisler, Zoltán Kacsuk, Tobias Malmsheimer, Anna Kleinhans, Samuel Wunderlich, and Sebastian Maisel. Without the fruitful topic-related discussions – balanced by occasional lighthearted nonsense – and the social activities, such as board game and Karaoke sessions, which have earned a unique reputation well beyond the research group, this thesis would not be what it is today.

In addition, I would like to thank all members of the CAIUS project, especially Jakob Kappenberger, Frederic Gerdon, Lea Cohausz, and Christoph Kern, for their inspiration and insightful discussions on fairness metrics. I would also like to thank the LSKI research group at the University of Mannheim for their

valuable feedback on several of the papers included in this thesis, which they have provided at regular research meetings and internal poster sessions.

And then there is the ”PhD student self-help group“ that met regularly to motivate each other with self-imposed thesis writing deadlines along with cookie penalties. Thank you to Nadine Auer, Veronika Schenk, and Korbinian Kuhn for helping me focus on thesis writing instead of yet another short paper.

For proofreadings and feedback on the thesis draft or parts of it I would like to thank Manuel Eberhardinger, Patrick Takenaka, Mike Preuss, Ronja Fuchs, Leonie Kallabis, and my dad. In addition, I thank Robin Schmöcker for his feedback on the mathematical notation.

And finally, I wish to thank my parents, my sister, my friends, and especially my girlfriend Natalie, for their emotional support over all the years.

Contents

List of publications

Most of the work presented in this thesis has been previously published in proceedings of international conferences and journals. This includes text, figures, and tables. We reference the corresponding publications in the respective chapters and list them here in inverse chronological order.

Florian Rupp and Kai Eckert. Level the Level: Balancing Game Levels for Asymmetric Player Archetypes With Reinforcement Learning. *Proceedings of the 20th International Conference on the Foundations of Digital Games (FDG)*, Graz, Austria, pp. 1–4, April 2025, doi: 10.1145/3723498.3723747.

Florian Rupp, Manuel Eberhardinger, and Kai Eckert. Simulation-Driven Balancing of Competitive Game Levels with Reinforcement Learning. *IEEE Transactions on Games (ToG)*, vol. 16, no. 4, pp. 903—913, December 2024. doi: 10.1109/TG.2024.3399536.

Florian Rupp and Kai Eckert. G-PCGRL: Procedural Graph Data Generation via Reinforcement Learning. *2024 IEEE Conference on Games (CoG)*, Milan, Italy, pp. 1–8, August 2024. doi: 10.1109/CoG60054.2024.10645633.

Florian Rupp, Alessandro Puddu, Christian Becker-Asano, and Kai Eckert. It might be balanced, but is it actually good? An Empirical Evaluation of Game Level Balancing. *2024 IEEE Conference on Games (CoG)*, Milan, Italy, pp. 1–4, August 2024. doi: 10.1109/CEC60901.2024.10612054.

Florian Rupp and Kai Eckert. GEEvo: Game Economy Generation and Balancing with Evolutionary Algorithms. *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, pp. 1-–8, June 2024. doi: 10.1109/CEC60901.2024.10612054.

Florian Rupp, Manuel Eberhardinger, and Kai Eckert. Balancing of Competitive two-player Game Levels with Reinforcement Learning. *2023 IEEE Conference on Games (CoG)*, Boston, USA, pp. 1–8, August 2023. doi: 10.1109/CoG57401.2023.10333248.

Appendix A contains an overview of the code and data published along with this thesis.

Moreover, the author participated in two doctoral consortiums, one at the beginning and one at the end of the thesis writing process. The following peer-reviewed publications were produced during the consortiums. We list them here in inverse chronological order.

Florian Rupp. Game Balancing via Procedural Content Generation and Simulations. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 21(1), Edmonton, Canada, pp. 442-445, November 2025. doi: 10.1609/aiide.v21i1.36856.

Florian Rupp, Learning the Generation of Balanced Game Levels. In *Proceedings of Doctoral Consortium at KI 2023*, Berlin, Germany, pp. 40–49, September 2023. doi: 10.18420/ki2023-dc-05.

In addition to the publications listed before, which are directly related to the content of this thesis, the author contributed to and published other research work during the course of his doctoral studies. We list them here in inverse chronological order.

Manuel Eberhardinger, **Florian Rupp**, Florian Richoux, Johannes Maucher and Setareh Maghsudi. Towards Cognitive-Plausible Explanations for Board Game Agents with Genetic Programming. *Proceedings of the 12th AAAI AIIDE Workshop on Experimental AI in Games (EXAG)*, Edmonton, Canada, November 2025 .

Jakob Kappenberger, Clara Strasser Ceballos, Frederic Gerdon, Daria Szafran, **Florian Rupp**, Kai Eckert, Heiner Stuckenschmidt, Ruben Bach, Frauke Kreuter, and Christoph Kern. Unintended Impacts of Automation for Integration? Simulating Integration Outcomes of Algorithm-Based Refugee Allocation in Germany. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 8(2), pp. 1375-1387. doi: 10.1609/aies.v8i2.36638 .

Johannes Bauer, Zheng Ji, **Florian Rupp**, Yavuz Caydamli, Klaus Heudorfer, Bruno Gompf, Stefan Carosella, Michael R. Buchmeiser, and Peter Middendorf. Optimizing Transparent Fiber-Reinforced Polymer Composites: A Machine Learning Approach to Material Selection. In *Composites Part B: Engineering* 307, p. 112799, 2025. doi: 10.1016/j.compositesb.2025.112799 .

Leonie Kallabis, Timo Bertram, and **Florian Rupp**. Deceptive Game Design? Investigating the Impact of Visual Card Style on Player Perception. *2025 IEEE Conference on Games (CoG)*, Lisbon, Portugal, pp. 1–8, August 2025. doi: 10.1109/CoG64752.2025.11114299.

Christoph Kern, Jakob Kappenberger, Frederic Gerdon, Clara Strasser Ceballos, Daria Szafran, **Florian Rupp**, and Ruben Bach. A Simulation Framework for Studying the Social Impacts of Algorithm-Based Refugee Matching. In: *Proceedings of Fourth European Workshop on Algorithmic Fairness (EWAF)*, Eindhoven, Netherlands, Vol. 294, pp. 487–491, 2025. Proceedings of Machine Learning Research. PMLR.

Manuel Eberhardinger, **Florian Rupp**, Johannes Maucher, and Setareh Maghsudi. Unveiling the Decision-Making Process in Reinforcement Learning with Genetic Programming. In *Advances in Swarm Intelligence*, pp. 349—365, Singapore, 2024. Springer Nature. doi: 10.1007/978-981-97-7181-3_28.

Florian Rupp, Benjamin Schnabel, and Kai Eckert. Implementing Data Workflows and Data Model Extensions with RDF-star. *The Electronic Library*, vol. 42, no. 3, pp. 393–412, 2024, Publisher: Emerald Publishing Limited, doi: 10.1108/EL-04-2023-0102.

Florian Rupp, Benjamin Schnabel, and Kai Eckert. Easy and complex: New Perspectives for Metadata Modeling using RDF-Star and Named Graphs. In *Knowledge Graphs and Semantic Web*, Cham: Springer International Publishing, 2022, pp. 246—262. doi: 10.1007/978-3-031-21422-6_18.

List of Figures

List of Tables

List of Acronyms

ABM Agent-based Modeling. 14, 15, 29, 53, 54
AI Artificial Intelligence. 11, 29, 86
ARLPCG Adversarial Reinforcement Learning for Procedural Content Generation. 28

CNN Convolutional Neuronal Network. 26, 27
CPU Central Processing Unit. 58

DDA Dynamic Difficulty Adjustment. 31

EDRL Experience-Driven Procedural Content Generation via Reinforcement Learning. 28

G-PCGRL Graph Procedural Content Generation via Reinforcement Learning. 9, 25, 26, 46, 95–99, 101, 103–106, 108–110, 112, 120, 124, 125, 128, 131, 133, 140
GAN Generative Adversarial Network. 26–28
GEEvo Game Economy Evolution. 9, 25, 29, 47, 95–98, 102, 103, 110, 111, 115, 116, 119, 120, 124–128, 131, 133
GNN Graph Neuronal Network. 98
GPU Graphics Processing Unit. 28

IGD Internet Gaming Disorder. 130

LLM Large Language Model. 24, 26–28, 45, 131

MCTS Monte Carlo Tree Search. 25, 32, 38
MDP Markov Decision Process. 8, 9, 18, 19, 27, 28, 34, 44, 49, 51, 54, 56, 59, 68, 69, 71–73, 91, 92, 95, 96, 98–100, 120–122, 124, 126, 128
MOBA Multiplayer Online Battle Arena. 81, 115

NMMO Neural Massively Multiplayer Online. 10, 42, 43, 55, 125

PCG Procedural Content Generation. 2, 4, 7–15, 18, 21, 23–29, 32–34, 41–45, 51–54, 69, 78–80, 82, 90, 92, 95–99, 105, 109–111, 119–122, 124–128, 130, 131

PCGML Procedural Content Generation via Machine Learning. 26, 27

PCGRL Procedural Content Generation via Reinforcement Learning. 8, 9, 27, 28, 34, 40, 43–45, 51–54, 56–59, 64, 68–75, 79–84, 86, 88–92, 96, 98–101, 103–105, 108, 109, 120, 122–124, 126, 128, 131, 133

PCGRLLM Procedural Content Generation via Reinforcement Learning and Large Language Models. 28, 131

PPO Proximal Policy Optimization. 20, 21, 54, 57, 58, 83, 88, 104, 131

PvE Player versus Environment. 29, 31

PvP Player versus Player. 31

RL Reinforcement Learning. 8–11, 14, 15, 18–20, 25–30, 34, 38–40, 44, 45, 51, 53–56, 66, 69–71, 73–75, 77–79, 84–86, 88, 95–99, 101–103, 105, 108–110, 120–124, 126, 128, 131

RTS Real-Time Strategy. 30–32

TRPO Trust Region Policy Optimization. 20

WFC Wave Function Collapse. 26

WHO World Health Organization. 130

Mathematical Notation

Throughout this work we adhere to the following notation.

Sets and Set Theory

$\mathbb{R}$	The set of real numbers
$[0, 1]$	The closed interval represents the set of all real numbers x such that $0 \leq x \leq 1$
$\mathcal{A}$	The set $\mathcal{A}$
$\lvert\mathcal{A}\rvert$	The size (cardinality) of set $\mathcal{A}$
$a \in \mathcal{A}$	Element a belongs to set $\mathcal{A}$
$\mathcal{A} \times \mathcal{B}$	The cartesian product of $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \cup \mathcal{B}$	The union of all elements of $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \subset \mathcal{B}$	$\mathcal{A}$ is a proper subset of $\mathcal{B}$, with $\mathcal{A} \neq \mathcal{B}$
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ is a subset of $\mathcal{B}$, allowing $\mathcal{A} = \mathcal{B}$

Matrix and Linear Algebra Notation

a	The variable a
α	The constant α
$\mathbf{A}$	The matrix $\mathbf{A}$
$\mathbf{A} \in \mathbb{R}^{m \times n}$	The matrix $\mathbf{A}$ of size $m \times n$ of real numbers
a_{ij}	The element at position i, j of matrix $\mathbf{A}$
$\mathbf{A}^T$	The transpose of a matrix $\mathbf{A}$
$\mathbf{A}'$	A derived form of matrix $\mathbf{A}$
$\lvert\lvert\mathbf{A}\rvert\rvert_F$	The Frobenius norm of a matrix $\mathbf{A}$

Logic and Quantifiers

$\forall\, x$	Universal quantifier, means "for all x"

Probability and Random Variables

A	The random variable A
$\hat{Y}$	The predicted value of Y
$P(A)$	The probability distribution over the random variable A
$P(A \mid B)$	The conditional probability distribution over A given B
$a \sim d$	Variable a is drawn from distribution d
$a \sim d(\cdot \mid p)$	Variable a is drawn from distribution d conditioned on parameter(s) p
$\overline{a}$	Average (arithmetic mean) of a

Functions and Operators

$f(\cdot)$	The function f
$f : \mathcal{A} \to \mathcal{B}$	Function f mapping set $\mathcal{A}$ to set $\mathcal{B}$
ΔX	The relative delta value of X
$\lceil a \rceil$	Ceiling of variable a to the next integer $\geq a$

Chapter 1

Introduction

Life is never fair. And perhaps it is a good thing for most of us that it is not.
Oscar Wilde, *An Ideal Husband*, Act II, 1899

We all play games[1]. The history of games dates back to the ancient past of humanity, as an integral part of many cultures and a form of social human interaction [Crist, 2019]. Psychologists and behavior scientists agree on the importance of play, starting in early childhood, to foster the development of social cognition [Piaget, 1951; Vygotsky, 1978]. Erikson [1977] shows that play allows children to experiment with social experiences and simulate consequences. Gottman [1986] shows how children use play for emotional mastery. Playing games, however, is not limited to children; play theorist Brian Sutton-Smith [1997] argues in his work *The Ambiguity of Play* that play serves the same purposes for adults as it does for children. The survey by Granic et al. [2014] highlights the positive effects of playing video games on various aspects, such as social, cognitive, motivational, and emotional skills. In addition, playing games engages various cognitive abilities [Togelius, 2019, p. 19] as defined in psychology by the Cattell-Horn-Caroll theory [Schneider and McGrew, 2012], which models intelligence as a hierarchy of different abilities.

Games have also contributed to scientific progress throughout the past directly, i.e., through the use of games to crowdsource information on protein folding [Cooper et al., 2010], which played a role in the scientific developments recognized by the 2024 Nobel Prize in Chemistry[2], or the breakthroughs in solving chess (Deep Blue, [Campbell et al., 2002]) or Go (AlphaGo, [Silver et al., 2018]) that have been used as benchmarks for progress in artificial intelligence. Moreover, playing games lies at the heart of society: According to the recent Global Games Market Report by Newzoo [2024], the number of video game players worldwide is estimated at 3.42 billion, and the industry generates revenues of USD 187.7 billion. Both figures have grown steadily over the past decades and are expected to continue to grow in the future.

"Game designers are wizards of engagement" [Granic et al., 2014, p.70]. However, to ensure this engagement, a game must be designed to appear fair

[1]The term *game* in this thesis refers to games for entertainment. We will define this term in more detail in the subsequent Section 1.1.

[2]Nature Portfolio: Nobel Price in Chemistry 2024 – Protein Structure Prediction. `https://www.nature.com/collections/edjcfdihdi`

and balanced for *all* players. According to the theory of the psychologist Csikszentmihalyi [1990], humans only stay in the *flow* (here: engaged) while solving a problem if the current challenge is appropriate. If the challenge is too low, humans (in this context: players) feel bored, whereas the opposite results in anxiety. This is important when designing multiplayer games, and particularly when designing *competitive* multiplayer games. A game which does not appear to be fair and balanced for all players will lead to frustration or boredom and players will quit playing [Becker and Görlich, 2020]. When creating new games, ensuring balance is, however, a challenging and time-consuming task [Adams, 2014; Schreiber and Romero, 2021]. For these reasons, this thesis explores methods to automate and support the process of game balancing.

Before defining our research questions and listing our contributions, we first clarify the definition of games for entertainment and balance.

1.1 Games and Balance

This thesis explores methods to automate game balancing. To lay the ground for our research, this section provides the definition of what constitutes a game in the context of this thesis, along with an overview of what is understood as balancing a game and which challenges it entails.

To illustrate theoretical concepts, we make use of the popular strategy board game *The Settlers of Catan (Catan)* [1995] by Klaus Teuber as a running example. In *Catan*, players collect and trade resources, such as wood and grain, to build roads, settlements, and cities. This allows them to expand their territory and earn victory points. It is a fitting example, as it is well-known and not simplistic. Catan also encompasses multiple aspects that we aim to highlight, such as balancing concepts in general up to Procedural Content Generation (PCG). In addition, in literature it is also referenced to explain game-theoretic concepts, for instance by Schreiber and Romero [2021].

We begin by defining what constitutes a *game* (Section 1.1.1) and clarifying the notion of *balancing* in this context (Section 1.1.2).

1.1.1 What is a Game?

The term *game* originates from Germanic and Old English (5th to 11th century), referring to ”an activity [...] or having the form of a contest or competition, governed by rules of play, according to which victory or success may be achieved through skill, strength, or good luck“ [Oxford English Dictionary, 2025].

From the perspective of game design, Salen and Zimmerman [2003] define a game based on the definition of Avedon and Sutton-Smith [1971] as follows: "A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome" [Salen and Zimmerman, 2003, p.11]. Based on their foundational book *Rules of Play: Game Design Fundamentals* [Salen and Zimmerman, 2003] we describe this quote in more detail:

- A game is a *system*: All games can be understood as systems that encapsulate several components. All components such as rules, players, and goals

can interact in complex ways. A key challenge is to keep the overall system balanced and fair to ensure an engaging player experience. In the example of *Catan*, the game system comprises the interaction between the rules, the economic model, the procedurally generated board, and the players.

- A game has *players*: One or more participants can actively participate and interact with the game's system. In *Catan*, three or four players are required for playing.
- A game embodies a *conflict*: All games involve a form of conflict that motivates the players to engage with the game in order to resolve it. This can range from cooperation to competition, solo challenges against a game system, or, for instance, multiplayer social conflicts. *Catan* is a competitive multiplayer game with a conflict of territory and limited availability of resources, forcing players to make strategic decisions on how to spend them. Interdependence and competition is additionally increased as players typically cannot produce all resource types by themselves.
- A game has *rules*: Rules provide the framework for play by defining what players can and cannot do. In *Catan*, the rules define how players interact with the game environment and with each other. They also specify the turn structure and the set of available actions.
- A game is *artificial*: Games exist intersubjectively in human minds, distinct from real life. All components of a game such as its system, rules, or conflicts are human-created. The same is true for *Catan*, where the concept of winning or generating resources has no meaning outside the context of play.
- A game has a *quantifiable outcome*: The quantifiable outcome of games distinguishes them from less structured play activities. Through winning or losing, a player receives an unambiguous evaluation based on their interactions with the game system. Alternatively, a numerical score can be assigned, allowing comparison between players. This is especially true in *Catan*, where players receive victory points. The player who reaches ten points first wins the game.

So when we use the term *game* in this thesis, we mean a game made for human entertainment. This differs from the definition of games in other disciplines and the term should therefore not be confused here, e.g., with game theory [von Neumann and Morgenstern, 1944] in mathematics. Although both treat games as rule-governed systems that define possible actions for multiple agents through formal rule sets, the primary focus of games in game theory is on abstract models of strategic interaction. These models are sometimes illustrated with simple narratives, such as the Prisoner's Dilemma [Rapoport, 1989]. However, the narrative is not an integral component of the game itself. Conversely, games for human entertainment directly integrate narratives, media elements such as visual aesthetics or sound, and most importantly, the player experience as core parts of the system. While both share a definition of outcomes, games for entertainment employ a broader definition, encompassing win conditions or narrative

progress. In game theory, outcomes are formalized as numerical payoffs, utilities, or equilibrium states. Another point of comparison is economic systems, which we will also address in Chapter 7. In entertainment games, their primary objective is to engage, incentivize, and reward players. By contrast, in game theory, they serve to model and predict rational behavior.

1.1.2 Game Balancing

Since games are systems composed of multiple components [Salen and Zimmerman, 2003], these must be coordinated to create a balanced experience, ensuring all players have an equal initial chance of winning. It should be noted that game balancing is primarily a practice rather than a formal academic field. The emerging niche field to which we contribute is automating game balance using various optimization strategies, with a particular focus on approaches based on PCG. We provide an overview of this field in the related work in Chapter 3. A foundational approach on game balancing for practitioners is presented by Schreiber and Romero [2021] in the book *Game Balance*, to which we refer to multiple times in this thesis. While game balancing is defined through multiple definitions, Becker and Görlich [2020] conclude in their survey that "no two authors share identical understandings" [Becker and Görlich, 2020, p.1] on what game balancing actually is. They continue to explain that game balancing must be tailored to the game's genre, it's competitive or cooperative system, as well as the specific design and goals of the game itself. Ensuring balance is, however, most important for competitive multiplayer games. But according to Becker and Görlich [2020], related works share a common denominator on the following points as well: difficulty, symmetry, the definition of good balancing, and the balancing process.

Difficulty refers to fine-tuning the level of challenge so that the game remains engaging without becoming too easy or too hard to beat. Alongside a user study, Klarkowski et al. [2016] show that the balance of challenge in a game, in combination with a player's skill level, has a further impact on their experience. In competitive multiplayer games like *Catan*, difficulty is mainly influenced by the other players' skill level. Difficulty balance is supported through limited forms of interaction, such as structured trading.

Symmetry refers to the design of the game's system and can tackle various facets, such as the design of the rules, players, or the game world (level). In a symmetrical game setup, all players have access to the identical setup of the same elements. This makes the game balanced in terms of providing equal opportunities for all players. *Catan* is designed to be entirely symmetrical. All rules apply equally to all players, and each player can choose from the same set of actions on their turn. Many modern games (e.g., Scythe [2016], League of Legends [2009], Starcraft 2 [2010], only to name a few) also include asymmetric elements in order to increase replayability and the overall entertainment. Due to the increased number of possible game states, this is, however, much harder to balance. In this thesis, we will address this challenge from the level, player archetype, and game economy perspectives. While *Catan* features a symmetrical design, players take turns placing their starting settlements which leads to asymmetries in resource availability. From a single player's perspective, this

may seem imbalanced; however, since most players lack one or more key resources, the game remains balanced overall.

A *well-balanced* game provides players with meaningful decisions with which they actively can influence their chances of winning or losing, so that that the better player always wins. In addition, there must not be a single dominant strategy to winning the game, but multiple strategies must be viable. *Catan* addresses this, by offering several ways to earn victory points. According to Becker and Görlich [2020] many authors agree that equal win rates in multiplayer games indicate balance; however, additional factors may need consideration depending on the context, such as how the players are matched according to their skill. In this thesis, we stick to this definition of balance and combine it with an existing metric in Section 4.2.2. Depending on the specific context, slight imbalances may, however, be accepted. We will discuss this in the context of this thesis in Section 4.1. The *process of balancing* a game is an iterative process involving several steps and tools. This includes, user data analysis, spread sheets, and mathematical formulas, among others. Since games are designed for humans, playtesting with the target player group is essential, followed by thorough analysis and iterative refinement.

Nevertheless, balancing a game remains a challenging, time-consuming, and often tedious task that requires a lot of manual effort. This thesis sets out several approaches to automating game balancing and addressing these issues. Nevertheless, some form of human evaluation is still necessary.

1.1.3 Game Economies

In Chapter 7 we will approach automated game balancing from the perspective of game economies. Here we provide a brief explanation of what a game economy constitutes and why it is important to consider in the context of game balance. This section is based on the definition of game economies in the foundational book *Game Balance* by Schreiber and Romero [2021].

A game's economy is a macro system that defines how virtual resources are created and transitioned to other resources within a game. As such it is a powerful tool that shapes player experience and progression speed, and, in turn, the overall balance of the game. Virtual resources are not limited to virtual currencies, as seen in games like Monopoly [1935], but extend much further – ranging from a player's time investment (e.g., a turn), to game-specific resources like mana, or tangible materials such as ore in a crafting systems. From an economy-balancing perspective, the narrative context of resources can be omitted to shift the focus to when and how specific resource trade-offs impact the game. Modern game economies are becoming increasingly complex and can be very sensitive to even minor numerical adjustments, which may have an unexpected impact on the overall gaming experience. Game economies are inspired by financial economic systems, and commonly present players with strategic trade-off decisions – for example, whether to invest resources for immediate early game power, or to allocate them differently for greater, but delayed, advantages in the late game.

Continuing with our running example, *Catan* implements an economic system in which players influence their access to virtual resources by building new settlements or upgrading them to cities. The game rules determine construction

costs. Thus, the game's economy incentivizes players to spend resources to increase their chances of gaining more resources in the future. To some extent, this simulates a financial economic systems. Game economic systems may implement so called *positive feedback loops*, in which *more* resources will sooner or later lead to *more* resources. This is a common and, from a game design perspective, highly motivating pattern that is frequently applied. While positive feedback loops can be an effective instrument for creating engagement from a game design perspective, they are difficult to balance from a game balancing standpoint. Due to the snowballing effect caused by the exponential creation of resources, even marginal changes can lead to drastic imbalances. To counteract snowballing, games can implement mechanisms to destroy virtual resources without any equivalent value. In *Catan* this is implemented by a rule that forces players to discard half of their resources if they hold more than seven when a seven is rolled. In addition, it is crucial to set appropriate building costs in Catan. If cities are too cheap to build, the game may snowball too quickly. If it is too expensive, progression becomes sluggish. This *progression* can be balanced through the game's economy and plays a key role in shaping the overall feel of the game [Schreiber and Romero, 2021]. In *Catan*, players begin with just two settlements and little resources. As the game unfolds, they experience a sense of growth by increasing their resource income and expanding their network of buildings. Furthermore, Catan features a trading system where supply and demand, combined with player negotiation, determine resource value. Additional balance emerges as players often refuse to trade with whoever is in the lead.

In Chapter 7 we explore ways to model a game economy as a graph inspired from previous work by Klint and van Rozen [2013] in combination with simulations and automated balancing.

1.2 Research Questions

In the previous section we have seen that game balancing is multi-faceted [Adams, 2014; Schreiber and Romero, 2021]. Based on related work and related foundational books, we identify game level and economy design as two crucial facets to target for automated game balancing. Therefore, we define four research questions (RQs) for this thesis, which we will explore and investigate in order to answer them thoroughly:

RQ 1 **Quantification of balance:** *What is a reliable, data-driven foundation that enables automated, game-independent measurement and quantification of balance in competitive, two-player game levels?*

In order to pursue automated game balancing methods, it is first necessary to develop a foundational method for determining balance that can express it in a form suitable for algorithmic processing. Since the balance of a game level depends on various aspects such as asymmetries in the level or character design, probabilistic elements, or simply the skill of the players, finding a robust and reliable metric to express the balance of a game level is crucial. Games inspire with their uniqueness,

therefore it is an additional challenge to develop this method independently of a particular game. Related works (e.g., Lanzi et al. [2014] and Lara-Cabrera et al. [2014]) incorporate domain-dependent information for automated balance estimation, but this makes them difficult or impossible to transfer it to another game.

RQ 2 **Accelerating automated level balancing:** *How can automated game level balancing be accelerated while maintaining content quality, diversity, and asymmetries?*

Balancing game levels is a time-consuming task involving a lot of manual work and playtesting [Schreiber and Romero, 2021; Becker and Görlich, 2020]. Moreover, as we will discover when quantifying balance (RQ 1), collecting information on the balancing state of a game automated is computationally intensive, but PCG methods should be fast and reliable [Togelius et al., 2011a]. Therefore, we aim to develop an approach to automatically generate balanced two-player game levels, with the overall goal of reducing the computational effort while still generating valid and diverse levels. In addition, we aim to explore other use cases for level balancing, such as asymmetric player setups or an "imbalancing" to favor a specific player.

RQ 3 **Human perception of balance:** *What is the human perception and evaluation of the artificially quantified balance of game levels through simulations?*

We aim to find a method to automatically determine the balance of a game level in a robust and quantified way (RQ 1), and to develop a method to accelerate balanced level generation based on this metric (RQ 2). Games are, however, made for human players, each of whom perceives balance from their own subjective perspective. For this reason, it is essential to investigate how human playtesters perceive the automatically created balance. Due to the perception of a level's balance involves also subjectivity based on the player's skill and experience for instance, it is an additional challenge how to ask people to express their opinions in an unbiased but accurate way.

RQ 4 **Automated game economy generation and balancing:** *How can automation techniques be integrated into the generation and balancing of game economies?*

With RQ 1–3, we explore automated game balancing on the game level facet. In addition to game levels, game economies are a powerful system for shaping the player experience while maintaining overall balance on a macro level. Moreover, they can be highly sensitive to small changes in their configuration, requiring a lot of manual work in fine-tuning. By transferring and adapting findings from RQ 1 and RQ 2, we aim to develop methods for the automated generation and balancing in the domain of game economies.

1.3 Contributions

In order to answer the research questions presented, we make the following contributions (C). The research question to which the contribution refers and the chapter in which it is included are also indicated.

C 1 A data-driven foundation to express the initial balance state of a competitive, two-player level.
Introduced in Chapter 4, applied in Chapters 5 and 6, RQ 1

To approach this question we first provide an in-depth discussion of what balance in a game actually means and how this concept differs from fairness. We conclude that, to isolate skill from randomness, a game should be balanced for all players in terms of an equilibrium of win rates, but must not be designed to be fair.

To quantify balance we transfer the Statistical Parity metric [Dwork et al., 2012] from the fair machine learning community to express the balance of a competitive two-player game. By only taking into account, how often each player wins in multiple simulations, our metric is independent of the game itself. While the Statistical Parity ensures equal probability between two groups, our method extends this concept by mapping it onto a numerical scale in the interval $[0, 1]$, where 0.5 indicates perfect balance, and 0 and 1 indicate maximal imbalance. This representation makes it well-suited for algorithmic processing. Since games often include probabilistic elements, simulations with the same configuration may have different outcomes. In order to ensure robust balance estimation, we propose a method for determining the minimum number of simulations required to obtain stable results.

C 2 An architecture that formulates level balancing as both a Markov decision process and a PCG problem. *Chapters 5 and 6, RQ 2*

Since game simulations to estimate the balance are computationally intensive (C 1), we frame tile-based game level balancing as a Markov Decision Process (MDP) in order to apply Reinforcement Learning (RL) to learn a trajectory of modifications that most influences the level balance. Therefore, we use, adapt, and extend the PCGRL (PCG via RL) framework by Khalifa et al. [2020] and propose new representations of the action space, in which the agent swaps two tiles at each time step.

Results show that our method can balance levels in fewer steps and with a better overall accuracy compared to other search-based approaches and the original PCGRL. This reduces the computational effort by avoiding unnecessary simulations and contributes to accelerating the generation. Moreover, we examine difference applications of the proposed method and show that it can also be used to generate balanced levels for asymmetric player archetypes, levels with a particular imbalance for a specific player, and its transferability to a different environment. We further op-

timize the action and observation space to improve the method's general accuracy.

C 3 The empirical evaluation of the artificially simulated and generated balance with human playtesters. *Chapter 5, RQ 3*

In an empirical study with human playtesters, we show with descriptive analysis and prove by hypothesis testing, that the balance determined in C 1 and used for accelerated level generation in C 2, is also perceived by humans in the indented way in most scenarios. Since game balance is multi-faceted and its perception is also partly subjective, we use a comparative survey design letting participants play and compare level versions before and after our method (C 2) has been applied.

C 4 G-PCGRL: A Markov decision process to generate graph data via reinforcement learning. *Chapter 7, RQ 4*

With G-PCGRL (Graph PCG via RL), we adapt the MDP of the PCGRL framework by Khalifa et al. [2020] in combination with our prior results from accelerated level balancing (C 2) to generate graph data through manipulating a graph's adjacency matrix according to a set of constraints. In comparison to other search-based methods, our approach is fast at inference. Moreover, G-PCGRL is controllable in terms of graph size and node types. We show the feasibility of our method with several different sets of constraints, however, we report a limited scalability when the graph size increases. We will address this limitation in our contribution with the GEEvo (Game Economy Evolution) framework (C 5).

C 5 GEEvo: A framework for the generation and balancing of graph-based game economies with evolutionary algorithms. *Chapter 7, RQ 4*

With GEEvo (Game Economy Evolution) we introduce a framework with two evolutionary algorithms to controllably generate and balance graph-based game economies. GEEvo comes with a lightweight game economy simulation framework to, like in previous contributions, simulate results in order to estimate the balance. To represent game economies as graph, we follow the formulation by Klint and van Rozen [2013]. Our evaluation shows that the method can balance arbitrarily configured economy graphs in most cases. Furthermore, it is capable of balancing two economies simultaneously, as demonstrated in our case study. In addition, GEEvo addresses G-PCGRL's limitation in scalability for increased graph sizes, however, in exchange for an increased runtime.

C 6 *Feast & Forage*: An environment for research on automated game balancing. *Introduced in Chapter 4, applied in Chapters 5 and 6, RQ 2 and 3*

Along with this thesis, we identified a lack of research environments for balancing games entirely through PCG. Therefore, we developed an en-

vironment inspired by the NMMO (Neural Massively Multiplayer Online) [Suarez et al., 2019] environment to apply, explore, test, and evaluate our method. While NMMO is designed for intelligent multi-agent research with up to 100 agents, *Feast & Forage* is made for research on generating small, asymmetric puzzle levels to be balanced for two players entirely through the spatial placement of tiles. Additionally, we define the win conditions differently in order to fit the context. Furthermore, we provide a playable prototype of the game for human playtesting, which was used for the empirical evaluation in contribution C 3. With easily adaptable and extensible rules as well as agent heuristics, it can serve as a versatile testbed for future research.

1.4 Thesis Structure

In general, this thesis is structured around the four main Chapters 4–7, which address the previously defined research questions from Section 1.2 and thoroughly describe our contributions, results, and findings (cf. Section 1.3). An introduction to games and balancing has been given in Section 1.1.

First, we introduce the relevant theoretical background consisting of a definition of PCG, agent-based simulations, search-based optimization, and RL in Chapter 2. Second, we discuss the related work on PCG and automated game balancing which is relevant for the entire thesis in Chapter 3. If a chapter requires a more focused review of related work or addresses a scope not relevant to the overall thesis, we include it directly within the respective chapter.

In Chapter 4, we introduce and define foundational concepts which are used throughout the thesis, such as quantifying balance automatically (RQ 1) and defining the environment *Feast & Forage*. Chapter 5 contributes methods and findings in order to answer RQ 2 and RQ 3, focusing on automated game balancing on the level facet in combination with an empirical evaluation. Chapter 6 extends and adapts the findings of Chapter 5 by providing additional insights and applications, such as asymmetries, imbalances, and transferability. These insights further support and strengthen our conclusions in the context of RQ 1 and RQ 2. Chapter 7 targets automated balancing on a second facet by transferring findings from Chapters 5 and 6 to generate and balance graph-based game economies in order to answer RQ 4. We discuss our contributions in a dedicated section in each main chapter.

Finally, Chapter 8 presents our conclusion and the implications of this thesis (Section 8.2). We also discuss ethical considerations in automated game balancing (Section 8.4), outline limitations (Section 8.3) of this thesis, and suggest directions for future work (Section 8.5).

Chapter 2

Theoretical Background

This chapter provides the theoretical foundation for the thesis by introducing key concepts used throughout. Section 2.1 introduces and defines Procedural Content Generation (PCG) in games in general, whereas Section 3.1 in the following chapter presents related work on PCG for this thesis. We then introduce agent-based modeling (ABM) in Section 2.2, which serves as the simulation framework to estimate a game's balance in Chapters 5 and 6. Finally, we describe several search-based optimization methods used in this thesis in Section 2.3, as well as Reinforcement Learning (RL) in Section 2.4. For better understanding and readability, the definition of what a game constitutes and its balancing has been given in the introduction in Section 1.1. Where applicable, we continue to use the running example of the game *Catan*.

2.1 Procedural Content Generation for Games

PCG [Shaker et al., 2016] is an active area of research spanning several computer science disciplines and is a subarea of game Artificial Intelligence (AI). This section is based on the foundational book *Artificial Intelligence and Games* by Yannakakis and Togelius [2025a], in particular the chapter on PCG [Yannakakis and Togelius, 2025b], and follows the proposed taxonomy. Before reporting on specific related work, we give a brief definition of what PCG actually means in the context of games and particularly this thesis, and why it is so relevant to games. While our focus is on PCG for games, it is not limited to games and can also be found in other domains, such as computer graphic [Merrell and Manocha, 2011], chemistry [Gómez-Bombarelli et al., 2018], or music [Hoover et al., 2011].

What is PCG? In the context of games, Togelius et al. define it as: "PCG is the algorithmical creation of game content with limited or indirect user input" [Togelius et al., 2011a, p.6]. According to the taxonomy by Yannakakis and Togelius [2025b], PCG approaches can be structured around the terms *content*, *methods*, and their *role*, each of which can be further divided into binary aspects. Content refers to the generated outcome and can be distinguished by its type, functionality, and spatiality. The *content* type can be necessary or optional, depending on whether it is required to complete the game. Its functionality distinguishes decorative content which is purely aesthetic and does not require any interaction,

from functional content. In terms of spatiality, methods either generate spatial structures such as 2D or 3D levels, maps, or objects, or non-spatial elements such as stories or narratives.

A PCG *method* can be further classified by its determinism, controllability, testing, learning, and iterativity. Methods are either deterministic or stochastic: the former can be seen as data compression, while the latter offers great variation. Controllability refers to whether parameters can control the output, a desirable property [Togelius et al., 2011b]. While respect to testing, constructive methods generate content once, whereas generate-and-test methods test the output and regenerate it, or parts of it, if constraints are violated. Methods can be authored or trained, depending on whether they rely on handcrafted rules or incorporate a model that can be learned via machine learning for instance. Finally, methods generate content iteratively in multiple steps or single pass.

In terms of *roles*, PCG approaches can be classified by temporality, autonomy, and adaptivity. Temporality distinguishes between generated at runtime and content created offline in advance. Autonomy refers to whether content is generated fully autonomously or support designers in a mixed-initiative setting. Adaptivity captures experience-driven methods that incorporate a player (experience) model or interact with players to evaluate the generated content, whereas experience-agnostic methods do not include players in any way. This distinction is, however, often blurred in practice.

For this thesis, we formally define a deterministic procedural content generator PCG_{gen}^{det} as:

$$PCG_{gen}^{det} := \mathcal{P} \times \{\mathcal{C}\} \rightarrow \mathcal{O}. \tag{2.1}$$

The generator outputs the content set $\mathcal{O}$ of all possible combinations of a set of input parameters $\mathcal{P}$ and the entire set of constraints $\mathcal{C}$. $\mathcal{O}$ in PCG refers to the contents of a game, such as levels, textures, narratives, quests, sounds and music, or even rules. In contrast, it does not include the game's program code itself or the engine [Yannakakis and Togelius, 2025b]. Conversely, stochastic content generator include probabilistic in order to increase the content diversity. In this work, we will introduce several of such generators and formally define them as:

$$PCG_{gen}^{stoch} := \mathcal{P} \times \{\mathcal{C}\} \rightarrow \mathcal{D}(\mathcal{O}), \tag{2.2}$$

where $\mathcal{D}$ is a set of probability distributions. The content o is then sampled from a selected probability distribution $d \in \mathcal{D}$:

$$o \sim d. \tag{2.3}$$

Catan also implements PCG by combining parameters and constraints with stochasticity in the distribution of hex tiles during setup. This can be understood as sampling from the distribution of all possible orders of tiles d, which has been predefined by the game designer, for example by specifying the number and types of tiles ($\mathcal{P}$) and the shape how the tiles are to be laid out ($\mathcal{C}$). The complexity of the content generation is further enhanced by additionally assigning probability values to each tile through dice number tokens.

Controllability refers to allowing the PCG method to be parameterizable via an input parameter $p \in \mathcal{P}$. This can allow a configuration as an input to influence the content, for instance based on the player's previous behavior or the

game designer's ideas. We define the controllable generation from a stochastic generator as:

$$o \sim d(\cdot \mid p). \tag{2.4}$$

Why do we need PCG? A driving incentive to procedurally generate content at runtime was the limited hardware resources for storing the game content in the early days of video games in the 1980's [Shaker et al., 2016; Yannakakis and Togelius, 2025b]. A popular example is the dungeon crawler game Rogue [1980], in which the levels are generated from scratch each time the game is started. The game's popularity led to the creation of the *Rogue-like* game genre, characterized by a heavy reliance on procedurally generated content. Another example is the game Elites [1984], which was really only about saving disk space by not allowing any variations in the generated content, so only $\mathcal{P}$ and $\mathcal{C}$ had to be stored. This is a deterministic PCG approach as defined in Equation 2.1.

PCG quickly caught the attention of the commercial gaming industry and is now present in many popular games across a range of genres, such as Diablo III [2012], Minecraft [2011] and Factorio [2020], to name but a few. Once motivated by limited disk space, PCG is now motivated by other reasons. One reason, of course, is the automation of human manual labor in order to reduce costs in the development process. Another reason is that PCG adds content diversity to a game which enhances the replay value. This is particularly true for *Catan*, where PCG varies the probability and spatial arrangement of resource tiles, making resource rarity different in each game. In addition, with adaptive PCG, content can be tailored to players' needs, for instance, it can be created based on previously played games to dynamically adjust the difficulty to a player's skill level. This is also known as dynamic difficulty adjustment. Finally, PCG can help designers to be not only more productive, but also to be more creative. PCG as an assistive toolbox can therefore be a valuable support within the process of design and development [Shaker et al., 2016; Yannakakis and Togelius, 2025b].

Moreover, when the problem of creating content is shifted to the metaproblem of designing systems that generate content, we gain deeper insight into the fundamental nature of the content [De Kegel and Haahr, 2020].

What are desirable characteristics of a PCG method? Which characteristics in particular are desirable or required is highly dependent on the game and the context in which a PCG method is used. Shaker et al. [2016] define the five characteristics speed, reliability, controllability, diversity and believability. Generation *speed* is particularly important when content is generated online while the player is playing, whereas it is only desirable in the game production process. Additionally, a PCG method must be *reliable* in terms of the validity, ergo playability, of the generated content. *Controllability* adds the ability to control the generated content with parameters from a set $\mathcal{P}$. This is particularly useful when player adapted content or a level with a specific difficulty is to be generated. A PCG method that always generates the same or mostly the same content is also not a desirable solution. For this reason, a PCG method should be able to generate *diverse* content. Finally, the generated content must be *believable* in the sense that players do not recognize it as artificially generated content. There-

fore, the PCG method should avoid revealing obvious patterns in the generated content, or otherwise players may lose interest in the game.

2.2 Agent-based Modeling and Simulations

Agent-based Modeling (ABM) is a computational approach to estimate the dynamic of a system by simulating agents that interact with an environment and with each other. To estimate a level's balance state we will use agent-based simulations in the Chapters 5 and 6. In this section we therefore give an overview of the theoretical foundations and explain advantages of this modeling approach. We base this section on the foundational books by Railsback and Grimm [2019], tailored to a mathematical and computer science audience, and Gilbert [2008], approaches ABMs from a computational social sciences perspective.

In contrast to expressing the whole system in terms of equations, simulating the actions and interactions of agents is beneficial for studying complex phenomena that arise from local interactions with the environment, such as in financial markets, social- or ecosystems, but also in games. To better understand the concepts of an ABM, we will break this down and describe and define the terms *agent*, *environment*, and *simulation*.

Agent: From the pure theoretic view, agents are abstract entities which act autonomously within an environment when simulating. Given a specific context they can represent barely anything from stock traders, animals, or like in this thesis, players. Each agent has a unique state, such as its position in the environment or other defining attributes. These attributes can be static or can change dynamically when the agent interacts with the environment or other agents. An agent can interact with the environment or other agents based on its internal heuristics or rules, but also learning algorithmic approaches such as RL are possible.

Agents can perceive the environment and other agents. A key distinction lies in the type of observability: in fully observable settings, all information is always available to the agent, while in partially observable settings, only a subset is accessible. Partial observability may depend on the agent's state, such as its position, limiting its perception to nearby agents.

Interactions may not only be with the environment, but also with other agents in order to communicate, cooperate, coordinate, or to compete. It must be further distinguished between a direct interaction, such as fighting or trading, and indirect, such as modifications of a shared environment. In Chapter 5 and 6 for instance, agents compete indirectly in a fully observable shared environment.

Environment The environment creates the space or context of the ABM's system in which the agents operate. It can be abstract or highly tailored to a specific use case, described by rules or a concise grid or network-based format for instance. Environments that model such a geographical space, as in our case game levels, are called spatially explicit. Internal constraints of the environment influence the agent behavior via shared resources or locations, but also global con-

straints can affect the dynamics of all agents simultaneously. The environment can be passive, merely reacting to agent interactions and providing information, or active, dynamically altering its state to influence agent behavior.

Simulation A simulation in the context of ABM is the process of executing the computer-based model defined by the environment and its agent(s). The execution is mostly done in discrete time steps, where the internal rules and constraints of both the agent(s) and the environment are executed. In order to observe how the system behaves as a whole, internal state representations of both the environment and the agent(s) are tracked at each time step. In this way, it is possible to analyze system behaviors that would be difficult or impossible to analyze purely analytically or to observe the real world.

2.3 Search-based Optimization

Search-based optimization is an umbrella term for algorithmic problem solving approaches that formulate optimization as an exploration of potential candidate solutions in a search space $\mathcal{S}$. The quality of a potential solution is evaluated by a fitness or objective function to guide the search in order to explore $\mathcal{S}$ efficiently to find near-optimal solutions. In the context of PCG, search-based optimization is widely applied (cf. Section 3). Given the formal definition of PCG in Equation 2.1, the search space would be the set of all possible contents $\mathcal{O}$. We will also use search-based optimization approaches in this thesis, such as hill climbing for level balancing (Chapter 6) and evolutionary algorithms for game economy generation and balancing (Chapter 7) and thus explain both approaches in more detail in the following.

2.3.1 Hill Climbing

Hill climbing [Eiben and Smith, 2015] is a family of trajectory-based local search algorithms for finding a local optimum $s^* \in \mathcal{S}$ within a given $\mathcal{S}$. As PCG can be defined as a search problem to find content within the defined $\mathcal{S}$, we use hill climbing approaches as baselines for our RL implementation in this thesis (Chapter 6).

A hill climbing algorithm operates iteratively in discrete timesteps t. It starts with an initial solution $s_{t=0} \in \mathcal{S}$ and then iteratively explores neighbors of $s_{t=0}$ using a neighborhood function N which yields a candidate solution $s_{t=1} \in N(s_{t=0}) \subseteq \mathcal{S}$. Domain knowledge, the best known solution so far, randomness, or their combinations can be used to initialize $s_{t=0}$. The quality of a candidate is evaluated with a function fitness$(s_t) \in \mathbb{R}$, often containing specific information about the problem domain. If the fitness $f_{t=1}$ of $s_{t=1}$ is superior then the one of $s_{t=0}$, $s_{t=1}$ is accepted as the new local optimum s^* and the search continues until a termination criteria is met. Such a criteria may be the number of allowed iterations T or a threshold when the actual solution is considered as good enough. A formal procedure of a hill climber is given in Algorithm 1. When implementing a hill climbing agent for *Catan* for instance, neighbors of s_t may be prioritized to directly increase the agent's total victory points.

Algorithm 1 Formal procedure of a hill climbing algorithm.

procedure HILLCLIMBING($s_{t=0}$)
  init $s^* \leftarrow s_{t=0}$
  init $f^* \leftarrow$ evaluate fitness$(s^*) \in \mathbb{R}$
  init $t \leftarrow 1$
  while $t \leq T$ **do**
    init $s_t \leftarrow$ select neighbor $s_t \in N(s^*) \subseteq \mathcal{S}$
    init $f_t \leftarrow$ evaluate fitness(s_t)
    if $f_t \geq$ threshold **then**
      return s_t ▷ success
    end if
    if $f_t > f^*$ **then**
      $s^* \leftarrow s_t$
      $f^* \leftarrow f_t$
    end if
    $t \leftarrow$ t + 1
  end while
  return s^* ▷ failure: return best-so-far
end procedure

Hill climbers can find a good solution quickly and are easy to implement, however, they tend to frequently find only local optima which may be significantly worse than the global optimum. Furthermore, depending on the evaluation function, there is often no information about how good the found solution is in relation to the global optimum.

2.3.2 Evolutionary Algorithms

Evolutionary algorithms [Eiben and Smith, 2015] are biology-inspired methods which can be applied to various optimization problems. They are widely used across domains such as chemical physics [Oganov and Glass, 2006], finance [Branke et al., 2009], and – relevant to this thesis – games [Togelius et al., 2024] (see also related work, Chapter 3), among others. In addition, evolutionary algorithms can also be used for *neuroevolution*, which is the usage of such an algorithm instead of the commonly used gradient descent for the optimization of the parameters of a neural network [Stanley and Miikkulainen, 2002; Floreano et al., 2008]. Risi and Togelius [2017] provide a survey on neuroevolution in games. In this thesis, we will apply evolutionary algorithms for the generation of valid graphs according to constraints and the balancing of game economies in Chapter 7.

Due to their flexibility, they can be applied to a wide range of search or optimization problems, including both discrete and continuous problems. Evolutionary algorithms typically employ crossover and mutation mechanisms which enable an extensive exploration of the search space $\mathcal{S}$. These mechanisms are often highly tailored to the target domain, allowing for strong adaptability — one of the key reasons for the widespread popularity of this family of algorithms.

Disadvantages include a strong reliance on randomness during search space exploration, which can lead to unpredictable behavior and make fine-tuning challenging. Additionally, similar to hill climbing methods, there is no guarantee of finding the global optimum — or even a satisfactory solution — within a limited number of iterations or computation time.

Functionality: Algorithm 2 describes the formal process of an evolutionary algorithm. Since they are often customized to solve specific problems, there is no single universal formal definition. This example provides a typical implementation that we will adopt in this thesis. The core concept of evolutionary algorithms is inspired by natural evolution: in a given environment, a population $\mathcal{P}$ of n individuals $i \in \mathcal{P} \subseteq \mathcal{S}$ competes for survival and reproduction. In the algorithmic context, each individual represents a potential solution, and the population maintains a diverse set of these solutions with varying quality (fitness). The fitness f_i of each individual is evaluated based on how well it performs in achieving the desired goal, essentially measuring how well it solves the given problem using a function $\text{Fitness}(i) \in \mathbb{R}$. The algorithm then simulates evolutionary cycles, known as generations, by selecting individuals for reproduction through crossover and applying mutations to introduce variation.

In the crossover step, individuals are typically paired randomly, with each pair serving as a parent unit. To create new individuals, crossover is applied. In biology, crossover refers to the exchange of genetic material between individuals, and in evolutionary computing, it similarly involves the exchange of "genomes" between parent solutions. These genomes must be modeled in a way that aligns with the problem domain, but ultimately, the problem's solution must be decomposable into individual genomes. The crossover process can be performed in various ways, depending on the data type of the genomes. For symbolic genomes, one common method is to randomly split the individuals at a chosen point and swap parts between them. For numerical genomes, mathematical operations can be applied to generate new solutions.

In the next step, mutations are applied to each $i \in \mathcal{P}$ with a probability μ. Similar to natural evolution, this involves randomly altering an individual's genome. Mutations are a crucial mechanism for introducing new genetic material into the population, allowing for the possibility of creating entirely new and potentially superior solutions. This mitigates the risk of the algorithm getting stuck in a local optimum by fostering diversity and encouraging the exploration of the search space. While mutations can thus be sufficient to create new genomes in order to explore the search space, the crossover step is not strictly mandatory in all evolutionary algorithm. The number of crossovers or mutations performed depends on the specific domain and the overall convergence of the algorithm. Therefore, the frequency and application of these operations must be evaluated and adjusted for each case individually. The crossover step then yields a new set of offsprings $\mathcal{C}$ which is then added to $\mathcal{P}$.

After crossover and mutation, the fitness f_i of all individuals is evaluated via a fitness function f, and the population is ranked according to individuals' finesses. For the next generation, only the n best individuals are selected to continue, ensuring that the most promising solutions are retained and further

evolved. This "survival of the fittest" selection ensures that the average fitness of the population increases over multiple iterations. The algorithm stops when a stopping criterion is met. As in hill climbing approaches, this can be a number of generations allowed or a threshold at which the current solution is considered good enough. Finally, the individual i^* with the best fitness is returned as solution.

Algorithm 2 Formal Process of an Evolutionary Algorithm

Initialize population $\mathcal{P} \subseteq \mathcal{S}$ with $|\mathcal{P}| = n$ individuals
repeat
  Select parents $\mathcal{P}_{parents} \subseteq \mathcal{P}, \quad |\mathcal{P}_{parents}| \geq 2$
  $\mathcal{C} = \text{Crossover}(\mathcal{P}_{\text{parents}}), \quad \mathcal{C} \subseteq \mathcal{S}$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{C}$
  for all $i \in \mathcal{P}$ **do**
    $i \leftarrow \begin{cases} \text{Mutate}(i), & \text{with probability } \mu \\ i, & \text{otherwise} \end{cases}$
    $f_i \leftarrow \text{Fitness}(i)$
  end for
  $\mathcal{P} \leftarrow \text{SurvivorSelection}(\mathcal{P}, n, f)$
until termination condition is met
return $i^* = \arg\max_{i \in \mathcal{P}} f(i)$

2.4 Reinforcement Learning

RL [Sutton and Barto, 2018] is a machine learning method inspired by learning from feedback of executed actions within an environment. Unlike supervised learning, RL does not require training data, making it particularly advantageous for applications in domains where usually no training data is available, such as procedurally generating content for games [Khalifa et al., 2020; Yannakakis and Togelius, 2025b], as in the context of this thesis. Since it learns to optimize a policy to optimize a trajectory, RL is less reliant on randomness once it has been trained, in comparison to other search-based methods (cf. Section 2.3). This characteristic makes RL well-suited for our use case, as it can speed up inference and, consequently, reduce generation time in the context of PCG. When implementing an RL agent for *Catan*, it may choose actions that temporarily decrease its victory points but lead to greater gains in the long term—offering an advantage over a hill climbing approach, which only considers immediate improvements. However, RL has some disadvantages, including limited generalization and scalability, as well as high computational costs during training.

Markov Decision Processes: In the basic framework of RL, two fundamental entities exist: the agent and the environment with which the agent can interact. This concept is similar to that of ABMs (Section 2.2). Environments can be complex and can contain uncertainty; the same action taken in the same state s_t may lead to different next states s_{t+1} due to stochastic transitions. A Markov De-

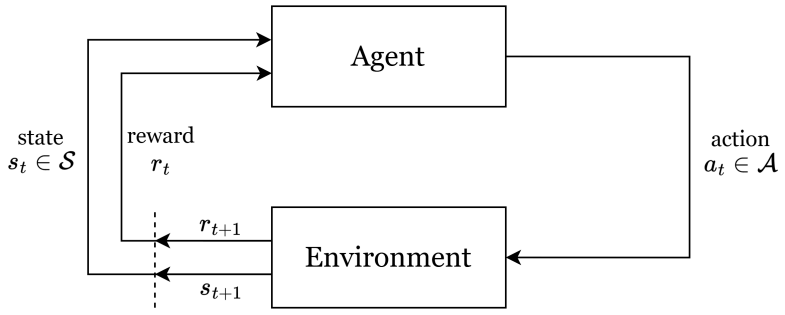


Figure 2.1: The interaction between the agent and the environment in a Markov decision process. A redrawn version by the author based on Sutton and Barto [2018, p.48].

cision Process (MDP) is a formal, structured approach to model such decision-making problems under uncertainty. By taking the consequences of different actions into account, MDPs also enable reasoning about the long-term implications of decisions. This is particularly advantageous, as it supports evaluating actions not just by their immediate outcomes, but by their expected cumulative rewards – allowing for the selection of actions that may appear suboptimal in the short-term but lead to better long-term performance. This is an advantage over search-based approaches such as hill climbing (cf. Section 2.3). Interactions with the environment occur in discrete time steps t and are defined by the quadruple $(\mathcal{S}, \mathcal{A}, P, R)$:

- $\mathcal{S}$ is a set containing all possible states $\{s_1, ..., s_n\} \in \mathcal{S}$. A state is the specific description of the environment in a distinct time step.
- $\mathcal{A}$ is a set containing all possible actions $\{a_1, ..., a_m\} \in \mathcal{A}$ in each state. An action is a specific interaction with the environment.
- $P(s_t, s_{t+1}, a_t)$ is the probability transition function describing the probability to enter state s_{t+1} from s_t by taking action a_t.
- $R(s_t, s_{t+1}, a_t)$ is the reward function describing the agent's expected reward given the state transition from s_t to s_{t+1} taken action a_t.

Figure 2.1 gives an overview of the agent-environment interaction. In each time step, the agent is in a specific state s_t of the environment. The agent can select an action a_t to interact with the environment and to enter the next state s_{t+1}. Subsequently, the agent receives a feedback, called reward r_t, from the environment. A positive reward reinforces the action taken in the context of s_t, a negative reward is considered a punishment which weakens the action. The agent's goal is to find a policy π which maximizes the cumulative reward when interacting with the environment. This sequence of actions is called a trajectory.

A challenge in RL is the trade-off between exploration and exploitation. Explorations refers to gaining new knowledge with potentially higher rewards by mainly choosing actions randomly, while exploitation is the usage of known

actions with high rewards. A simple example is the *ϵ-greedy strategy*, which selects the best known action with a certain probability ϵ. This may be combined with a decay mechanism. In the beginning of training where little knowledge is present, ϵ is high for choosing a random action to encourage exploration, while as training proceeds, ϵ decays to favor actions known to yield a high reward. Other approaches include *Upper Confidence Bounds*, such as Auer et al. [2002], which select actions based on both expected reward and and how often they have been tried. Another is *Entropy Regularization*, as used by Haarnoja et al. [2018], which encourages exploration by maintaining a more stochastic policy to prevent it from converging too quickly.

Methods: In this thesis the RL method Proximal Policy Optimization (PPO) [Schulman et al., 2017] is applied. This paragraph aims to categorizes it in the context of other RL methods in the literature. A detailed explanation of PPO is given in Section 2.4.1.

Multiple RL methods have been proposed which can be separated in different categories. *Value-based* methods aim to estimate a value function, the policy then selects state-action pairs based on their estimated value. Examples are Q-learning [Watkins and Dayan, 1992], where the agent learns an action-value function (Q-value) or Deep Q-learning [Mnih et al., 2015] which extends Q-learning with a deep neural network to approximate the Q-value function.

Instead of estimating value functions, pure *policy-based* methods directly learn a policy that maps states to actions. Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] constrains policy updates using a so called *trust region* to prevent it from too large updates. This trust region is defined by a Kullback-Leibler divergence threshold between the old and the new policy, measured over the state distribution, to avoid overly large changes. PPO improves TRPO by using a clipped surrogate objective function in order to constrain the policy updates, achieving similar stability with a simpler implementation.

Actor critic methods are a combination of both, value-based and policy-based methods. While the action is in charge of selecting good actions, the critic evaluates the actions by estimating a value function. Examples for actor critic methods are PPO or TRPO. Both use (deep) neural networks to represent the critic and the actor.

Another distinction of RL methods can be made by separating them into *model-free* and *model-based* methods. Model-free methods do not require a specific type of environment to learn an optimal policy, whereas model-based methods require a model of the environment. Examples for a model-free method are PPO or Q-learning. An example for a model-based meth-od is AlphaZero [Silver et al., 2018], which uses a model of the environment for better planning of the state-action pairs.

2.4.1 Proximal Policy Optimization

This thesis applies PPO [Schulman et al., 2017] as RL algorithm, as it is widely used due to its balance of simplicity and robustness. Since then it has been applied in various domains, such as fine-tuning of large language models [Ouyang

et al., 2022], robotics [Shahid et al., 2020], stock trading [Yang et al., 2020], and, as the focus of this thesis, PCG for games [Khalifa et al., 2020], to name just a few.

Functionality: PPO is model-free, policy-based, and actor-critic. In contrast to other policy gradient methods, PPO updates the policy in mini batches more frequently to ensure a more efficient training. It's main novelty lies in the introduction of a clipping mechanism to avoid overly aggressive policy updates and a better training stabilization. This is achieved by its *Clipped Surrogate Objective* function (Equation 2.5). To avoid excessive updates of the policy, PPO adds a constraint to penalize too big changes to the policy parameters θ.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{2.5}$$

The probability ratio $r_t(\theta)$ of choosing a_t in s_t under the actual policy π_θ and the old policy $\pi_{\theta_{old}}$ is defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2.6}$$

Before starting with any policy updates, $r_t(\theta)$ will be 1. While updating the policy in mini batches, $r_t(\theta)$ will move away from 1. To counteract too large changes to θ, updates exceeding $[1-\epsilon, 1+\epsilon]$ are clipped, where ϵ is a configurable parameter. The estimator of the advantage function $\hat{A}_t$ in Equation 2.5 measures the relative benefit of taking action a_t in state s_t compared to the expected return from the state under the current policy. The average value of the objective function $\hat{\mathbb{E}}_t$ represents the sampled trajectories from interacting with the environment.

Chapter 3

Related Work

Before providing a detailed overview of the related work, we categorize this thesis within its research domain. A schematic categorization is given in Figure 3.1. This thesis is situated within the broad field of game research, specifically at the intersection of two subdomains: Procedural Content Generation (PCG) and automated game balancing. Game balancing itself is rather a practice which is applied in industry than an academic field. However, there are emerging fields focused on optimizing and automating this process. An example is the investigation of interactions via PCG elements, a subject to which our work contributes. Within this intersection, we further focus on methods that utilize machine learning.

This chapter is therefore structured as follows: we first provide related work on PCG in games in Section 3.1, the definition and background of PCG has been given before in Section 2.1. Section 3.2 covers related work on automated game balancing. Within this section, we include a dedicated paragraph that presents work that is directly at the intersection of PCG and automated game balancing.

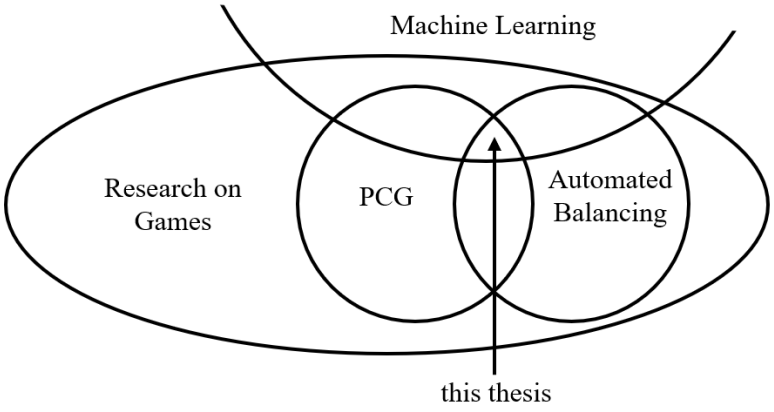


Figure 3.1: A schematic categorization of this thesis within the academic literature.

3.1 Procedural Content Generation in Games

The academic literature on PCG in games covers multiple different algorithmic approaches and genres for which the content is to be created. In this thesis, we are interested in the algorithmic approaches and thus structure this section accordingly. We will first give a broad overview of the methods in general, before going into more detail by describing specific works. Since this thesis is at the intersection of PCG, game balancing, and machine learning (cf. Figure 3.1), the intention here is to give the reader an overview of existing techniques in the literature, rather than a comprehensive enumeration of all existing works.

PCG methods proposed in literature are mostly search-based [Togelius et al., 2011b], such as evolutionary algorithms [Togelius et al., 2024], constraint-solving [Cooper, 2022], machine learning [Summerville et al., 2018] and deep learning [Liu et al., 2021] approaches, which have recently been enhanced by the use of Large Language Models (LLMs) [Gallotta et al., 2024]. Besides the mentioned approaches, dedicated algorithms tailored to a particular game are widely implemented in the industry, such as in games like Minecraft [2011] or Factorio [2020]. However, since these approaches are neither academic nor publicly available, we exclude this category from this section.

While many different works on PCG for games have been introduced with different algorithmic approaches, the evaluation of a content generator is not standardized. To fill this research gap, Khalifa et al. [2025] introduce the *PCG Benchmark*, a testbed decoupled from a specific algorithm, to evaluate PCG approaches in terms of the quality and diversity of the generated content, as well as the method's controllability. We will evaluate our content generators on comparable aspects, but we have not included the proposed test benchmark as it was published at the end of the writing of this thesis.

Although many different methods for PCG have been proposed in the literature, we find that the majority of works focus on single-player and tile-based or voxel-based games. Popular games used as testbeds are for instance the puzzle game Sokoban [1982], the platformer Super Mario Bros. [1985] or the sandbox game Minecraft [2011]. In contrast, in this thesis we focus on competitive two-player games in the context of balancing entirely through PCG and the generation of graph-based game economies.

3.1.1 Search-based Approaches

Search-based approaches (cf. Section 2.3) are particularly suitable to create game content since they do not require existing game content to learn from as in contrast supervised machine learning methods do. Furthermore, they are very flexible in terms of the design of their objective function. For these reasons they are a popular and widely applied approach in research.

Togelius et al. [2011b] provide a taxonomy and survey on search-based PCG. Yannakakis and Togelius [2011] introduce experience-driven PCG (EDPCG), a framework to use PCG to enhance the user experience by exploring the search space to find candidates which maximize the player experience. Therefore, a player experience modeling is included into the objective function of a search-based algorithm, using subjective and objective player or gameplay data. Ac-

cording to the authors, gameplay data used within an objective function can be considered a static simulation-based evaluation function. We will use this concept in Chapters 5 and 6, however, applied in the reward function for Reinforcement Learning (RL).

Evolutionary algorithms are a popular algorithmic approach for PCG. An up to date survey how evolutionary algorithms are used in the context of games, particularly also for PCG, is given by Togelius et al. [2024]. Many related works also combine evolutionary algorithms with an additional method: Hastings et al. [2009] propose an algorithm in combination with an evolutionary algorithm to generate both graphics and game content tailored to a player's preference. Cardamone et al. [2011] introduce an evolutionary algorithm for the generation of maps for a first-person shooter using simulations with artificial agents to evaluate the generated content. Shaker et al. [2012] generate new levels for the game Super Mario Bros. [1985] by combining an evolutionary algorithm with a grammar for defining constraints. Holmgård et al. [2019] generate personas using Monte Carlo Tree Search (MCTS) in combination with an evolutionary approach to generate personas for game testing. Liapis et al. [2013] present the *Sentient Sketchbook*, a game level design tool to assist designers in the development phase. The tool allows to create sketches of levels – based on which the tool then automatically generates levels for proposal using a genetic algorithm, and tests them for playability, among other things. Charity et al. [2020] generate new levels for the game Baba Is You [2019] with a mixed-initiative approach of human users and an evolutionary algorithm. Rogers et al. [2023] use an evolutionary algorithm for the generation of graph-based game economies to target different complexity levels. Along with a user study the authors show that participants perceived the targeted complexities as intended. We will discuss this work in more detail when we are dealing with game economy generation and balancing in Chapter 7.

Besides evolutionary algorithms, other search-based approaches are also applied for PCG. Summerville et al. [2015] present the improvement of Markov chain-based PCG level generators with MCTS. De Kegel and Haahr [2020] provide a survey of PCG for puzzles games, highlighting several works that use search-based approaches such as MCTS, Breadth First Search, and evolutionary algorithms.

Inspired by Rogers et al. [2023], we will apply evolutionary algorithms to the largely under-explored field of game economies in the context of GEEvo (Game Economy Evolution) in Section 7.4. We will investigate their use for both generating and balancing graph-based game economies and employ them as a baseline for G-PCGRL (Graph PCG via RL) in Section 7.3.

3.1.2 Constraint-solving Approaches

In the context of PCG for games constraint-based solving approaches generate content by systematically satisfying predefined rules (constraints) that specify limitations of the content. While this family of algorithm ensures content validity by design and offers fine-grained control through explicitly defined constraints, solving the constraint-based problem can quickly become computationally expensive and often results in repetitive content which is generated.

A popular representative for the generation of tile-based game content is the algorithm *Wave Function Collapse (WFC)*, inspired by quantum mechanics and originally introduced by Maxim Gumin in 2016 in a non-academic context.[3] WFC generates content greedily and without backtracking, filling a grid by assigning tiles to each cell in order to satisfy local adjacency constraints. Each cell starts in a superposition where all tiles are possible. In order to lower the entropy of the grid, the algorithm iteratively reduces the uncertainty by eliminating invalid options by propagating constraints from neighboring cells until the whole grid is resolved or a contradiction occurs. Due to its versatility in creating content, WFC has been recognized by researchers, e.g., in the investigation as a general constraint solving approach for PCG by Karth and Smith [2017] or in the survey by Summerville et al. [2018]. Since its introduction, many works extended WFC, such as through incorporating design-level constraints [Sandhu et al., 2019], applying it hierarchically [Beukman et al., 2023], or including also non-uniform tiles [Piepenbrink and Bidarra, 2025], to name just a few.

Other works use SMT (Satisfiability Modulo Theories) solvers, such as the one described by Whitehead [2020], to generate dungeon layouts. Others employ Answer Set Programming (ASP), as demonstrated by Smith and Mateas [2011], in various game domains. *Sturgeon*, an approach introduced by Cooper [2022], is a system for generating tile-based levels using a constraint solving approach. The author evaluates his method by comparing the different solvers and testing it with various game environments, such as Super Mario Bros.

While constraint-solving approaches can be a powerful tool for PCG, we avoid them in this thesis since they require a manual encoding of human and domain knowledge through constraints [Guzdial et al., 2025]. In particular, the latter is what we explicitly do not want to do in this thesis to ensure that our approaches are easily transferable to other games. In the context of automated balancing through PCG, capturing the nuances of balancing via a set of constraints is especially prone to errors and cumbersome. In the context of G-PCGRL (Graph PCG via RL) in Section 7.3 we deal with sets of constraints. However, we approach this from a different perspective: learning these sets via RL in order to speed up generation time.

3.1.3 Machine Learning and Deep Learning Approaches

Besides search-based and constraint solving approaches, machine learning and deep learning methods can be applied to train models for content generation on existing content. Summerville et al. [2018] present a survey on PCG methods using machine learning (PCGML) and Liu et al. [2021] for deep learning. While search-based methods can use machine learning for evaluation, but generate the content still by exploring the search space, PCGML methods directly use data to train a model which generates the content. PCG with deep learning enhances neural network based PCGML using advanced neural network architectures, such as Generative Adversarial Networks (GANs), Convolutional Neuronal Networks (CNNs) and LLMs. GANs are applied for instance in the work presented by Volz et al. [2018] and *TOAD-GAN* [Awiszus et al., 2020] to

[3]Github: `https://github.com/mxgmn/WaveFunctionCollapse`

generate Super Mario levels. Minecraft [2011] worlds are generated in *World-GAN* [Awiszus et al., 2021]. For the tile representation of the levels, techniques known from natural language processing are used, for instance TF-IDF (TOAD-GAN) or embedding representations (World-GAN).

Merino et al. [2023] introduce the *5-dollar model*, a lightweight sentence transformer architecture to generate pixel art sprites for games. Bontrager and Togelius [2021] present *Generative Playing Networks*, a PCG architecture for level generation consisting of two parts: a CNN-based generator and a RL agent which learns to play generated levels. In contrast to other PCGML approaches, this method does not require existing training data, as the agent guides the generator in terms of what content to generate.

Recently, LLMs gained a lot of attention and are also applied to various PCG problems. LLMs enhance PCG to provide control and descriptions of the desired content via natural language. Gallotta et al. [2024] present a comprehensive survey on the use of LLMs in games, including PCG, while Maleki and Zhao [2024] provide a focused survey specifically on LLMs for PCG in games. Sudhakaran et al. [2023] introduce *MarioGPT*, a fine-tuned LLM to generate Super Mario levels and Todd et al. [2023] explore LLMs to generate levels for a puzzle game. Whitehead et al. [2025] empirically evaluated the generation of tile-based worlds via LLMs through conversational interaction. They concluded that large parameter models could quite accurately reason about game world information in the tested setting but performed poorly when reasoning about pure tilemap representations or tile image data. We will describe LLM approaches explored in combination with RL in the subsequent paragraph on PCG with RL.

Since our use cases involve problems for which no training data is available, the approaches in this paragraph are not applicable, but we include them here to provide a thorough overview of the research area.

3.1.4 Reinforcement Learning Approaches

RL lies at the intersection of machine learning and optimization (cf. Section 2.4). In the context of games, RL is widely applied to *play* them (e.g., Silver et al. [2018] and Vinyals et al. [2019]), whereas in this paragraph we focus on the use of RL to generate game content. Khalifa et al. [2020] introduce the PCGRL (PCG via RL) framework to frame level generation as a Markov Decision Process (MDP). Since this thesis uses and extends PCGRL (Chapters 5, 6, 7), we give the reader a thorough explanation on PCGRL in Section 4.4 and focus here on the related work in terms of PCGRL.

Earle et al. [2021] extended PCGRL to be controllable. Controllability is provided by including additional information in the observation space and also adjusting the reward function dynamically. For example for the game Sokoban, the PCGRL generator can be controlled in terms of how many boxes should be in the level. Jiang et al. [2022] further adapt PCGRL to generate 3D levels and demonstrate this in an Minecraft environment.

However, RL suffers from limited scaling, in particular when the action space or the observation space increase. When using RL as a procedural content generator this is a problem when larger levels or levels with many different types of tiles should be generated. Earle et al. [2024] try to address this issue and

experiment with different definitions of the observation space (representations) like partial observability. Additionally, they propose using PCGRL on a GPU to enhance computation speed. While this approach slightly improves PCGRL's scalability, scalability limitations still arise as the level size increases. In comparison to this work, Earle et al. use comparably simpler environments where the rewards can be calculated directly from the environment state. For our use case with a simulation-driven evaluation function within the reward function, this is not an efficient solution, as this would require a reimplementation of the whole game environment for a GPU, among other things. Additionally, heuristic agents that use path-finding algorithms, such as A*, cannot be efficiently run on a GPU. For example, the inherently sequential nature of A* affords limited opportunities for parallelization [Hart et al., 1968].

Shu et al. [2021] introduce Experience-Driven Procedural Content Generation via Reinforcement Learning (EDRL) based on PCGRL and the experience-driven PCG by Yannakakis and Togelius [2011] mentioned earlier. On the example of Super Mario Bros. levels the authors show how to embed different constraints in the reward function to generate levels varying in fun and diversity based on the *theory of fun in game design* by Koster [2013]. The action space of the MDP definition is to select level segments generated with a pretrained GAN. EDRL has than been extended by Wang et al. [2022] to generate further facets of game creativity.

PCGRL has been further investigated by incorporating natural language processing methods. Baek et al. [2025b] introduce IPCGRL (Language-Instructed PCGRL), a PCG method that incorporates a sentence embedding model to provide controllability via natural language-based instructions. With *ChatPCG*, Baek et al. [2024] show that reward functions which are generated as program code with an LLM enhance the content quality by better emphasizing certain game design aspects. With PCGRLLM (PCG via RL and LLMs), Baek et al. [2025a] enhance ChatPCG by employing a reward refinement approach and multiple reasoning-based prompt engineering techniques. The authors demonstrate their method on a previously GPU optimized tile-based 2D game environment. For instance, a reward function generated with PCGRLLM from natural language can teach a RL agent to generate levels where the player must first encounter a bat and then find a key to escape through a door.

Gisslén et al. [2021] propose ARLPCG (Adversarial RL for PCG) with two RL agents. The adversarial design of these two RL agents lies in two separated RL agents, one for the PCG part and one for the playtesting of the generated content. To generate challenging but not impossible content, the generator receives a reward signal from the playtesting agent. For better generalization the authors use an auxiliary input within the observation space to improve the model in terms of generalization. We will use the idea of dividing the content generation process into distinct units, such as in the context of generating balanced levels (Chapter 5) and game economies (Section 7.4).

With this thesis, we contribute to using RL for PCG in particular to generate balanced game levels, where we use, adapt, and extend PCGRL (Chapter 5 and 6). In addition, we contribute on learning sets of constraints in order to generate graph data via RL (Section 7.3).

3.2 Automated Game Balancing

Several methods in the context of automated game balancing on the examples of different board and video games and genres have been proposed. In Section 1.1.2, we had previously given a definition of what game balancing is. Related works on automated game balancing are to be separated into two areas: first, methods or processes which can be used to automatically generate insights on a game's balance (Section 3.2.1) and second, methods that use e.g., algorithmic approaches to balance games (Section 3.2.2). Both types aim to support the design process of a game and to reduce manual effort, e.g., to reduce the required time for playtesting. The main takeaway for this work is the overview and the way in which other works use simulations to empirically determine the balance of a game.

3.2.1 Methods for Automated Game Balance Analysis

Jaffe et al. [2012] propose an "early warning" system for balance for game developers. Like in this work, the authors estimate the balance using artificial agents for competitive two-player games, but for a card game in an educational context. The distinction to this work is their focus on providing game-specific information to the developer, for instance, which actions may be imbalanced in which state of the game.

Pfau [2025] presents a method for balancing the progression in a game using Baldur's Gate 3 [2023] as an example. While other works on automated game balancing focus on selected aspects at a specific state of the game, the author highlights that this is not enough and that a game in order to be balanced must "shine" in all its times and places [Pfau, 2025, p.10]. For this reason, Pfau proposes using game simulations with game-playing Artificial Intelligence (AI) to explore all possible game states across different configurations. This is similar to our approach of using an Agent-based Modeling (ABM) to simulate balance, however, our focus is on competitive games, while Pfau focuses on progression balance in a (cooperative) Player versus Environment (PvE) context. In addition, Pfau analyzes the progression of game combats and compares how much damage selected classes deal compared over multiple rounds of combat based on data from simulations. We follow a similar process with GEEvo (Chapter 7), however, we model the simulation as a graph-based game economy and use the data to optimize the economy configuration and so the overall balance.

Istamar et al. [2023] propose to train a RL agent to play a game and analyze its behavior to identify imbalances. In a case study, the agent discovers a dominant strategy, providing insights that can be used to adjust the game's balance.

The focus of this thesis is on game balance through PCG and does not develop new methods for automated game balance analysis directly. However, our simulation and modeling framework, introduced in the context of GEEvo, enables the investigation and automation of game economy balance.

3.2.2 Methods for Automated Game Balancing

Algorithmic methods for automated game balancing often use search-based approaches to optimize balance. An overview of search-based optimization has been given in the background chapter in Section 2.3.

In an initial approach, Hom and Marks [2007] use an evolutionary algorithm to balance board games by adapting their game rules. Volz et al. [2016] propose multi-object evolutionary algorithms to balance the decks of a card game. The algorithm optimizes card decks towards a pareto optimality of balance and excitement. The objectives for the fitness functions are estimated with a simulation of the game. The authors conclude that automated balancing is feasible, however, they note that the human perception of the actual balancing is the "only acceptable way" to accurately achieve balance and to maximize the enjoyment for human players [Volz et al., 2016, p. 276].

Beyer et al. [2016] introduce an integrated, iterative process for automated game balancing using a notation for business process models. However, automatically tested gameplay with artificial agents may be too different from human gameplay. Therefore, like Volz et al., the authors highlight the importance of human playtesters and therefore the process provides for both, automated testing and manual testing of the game's balance. While the process itself is game independent, the proposed application still requires the incorporation of game dependent parameters in the objective function of the applied evolutionary algorithm.

Preuss et al. [2018] apply the balancing process from Beyer et al. in a case study to an open source Real-Time Strategy (RTS) game with small modifications. Game parameters are tested with simulations of the game and are automatically optimized with an evolutionary optimization strategy. The authors highlight the computational effort of the simulations, restricting them to evaluate each parameter configuration only three times. As we will explore in Section 5.2.2, the minimum number of simulations to run the game is notoriously important in order to determine the game's true balance with acceptable variance. As an objective function, the authors use highly game specific parameters such as how many units are killed. Using game specific information, also like Beyer et al. do, makes the method cumbersome to transfer to other games, as for each new game a new function must be created respectively, tested and evaluated. For this reason, we will not include any game specific information in any of the proposed methods in this thesis (cf. Chapters 5, 6, and 7).

Politowski et al. [2023] introduced an approach with autonomous agents that are trained via RL to target game balancing from a game testing perspective. The study focuses on the two types of balance: challenge versus success and skill versus chance. In this context, they systematically evaluate their method using a feedback loop that includes different versions of a game and agents with different skill levels, among others. Hernandez et al. [2020] present a framework for balancing the metagame of a multiplayer game. The desired metagame can be expressed using a graph representation, and the balance is optimized using a Bayesian optimization approach. As in this work, the authors use simulations with artificial agents of the game. The framework is then tested using the simple

Rock-Paper-Scissors environment, as well as a more complex combat environment.

Mahlmann et al. [2012] propose a method based on evolutionary algorithms to compile card decks to balance the deck-building board game Dominion [2008]. The authors experiment with different fitness functions and also evaluate the balance with artificial agents. In particular three different agents each with different skill levels are used. The authors conclude, that there are cards in the game which bring more balance into the game than others, independently of player skill and behavior.

Beau and Bakkes [2016] target automated game balancing for asymmetric games on the example of a tower defense game. Using Monte Carlo simulations, the proposed method aims to find the actions or the trajectory of actions that lead to imbalanced states, and then iteratively adjust the identified actions. The overall balance is then expressed in terms of the proportion that a player wins on average; imbalances exist when the proportion is skewed toward a single player. In Chapter 5 we use a similar approach, but motivate it from a different angle: a fairness metric to assess algorithmic fairness.

The method introduced by Pfau et al. [2020] addresses automated game balancing differently in comparison to the previously described works by introducing deep player behavior models (DPBM), neural network-based models trained on user data to replicate human behavior for a MMO game. The advantage of this method is that it allows human play styles to be replicated in a data-driven manner. Using data from simulations, balance is assessed using game-independent information, such as whether an agent wins, combined with game-dependent data like fight durations and the remaining health of both the agent and its opponent. The replicated agents are then tested in a PvE and Player versus Player (PvP) setting. In their follow-up work [Pfau et al., 2023], the authors provide additional insights by analyzing the distribution of optimal value configurations for enemies based on the available player data, among other things.

Dynamic Difficulty Adjustment (DDA) is also a researched field in that context as it aims to automatically balance the game's current level of difficulty for a better and personalized player experience. Zelada and Gutierrez [2023] propose an algorithm to use DDA based on players' heart rate for a platformer game. While they increase the game's difficulty, for example, by adding more obstacles when the heart rate decreases, they find in an accompanying user study that the player experience increases when the difficulty is adjusted to keep the heart rate within a predefined range.

Automated game balancing through PCG: Whereas most works focus either on automating game balancing or procedurally generating playable content, this thesis combines both approaches to target the balance through the design of the content (cf. also Figure 3.1). Lara-Cabrera et al. [2014] generate balanced and dynamic maps for an RTS game using an multi objective evolutionary algorithm. As in this work, balance is estimated based on data from simulations, however, a fuzzy rule base to incorporate game-specific information and expert knowledge is used. In addition, no description of how many simulations are conducted

or how the agents in the simulation behave is given. Similar to our findings for certain generated levels, which we will report in Chapter 6, Lara-Cabrera et al. found, among other things, that their method generates perfectly balanced levels at the expense player inaction.

Lanzi et al. [2014] generate balanced maps for a first-person shooter, also using an evolutionary algorithm along with a balance estimation based on statistics from simulations with artificial agents. The authors are also experimenting with balancing maps for asymmetric setups by pitting agents with different weapons and skill levels against each other. Inspired by this approach, we will also experiment with asymmetric setups of player agents in Chapter 6. Like previously discussed works on automated balancing, this work also incorporates game-specific information into the fitness function of the algorithm to estimate the balance of a map in a given state. Since the game environment is deterministic, the simulation is run only once, although different skill levels of bots are created by setting a value for the respective accuracy when shooting. Many games include, however, probabilistic elements in order to increase replayability and to encourage strategic thinking under uncertainty. For this reason, we focus on level balancing with probabilistic elements in Chapter 5 and 6, but also in the context of game economies in Section 7.4.

Besides levels or maps, other forms of balanced content can be generated in the context of a game. Sorochan and Guzdial [2022] generate balanced units for the RTS game environment Micro RTS[4] using a search-based approach. The balance of a unit is evaluated using game simulations with MCTS agents. As a result, the authors propose ten generated units with distinct value configurations that remain balanced, for instance, being worth their cost relative to their in-game functionality. This approach enables the generation of diverse and viable strategies by increasing a game's asynchrony.

We have observed that many works on automated game balancing rely on simulations of the game to estimate balance. We adopt this approach, but address two key limitations of existing works: (1) simulation setups and configurations are often under-specified or not specified at all, and simulations are typically run only a few times for each balance estimate. In addition, (2) most methods rely on game-specific information. In contrast, we will propose an approach to estimate the minimal number of simulations required to reduce randomness-induced fluctuations, regardless of game-specific knowledge. This approach provides a foundation for reliably estimating a game's balance simulation-driven. Lastly, we will contribute to automated level balancing entirely through PCG, thereby addressing the two mentioned limitations of existing works. In addition, while other works focus on the balancing of game maps, classes, or units, we will explore the balancing of game economies.

[4]Github: `https://github.com/Farama-Foundation/MicroRTS`

Chapter 4

Foundational Concepts

Partial results from the author's publications are included in this chapter:

- Section 4.2 and 4.3: **Florian Rupp**, Manuel Eberhardinger, and Kai Eckert. Balancing of competitive two-player Game Levels with Reinforcement Learning. In *2023 IEEE Conference on Games (CoG)*, pp. 1-8, Boston, USA, 2023. doi: 10.1109/CoG57401.2023.10333248 .

- Section 4.1, 4.2, and 4.3: **Florian Rupp**, Manuel Eberhardinger, and Kai Eckert. Simulation-Driven Balancing of Competitive Game Levels with Reinforcement Learning. *IEEE Transactions on Games (ToG)*, vol. 16, no. 4, pp. 903–913, 2024. doi: 10.1109/TG.2024.3399536 .

- Section 4.5 **Florian Rupp** and Kai Eckert. GEEvo: Game Economy Generation and Balancing with Evolutionary Algorithms. *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, pp. 1–8, doi: 10.1109/CEC60901.2024.10612054 .

This chapter presents the foundational concepts that are used throughout the thesis, most of which are developed as part of this work. These concepts establish the basis for our contributions and help us address the research questions.

This thesis contributes to the automated balancing of games through algorithmic approaches. First, we clarify and refine the concept of balance to ground our work. Since games for entertainment are designed and intended to be played by human players, it is important to distinguish balance from the related yet distinct concept of fairness. Thus, we discuss fairness and balance in general, and subsequently in the context of games, to establish a more precise foundation for our balancing methods (Section 4.1). Based on that foundation, we introduce a metric in Section 4.2 to express the balance of a two-player game numerically. We derive this metric from an existing fairness metric, which constitutes our contribution C 1. We will use this metric throughout the Chapters 5 and 6.

Due to the lack of research environments for game balancing through Procedural Content Generation (PCG), we introduce the *Feast & Forage* environment, which we will mainly use in Chapters 5 and 6. It can be found in Section 4.3 and is our contribution C 6. In Chapters 5 and 6, we formulate game balancing as

a Markov Decision Process (MDP) and in Chapter 7, we formulate constraint-based graph data generation as an MDP. For both, we use the PCGRL (PCG via Reinforcement Learning (RL)) framework by Khalifa et al. [2020] as a foundation, which we adapt, extend, and build on. We describe Procedural Content Generation via Reinforcement Learning (PCGRL) in Section 4.4.

In Chapter 7, we address game balancing from the perspective of game economies. To simulate graph-based game economies, we introduce a lightweight game economy simulation framework inspired by the formal definition by Klint and van Rozen [2013]. This framework is presented in Section 4.5.

4.1 Balance and Fairness – A Distinction Between the Concepts

With this section, we define and discuss what balance in games means, as well as how this concept differs from the term fairness. It is important to note that the concept of fairness is a social and ethical concept, and not a statistical one [Chouldechova, 2017]. Games are, however, designed to be played by humans. Therefore, when applying an automated balancing process, particularly in a competitive environment with multiple players, social and ethical considerations should be taken into account.

This section is structured as follows: First, we give an overview of how the terms fairness and balance are used in other fields, and explain their meaning in the context of games (Section 4.1.1). Next, we introduce our method for estimating the balance of a two-player game using a simulation-driven approach and expressing it numerically (Section 4.2). Finally, we discuss the extent to which a game can be considered balanced (Section 4.2.4).

4.1.1 Fairness and Balance in General

In this thesis, the terms *fairness* and *balance* are key concepts that are frequently referenced, each however addressing different aspects with different conceptual focuses. Before we discuss them in more detail in the context of games, it is important to define them and distinguish between them.

Fairness has been defined across various disciplines, reflecting diverse perspectives of equity, justice and inclusion. The philosopher John Rawls [1971] introduced the concept of justice as fairness by treating all individuals equally and impartial. His thought experiment the *Veil of Ignorance* puts individuals without knowledge of their own characteristics in a decision-making problem where they can decide about the design of a future social order, but they don't know in which place of the order they will be. According to Rawls, this ignorance ensures that their decisions are not influenced by self-interest and people would improve conditions for the most disadvantaged to mitigate their own risk, ensuring fairness and impartiality.

In economics, fairness can be understood as the equality of opportunity, which means that an outcome for an individual with e.g., disadvantageous circumstances only depends on factors for which the individual could be considered responsible for [Roemer and Trannoy, 2015]. To ensure fairness, policies can

be applied to govern systematic biases in economic systems and the distribution of resources. In healthcare, the allocation of limited medical resources, particularly during the COVID-19 pandemic, became a critical challenge. Emanuel et al. [2020] define six recommendations to ensure patients are treated fair across multiple areas and to prevent that individual doctors are left with the decision which patients to prioritize. In the context of fair machine learning, fairness is expressed technically through constraints and metrics to ensure equal treatment of individuals and groups by preventing biases in data, predictions and models [Barocas et al., 2023].

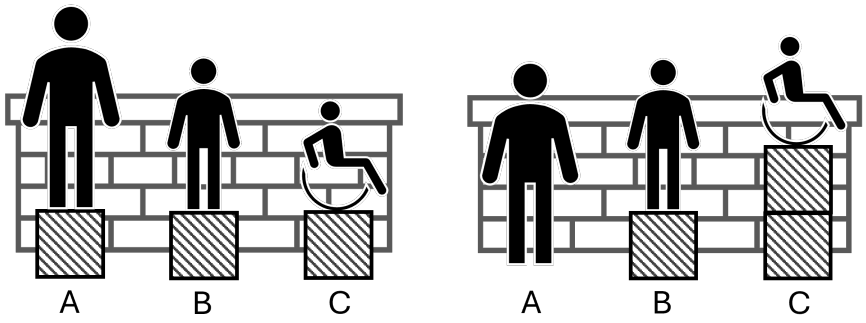
The described definitions, though emphasizing different aspects of fairness, converge on a common principle: ensuring equitable treatment and addressing inequalities of individuals or groups while also taking their contextual background into account. We therefore refer to the term fairness as ensuring equity while also taking into account a person's or group's contextual background.

Balance in contrast refers to managing competing objectives by ensuring a state of equilibrium. The philosopher Thomas Nagel [1995] explores balance as managing ethical tensions in a way that acknowledges both universal and moral considerations. He states as an example how individuals and societies must find a balance between impartial treatment for all, and partiality, such as a special treatment for friends or family members.

In economics balance refers to a trade-off when addressing competing principles, in particular in the context of resource distribution. Sen [1986] for instance links this to the balance between efficiency (maximizing total benefits and welfare) and equity (fairness for the disadvantaged), in other words the balance between moral and pragmatism. Balance is also important in the context of ecosystems. An example is the balance between carbon dioxide emissions and its sequestration e.g., through photosynthesis. The human-caused imbalance heavily impacts the entire ecosystem, resulting in global warming, permafrost thawing, and even more carbon dioxide being emitted [Schuur et al., 2008]. In machine learning, balance involves the trade-off between performance metrics such as, in the context of a classification problem, recall and precision [Bishop and Nasrabadi, 2006]. Imbalance is also present in data, where a class label is underrepresented, which is a challenge for many algorithms [He and Garcia, 2009].

Based on these definitions, we use the term balance when ensuring an equilibrium. This does not take the contextual background of individuals or groups into account by design.

The difference between fairness and balance can be further illustrated with the fence metaphor by Angus Maguire [2016] in Figure 4.1. Three people of different heights and conditions stand behind a fence and try to look over it. There are only a limited number of three boxes to help the people – how should we distribute them? Figure 4.1a shows a balanced distribution of the resources: all three people have been given the same help – one box to stand on. With the aid of the box, however, person C in the wheelchair is still unable to see over the fence, whereas persons A and B can. Although this distribution is balanced, it is not fair, since not all parties are treated according to their contextual background and thus cannot all see over the fence. In contrast, Figure 4.1b shows a fair



(a) Balance: an equal treatment. (b) Fairness: an equitable treatment.

Figure 4.1: Balance and fairness in the fence metaphor. A redrawn version by the author based of an original illustration by Angus Maguire [2016].

distribution of boxes, that allows everyone to see over the fence. In this case, the distribution is not balanced, but the outcome is fair, assuming that the overall goal is to design a process that allows all parties to look over the fence. With that in mind, we now discuss both terms in the context of games.

4.1.2 Fairness and Balance in Games

Based on the survey by Becker and Görlich [2020], we had discussed that there is no clear consensus among researchers what balance in the context of games accurately means (Section 1.1.2). Balancing a game in a competitive multiplayer context, such as in this thesis, differs from balancing solo player games or co-operative multiplayer games. Here, balance must ensure equal chances of winning for all players, for example by providing equal initial conditions. A well-balanced game should allow players to win based on their decisions – and therefore their skills – rather than relying on luck. If we refer to the metaphor in Figure 4.1a again, we assume in this context that all players are identical and thus provide them with the same number of equally-sized boxes. By design, we do not consider that the players, the people playing the game, *are* actually different. Exactly this difference between players should determine the outcome of winning or losing, not an imbalanced game setup nor pure luck.

This is what the term fairness in a game references to. In a fair game, players *feel* that a win or lose is earned, because their own decisions and condition have influenced the outcome, not because there is a bias or impartial treatment in the game design. Fairness always includes the players' individual perceptions, whereas balance is to ensure the mathematical equilibrium.

In the opening quote on page 1, Oscar Wilde [1899] states that "life is never fair" and that "it is perhaps a good thing for most of us that it is not". Wilde points out to an inherent truth: life is unfair. Circumstances like birth, talent or luck are not distributed in a way that each individual has equal chances to succeed in life and human social systems can also not fully compensate this (even if they try). In Wilde's context, this is a cynical remark about the society of the time, where the upper class — to which Wilde also belonged — owed their wealth and status to inheritance. For them, life's unfairness was beneficial, as

they had received advantages they hadn't earned. However, unfairness in life can let people think outside the box, being creative, innovate and try out various experiences. If life was perfectly fair, most things would be highly predictable and there would be only little room for surprises.

And this is exactly what makes games interesting: Every human player of a competitive game is different. They might be of different ages, come from various backgrounds, have a different education, or be in a better physical condition than others. All these circumstances may affect their problem-solving strategy and ability to react to unforeseen challenges resulting in unique behaviors in order to win the game. For example, as shown by Kallabis et al. [2025], simply designing identical game elements differently can lead people to perceive the intended strength in the game differently. If all humans were the same, a game's outcome would be almost always the same, merely dependent on the game's probabilistic elements. In order to estimate balance isolated from fairness, we will therefore use heuristic player agents that always behave deterministically and consistently, to measure the effect of the system design rather than players' differences (Chapter 5 and 6). Since humans are different, also their perception of fairness is different. The one player might think he lost because of bad luck, while another may perceive it was because of a wrong strategy she chose.

For these reasons, we cannot and do not want to create a game for entertainment that is fair in any way in this thesis. What we can do and actually should do instead is to ensure that the game is balanced. As we have seen in the previous paragraphs, balance ensures an equilibrium, excludes human disparities by design, and focuses on the system itself (also cf. Figure 4.1). Nevertheless, we cannot completely exclude the human from this process. In the end, it is still up to humans how they respond to the intended balance, perceiving fairness from their individual point of view. For this reason, we will conduct an additional empirical evaluation of the automated balance with human playtesters in Section 5.5.

4.2 A Balance Metric for a Competitive Two-Player Game

This section addresses research question RQ 1, which aims to find a foundation for quantifying balance automatically with a data-driven approach. This is our contribution C 1.

Still, a challenging question is how to actually measure whether a game is properly balanced, or even better, how to numerically express *how* balanced a game is in its initial state. Related works [Volz et al., 2016; Preuss et al., 2018; Morosan and Poli, 2017] suggest using simulations of the game to collect data that can then be used for metrics. However, a limitation of these metrics is that they are domain-specific and contain highly game-dependent information. These metrics, often hand-crafted for a particular game, may also distort the measured balance if certain aspects included are overestimated or misjudged. For these reasons, it is difficult or even impossible to transfer them to other games. Therefore, we aim to develop a method that does not contain any domain-dependent information, allowing the metric to be transferable to other games and settings.

The first step in automating game balance is to automate the balance estimation of a game level with a metric in order to process it algorithmically. Related work proposes heuristic approaches [Lanzi et al., 2014; Lara-Cabrera et al., 2014] where the balance is computed from game-specific information, such as the states of the players or spatial distances of game entities within the level (cf. Section 3.2.2). Since these approaches contain domain-specific information, they are not transferable to other games and additionally rely on the quality of hand-crafted rules within the metric. The latter can be in particular an issue, if a designer under- or overestimates the impact of a game parameter, which can lead to an imprecise assessment of the balance. In addition, it is important that this metric is robust in terms of scatter in order to provide comparable and reproducible results that are not too influenced by probabilistic elements within the game.

To address this shortcoming, we propose the use of a more general, game-domain-independent balance metric, which by design treats a game as a black box, depending only on how often each player wins. In Section 1.1.1, we have seen, that a quantifiable outcome is essential to making a game a game. Therefore, we only require that the game has at least one winner at the end to fit for our method. Since draws can also provide information about the balance, our metric takes this into account as well. We will discuss the influence of draws on our proposed metric in the context of results in Chapter 6.

In order to automate this process, human playtesters are not an option, and thus we stick to deterministic heuristic agent-based simulations of the game. To replicate human behavior, the artificial agents should behave in a comparable way to humans. Their configuration, however, is independent of the approach and can be seen as a configurable parameter of the method. Instead of heuristics, the usage of other approaches, such as Monte Carlo Tree Search (MCTS) or RL agents, would be an option, but we argue that using deterministic agents that follow an interpretable heuristic is a better option for a robust and reproducible balance estimation. Since these heuristics significantly influence the artificially estimated balance, they should be carefully evaluated. We will evaluate the generated levels and their balance later with human playtesters in an empirical study in Section 5.5.

Furthermore, the estimated balance will solely depend on the heuristic used in the simulations. We will thus show that heuristics can easily by exchanged. Additionally, agents controlled by different heuristics can be used at once in order to balance asymmetric player archetypes for instance (Section 6.2.2).

4.2.1 Derivation from the Statistical Parity Metric

As we saw in Section 4.1, an equilibrium must be established between two entities in order to create a balance. An existing metric that aims to ensure this is the *Statistical Parity* metric [Dwork et al., 2012], which is used, for example, in the fair machine learning community. Part of this research community is the evaluation of fairness in automated decision-making [Barocas et al., 2023], such as algorithmic fairness [Chouldechova, 2017]. Since it puts the human into the center and asks, if the automated decision is the same for e.g., *all* genders, it is used in the context of a collection of metrics known as *fairness metrics*. An overview

of these metrics is given by Makhlouf et al. [2021]. In that context, the Statistical Parity metric (Equation 4.1) can measure the fairness between two groups, e.g., the classification result of a binary classifier for two groups $G = 1$ and $G = 2$:

$$P(\hat{Y} \mid G = 1) = P(\hat{Y} \mid G = 2) \tag{4.1}$$

The decision for the two groups is considered fair if the conditional probability of the same result $\hat{Y}$ is the same for both groups. For example, if men and women are treated equally by an automated credit scoring system, the conditional probability of getting a loan must be the same for both groups. In other words, the decision must not depend on a person's gender to be creditworthy.

In terms of game balance, we want to make sure that the outcome (here the chance of winning) is the same for both groups (the players) and is not dependent on a particular player. In the context of a level, this can be influenced, for example, by the spatial distribution of game elements. In contrast to other fairness metrics, such as *Equal Opportunity* and *Equalized Odds*, the Statistical Parity solely focuses on the overall accuracy per group (here: win rate per player) and does not take the actual background or qualification of individuals into account [Makhlouf et al., 2021]. Since we aim to treat both players independently of factors like their individual skill or player archetype (cf. Section 6.2.2), the Statistical Parity metric is the most appropriate fairness metric for this problem. As we use it to create an equilibrium between the win rates of heuristic player agents, we can speak of a balance metric.

4.2.2 Metric Design

In this section, we transfer the idea expressed in the *Statistical Parity* fairness metric to the domain of game balancing. However, the direct use of Equation 4.1 is not applicable for a search-based or RL-based approach, since a numerical representation of the balance quality is required for each time step t. We therefore define the win rate $w_{p_i t}$ per player $p_i \in \mathcal{P}$, where $\mathcal{P} = \{p_1, p_2\}$, as:

$$w_{p_i t} = P(w_t \mid p = i), \quad i = 1, 2\,. \tag{4.2}$$

$w_{p_i t}$ is then inferred from the results of the n-runs of the agent-based simulations per step. This can be considered as a sampling from the true win rate distribution. Second, we additionally want to make the metric design configurable to a certain balance state b and thus rewrite the balance calculation b_t as:

$$b_t = |\, w_{p_1 t} - b \,|, \quad 0 \leq b \leq 1\,. \tag{4.3}$$

Due to the use of the absolute value, $w_{p_i t}$ of both players could be used in Equation 4.3, since their values describe probabilities that sum up to 1. b_t is defined in $[0, 1]$ as well, where 0 indicates that b_t is exactly equal to b; 1 indicates the maximum deviation from b. In the example where an exact balance between both players is desired ($b = 0.5$), $b_t = 1$ indicates that a particular player wins every game. With b_t in this range, the metric can be adjusted flexibly, for example, to also favor or disadvantage a particular player. In addition, since intermediate rewards that compare t to $t - 1$ have shown good results in

PCGRL, we include this design here as well. We will apply this metric in the context of our adaption of PCGRL (Chapter 5 and 6) and thus refer to it here as a reward. Finally, the reward r_t is:

$$r_t = b_{t-1} - b_t + \alpha\,. \tag{4.4}$$

To reward the RL agent additionally, a reward α is given if b_t is exactly b. Otherwise, α is 0. If $\alpha > 0$, the episode always ends since $b_t = b$. As a result, the reward will be positive if the agent improves the balance state, negative otherwise. For no impact on the balance state, the agent does not receive a reward (value 0). Using this reward design, the RL agent is gradually incentivized to reduce the absolute difference from the current balance state to the desired one.

4.2.3 Estimation of a Suitable Number of Simulations

Even when simulating with the same deterministic heuristics, the winners may be different each pass if the game environment contains probabilistic mechanics. Probabilities are important mechanics that make games interesting, such as dice rolling or card drawing [Schreiber and Romero, 2021]. A player can win with luck, but when playing many times, skill should make the difference, otherwise it is a gambling game. Since two simulation runs of an identical level with the same deterministic heuristics can produce different results, the question arises as to what number n of simulations is a good number to minimize the noise in the probability and thus produce robust results. Since the simulations are computationally intensive, it is of additional interest to find the number of minimum runs where the results vary at an acceptable level.

We approach this by investigating how much the average win rate $\overline{w}_n$ of a given number of simulations n differs from that of $\overline{w}_{n+2}$. Only even values of n are applicable, since otherwise a balanced game is not possible (unless there are draws). We increase n up to $N = 30$ and use the same set of levels $\mathcal{S}$, with $|\mathcal{S}| = 500$ for all values of n. The average deviation μ_n of a particular n is expressed as follows:

$$\mu_n = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}}^{\mathcal{S}} |\, \overline{w}_{s,n} - \overline{w}_{s,n+2} \,|, \quad n \in 2\mathbb{Z},\ n \geq 2\,. \tag{4.5}$$

We show the course of the number of simulations with two identical heuristics for the environment Feast & Forage (Section 4.3) in Figure 4.2. We will use this setup in Chapter 5 and 6. In addition, the standard deviation is given with σ and 2σ. It is clear that the larger n, the smaller μ_n. To determine a reasonable value for n, we set the threshold: $\mu_n + \sigma < 0.05$. For this environment, this is $n \geq 14$. Therefore, we use $n = 14$ in this thesis for level balancing with the Feast & Forage environment. This method can be used to determine n for any competitive game environment. We will also use it to determine an appropriate number of simulations for a different environment in Section 6.3.

4.2.4 An Acceptable Range of Win Rates

So far, we have only considered the balancing process to be successful if the metric exactly matches the predefined balancing goal (e.g., $b = 0.5$). In many

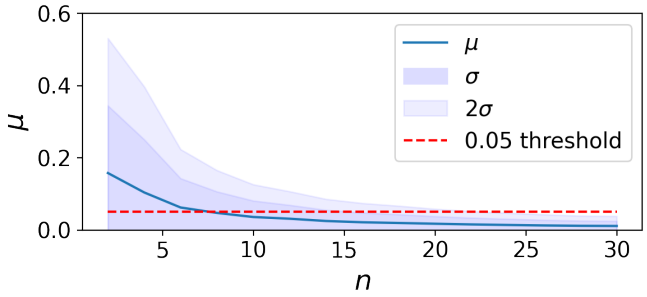


Figure 4.2: How many times n should we run the simulation to approximate the balance state? We figure that out by calculating the mean deviation μ_n of win rates from $n-2$ to n for the investigation of fluctuations in win rates. The Figure shows the course for the *Feast & Forage* environment (Section 4.3) which we will use in the Chapters 5 and 6.

cases, however, it is not possible to measure a certain value exactly by sampling. Although we have examined how often we should sample to find a balance between the number of simulation runs to minimize the uncertainty in estimating the actual game balance, there is still a residual uncertainty. This is also a problem that often arises when collecting data about the real world, such as surveys. To address this issue, a solution is the introduction of a bias and define the optimal value not as a single value, but as a range.

The *Disparate Impact* metric [Feldman et al., 2015; Saleiro et al., 2018] expresses fairness with the ratio of a predicted outcome of two groups. The fairest ratio here would be represented by a value of 1. A commonly used range for this metric to consider an outcome as fair is ≥ 0.8 as noted by Saleiro et al. [2018]. That being said, applied to game balancing, values in a range to the desired value can be considered as balanced. For this work for instance, we could then define a balanced game state as: $1-(w_{p1}-w_{pt}) \geq 0.8$. Introducing this bias would increase the range of levels considered balanced, resulting in an even better proportion of balanced levels in our method. Since this bias has a strong influence on this proportion, it should be adapted to the game domain in terms of residual uncertainty when transferred to game balancing. For example, in the context of Section 6.3 – where we aim to balance paths for asymmetrical player setups in a city environment – a perfectly fair ratio of 1 is not feasibly.

4.3 Game Environment: Feast & Forage

Both Chapter 5 and 6 focus on balancing game levels through PCG in order to create balance through the spatial distribution of tiles. Although many different game research environments have been introduced in the scientific community and literature, they focus on the research of game-playing agents [Hu et al., 2024; Bamford, 2021]. To the best of our knowledge, there is currently no research environment for game balancing through PCG. For our research, we require a tile-based, competitive environment that contains probabilistic game elements,

and where, most importantly, spatial tile placement has a significant effect on the overall balance.

An existing research environment that comes as close as possible to the listed requirements is the Neural Massively Multiplayer Online (NMMO) environment [Suarez et al., 2019]. The NMMO environment simulates a typical setting in the MMO game genre, where multiple players of different character classes can forage for resources, craft items, enhance their character, fight, or trade items at a market. This provides a great environment to test and explore various aspects of balance, such as asymmetries or level design. NMMO has been developed for the study of intelligent multi-agent systems, and there have been several competitions using NMMO at the NeurIPS conference[5] for instance. NMMO also employs a PCG method based on Perlin noise [Perlin, 1985] to generate the game world for each round. However, its focus is not on balance, but on providing a diverse environment to encourage agent generalization in a multi-agent content.

Since our focus is on creating a method for generating levels that are balanced entirely through the level design, the NMMO environment contains too much content that adds unnecessary complexity. To better control the actual impact on balance, we focus on the competitive race for resources and their spatial accessibility as the key mechanic. Therefore, we are disabling parts of NMMO's functionality, such as combat, crafting, and trading items on a global market, and limiting the number of players to two. Along with this thesis, we thus introduce the *Feast & Forage*, which is set up within the NMMO environment, in order to test and evaluate the level balancing methods developed in this thesis. While an environment with a similar idea called *Foragers*[6] exists on the Griddly platform [Bamford, 2021], it is cooperative, whereas Feast & Forage is competitive.

In the following paragraph we will describe the rules for Feast & Forage which are used in this chapter and we consider to be the basic game. In Chapter 6 this basic game will be adapted for additional studies, such as on asymmetric balancing by changing the winning conditions, or which tile types block the movement for a particular player. Table 4.1 provides an overview of all tiles in the basic game, including their functionality and sprites[7]. Figure 4.3 shows an example level of the game.

Environment: Tiles are limited to the seven types: grass, rock, water, food, scrub, player one, and player two. Rock and water tiles impede movement.

[5] NeurIPS NMMO Competition 2023: `https://neurips.cc/virtual/2023/competition/66597`

[6] Griddly – Foragers: `https://griddly.readthedocs.io/en/latest/games/Foragers/index.html`

[7] The grass, water, and stone sprites are taken from the *RPG Nature Tileset – Seasons* by Stealthix, available at `https://stealthix.itch.io/rpg-nature-tileset`, licensed under CC0 1.0. The food and scrub sprites are created from the *Pipoya Free RPG Tileset 32x32* by Pipoya in combination with the grass tile from the previous one. It is available at `https://pipoya.itch.io/pipoya-rpg-tileset-32x32` and is free for commercial and personal use; redistribution is not permitted. The player tiles have been created by Alessandro Puddu in the context of our joint paper, also using the grass tile as background.

Name	Blocks	Resource	Sprite
Grass	No	No	
Rock	Yes	No	
Water	Yes	Yes	
Food	No	Yes	
Scrub	No	No	
Player 1	No	No	
Player 2	No	No	

Table 4.1: Overview of the tile types and their properties in the basic game.

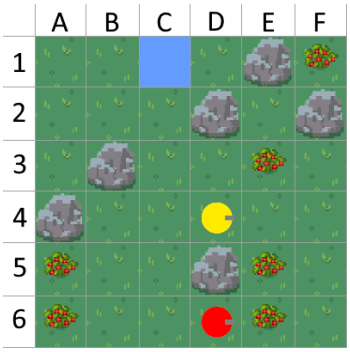


Figure 4.3: Example level.

Players win by either collecting five food resources or by surviving longer than their opponent.

NMMO randomly assigns start positions at the beginning of each game. Since starting positions have, however, a significant impact on a level's balance by defining the player's spatial distance to resources, we add additional corresponding tiles. Moreover, this allows to directly influence the balance by placing the players tiles in the level. The starting positions of both players are always on grass tiles and are indicated by the red and yellow player tiles. Beginning with this chapter, we use a single tile to represent both players. However, as we will see in Chapter 6, we improve the method's performance by assigning each player its own tile.

Players take their turns simultaneously and can choose from five actions: move one tile up, down, left, or right, or to do nothing. Like in NMMO, a player's state is defined by their spatial 2D position on the level, and their health, water, and food levels. On each turn, players lose water and food, with health depleting when both reach zero. We use the same values as in NMMO, an overview of these values and their function is given in Table 4.2. If a player's health is zero, he has lost immediately. Players can replenish their food level by consuming food tiles by moving onto them, which then become scrub tiles. There's a 2.5 % chance per turn that scrub tiles will respawn as food. Water can be replenished by moving onto tiles adjacent to water tiles, which are never depleted. Health is gradually restored when food and water levels exceed 50 %.

4.4 PCGRL: Procedural Content Generation via Reinforcement Learning

In this thesis, we use, adapt, and extend the PCGRL framework [Khalifa et al., 2020] for generating and balancing levels (Chapters 5 and 6) and generating graph data (Chapter 7).

In PCGRL, PCG is formulated as a sequential decision-making task to maximize a given reward function, where semantic constraints can be expressed and thus, no training data is needed. In the context of games and context generation, this is a huge advantage, because when new games are created, there is often no

Table 4.2: Overview of the used configuration values in the basic version of the game environment Feast & Forage.

Description	Value	Additional information
Maximal Health	100	Health is initially at maximum.
Maximal Food	100	Food is initially at maximum.
Maximal Water	100	Water is initially at maximum.
Food Drop	10	The loss of food per turn.
Water Drop	10	The loss of water per turn.
Health Regeneration	10	Per turn if water and food levels are $\geq 50\,\%$.
Water Regeneration	Maximal Water	Set water to maximal water value if a player is adjacent to water.
Food Regeneration	Maximal Food	Set food to maximal food value if a player moves on a food tile.
Victory points (Collected Food)	5	The number of food resources required to win.
Food Respawn Probability	2.5 %	The probability of a scrub tile transitioning back to a food tile.

data from which to train a model. A level is represented as an integer matrix, where each integer denotes a distinct tile type. PCGRL modifies the level by changing tiles within this matrix.

To apply RL, the PCG problem is framed as an MDP. Therefore, PCGRL introduces three different MDP representations for level generation which differ in their definition of the action $\mathcal{A}$ and observation space $\mathbf{S}$. For each, $\mathcal{A}$ is a discrete set and $\mathbf{S}$ is a matrix with a one-hot encoded representation of the integer level matrix. Across this work, we will introduce new swap and graph representations based on the existing representations, and show that they are better tailored to the specific problem domain. The original representations in PCGRL are:

- **Narrow:** This representation randomly selects a tile in the matrix and the agent can only decide what type of tile to place on the selected position. It has a small action space because it consists only of the different types of tile. For an environment with n unique tiles the action space would results in a size of $|\mathcal{A}| = n$. The observation space is cropped around the randomly selected tile position where an action is to be taken. The crop size c is configurable which then results in the matrix $\mathbf{S} \in \{0,1\}^{(2c+1)\times(2c+1)\times n}$. To yield a matrix in the correct shape when a border position is selected, a tile type for padding must be specified.

- **Turtle:** The turtle representation starts at a random position and then allows the agent to move to an adjacent position on the map and then to decide what type of tile to place on the current position. The advantage of the turtle representation is that the agent is not restricted to the randomly assigned position and can therefore learn where to move next. Its action space is $|\mathcal{A}| = 4 \cdot n$. The observation space is the same as for the narrow representation and cropped around the current position of the agent on the map. Additionally, the *warp* mode can be enabled or disabled. When enabled, if the agent selects an action that would move it outside the matrix, its position is set to the maximum value along that dimension. When disabled, the agent's position remains unchanged.

- **Wide:** The wide representation gives the agent full control over the level generation process, as the agent can decide which tile of the whole grid should be changed. This greatly increases the action space, as each position of the grid multiplied by the number of tiles represents an action. This is the most human-like representation, as the agent can change everything directly according to a plan it has constructed. The size of the action space for a level with the dimensions k and l is $|\mathcal{A}| = (k \cdot l)^2 \cdot n$. The observation space is the full level resulting in the matrix $\mathbf{S} \in \{0, 1\}^{k \times l \times n}$.

An episode ends when the level is valid, or a specific number of changes or steps are exceeded. Each modification to the level matrix is considered to be a change. Restricting the agent to a certain number of possible changes is crucial to stabilize the training for better convergence, and also to ensure content diversity in order to prevent the agent from always ending up with the same level. This parameter, called the change percentage, should depend on the level size, and according to Khalifa et al. [2020], 20 % is a good value. For level balancing we will deal with 6×6 levels, resulting in eight changes (rounded up).

There have been several works which adapt PCGRL such as for controllability [Earle et al., 2021], 3D levels [Jiang et al., 2022], scalability [Earle et al., 2024], and the tailored generation of reward functions using a Large Language Model (LLM) [Baek et al., 2025a]. For more details on PCGRL's related work, see the the related work chapter, Section 3.1, specifically the paragraph on PCG with RL.

4.5 Game Economy Simulation Framework

Based on the existing formal definition by Klint and van Rozen [2013] a game economy can be considered as a directed graph. It defines how resources of different types are generated and transitioned to other resource types. A basic introduction of game economies has been given in Section 1.1.3. We only deal with fixed economic systems, where all resources come from the economy itself. In this context, nodes in that graph represent different functional components such as the creation of resources or conversion to different types. Weights on the edges describe how many resources flow from one node to the adjacent one. Depending on the node type, the weights are absolute values or probabilities.

For this thesis, we will use a subset of five different node types in our framework of all available nodes defined by Klint and van Rozen [2013]. To create valid and executable economy graphs, we establish the set of constraints $\mathcal{C}$ similar to as in the context of Graph Procedural Content Generation via Reinforcement Learning (G-PCGRL) (cf. Section 7.3.2), specifying for each node type the permissible connections to other node types (Table 4.3). The values for maximum (max) and minimum (min) output sizes are each chosen to keep the economies more manageable. However, these values could also be changed to allow for more complex economies (e.g., increase max output). The different types of nodes are:

- **Sources:** Sources are entry points creating resources and adding them to the economy.
- **Random gates:** A random gate distributes incoming resources based on the probabilistic weights of its outgoing edges. It must be ensured that the sum of the weights of all outgoing edges equals one. This can be used to model e.g., critical attacks or random drops.
- **Pools and fixed Pools:** Pool nodes have an intern memory to store incoming resources. They serve as buffers for outputs from sources, random gates, or converters or as end points of the economy. For the analysis of the economy, we can monitor the fluctuations of the resources within pools over time. Fixed pools, a subform of a pool buffer a maximum of the number of resources equal to the highest outgoing weight. This is particularly useful when modeling ability cooldowns.
- **Converters:** A converter transitions one or multiple incoming resources to one outgoing resource.
- **Drains:** Drains permanently remove resources from the economy. As with pools, drains can be monitored. Their main distinction from pools is that they do not allow outgoing connections, thereby creating a termination of the economy.

An example of how to model an existing economy from the sandbox game Minecraft [2011] is given in Figure 4.4a. It shows the torch crafting process in an automation setting where specific amounts of resources are added to the economy via sources per time step. Using pool and converter nodes we can define this economy for torches from wood and coal sources. While coal can be directly used for crafting torches, the wood resource must first be converted into sticks. According to the original implementation[8], the conversion to sticks yields four sticks per two wood entities. In this example, other resources such as the time to collect resources or the need for a crafting table for resource conversion are neglected.

[8] `https://www.minecraft-crafting.net/`

Table 4.3: The set of constraints $\mathcal{C}$ for Game Economy Evolution (GEEvo) game economies: An overview of the different node types, including their allowed connection types and the permissible number of connections to other node types.

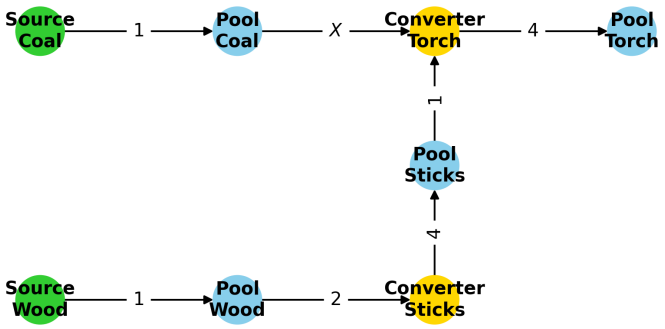
Node types $\mathcal{N}$	**Max in**	**Max out**	**Min out**	**Min in**	**Color**
Source	0	3	1	0	green
Random Gate	1	3	2	1	red
Pool	2	3	0	1	blue
Converter	3	1	1	1	yellow
Drain	2	0	0	1	orange

Node types $\mathcal{N}$	**Allowed inputs**	**Allowed outputs**
Source	-	Pool, Random Gate
Random Gate	Source, Converter	Pool, Converter
Pool	Source, Random Gate, Converter	Converter, Drain
Converter	Pool, Random Gate	Pool, Random Gate
Drain	Pool	-

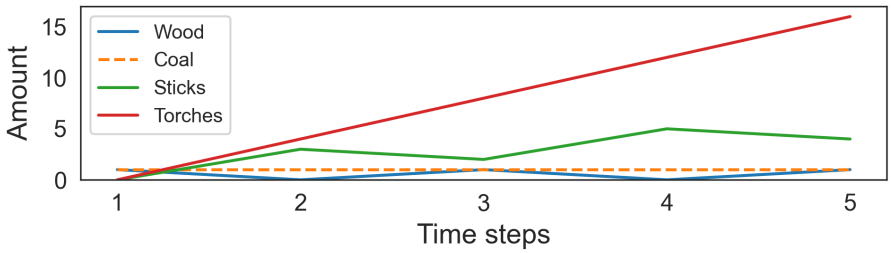
4.5.1 Execution of Game Economy Simulations

The directed graph G of a game economy can be denoted as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices (nodes) and $\mathcal{E}$ is the set of the edges connecting vertices. For each vertex $v_i \in \mathcal{V}$ a subset $\mathcal{A}^{v_i} \subset \mathcal{E}$ exists containing all outgoing edges. The types of all allowed node types (such as source or pool) are described in the set $\mathcal{N}$ (cf. Table 4.3). The execution of the graph economy starts by iterating over the subset of all source nodes $\mathcal{S} = \{v \in \mathcal{V} \mid \tau(v) = source\}$ and then recursively executing all connected nodes. For each v_i all edges $e_j \in \mathcal{A}^{v_i}$ are executed by a function $h_\tau(v_i, e_j)$, $\tau \in \mathcal{N}$, h_τ respectively to the type τ of v_i.

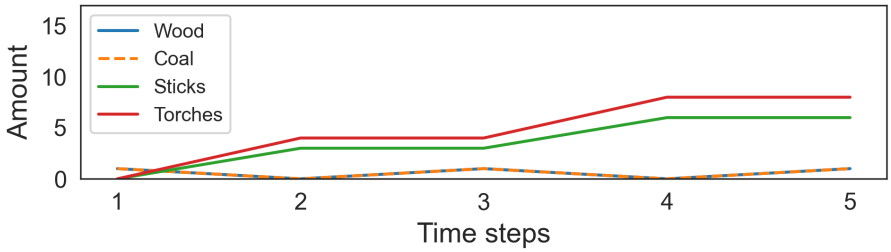
We plot two example courses of simulations of the previously used example with different weights of the economy in Figure 4.4a in the Figures 4.4b and 4.4c. For this example, we assume both resources, wood and coal, to create one of each per time step. By changing the needed amount x for coal to converse coal and sticks to torches, this example demonstrates how small changes to a single weight can impact the whole economy. While e.g., for $x = 1$ the number of torches grows linearly per time step; for $x = 2$ it is gradual. Also, the curve progressions of available wood and coal is different.



(a) Example economy from the game Minecraft for crafting torches from basic wood and coal resources.



(b) Simulation 1: $x = 1$ (original value).



(c) Simulation 2: $x = 2$ (value for demonstration).

Figure 4.4: Small changes can have a huge impact: An example of a game economy using the proposed framework and two simulations of it, each with a different configuration. The graph (a) shows the economy from the game Minecraft to craft torches from wood and coal in an automation setting. The Figures (b) and (c) show the monitoring of the pool nodes simulating the economy in (a). By only changing the amount of coal needed to craft torches, the entire economy behaves differently.

Chapter 5

Level Balancing via Procedural Content Generation and Reinforcement Learning

This chapter is based on and extends the author's publications:

- **Florian Rupp**, Manuel Eberhardinger, and Kai Eckert. Balancing of competitive two-player Game Levels with Reinforcement Learning. In *2023 IEEE Conference on Games (CoG)*, pp. 1-8, Boston, USA, 2023. doi: 10.1109/CoG57401.2023.10333248 .
- **Florian Rupp**, Alessandro Puddu, Christian Becker-Asano and Kai Eckert. (2024). It might be balanced, but is it actually good? An Empirical Evaluation of Game Level Balancing. *2024 IEEE Conference on Games (CoG)*, Milan, Italy, pp. 1-4, 2024. doi: 10.1109/CoG60054.2024.10645642 .

Partial results in this chapter are included from:

- **Florian Rupp**, Manuel Eberhardinger, and Kai Eckert. Simulation-Driven Balancing of Competitive Game Levels with Reinforcement Learning. *IEEE Transactions on Games (ToG)*, vol. 16, no. 4, pp. 903–913, 2024. doi: 10.1109/TG.2024.3399536 .

This chapter is about the automated generation and balancing of competitive two-player tile-based game levels. We address the research questions RQ 2 and RQ 3. In particular, we accelerate automated level balancing (RQ 2), and evaluate the human perception of balance (RQ 3). In this context, we apply and evaluate our approach to automatically estimating a game's balance simulation-driven (C 1).

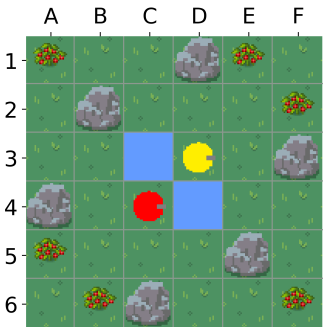
We answer these research questions with our contributions C 2 and C 3. C 2 presents an architecture that formulates level balancing as a Markov Decision Process (MDP). With C 3, we provide an empirical evaluation of the automatically balanced levels with human playtesters. However, we also report several shortcomings and limitations, that motivate a deeper investigation of the proposed method in Chapter 6. It will further contribute to answering RQ 1 and

RQ 2 with additional applications in the problem domain, presenting further insights and results.

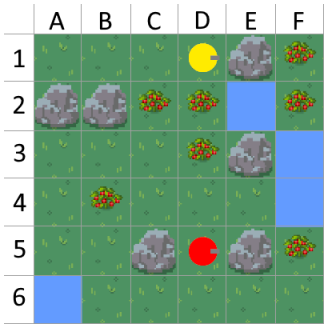
5.1 Overview and Motivation

Level design is a key concept when creating games. In order to keep players engaged, a balance must be struck between a challenging and enjoyable experience. This is generally not an easy task because it depends on two factors: the game environment itself and the skill and experience of the players [Schreiber and Romero, 2021]. In addition, each game is unique by design. An integral part of the entertainment potential of a game are its unique rules and its narrative setting, or both in combination, and how this interacts with the environment e.g., the level design. Furthermore, game levels for competitive multiplayer games must be designed to be balanced towards equal initial win chances for all players. Imbalanced levels will lead to boredom or frustration, and players will quit playing [Andrade et al., 2006; Becker and Görlich, 2020]. For these reasons, level balancing remains a challenging and time-consuming part of the entire development process requiring a lot of manual work and playtesting.

To ensure balance through level design, game designers often rely on nearly (point) symmetric map architectures. This can be seen in popular competitive e-sports titles such as League of Legends [2009] or Starcraft II [2010], but also in other competitive tile-based games such as Advance Wars [2001] or Bomberman [1983]. In addition to symmetrical levels, alternative approaches are also possible. Non-symmetrical levels offer more variety and can create new ways for playful creativity to be entertaining and challenging. Figure 5.1 contrasts a symmetrically designed level and an asymmetrical level which are both balanced in order that both players (red and yellow) can win equally often. Whereas symmetrical levels can be easily designed by mirroring part of the level for the second player in order to ensure balance, asymmetrical levels must be thoroughly designed and tested.



(a) A symmetrical level.



(b) An asymmetrical level.

Figure 5.1: A symmetrical and an asymmetrical level: Unlike symmetric levels, asymmetric levels require an increased deal of design and tweaking for balance, but can offer more variety in the gameplay, since players' turns cannot be the same every round.

Since in this chapter we are only affecting game balance through level design, we frame this problem as a Procedural Content Generation (PCG) task. There are a lot of works on PCG for games (cf. related work on PCG in Section 3.1), but creating balanced content is more complex than just creating a playable level, since it requires a deeper understanding of the overall game, its rules and mechanics, such as win conditions and so on. Among other things, a PCG method should serve content validity, diversity, and should generate the content fast [Togelius et al., 2011a]. Recently, the PCGRL (PCG via Reinforcement Learning (RL)) framework [Khalifa et al., 2020] has been introduced to generate tile-based game content, framing this problem as a MDP. Applying RL to PCG has the advantage that once the model has been trained, content generation is very fast. To estimate balance, we will use the simulation-based method for balance estimation proposed earlier in Section 4.2. Since simulating the game multiple times to measure the game's balancing state is computationally intensive, RL helps to reduce computational costs by avoiding unnecessary inference steps since the search space does not need to be explored from scratch each time.

In order to combine RL, PCG, and simulations to balance game levels, we will introduce an architecture and a new definition of an MDP, where the model optimizes an existing level by swapping tile positions. This introduction raises new questions and concerns that will be systematically explored and answered:

- Which tile types have more or less influence on the balance? Therefore, we analyze the model's behavior in the balancing process.
- The heuristics for the agent-based simulations must be designed carefully. Since games are made for humans, they need to be balanced for humans, so the heuristics need to replicate human behavior as closely as possible. How is this to be determined?

The latter will be answered by an empirical evaluation involving human game testers who will be asked how they perceive the intended balance. Since a game's balance is multifaceted on multiple layers, human perception of balance can depend on different aspects that are highly subjective and moreover, difficult for humans to express objectively and numerically. We will therefore break down the overall balance into smaller and more tangible items. Furthermore, we will use a comparative survey design, as it is much more easier for humans to express differences between two states [Thurstone, 1927]. More specifically, participants are asked how they perceive differences in a particular balance aspect of a level version one and two. To prevent participants from being influenced by prior knowledge, they do not know which version of the level they are playing. In order to analyze the data, we propose an approach to convert the relative values of the comparison into absolute values. Using descriptive statistics and hypothesis testing, we show that for most of the levels tested, humans perceive a significantly better or even balance for certain aspects after applying the proposed balancing method to a previously imbalanced level.

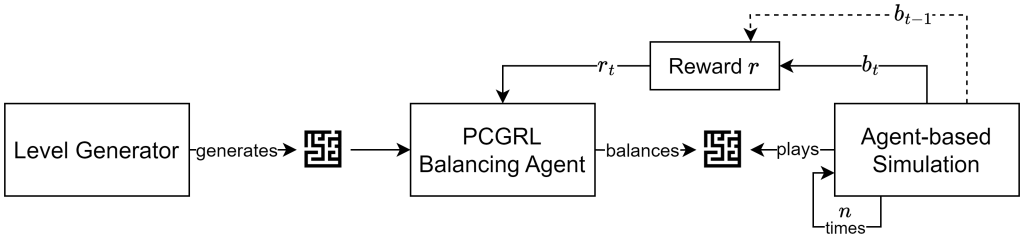


Figure 5.2: Overview of the balancing architecture. It is separated into three units: A level generator, a level balancing agent, and a game playing simulation. In the latter, the game is simulated by playing it n-times with heuristic player agents. The reward r_t for training the balancing agent is computed from the balance states b_t and b_{t-1} of the simulations. The reward has been defined in the context of the foundational concepts in Section 4.2.2.

5.2 Balancing Architecture

This section and the following ones address RQ 2 in order to accelerate automated level balancing. This is our contribution (C 2). As we identify limitations and areas for further exploration, we will continue to strengthen our response to RQ 2 in Chapter 6.

To automate the process of balancing tile-based levels, we propose an architecture consisting of three separate parts: a level generator, a balancing agent, and agent-based simulations (Figure 5.2). The core idea is to divide the level generation and balancing process into separate units. We will show that this approach yields better results than doing both in the same step. Moreover, this gives, game designers for instance, the opportunity to design a level which can be automatically balanced. With this architecture, level balancing is framed as a PCG task and can be seen as a fine-tuning for balance of existing content.

The level *generator* constructs a playable level from random noise as input which is then fed to the balancing agent. In this work, we use PCGRL for the generator, but any other PCG method will serve comparable at this point. The generator is trained separately before training the balancing agent.

The core idea of level balancing is not to generate a new level but to modify the given input level to improve the overall balance. For more information on balance estimation see Section 4.2 At each time step, the balancing agent can decide to swap the positions of two selected tiles. We will introduce the swap-based representation pattern in Section 5.3. If a swap has been made, the level is played n times in a agent-based simulation by heuristic player agents to collect data for the balance estimation. Subsequently, the balancing agent is rewarded based on how this action affected the balance state in simulations (Section 4.2.2). The simulations can therefore be understood as the basis for a static simulation-based evaluation function as described by Yannakakis and Togelius [2011]. More details are given in the implementation details (Section 5.2.1).

Classification within the PCG taxonomy: According to the PCG taxonomy by Yannakakis and Togelius [2025b] (described in Section 2.1), our balanced level

generation method can be classified as follows: In terms of content, it is *necessary*, *functional*, and *spatial*. In terms of the method, it is *stochastic*, *generate-and-test*, *trained*, and *iterative*. Although it is not designed for controllable RL, it supports *controllability* to the extent that both generated and user-defined levels can be passed to the balancing agent to control the output. In terms of its role, our method can generate content both at *runtime* and offline and operates *autonomously*. In practice, adaptivity is often blurred, as is also the case here. Although it does not explicitly model player experience or use player data, it incorporates player behavior via an Agent-based Modeling (ABM) in the simulation step for balance estimation.

5.2.1 Implementation Details

In this subsection, we provide more detailed information on the three individual components of the architecture.

Level Generator

The task of this unit is to generate playable levels for the subsequent balancing using a model trained with the PCGRL framework. The generator takes in a random integer matrix (the level) as candidate level p from the set of all possible levels (parameters) $p \in \mathcal{P}_{noise} = \mathcal{T}^{k \times l}$. The set $\mathcal{T}$ is the set of all possible tile types a level of the size $k \times l$ can be constructed from (cf. Table 4.1). The reward for the training process is designed to ensure the validity (a playable level) of the producible content $\mathcal{O}_{playable}$. This can be understood as incentivicing the agent to satisfy the constraints in the set $\mathcal{C}_{gen}$. It requires having exactly two players and a valid path between them. To ensure direct competition, the latter constraint ensures that both players have access to the same area of the game level. Additionally, it prevents single players from being locked behind stone walls. In the context of RL, a policy π is trained to manipulate p in order to satisfy all constraints in $\mathcal{C}_{gen}$. The set of possible distributions (the models) of playable content is then $\mathcal{D}_\pi$. This approach can formally be described as the stochastic PCG method (cf. Section 2.1):

$$PCG^{RL}_{level} \colon \mathcal{P}_{noise} \times \{\mathcal{C}_{gen}\} \to \mathcal{D}_\pi(\mathcal{O}_{playable})\,. \tag{5.1}$$

If there is a valid path between both players, the agent receives a positive reward, otherwise a negative one. Additionally, at each step the agent is rewarded with the difference of the players that are and should be (similar to Khalifa et al. [2020]). We get the best results using the *wide* representation. An episode ends when both constraints are met, or when the agent exceeds a fixed number of permitted steps or changes. After training, we get a policy π_θ with the parameters θ, yielding the distribution $d_{\pi_\theta} \in \mathcal{D}_\pi$. A playable level o_p is then generated by rolling out π_θ in the generator environment given a particular random initial level $p \in \mathcal{P}_{noise}$. This is defined as sampling from d_{π_θ}:

$$o_p \sim d_{\pi_\theta}(\cdot \mid p)\,. \tag{5.2}$$

The model can produce levels satisfying the given constraints in 98,7 % levels out of an evaluation sample of 5000. The generated levels achieve a maximum

diversity at 100 % in an automated comparison, so no two levels are completely identical. An example of a generated level is shown in Figure 4.3. The functionality of the game and tiles has been given in Section 4.3.

Level generation in this step continues until the agent has produced a valid level that satisfies the constraints to ensure that the balancing agent receives playable levels only. If a valid level has been generated, it is immediately passed on to the balancing agent.

Balancing Agent

The balancing agent is the core component of the architecture in Figure 5.2. We model the agent as a PCGRL agent to fine-tune a previously generated playable game level $o_p \in \mathcal{O}_{playable}$. To support this process, we use our swapping representations as action space definition for the MDP which we will introduce in Section 5.3. The reward is computed from the results of multiple simulations of the level (Section 5.2.2), thereby implementing the balancing constraint b (cf. Section 4.2.2). The observation is the current level one-hot encoded. In detail, this depends on the chosen action space representation (cf. Section 4.4). For all RL problems, we use Proximal Policy Optimization (PPO) [Schulman et al., 2017] as algorithm in this thesis. We use PPO not only because PCGRL employs it, but also because it is a robust and computationally efficient RL approach, due to its mechanisms for training stabilization and to avoid overly aggressive policy updates [Schulman et al., 2017] (also cf. Section 2.4.1). Additionally, its integration with deep RL enables the learning of a feature extractor and a value function using a (deep) neural network. Since we are dealing with computational intense simulations for reward calculation, its computational efficiency is of particular interest. A description of RL in general (Section 2.4) as well as a detailed one of PPO (Section 2.4.1) can be found in the Background Chapter. Balancing as a fine-tuning process of $\mathcal{O}_{playable}$ with a policy π can formally be described as a stochastic PCG method with:

$$PCG^{RL}_{balance} : \mathcal{O}_{playable} \times \{b\} \rightarrow \mathcal{D}_{\pi}(\mathcal{O}_{balanced}). \tag{5.3}$$

An episode ends when the level is either balanced, or a fixed number of steps or changes is exceeded. In PCGRL, the number of allowed changes is determined relative to the grid size (cf. change percentage, Section 4.4). As in the original PCGRL we use a change percentage of 20 %. We set the value for the number of allowed steps within an episode to 100. After training, the yielded policy π_{ψ} with the parameters ψ generates a balanced level $o_b \in \mathcal{O}_{balanced}$ from a previously generated playable one o_p. This can be formulated as sampling from the distribution (model) $d_{\pi_{\psi}} \in \mathcal{D}_{\pi}$ of balanced levels by rolling out π_{ψ} in the balancing environment:

$$o_b \sim d_{\pi_{\psi}}(\cdot \mid o_p). \tag{5.4}$$

5.2.2 Player Simulation

To compute a levels balance b_t in a time step t for the balancing reward (Section 4.2.2), we run an agent-based simulation n-times of player agents playing the game which can be understood as an ABM (cf. Section 2.2). Using an ABM

here is beneficial to investigate the interactions between the player agents with the game (the environment). As part of the reward function in RL, the necessary data cannot be obtained from the real world, making game simulation essential. The simulation of a game is executed in discrete time steps. Since the simulated environment models the game level, where each agent is assigned a position in all time steps, it is spatially explicit. The environment influences agent behavior both passively by defining static win conditions, and dynamically, by allowing elements like food resources to respawn, making the environment probabilistic as well. In the context of introducing the method for balance estimation, we have determined a suitable n for the Feast & Forage environment as $n = 14$ in Section 4.2.3.

Heuristic Player Agents

The player agents can be any solution that can simulate the behavior of a player with a desired quality. In this thesis, we simulate this behavior with deterministic heuristics. We prefer the interpretable behavior over black box modeled agents, e.g., RL, and the determinism over approaches like e.g., Monte Carlo Tree Search. See the discussion in Section 5.8 for more information. Both contribute to a better control of the heuristic and offer a versatile way for further extension and adaptation, as we will explore in Section 6.2.2. So far in this chapter, we will only use one type of heuristic controlling both players, which is inspired by a baseline agent[9] from a Neural Massively Multiplayer Online (NMMO) competition, which we call archetype A.[10] A description of the heuristic is given in Algorithm 3.

Algorithm 3 Heuristic Agent Archetype A.

```
1:  procedure STEP(gameState)
2:      init action ← DoNotMove
3:      init foodReachable ← FOODREACHABLE(gameState)
4:      init waterReachable ← WATERREACHABLE(gameState)
5:      if foodReachable then
6:          action ← FINDSHORTESTPATHTO(food)
7:      else if waterReachable then
8:          action ← FINDSHORTESTPATHTO(water)
9:      end if
10:     return action
11: end procedure
```

The task of the heuristic is to decide, given the current state of the game, which of the available actions of the environment (see Section 4.3) to choose. Agents are implemented by a STEP function, which is called once per turn for each agent. The heuristic of archetype A is designed to always gather the nearest available food resource using the path-finding algorithms A* [Hart et al., 1968] and Dijkstra [1959] in the helper functions FOODREACHABLE and WATER-

[9] Github: `https://github.com/NeuralMMO/baselines`

[10] We will introduce and experiment with further archetypes in Section 6.2.2.

REACHABLE. When the shortest path to a resource is found, food resources are always favored over water as they have a greater impact on winning the game. If no food resource is reachable or there is no valid path to a food resource, the agent favors the nearest reachable tile that is adjacent to a water tile.

5.3 Swap-based Action Space Representation Pattern

To formulate level generation as MDP, PCGRL introduced three different MDP representations. A detailed description of them is given in the foundational concepts chapter in Section 4.4. In all of them, the agent can decide to replace a tile in a particular position. Since we estimate the balance simulation-driven, PCGRL's approach can, however, lead to unplayable levels at a time step. Furthermore, to move the position of e.g., a player tile somewhere else, the agent would first have to remove the player tile before creating it at a different position. In this time step, the level would not be playable for the player agents, and thus, no reward could be computed in the simulation step. Additionally, the agent would first receive a negative reward due to the number of players is now invalid. The trajectory of a subsequent creation at a different position under the previously given negative reward is more difficult to learn for RL agents.

To address this issue, we introduce a swap-based representation pattern. In these representations, the agent can decide to swap the positions of two tiles per time step. Not adding or removing tiles entirely is a more robust approach to ensuring level playability. Like in PCGRL the action space is discrete and the observation space is a one-hot encoded representation of the level's integer matrix.

However, there may be game domains where multiple tiles have a semantic connection, such as multiple water tiles forming a river. Swapping these tiles around can break playability. Therefore, in the balancing step, we suggest sending unplayable levels back to the generator for repair. This repair is similar to the level-fixing approach seen in [Siper et al., 2022], but would also involve an additional increase in the computational effort. In this thesis, we demonstrate the power of swapping with a simpler domain where there are no semantic relationships between multiple tiles. In addition to ensuring playability, swapping tiles contributes to not simply regenerating a new level from scratch, since that is the job of the level generator. Swapping two tiles of the same type has no effect on balance. Thus, we prevent these swaps to reduce the computational effort. In these cases, the agent is rewarded with 0.

To formulate our approach as MDP we directly adapt PCGRL's narrow, turtle and wide representations [Khalifa et al., 2020] with a swapping mechanism. Like in PCGRL, observations are one-hot encoded in all representations. The detailed description for each swap representation is provided in the following:

- **Swap-Narrow:** At each time step, two random tile positions are presented to the agent, and it can decide to swap the tiles or not. The agent's limited positional control results in a very small action space $\mathcal{A}$ with the only actions: swap or do not swap. The size of $\mathcal{A}$ is therefore $|\mathcal{A}| = 2$. Like in PCGRL, the observation space $\mathbf{S}$ is cropped around the selected position

with a configurable parameter c. The cropped observation around the second position is concatenated to the matrix of the first position. $\mathbf{S}$ is then: $\mathbf{S} \in \{0,1\}^{(2c+1)\cdot 2\times(2c+1)\times n}$.

- **Swap-Turtle:** Starting from two random positions, the agent can decide to swap the tiles at the current positions in each time step. If no change is made, it can decide to which adjacent tiles to move next. As in the original PCGRL turtle representation, staying at a position and not changing a tile is not an option. $\mathcal{A}$ is therefore $|\mathcal{A}| = 4 \cdot 4 \cdot 2 = 32$. The observation space is handled identically to the swap-narrow representation.

- **Swap-Wide:** In this representation, the agent sees the entire level and can freely determine the tile positions and whether to swap them. It can be interpreted as looking at the whole level and then deciding what to move where. A drawback here is the large action space, since it scales twice with the level width l and height k. The size of $\mathcal{A}$ is therefore $|\mathcal{A}| = (k \cdot l)^2 \cdot 2$. In the case of a square grid, as in this work with a size of 6, $\mathcal{A}$ has 2592 possible actions. The observation space remains the same as in the PCGRL wide representation.

  As we will see throughout this work, this representation yields the best results, and we further refine it to reduce the size of the action space and the definition of the observation space (Section 5.6). However, its strong dependence on level size limits the method's scalability for larger levels.

5.4 Experiments and Results on General Feasibility

The evaluation of our method is done in several steps: First, we sample a fixed set of levels to use for direct comparison with the generator. Second, we evaluate the overall performance by comparing the proportion of balanced levels of this set across the three swap representations introduced in Section 5.3 with the original PCGRL method as a baseline (Section 5.4.1). To generate balanced levels with the original PCGRL method, we use the same reward function as introduced in Section 4.2.2.

Third, we investigate the levels the models created (Section 5.4.2) and examine which tiles the models swapped in the level-altering process (Section 5.4.3). The latter provides insights into which tiles actually affect the balance. Finally, we conduct a thorough empirical evaluation with human playtesters in the next section (5.5).

Experimental setup: For all PPO models we use 3-layered multi-layer perceptrons with layers of sizes 64, 128, and 64 for both the feature extractor and the value function respectively. We use the PPO implementation of the Python library Stable Baselines 3.[11] To train such a model, a number of total time steps T must be defined, which, in conjunction with a step size n_{steps} and the number of parallelized environments n_{envs}, determines how many updates $N^{(\pi)}_{updates}$ of the

[11] `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`

policy π are made with how much data from collected trajectories through interactions with the environment. The number of policy updates can be calculated with Equation 5.5.

$$N_{updates}^{(\pi)} = \frac{T}{n_{envs} \cdot n_{steps}} \tag{5.5}$$

We use a step size of 512 and 60 parallelized environments, running on a total of 60 CPU cores, so each environment gets a single core. Using multiple environments in parallel to collect trajectories reduces the overall required computation time only, but does not change the training behavior. For example a T of 10 million steps of training with this configuration would result in 326 policy updates[12], which are 30,720 trajectories for updating the policy via training the neuronal networks per each single update. On the one hand, the number of collected trajectories ($n_{envs} \cdot n_{steps}$) used for policy updates should not be too small, as too little data hinders learning. On the other hand, if it is too large, it leads to an unnecessary computational overhead. In preliminary tests, we found a sweet spot in using a value of 512 in combination with 60 parallel environments. The latter was given by hardware limitations. For all other PPO hyperparameters, such as the learning rate, we use the default configuration.

Since the swap-narrow and swap-turtle representations are derived from their analogs in the original PCGRL, they require cropped observations centered on the selected swap positions (cf. Section 4.4). For both we use a crop size c of $c = 5$, allowing it to see most of the level around the selected position, and a padding value of the tile type grass. We chose the grass tile because it has the least semantic impact on gameplay: it is not a resource, not a player, and does not block movement. Introducing a dedicated empty tile could also be an option, however, this would expand the observation space by an additional dimension and further increase the difference compared to the wide representation, where such a tile is unnecessary. Finally, in the swap-turtle representation, we disallow warping to prevent movements such as shifting from the far left to reappear on the far right.

5.4.1 Performance and Feasibility of the Swapping Representations

We compare the performance of our swap representations against each other. In this section we focus initially on the general feasibility of our approach; in Section 5.7, we present an in-depth comparison with additional baselines, including the original PCGRL and search-based methods.

For the evaluation, we create a dataset of 1000 levels with our PCGRL generator. This dataset is then used for all three models for evaluation. First, we examine the distribution of the initial balance states of the levels in the dataset (see Figure 5.3). Balanced levels are those in which both players win equally often ($b = 0.5$). It is important to note that the distribution is not uniform. Except for the states 0 and 1, the levels seem to be normally distributed around the state of 0.5. Initially, 13.6 % of the levels are balanced. However, peaks towards maximally imbalanced levels can be observed at the outer edges of the distribution. Maximally imbalanced levels make up 26.6 % of the dataset.

[12] Always rounded up to the next integer.

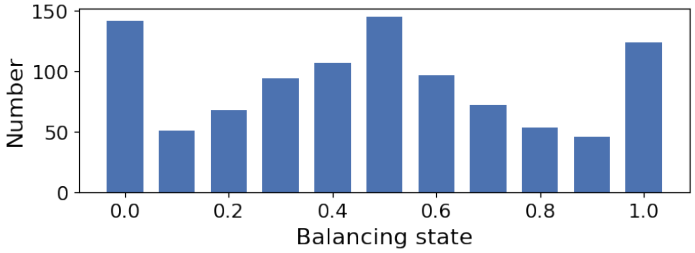


Figure 5.3: Distribution of the initial balance states based on the players' win rates in the generated dataset of 1000 levels. We use this dataset to compare the different representations.

Table 5.1: Performance overview of the swapping representations. Balanced levels are those in which both players win equally often ($b = 0.5$). For each representation, we provide the proportion of these levels in the evaluation dataset. Initially balanced levels were not considered in order to present the true influence of the models.

	Swap-narrow	**Swap-turtle**	**Swap-wide**
Balanced (%)	**48.1**	42.5	**48.1**
Improved (%)	63.3	56.8	**64.1**
Avg. changes	4.6±1.9	4.7±1.9	4.9±1.9
Avg. ep. length	11.3±6.1	25.4±17.7	14.1±7.9
Size action space	2	32	2592

To evaluate the performance of the balancing method we compare the balance state before and after balancing per representation. A general overview is given in Table 5.1. Histograms of the balancing improvement are shown in Figure 5.4. We train all models for the same total number of training steps of 12 million.

The performance of all three swap representations is of comparable quality. Each representation managed to improve the proportion of balanced levels. The swap-narrow and swap-wide representations perform slightly better than swap-turtle in terms of balancing. However, imbalanced levels remain in all results. The largest proportion remains for the balance states 0 and 1. For representations with a higher proportion of balanced levels, the average episode lengths are shorter. This is caused by the remaining imbalanced levels maximizing the episode length. Since the proportion of balanced levels is generally improved, we conclude that our approach is feasible, albeit with limited performance. For this reason, we will further explore our method and improve the results by refining the definition of the MDP (Section 5.6). A thorough comparison to baselines, such as the original PCGRL and search-based approaches (hill climbing and random search), is also provided in this context (Section 5.7).

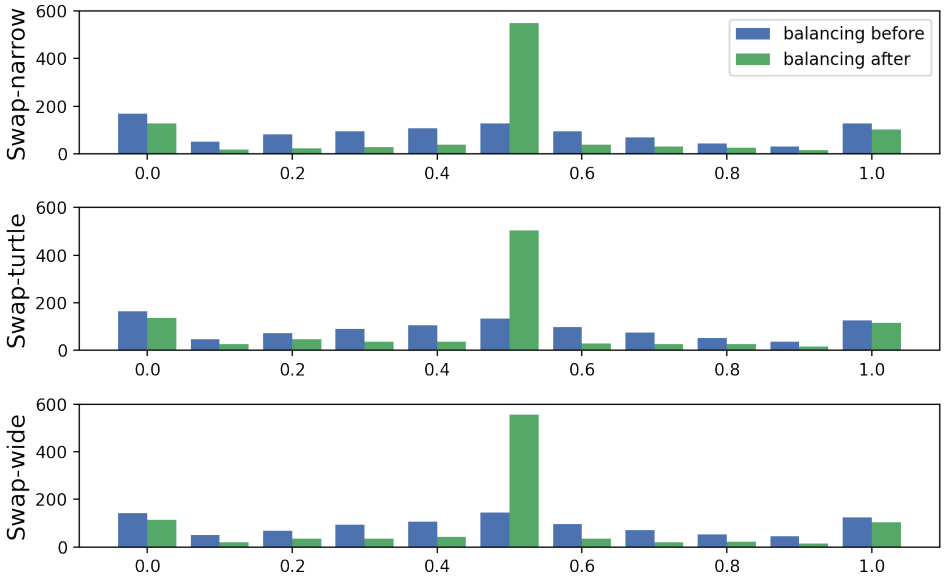


Figure 5.4: Comparison of the balance state distributions before and after the balancing process for each representation using the evaluation dataset of 1000 levels (Figure 5.3).

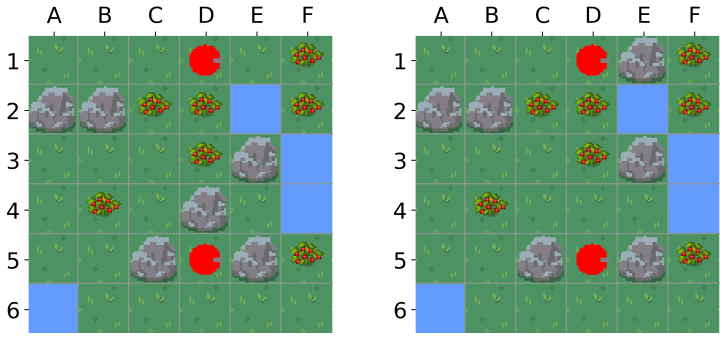
5.4.2 Generated Levels

Figure 5.5 shows examples of the different types of generated levels taken from the samples in Section 5.4.1. Sample 1 in Figure 5.5a is an example where the balancing agent improved the given level to a balanced level by swapping only one tile. By swapping the highlighted grass tile with the rock tile the path to the resource (food) tiles is now blocked. This results in a more equal availability of food resources for both players. In Figure 5.5b, the agent has balanced the level from an initial balance state of 0.3 to 0.5. By swapping the marked water tile to a more central position for both players, the balance is improved. However, in Figure 5.5b the agent failed to balance the initial level with $b_0 = 0$. The balancing process terminated after reaching the maximum number of changes permitted. In the end, b is still 0.

5.4.3 Evaluating the Impact of Different Tile Types on Balance

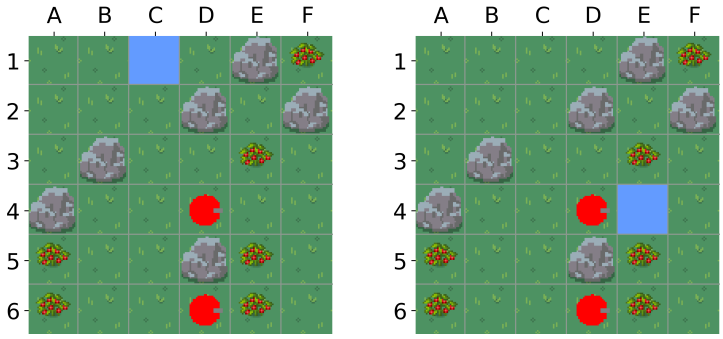
The analysis of the actual swaps made by a model gives insight into its behavior. Thus, we have shown that the model could improve the balance state of the given levels, we can further argue that the swaps made by the model have an impact on the balancing. Conversely, we can infer from this behavior which tiles in the game have the most impact on the balance.

For each pair of swapped tile types, t_1 and t_2, we calculate the difference in frequency of randomly swapping them in the context of tile frequencies in the entire dataset $f^{rand}_{t_1,t_2}$, relative to the observed frequency when using a trained model $f^{obs}_{t_1,t_2}$. To emphasize relative deviations from chance $\Delta f^{rel}_{t1,t2}$, this difference is then weighted by the inverse of the expected random swap probability:



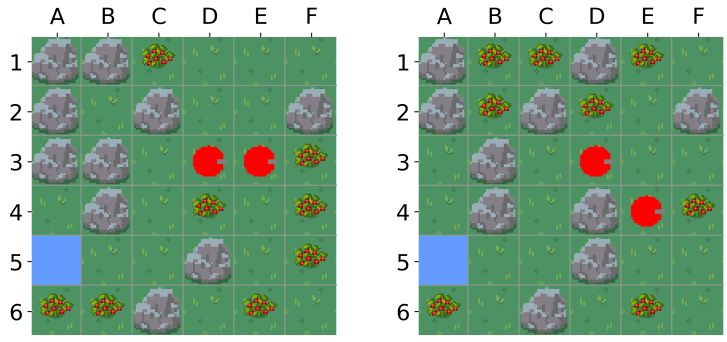
Before ($b = 0.0$) After ($b = 0.5$)

(a) Sample 1: The agent balanced a maximum imbalanced level (left) to a balanced level (right) by swapping the tiles D4 and E1.



Before ($b = 0.3$) After ($b = 0.5$)

(b) Sample 2: The agent balanced an imbalanced level (left) to a balanced level (right) by swapping the only water tile available in C1 to a more central position for both agents in E4.



Before ($b = 0.0$) After ($b = 0.0$)

(c) Sample 3: The agent could not change the initial imbalance of 0. The generation stopped after exceeding the allowed number of changes.

Figure 5.5: Examples of levels modified by the balancing agent. The left image of each subfigure shows the generated levels before balancing. The right subfigures are the resulting levels after the balancing.

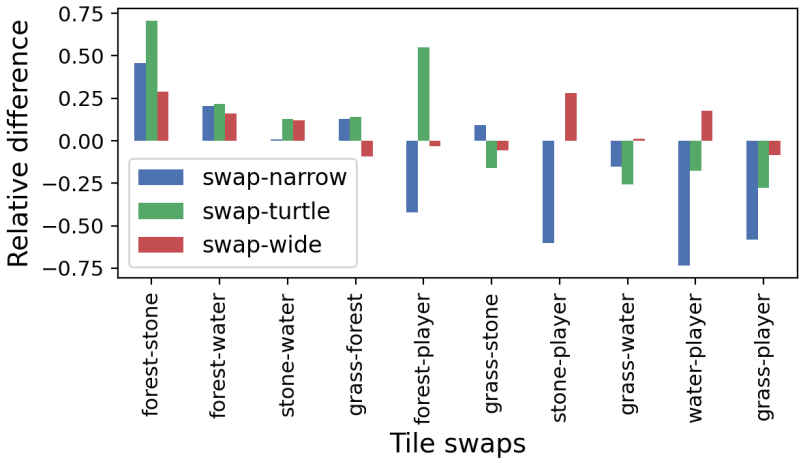


Figure 5.6: Comparison of the swapped tiles by model per representation on the generated 1000 levels. The comparison is made with respect to the inverse tile type distribution of all levels. This shows the difference to random swapping.

$$\Delta f^{rel}_{t1,t2} = (f^{rand}_{t1,t2} - f^{obs}_{t1,t2}) \cdot \frac{1}{f^{rand}_{t1,t2}} = 1 - \frac{f^{obs}_{t1,t2}}{f^{rand}_{t1,t2}} \tag{5.6}$$

Due to swapping the same tile types is prohibited by the representations, ten different combinations are possible. This result is shown in Figure 5.6. The figure shows that the three representations have different behaviors, but they agree on particular points. Swapping food for stone tiles has the most impact, followed by swapping food for water tiles. This makes sense in the context of the resource foraging win condition. Moving the resource tiles food or water is likely to affect the balance as they are included in the win condition. A stone tile can be used to block a player's path. Therefore, swapping food tiles for stone tiles has an additional powerful effect. Surprisingly, swapping a player's spawn position with other tiles is not a favored action in all cases.

5.5 Empirical Evaluation

This section addresses RQ 3 in order to evaluate the perception of the simulated balance empirically with human playtesters. This is our contribution (C 3).

A level's balancing state is evaluated based on data from multiple heuristic agent-based simulations (cf. Section 5.2.2) and is considered balanced when all agents win equally often (cf. Section 4.2). The simulated balance, however, solely depends on the heuristic used to control the behavior of the agents (Algorithm 3) in the simulations. In Section 5.4 we could present results indicating that our approach can learn a trajectory to balance tile-based levels according to the simulation metric. Games are, however, meant to be played by human players. Therefore, we ask the question: do humans actually perceive this simulated balance as intended?

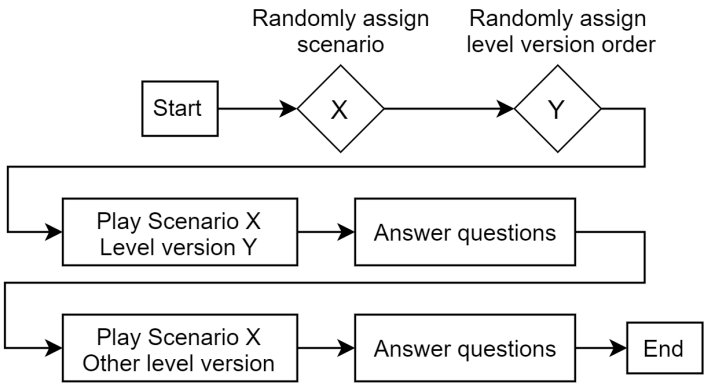


Figure 5.7: The design of the survey. Each participant is randomly assigned a scenario and which version of the level to play first.

To answer this question, we conduct a survey based on human playtests of the generated content. Playing the game gives humans a more detailed insight in the actual balance than just looking at the level. Each participant is asked to play one out of four randomly assigned scenarios, each consisting of an imbalanced and a balanced version of a level. Random scenario assignment follows a uniform distribution. After each playtest, participants are asked to answer questions about their perceptions of the balancing. As evaluation of game balance on an absolute scale leads to very subjective results [Schreiber and Romero, 2021], we use a comparative rating [Thurstone, 1927] by asking participants how they rate the balance of a level in comparison to a different version of the level. We provide the question catalog for the survey in the Appendix B. All tested levels along with the survey results are available on Github (see Table A.1 in Appendix A).

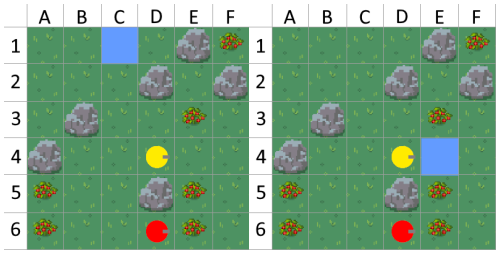
5.5.1 Method

We present the empirical evaluation method in two parts: first, the design and process of the user study (Section 5.5.1), and second, the foundation of the descriptive and statistical analysis of the survey data (Section 5.5.2).

Comparative survey design

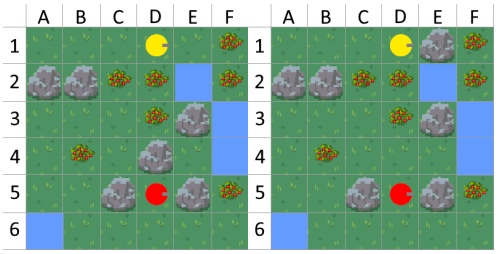
The survey's design plays a crucial role in empirically assessing balancing from the perspective of human players. Figure 5.7 outlines the survey process, which involves defining four scenarios, each consisting of paired imbalanced and balanced versions of the same level. Figure 5.8 gives and overview of all four scenarios with the imbalanced and the balanced level version. These levels are taken from trained models from Section 5.4. In addition to two newly generated and balanced levels, we evaluate the previously generated levels and their balanced version (Figure 5.5a and 5.5b).

Participants are randomly assigned a scenario at the onset and play versus the heuristic (Algorithm 3) used to evaluate the balance. When designing the



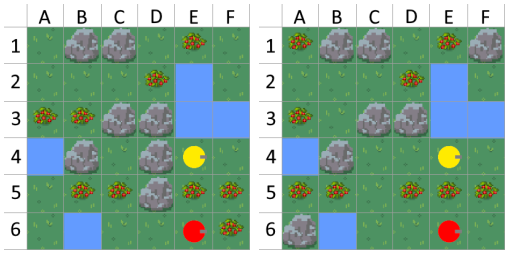
Imbalanced, 0.3 Balanced, 0.5

(a) Scenario 1 (cf. Figure 5.5b)



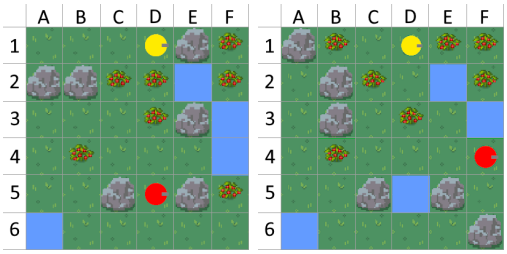
Imbalanced, 0.0 Balanced, 0.5

(b) Scenario 2 (cf. Figure 5.5a)



Imbalanced, 0.2 Balanced, 0.5

(c) Scenario 3



Balanced, 0.5 Imbalanced, 0.0

(d) Scenario 4

Figure 5.8: The four scenarios included for playtesting. Levels in scenarios 1-3 were balanced ($b = 0.5$) from a previously imbalanced version ($b \neq 0.5$) by swapping tiles using the method described in Section 5.2. Scenario 4 presents the balanced version of scenario 2, which was subsequently imbalanced again using our PCGRL model. The scenarios 1 and 2 were taken from the previous results in Section 5.4, whereas Scenarios 3 was newly generated for this study. Scenario 4 investigates the case where an already balanced level Figure (5.5a) is imbalanced again. We will examine unbalancing in more detail in the next Chapter in Section 6.1.

survey, we first randomly assign, once for all scenarios, which player is controlled by a human and which is controlled by the heuristic agent. The red player is controlled by the human, and the yellow player is controlled by the heuristic. In Chapter 6 we will explore the method of directly dealing with two differently labeled tile types for each player in order to improve the performance.

To ensure participants' impartiality, the survey allows participants to sequentially experience both versions of a level without knowing which they are playing. This is reinforced by randomizing the order in which participants encounter the versions, facilitating bi-directional evaluations. Following each playtest, participants provide feedback on their perceptions of the balancing of the level they just experienced.

Survey Questions

Participants are asked to answer six questions (*items*) for each level version, breaking down the abstract concept of balance into more tangible, game-related

aspects for easier comprehension. These questions assessed the proximity of the number of resources (food and water) to the player's spawn point and the difficulty of moving around the level due to impassable rocks and water tiles. Additionally, participants evaluated their perception of their opponent's resource access based on its spawn position, resulting in six questions in total. In the second level, participants answered the same six questions, but always in *comparison* to the corresponding previous version, as comparative judgments are more intuitive for humans than absolute scales [Thurstone, 1927].

We use a five-point Likert scale for all questions. Positioned in the middle of the scale is the choice indicating optimal balancing. To express either a surplus or a deficit, participants can choose between two levels respectively. For example for the item *Amount Food*, the scale ranges from *way too few/none* (-2), *too few* (-1), *sufficient* (0), *too much* (1), and *way too much* (2). The design is consistent across all questions in the first level, with slight variations in formulation based on the items. The mapping to numerical labels is not shown to participants. The complete catalog of questions and possible choices is provided in the Appendix B. For instance, the corresponding question for the item *Amount Food* is: *How would you rate the amount of berry bush tiles (food) near your starting position?*

5.5.2 Data Analysis

The data analysis is twofold: firstly, a descriptive comparison of median values per item before and after balancing, and secondly, Wilcoxon signed-rank hypothesis tests to assess the statistical significance of items before and after balancing.

Data preparation

To accurately compare median values and conduct hypothesis tests based on the previously mentioned relative data (cf. 5.5.1), it is imperative to convert it into absolute representations.

Per item we create a proxy to obtain the absolute value for the second level played $L2$ using the absolute value from the first level $L1$ and the relative one from the second level $\Delta L2$. This can be expressed as:

$$f(L1, \Delta L2) = L1 + \Delta L2\,. \tag{5.7}$$

To revert the data for cases where the balanced version has been played first we retrieve the absolute value using $f(L2, \Delta L1)$. For instance, if a participant rates the available amount of food for the imbalanced version $L1$ as *way too much* (2) and in the balanced version $\Delta L2$ as *fewer* (-1) this results in a score of 1, being still (*too much*), but not *way too much* anymore. We fix the scale to the interval of $[-2, 2]$ and round values exceeding this interval to the closest possible value.

Hypothesis tests

To evaluate changes in participants' perceptions of items before and after balancing, we utilize the two-tailed Wilcoxon signed-rank test, suitable for paired,

ordinal, and non-normally distributed data. A preliminary Shapiro-Wilk test revealed that neither case exhibited a normal distribution for both distributions respectively. The null hypothesis H_0 states that the data distribution does not significantly change after balancing the level. We choose a significance level of 0.05. If the test's p-value falls below 0.05 and the test statistic W is less than the critical value W determined by the sample size, we reject H_0 and accept H_1, indicating a change in the data distribution. Despite multiple tests, we don't apply any measure to correct the p-values, such as the Bonferroni correction, as this would increase the risk of Type II errors due to the relatively small sample size in each scenario.

5.5.3 Results

Participants were primarily recruited from two groups: students and academic staff. We only included completed responses from participants for further analysis, for a total of 71 valid responses. The number of submissions per scenario varies as some surveys were not fully completed. Table 5.2 presents the descriptive results (median) per item in each scenario, along with the results of Wilcoxon signed-ranked tests comparing the data before and after balancing.

The results indicate significant changes in the perception of particular items across Scenarios 1, 2, and 3. Notable findings underscore variability in the perception of items across different levels. For instance, in Scenario 1 (Figure 5.8a), there is a significant increase in the player's accessibility to the water resource (*Amount Water Player*). Also in the descriptive results is a noticeable improvement, as evidenced by the median shifting from -1.5 (*too few/way to few*) to 0 (*adequate*) in the balanced version. This suggests that participants perceived the single water tile swap as affecting the level's balance positively. Conversely, in Scenario 3 (Figure 5.8c), perceptions of player movement difficulty varied, with no discernible shift in perceptions of water accessibility.

In Scenario 2 (Figure 5.8b), numerous items exhibited significant perceptual changes after balancing. Particularly noteworthy are the shifts in the items *Movement Difficulty Player* and *Amount Food Player*. Additionally, descriptive results suggest changes in the distribution towards a better balancing. For instance, the median value shifted for the item *Movement Difficulty Player* from *very inconvenient* (-2) to *adequate* (0) indicating balance after the balancing procedure. Exemplary we provide the data for both level versions of this item of Scenario 2 in the violin plot in Figure 5.9.

A comparable improvement can also be observed for the *Amount Water Opponent* item, where the access to water resources has been reduced from *slightly more* (1) to *same amount* (0). The perception of *Amount Food Player* also shows significant variation; however, the descriptive results suggest that the player now has slightly excessive food resources. Nevertheless, in sum this improvement remains positively valued. In Scenario 4, we evaluated the capability of RL to unbalance the initially balanced level from Scenario 2 in favor of the opponent.[13] Although players did not perceive statistically significant differences,

[13]We will examine unbalancing in more detail in the next Chapter in Section 6.1.

Table 5.2: The descriptive and hypothesis testing results: For each item per scenario, the medians of the imbalanced (U) and the balanced (B) version are compared. The Wilcoxon signed-rank tests for each scenario compare the imbalanced with the balanced version per item. Significant values ($p \leq 0.05$) and median changes are highlighted. The terms *Movement* (Movt.) and *Amount* (Amt.) in the item descriptions are abbreviated for visualization purposes.

	Scenario 1 Samples: 16				**Scenario 2** Samples: 20			
	Median		Wilcoxon		Median		Wilcoxon	
Item	**U**	**B**	**W**	**p**	**U**	**B**	**W**	**p**
Movt. Diff. Player	1	1	35	0.225	**-2**	**0**	**5.5**	**0.00002**
Movt. Diff. Opponent	1	1	20	0.740	2	2	**0**	**0.007**
Amt. Food Player	0	0	3	0.180	**-1.5**	**1**	**7.5**	**0.0007**
Amt. Food Opponent	**0**	**-1**	**4**	**0.035**	2	2	3	1.0
Amt. Water Player	**-1.5**	**0**	**10.5**	**0.007**	0	0	**0.0**	**0.006**
Amt. Water Opponent	**0.5**	**1**	19.5	0.083	**1**	**0**	**11**	**0.019**
Critical Value for W			29				52	

	Scenario 3 Samples: 23				**Scenario 4** Samples: 12			
	Median		Wilcoxon		Median		Wilcoxon	
Item	**U**	**B**	**W**	**p**	**B**	**U**	**W**	**p**
Movt. Diff. Player	2	2	**4.5**	**0.034**	**-0.5**	**-1**	15	0.357
Movt. Diff. Opponent	**1**	**2**	**0**	**0.005**	**1**	**2**	6.5	0.190
Amt. Food Player	1	1	20	0.740	-1	-1	7	0.206
Amt. Food Opponent	0	0	12	0.705	2	2	2	0.564
Amt. Water Player	0	0	22.5	1.0	0	0	9	0.194
Amt. Water Opponent	1	1	10.5	1.0	0	0	2	0.257
Critical Value for W			73				13	

a descriptive contrast emerged: the player's movement slightly worsened, while the opponent's movement slightly improved.

Not all balanced versions were universally perceived as improved towards the center of the Likert scale. In Scenario 1 the perception of the *Amount Food Opponent* is slightly worse in the balanced version. Especially in Scenario 3 regarding the item *Movement Difficulty Opponent*, participants rated their opponent's freedom of movement even better than in the imbalanced version, as indicated by the median and the significance of the hypothesis test. Despite this, both

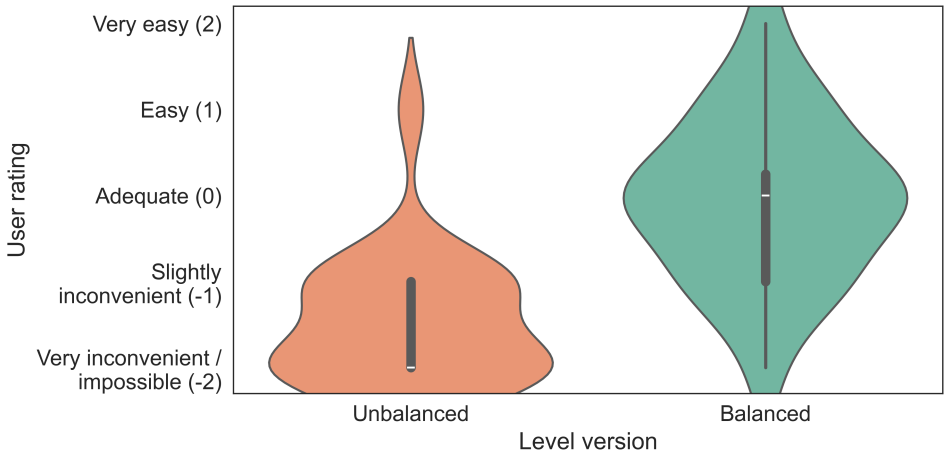


Figure 5.9: An exemplary data distribution of participants' normalized rating: Comparing the imbalanced and balanced level version for the item *Movement Difficulty Player* for Scenario 2. The sample size is 20. An overview of all the descriptive and statistical results is provided in Table 5.2.

player and opponent are now rated similarly. This suggests that while movement is still relatively easy, it is now comparably easy for both players.

5.6 Improvements in the Markov Decision Process Definition for Level Balancing

So far we have defined an MDP for balancing tile-based levels in this section, extending the PCGRL framework with a novel swap-based representation pattern. While the initial results (cf. Table 5.1) indicated feasibility in comparison to the original PCGRL, we now aim to further improve the performance by sharpening the definition of the observation and the action space.

5.6.1 Observation Space

One takeaway from the evaluation of the different tile types on the balance (Section 5.4.3) was that the spatial distribution of players' starting positions plays a substantial role. We also noticed this limitation in the context of the empirical study (Section 5.5), where we had to initially assign the players to their positions. Until now, both players' starting positions were equally encoded in the model's observations space, making it impossible to distinguish between them. The size of the one-hot encoded observation space $\mathbf{S}$ was $\mathbf{S} \in \{0,1\}^{6\times6\times5}$, resulting in a matrix of 180 elements.

We address this by assigning different tile representations to each spawn position. To differentiate between the two, we assign red to player 1 and yellow to player 2 (cf. Table 4.1). The different encoding of the two positions adds more information and thus slightly increases the observation space, but the action space remains the same. The observation space is now: $\mathbf{S} \in \{0,1\}^{6\times6\times6}$, resulting in

a matrix with 216 elements. Moreover, having both players unambiguously defined through the level makes it also easier when assigning human players like in Section 5.5.

Result: When comparing the performance of this new definition of the observation space under the same conditions as in Section 5.4, the model using the swap-wide representation can now balance 68.0 % of the given levels. This is an improvement over the previous result of 48.1 % (cf. Table 5.1).

5.6.2 Action Space

The *swap-wide* representation's action space allows the RL agent to swap the tile of a predicted location $(x1, y1)$ with the tile of a second predicted location $(x2, y2)$ of a level. With another binary decision it can predict if the swap should be done or not. This additional prediction was directly derived from the wide representation introduced by Khalifa et al. [2020] and resulted in an action space depending on the height k and width l of the level. For a 6×6 level this results in 2592 actions. Compared to the swap-narrow and swap-turtle representations, it has the largest action space by far.

Since the model already makes a prediction about the positions where to swap, the additional prediction of whether to swap or not makes the action space unnecessarily complex. Therefore, we reduce it to predict only the two swap positions, resulting in the action space size of $|\mathcal{A}| = (k \cdot l)^2$. For a 6×6 level this reduces the action space to the half of 1296 actions.

Result: When comparing the performance of this optimized definition of the action space and the improved observation space under the same conditions as in Section 5.4, the model using the swap-wide representation can now balance 91.5 % of the given levels. This is a further improvement in comparison to the first result of 48.1 % (cf. Table 5.1) and the improvement through the observation space (68.0 %, Section 5.6.1).

5.7 Comparison to Baselines: PCGRL and Hill Climbing

In Section 5.4, we evaluated our definition of the MDP primarily in terms of feasibility by comparing the three swap-based representations against each other. This alone, however, does not sufficiently evaluate the method's performance. We therefore extend the comparison to include the original PCGRL representations. Yet, even this does not fully justify the applicability of RL to the problem: is *learning* a trajectory for balancing actually beneficial? For this reason, we further compare the PCGRL-based approaches with further algorithmic, model-free PCG methods (hill climbing) and a random method in order to demonstrate the advantages of learning in advance.

In the related literature (cf. Section 3.1), search-based approaches are a commonly used method for game level generation. In order to accelerate level generation, evolutionary algorithms are, for instance, not a suitable baseline as they

explore the search space across several individuals in parallel, requiring a massive number of fitness evaluations per generation. Given that our simulation-based evaluation function is computationally expensive, our goal is to minimize this effort. In contrast, hill climbing approaches are lightweight and can serve as a suitable baseline, as they are trajectory-based like RL. This allows the action spaces of the narrow and swap-narrow representations to be transferred easily, ensuring a fair comparison. For this reason, we compare our method's performance in balancing games with a limited of allowed number of changes to two hill climbing approaches.

Experimental setup: We run all these approaches on the dataset of 1000 generated levels (cf. Section 5.4) with the same model configurations and the improved definitions of action and observation space from Section 5.6. The hill climbing approaches can resemble the PCGRL narrow and turtle representations (cf. Section 4.4), as well as the swap-narrow and swap-turtle representations introduced earlier (cf. Section 5.3), but without the ability of learning trajectories of actions. Since they incorporate no model to predict a position on the level to change or swap, the representations based on the wide representation is not applicable.

A formal definition of hill climbing has been given in Algorithm 1 in Section 2.3.1. Our swap-narrow hill climbing baseline implementation is given in Algorithm 4 and operates as follows: During each iteration, the method randomly selects two positions and swaps the tiles (SAMPLERANDOMSWAPACTION). This can be interpreted as sampling from the action space of the swap-wide representation, which resembles the swap-narrow representation. We use the same reward function to evaluate the balancing (Equation 4.2.2) and simulation setup (cf. Section 5.2.2), but with the enhancement for the labeling of the second player through the newly introduced tile labeling (cf. Section 5.6.1). If the level balance is not improved, ergo the reward in time step t is not positive, the level state is reverted to the state in $t-1$. As in the experiments in the previous chapter, levels that are initially balanced are not considered to prevent the results from being disproportionately amplified in the positive direction.

The execution terminates either when the desired balancing is achieved or when the maximum of eight changes is reached. This is the same value used in our previous experiments and in accordance with the proposed one by Khalifa et al. [2020]. Since we consider the balancing phase as a fine-tuning stage for pre-existing content, it's crucial to restrict the number of changes. Excessive changes can lead to a total overhaul, transforming the content into an entirely new level. Moreover, adhering to the PCGRL's constraint on the number of changes also promotes content diversity, as the approach cannot consistently yield identical outcomes [Khalifa et al., 2022].

The narrow and turtle hill climbing approaches resembling the narrow and turtle representation from PCGRL operate similarly, however, due to their ability to freely exchange tiles from the pool of available tiles, it may happen that levels remain unplayable after the execution has finished. When using these representations from PCGRL, the behavior of the SAMPLERANDOMSWAPACTION function is configured to resemble this behavior accordingly. In general, no sim-

ulations can be run if a level is in an unplayable state. In these cases, we assign a negative reward of -1, which can occur in all non-swap-based representations (original PCGRL, hill climbing, and random)

Algorithm 4 Hill climbing baseline resembling the swap-narrow representation.

```
1:  procedure HILLCLIMBING(level_{t=0})
2:      init bestLevel ← level_{t=0}
3:      init t ← 1
4:      while t ≤ changePercentage do
5:          init action_t ← SAMPLERANDOMSWAPACTION()
6:          init level_t ← APPLYACTIONTOLEVEL(action_t)
7:          init balance_t ← SIMULATEBALANCE(level_t)
8:          init reward_t ← CALCULATEREWARD(balance_t)
9:          if balanced_t is true then
10:             return level_t                          ▷ Success
11:         end if
12:         if reward_t > 0 then
13:             bestLevel ← level_t
14:         end if
15:         t ← t + 1
16:     end while
17:     return bestLevel                                ▷ Failure
18: end procedure
```

Results: The results in Table 5.3 indicate that PCGRL using the swap-wide representation significantly balances the largest portion of levels (91.5 %) within the prescribed limit of changes compared to the other swap-based approaches, the original PCGRL, as well as all other search-based approaches. The swap-narrow hill climbing approach achieves a noteworthy balancing percentage across (73.9 %), however, not being able to achieve a better performance than our PCGRL approach. With this result, it ranks as the second-best performance, outperforming both the original PCGRL and the swap-narrow and swap-turtle representations.

When directly comparing our swap-based MDP definition with the original PCGRL ones, ours show significantly better performances than PCGRL for the turtle and wide, and slightly better than for narrow. Notably, the original PCGRL representations manage to achieve high proportion of playable levels.

Although the swap-based hill climbing approaches cannot outperform the swap-wide PCGRL approach, they produce superior results compared to their non-swap-based counterparts. This can also be observed when the actions are executed randomly. Therefore, we conclude that swapping existing tiles is a superior approach to exchanging tiles arbitrarily (original PCGRL). In addition, using RL further improves the results, since the RL-based swapping approaches each outperform their purely search-based counterparts.

For each experiment, the average number of changes made to the level $\overline{C}$ and the average length of an episode $\overline{L}_{ep}$ are listed. The average episode length

Table 5.3: Overview of baselines comparisons: We compare our PCGRL-swap MDP definition with the original PCGRL, as well as with hill climbers and fully random approaches. All methods use the improved versions as described in Section 5.6. Each is evaluated using the same dataset of 1000 levels (cf. Section 5.4.1). Since hill climbing and random methods do not use pretrained models, the wide and swap-wide representations are similar to their corresponding narrow representations. We list the proportion of balanced levels ($b = 0.5$) as the main metric for comparison. In addition, we provide the proportion of levels that have improved towards $b = 0.5$ (Impr.), the average number of changes made ($\overline{C}$), and the length of the episode ($\overline{L}_{ep}$), both along with their standard deviations (ρ). Finally, we list the proportion of levels which are playable after balancing (Play.).

	$b = 0.5$ **(%)**	**Impr. (%)**	$\overline{C} \pm \rho$	$\overline{L}_{ep} \pm \rho$	**Play. (%)**
PCGRL-swap					
swap-narrow	62.0	71.5	5.3 ± 2.7	17.1 ± 11.3	100.0
swap-turtle	63.2	75.0	5.1 ± 2.8	24.9 ± 19.6	100.0
swap-wide	**91.5**	**93.7**	2.7 ± 2.2	4.8 ± 11.7	100.0
PCGRL [2020]					
narrow	59.2	76.2	5.4 ± 2.7	7.9 ± 4.6	98.5
turtle	29.4	50.2	6.9 ± 2.1	8.8 ± 4.8	96.4
wide	37.2	55.0	6.7 ± 2.1	7.1 ± 3.9	89.3
Hill Climbing					
narrow	47.0	55.0	6.0 ± 2.6	8.4 ± 3.9	75.8
turtle	43.8	54.0	6.1 ± 2.6	12.0 ± 6.1	73.6
swap-narrow	73.9	85.3	4.9 ± 2.6	7.3 ± 4.4	100.0
swap-turtle	53.4	66.8	5.6 ± 2.8	18.2 ± 12.5	100.0
Random					
narrow	13.0	14.4	7.3 ± 1.9	10.3 ± 3.4	83.7
turtle	22.4	28.3	6.9 ± 2.2	13.9 ± 5.8	57.4
swap-narrow	59.8	72.9	5.5 ± 2.7	8.2 ± 4.4	100.0
swap-turtle	46.8	64.1	6.0 ± 2.7	18.9 ± 12.3	100.0

is always higher because it includes the full length of the trajectory. For example, it includes actions that are not executed as changes, such as when the same tile types are chosen for swapping. The results show that as the proportion of balanced levels increases, the number of average changes and episode length decrease. In the context of RQ 2, our goal is to accelerate the automated level balancing process. Therefore, it is important to perform well in general level balancing but also to make as few changes as possible, since n simulations must be run for each change to estimate the balance. With an average of only 2.7 changes, the swap-wide representation requires the fewest changes. This further distinguishes it from the second-best result, the swap-narrow hill climbing ap-

proach, which still requires an average of 4.9 changes. With an average of over two additional changes required, this amounts to 28 extra simulation runs for this environment (previously, 14 simulation runs were identified sufficient per one change; cf. Section 4.2.3). The turtle-based representations have the largest average episode lengths, in particular the swap-turtle ones. This is due to the limitation of tiles that can be selected for exchange or swap, since the agent can only move to an adjacent tile at each time step.

5.8 Discussion & Conclusion

In this chapter, we addressed RQ 2 by introducing a method that uses RL to learn a policy for game balancing to accelerate the generation process of tile-based levels (C 2). While we could demonstrate the feasibility of our method, we identified limitations and cases that require further examination in order to strengthen our response to RQ 2 (see the discussion below and the limitations in Section 5.9). We will thus address further applications, such as exploring asymmetric player archetype setups or the transferability of the method to another environment in Chapter 6.

In order to quantify balance automatically, we applied the balancing metric paired with multiple agent-based simulations, as previously introduced in the chapter of foundational concepts in Section 4.2 (C 1). This allowed for a reliable, game-independent measurement of balance in a competitive two-player game. Since we used agent-based simulations and heuristics to automate level balance evaluation, we further evaluated balanced levels through a user study with human playtesters, confirming their accuracy in most cases. This constitutes our contribution 3 (C 3) in addressing RQ 3. To test the presented approach, we used the *Feast & Forage* environment as introduced in Section 4.3. Our experiments showed superior results within fewer training steps compared to the original PCGRL, while also being more robust to ensure playability. In addition, our approach of using a swap-wide representation for the MDP showed significantly better results than the search-based baselines and other swap-based representations. There are, however, several aspects that need to be discussed.

The reward of the balancing agent represents the balancing state of multiple simulations on the current level state. By design, this is achieved with the use of the information of which player actually won per simulation pass. Using game-specific information, such as the health state of players inside the reward function, could potentially improve the results. This is, however, not desirable since including domain specific information creates dependencies on the particular game. In this case, a special reward function would need to be developed for each game. Furthermore, including too much or the wrong information could bias the reward, resulting in poor model performance or unwanted behavior. As a consequence, the model would not learn what *really* influences balance, or it might exploit unforeseen loopholes.

Splitting up the level generation and balancing process into separated units yielded much better results than executing it in the same step, as in the original PCGRL. Generating and balancing levels in the same step might achieve comparable results if trained for a significantly longer time. However, the architecture

proposed here converges fast with a high performance in the best configuration (91.5 %). This is because, in the second step, the agent can focus exclusively on balancing, reducing its reward function to that single objective.

We compared our three PCGRL-adapted swap-based representations among each other, to those from the original PCGRL, hill climbing, and a fully random approach. Overall, the swap-wide representation showed the best results by far with a proportion of 91.5 % of balanced levels ($b = 0.5$) out of an evaluation set of 1,000 levels. In general, the higher the proportion of levels a method can balance, the fewer changes are required to balance each level. This accelerates the level-balancing process by eliminating the need for unnecessary simulation runs after each change to estimate the balance state. On average, the model using the swap-wide representation balances levels within 2.7 changes. For these reasons, we recommend using the swap-wide representation when balancing tile-based levels.

Notably, all approaches using a swap-based representation outperform their non-swap-based counterparts, regardless of the method. This highlights the superiority of our proposed swap-based approach for level balancing as fine-tuning. While not as effective as the PCGRL swap-wide model, the swap-narrow hill climbing approach achieves the second-best result, with 73.9 % balanced levels – approximately 25 percentage points better than hill climbing with the narrow representation. The original PCGRL approach outperforms its respective analog hill climbing approach and the PCGRL swap-wide outperforms hill climbing in general. Thus, we conclude that good performance generally arises from two factors: (1) using the swap-based representation pattern in combination with (2) using RL.

Since we showed that our approach improves the levels' balance significantly, it is possible to draw conclusions about which tile types have most impact on balancing regarding their swap-frequencies. For this domain, swapping resource tiles (e.g., food) for blocking elements (stone) had the greatest impact. This is reasonable because swapping a resource for a tile to block movement, such as stone, can greatly impact the balance of the level.

By simulating the game n-times with player agents the balance state is evaluated. This metric is of course dependent on exactly these types of players. When using players with e.g., different skills or types the balancing would be different. That is a limitation, however, being also an advantage. By using different types of player agents, the game level could be balanced to compensate for skill differences of the players by solely adjusting the level not the players themselves. This is of high interest when balancing levels for different player types such as a mage and a fighter. It can be applied in e.g., role play games where gear levels are an indicator of character strength. So, players can use their long-farmed equipment in a competitive setting, and the game could be balanced through the environment only. For this reason, we will explore the setup of two different heuristic agents in the simulation in the next chapter in Section 6.2.2.

To answer RQ 3, we have additionally designed and conducted a survey paired with human playtesting, to empirically evaluate automated game balancing based on heuristics and RL. Participants were presented with one of four scenarios, each consisting of a pair of imbalanced and balanced level versions

in random order. To make it easier for the participants, we implemented a comparative survey design in which the participants always rate the second level version compared to the first level version played. In order to use the data in a direct descriptive comparison and paired hypothesis testing, we therefore introduced a way to normalize the data. Since game balance is multifaceted and can depend on different game entities, we split the general question of game balance into separate questions tailored to the spatial availability of resources or freedom of movement. These questions always asked both how the participants perceived their situation and that of their opponent.

Descriptive analysis coupled with hypothesis testing revealed significant differences in the perceived balance distribution of levels pre- and post-automated balancing. Notably, participants perceived balance differently across various aspects such as resource availability and freedom of movement within each scenario. So, is the balancing actually good? Our findings suggest that our balancing approach influences balance perception in most cases positively; however, human perceptions may differ in certain aspects, depending on the level. We conclude that while the investigated automated method can balance levels accordingly to reduce the need for manual human work, a final human evaluation still remains essential.

For these reasons, we encourage authors to rely not only on the validity of the generated content, but also to include human feedback in a method's evaluation process. Moreover, we the survey design presented here can be applied to other empirical evaluations of content that has undergone procedural optimization for a specific objective. Essentially, it is applicable to scenarios in which both the original and optimized versions of the content are available for comparison.

5.9 Limitations

While we concluded that our method is generally feasible and superior in comparison to other baselines, we identified limitations in the proposed method. We describe these limitations in this section and provide recommendations for addressing them in subsequent research.

We have presented a method for using PCGRL to balance tile-based levels in a fine-tuning process after level generation. Although we can accelerate automated level balancing using RL, the proposed method is still computationally intensive in the model training, which lies in the simulation step for balance estimation and rewarding the agent. Multiple simulations must be performed for each swap action the agent takes on the level. In this context, we determined that this reward is mostly sparse during training. This indicates that only a small sub sample of the action space are actually actions which influence the balancing. Especially in the beginning of the training, this must be learned by the model first. To fasten the training process, we think of methods to reduce the computational cost which arise through simulations. That would be also of high interest for the application in more complex environments. One solution might be to reward the agent after several time steps only or even use targeted sparse rewards. Despite the learning process would then be harder for the agent, simulation steps for the reward are omitted. Thus, the training process is sped

up and the agent can explore faster. Another approach could be the usage of a reward model. Therefore, a model is trained to predict the balancing state of a level. This would speed up training a lot, however, the model's accuracy must be high enough to give suitable rewards to ensure correct training. This would, however, require a lot of manual engineering work and fine-tuning of the model, adding a new layer of complexity and dependence. In line with our goal of avoiding overengineering balance estimation and creating dependencies on the game domain (RQ 1), we refrain from taking such an approach in this thesis.

An extensive comparison to baselines revealed that the swap-wide representation yielded the best results. This representation's action space is, however, depends on the size of the level, which limits the applicability of the method to larger levels. PCGRL's issues with scalability have also been reported by Earle et al. [2024]. Although we explored options to reduce the size of the action space and improved the observation space to enhance overall performance, we did not increase the scalability for larger level sizes.

A motivation for this research is to generate asymmetric levels which are balanced for both players. But we did not analyze their asymmetric structure, nor did we prove that they are indeed asymmetric enough to be not trivially different from just a symmetric layout. We will explore this thoroughly in Chapter 6.

Lastly, we have conducted an empirical study to evaluate the generated content with human playtesters. A limitation arises from the time-consuming nature of playtests involving humans, restricting our ability to evaluate only a small subset of levels, despite the potential to generate thousands using our PCGRL method. The balance across all levels was heuristically estimated through multiple simulations involving scripted agents. These agents consistently behave identically, varying only due to the probabilistic nature of the game environment. However, humans adapt their strategies as they play and learn from each experience. Consequently, they adjust their strategy upon replaying a level. This contrasts with the setup when balancing the levels, where two static deterministic heuristic agents of always precisely equal skill face each other.

Chapter 6

Applications of Automated Level Balancing: Imbalances, Asymmetries, and Transferability

This chapter is based on and extends the author's publications:

- **Florian Rupp**, Manuel Eberhardinger, and Kai Eckert. Simulation-Driven Balancing of Competitive Game Levels with Reinforcement Learning. *IEEE Transactions on Games (ToG)*, vol.16, no. 4, pp. 903–913, 2024. doi: 10.1109/TG.2024.3399536.
- **Florian Rupp**, and Kai Eckert. Level the Level: Balancing Game Levels for Asymmetric Player Archetypes With Reinforcement Learning, *Proceedings of the 20th International Conference on the Foundations of Digital Games*, pp. 1-4, Graz, Austria, 2025. doi: 10.1145/3723498.3723747 .

We additionally contribute the following to the underlying publications:

- Section 6.3: A study on the transferability of our method (C 2) from Chapter 5 to another environment.

This chapter investigates applications of the method proposed in Chapter 5, using them to both strengthen our contribution and address several shortcomings and limitations. In Chapter 5, we introduced a method for generating tile-based levels using Reinforcement Learning (RL) and estimating a game's balance with agent-based simulations, addressing research question RQ 2. In addition, we evaluated the human perception of the generated content in an empirical study to address RQ 3. There are, however, particularly concerning RQ 2 and the general appliance of the results of RQ 1, several additional aspects which deserve a further investigation and a deeper understanding and discussion. Therefore, this chapter investigates additional applications of our automated level balancing approach, each exploring, improving, and enhancing a dedicated case. These strengthen our contributions to automatically estimating balance (C 1) and accelerating balanced level generation (C 2) with additional results in order

to answer RQ 1 and RQ 2. This chapter is structured according to the three applications. An overview and motivation for each application is provided in the following paragraphs.

Degrees of Imbalance: We have proposed a metric to numerically determine how balanced a level is, using an existing fairness metric, and demonstrated it on the case of balancing where all players win equally often. But can we use it also to achieve the opposite, i.e., to train a model that intentionally produces imbalanced levels? Therefore, we explore how configurable our method is in terms of unbalancing levels to a certain value of imbalance in order to produce levels, where one player wins seven out of ten games for instance. Results show that our methods can generate imbalanced levels based on a given balance configuration. We find that performance is highest for generating either balanced levels or maximally imbalanced levels, whereas generating levels with a precise degree of imbalance is more difficult. This can be found in Section 6.1.

Asymmetric levels and player archetypes: One motivation for the work in the previous chapter is to generate levels which are not symmetric. Symmetric levels could have easily been generated by mirroring the content and assigning a player to each level region. We thus provide a detailed analysis of the asymmetry of the generated levels. Results indicate, that our generated levels indeed do not have symmetries. In addition, a Procedural Content Generation (PCG) method should not only produce valid content, but also diverse content [Togelius et al., 2011a]. We will thus proof the diversity of generated levels in a dedicated examination. This can be found in Section 6.2.1.

In Chapter 5, all agents in the simulation where controlled by the identical heuristic. While the focus was on creating asymmetric levels for symmetric player archetypes, we now explore the next step: asymmetric levels for asymmetric player archetypes. Modern games (e.g., Scythe [2016] or League of Legends [2009]) are increasingly focusing on asymmetric designs to increase diversity and the replay value. For this reason, we extend the existing heuristic and define additional player archetypes, which differ in abilities, having drawbacks or advantages. Since we designed our method to be as flexible as possible and not to use any game-dependent information, the heuristic can be easily adapted while still using the same reward function, making the results directly comparable. The results show that our RL-based method can learn a trajectory to create balanced levels even for asymmetric player setups. However, we observe performance variations corresponding to the disparity in archetype strengths, with greater initial unfairness leading to larger declines in performance. This can be found in Section 6.2.2.

Transferability to another environment: We have introduced a method for balancing tile-based levels in Chapter 5 and will further improve and explore it in this chapter. We highlighted limitations in related works (e.g., Lara-Cabrera et al. [2014] and Lanzi et al. [2014]) that introduce approaches using domain-specific information, so our goal was to develop a method that does not include any domain-specific information in order to be transferable to other games. So

Table 6.1: Degrees of imbalance – a performance overview of unbalancing levels: Six swap-wide models trained to optimize different balance values b over the interval $[0.0, 0.5]$ of the balance metric (cf. Section 4.2.2) using the action and observation space definition as described in Section 5.6. For direct comparison, the result for $b = 0.5$ is taken from the previous results in Table 5.3. All values are given as a percentages.

b	**Balanced to** b	**Improved towards** b	**Initial** $= b$
0.0	83.2	92.6	9.4
0.1	39.6	64.0	5.0
0.2	51.0	62.0	10.4
0.3	42.8	56.8	7.8
0.4	45.6	56.6	7.6
0.5	91.5	93.7	16.6

far, however, we have not demonstrated this on an additional environment. Moreover, as we have discussed in Section 5.8, the intent of our swap-based action space for PCGRL (PCG via RL) is to reduce states where the game is not playable and no simulations can be run in order to estimate balance. Using our swap-based approach, the Feast & Forage environment is always playable. However, there may be games where playability depends on the adjacencies of the tiles. For this reason, we will additionally experiment with cases where playability is not always guaranteed. This can be found in Section 6.3.

6.1 Degrees of Imbalance

So far we have demonstrated the feasibility of our approach to balance two-player tile-based levels towards exactly equal wins for both players (Chapter 5). In more detail, this means that we set the balancing value b to $b = 0.5$ (cf. Section 4.2.2). From the perspective of game balancing, there can be, however, other situations in which a non equal balance for both players can be required. As explained earlier, this can involve special scenarios that are particularly more difficult to solve or designed to favor a less skilled player. Moreover, this study explores our method's general capability of balancing levels to a configurable value, such as to specific degrees of imbalance as well.

In order to explore this, we test our method to balance levels for different configurations of b in the range from 0.0 to 0.4, with a step size of 0.1, creating different degrees of imbalance. We run the experiments with both, the improved action and observation space as defined in Section 5.6. We use the exact same experimental setup as in Chapter 5, including the model configurations and the test set for evaluation. In addition, we compare the unbalancing approach with the hill climbing baselines from Section 5.7 in order to evaluate the RL approach with a model-free search-based approach.

Results: The results for various configurations of b are given in Table 6.1 and clearly indicate that the PCGRL approach using the swap-wide representation

Table 6.2: Comparison with hill climbing baselines across multiple degrees of imbalance: An overview of the performances of our PCGRL method optimizing different win rates for players (cf. Table 6.1). The different win rates are expressed with the balancing constraint b and the results are compared to the same hill climbing approach as introduced earlier. All values are expressed as percentages.

b	**PCGRL** Swap-wide	**Hill Climbing** Swap-narrow	**Hill Climbing** narrow
0.0	**83.2**	33.7	5.4
0.1	**39.6**	31.9	10.8
0.2	**51.0**	36.6	7.9
0.3	**42.8**	39.6	8.9
0.4	**45.6**	45.2	12.2
0.5	**91.5**	73.9	47.0

can improve the proportion of accordingly balanced levels of the evaluation dataset across all different configurations of b. A larger proportion could at least be improved towards the given b. The results for b in $[0.1, 0.4]$ are similar, indicating only small improvements for these cases. The proportion of balanced levels could, however, significantly improved for the unfairest case – the levels where exactly the same player should always win. For this configuration, the results are on a comparable level as balancing levels for equal win rates.

As shown in Table 6.2, the results of comparisons to the hill climbing baselines indicate that our PCGRL approach is superior across all configurations of b, except for $b = 0.4$, where a comparable result is observed. For $b = 0.0$, PCGRL using the swap-wide representation outperforms the hill climbing approaches the most. While PCGRL performs well for balanced and imbalanced cases, we observe that the model-free approaches perform worse the more unfair the setup becomes. This will be examined in more detail in the Discussion section (Section 6.4) of this chapter. As in Section 5.7, we also observe here that the hill climbing approach with the swap representation outperforms the analog PCGRL narrow representation, indicating the superiority of swapping tiles for level balancing as a fine-tuning process.

6.2 Asymmetric Balance

A motivation behind this work is the creation of asymmetric game levels to enhance diversity for competitive play, which is also of general interest when implementing PCG methods [Togelius et al., 2011b].

6.2.1 Level Asymmetry and Diversity

For all generated levels per model, we evaluate their asymmetry and diversity with matrix comparisons. For each b, we determine a content diversity of 100 % of the balanced levels, which means that no level is identical to another one.

Table 6.3: Overview of symmetry scores for generated levels across the four axes using Equation 6.1. Lower values indicate more symmetry along an axis, 0 indicates perfect symmetry.

	diagonal	**counter-diagonal**	**vertical**	**horizontal**
Mean	4.52	4.51	4.95	4.96
Std.	0.41	0.40	0.40	0.39
Min.	2.83	2.83	3.16	3.46
Max.	5.48	5.48	6.0	6.0

This result is also consistent with findings of Zakaria et al. [2022], which show that PCGRL yields highly diverse content compared to other approaches.

We evaluate the asymmetries across all generated levels of all models outlined in Section 6.1 regarding their symmetric properties along the horizontal, vertical, diagonal, and counter-diagonal axes. The diagonal symmetry of an $n \times n$ matrix $\mathbf{M}$ can be evaluated using the Frobenius norm with $||\mathbf{M} - \mathbf{M}^T||_F$ which computes the sum of the absolute differences of the matrix elements [Golub and Van Loan, 2013]. As we are dealing with non-numerical tile representations, we use a distance metric h which returns 1 for an element i, j if $|\mathbf{M}_{i,j} - \mathbf{M}^T_{i,j}| > 0$, otherwise 0. To compute symmetries along all four axes, we rotate or transpose $\mathbf{M}$ accordingly which yields a matrix $\mathbf{M}'$. The symmetry between $\mathbf{M}$ and $\mathbf{M}'$ can then be computed using Equation 6.1. Lower values indicate greater symmetry, a value of 0 indicates perfect symmetry.

$$sym(\mathbf{M}, \mathbf{M}') = || \; h(\mathbf{M} - \mathbf{M}') \; ||_F \tag{6.1}$$

A counterclockwise rotation of $\mathbf{M}$ by d degrees is denoted with $\mathbf{M}_{rot,d}$. The transposed matrix of $\mathbf{M}$ is denoted with $\mathbf{M}^T$. To compute the symmetry score for the diagonal axis we compute $sym(\mathbf{M}, \mathbf{M}^T)$, for the counter-diagonal $sym(\mathbf{M}, (\mathbf{M}^T)_{rot,180})$, the vertical $sym(\mathbf{M}, (\mathbf{M}_{rot,90})^T)$, and the horizontal $sym(\mathbf{M}, (\mathbf{M}^T)_{rot,90})$. The mean, standard deviation, minimal, and maximum values are presented for the four symmetry axes in Table 6.3. The results show that none of the generated levels for any model are symmetric.

6.2.2 Level Balancing for Asymmetric Player Types

Well-designed games ensure the viability of multiple strategies for players to choose from, all of which, if played well, can lead to victory [Schreiber and Romero, 2021]. As a result, many modern games make use of asymmetric balancing strategies, such as putting heroes of different abilities against each other. This can be seen in dungeon crawlers like *Decent: Journeys in the Dark* [2012] or MOBA (Multiplayer Online Battle Arena) games like *League of Legends* [2009]. The asymmetric balance is mainly achieved by balancing the numerical values of the game units, such as health or attack values. We will also consider this aspect of balancing in the context of game economies in the next chapter in Section 7.4.

Limited work [Beau and Bakkes, 2016; Lanzi et al., 2014], however, has been published to automatically balance asymmetric games, i.e., players with different abilities and stats. For this reason, this section focuses on balancing an asymmetric game setup with our method on the example of players with different abilities entirely through *level design*, e.g., where to place resources in relation to the players' spawn positions. Therefore, this problem is also formulated as a PCG problem. This approach is further motivated to be used in settings where players of different skill levels face each other, such as experts and beginners, or adults playing against children. Another use case is to ensure balance in competitive settings where players with different gear levels have different strengths.

Since then our approach is, however, limited by the fact that it only uses exactly the same archetype of agents playing against each other (cf. Section 5.2.2). To address this shortcoming, we adds four new archetypes to investigate the ability of the method to achieve balance for *different* agent archetypes playing against each other, for instance, an agent playing against a handicapped agent. We will evaluate and compare our results to a random search and also to the hill climbing baseline (Algorithm 4).

Definition of archetypes

In this work, we introduce the four new agent archetypes B, C, D1, and D2, which extend the existing archetype A (Section 5.2.2, Algorithm 3). An heuristic archetype definition always returns a decision on one of the five actions per game step, describing which of the four adjacent tiles to move to next, or to do nothing (cf. Section 4.3).

We define the new archetypes each to address different specific aspects of the balance, such as movement advantages or the required number of food resources to win the game. The list below gives a brief description of the agents used and how this affects their chance of victory compared to archetype A:

- Archetype A is the *Base Agent*, as has been described earlier in Algorithm 3 and applied as heuristic for all experiments so far. It cannot move over rock and water tiles and wins with five victory points.
- Archetype B, the *Rock Agent*, has the additional ability to cross rock tiles, being blocked only by water tiles. This gives it an advantage over archetype A agents.
- Archetype C, the *Handicap Agent*, can only perform one action every second turn. This agent is at a huge disadvantage when playing against archetype A agents.
- Archetype D, the *Food Agent*, already wins the game with four (D1) or three (D2) collected food resources instead of five. This gives it an advantage over archetype A agents.

Experimental setup

For all experiments, we use the same dataset of generated levels (cf. 5.4) to ensure a fair comparison. We compare our PCGRL method with two baselines:

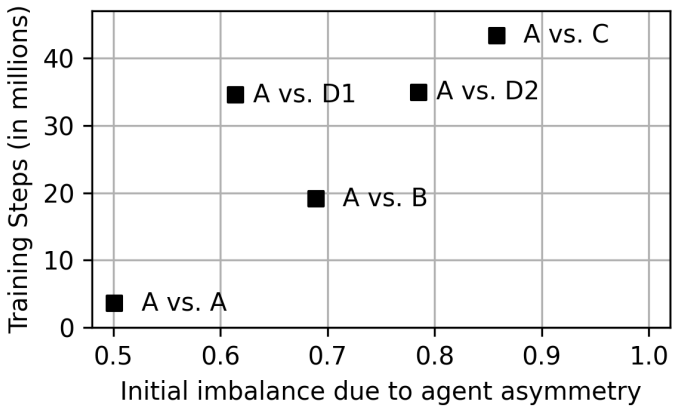


Figure 6.1: Required training steps for models to first exceed a reward of 9 compared to the initial imbalance due to agent asymmetries per setup. A value of 0.5 (50 %) indicates equal wins, while 1.0 means one agent always wins.

a random agent and the hill climbing approach as used before in Section 5.7. The hill climbing approach uses the same swapping mechanism as PCGRL, but chooses the positions randomly. If the reward is not positive, it transitions back to the previous state. A key metric for evaluation is the proportion of how many levels a method can balance.

We train multiple models where archetype A faces the new ones in a one-versus-one setting. For all Proximal Policy Optimization (PPO) models we use the same setup and training conditions as described in Section 5.4, but with the improved definitions of the action and observation space from Section 5.6.

Relationship between initial imbalance and model convergence time

We paired archetypes of different strengths against each other, but how can we determine *how* different their strength is? In the previous chapter, Figure 5.3 indicated a balanced distribution of the randomly generated levels that are initially imbalanced and favor a particular player. Setups with asymmetric archetypes skew the initial balance of levels toward the stronger agent, which wins more often in simulations than its opponent. We can then estimate the absolute proportion of levels which are initial favoring a specific player to express an archetypes advantage over another numerically. In other words, if both archetypes are of equal strength, both should win the same proportion of imbalanced levels, so the ratio should be 0.5 (50 %).

Figure 6.1 shows the initial imbalance of archetype setups compared to type A, in relation to the number of training steps required to surpass a reward threshold. We measure the number of steps each model takes to first exceed a reward of 9 — the lowest peak reward achieved across all models — to ensure a fair comparison. As expected when the archetypes setup is symmetric (A vs. A), the proportions of levels that initially favor a single player are almost equal (50.1 %). In contrast, the asymmetric setups strongly favor one specific player indicating that archetype C is in the largest disadvantage when paired

up with type A. Comparing the initial imbalance with the number of training steps required for model convergence shows that the greater the disparity in strength between two archetypes of a setup, the more training steps are required. This method is also useful for accurately measuring the disparity in asymmetry caused by differing abilities.

Performance overall

Table 6.4 shows the performance of the various setups compared to the hill climbing approach and a random search. Unlike the hill climbing approach, the random search explores the search space fully randomly and does not reset the state to $t - 1$ if the cumulative reward worsens. Initially balanced levels are not taken into account. While the hill climbing approach achieves reasonable results and can beat the random search approach in all cases, our PCGRL approach remains the best in comparison for all setups. This is due to the advantage of RL to learn during training which trajectories have the best impact on the balance.

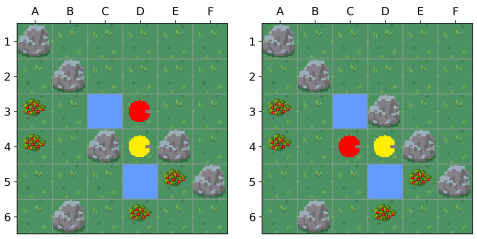
We also see that the performance of the different archetypes varies by about 20 percentage points. In relation to Figure 6.1 we observe that the greater the initial disparity between the two archetypes is, the more the performance of the models decreases. We can therefore conclude that the greater the initial unfairness of a setup, the harder it is for the model to learn how to compensate the balance by modifying the level alone.

Table 6.4: Comparison of the proportions of balanced levels on a set of 500 generated levels with two baseline approaches. A level is balanced when both players win equally. Since the swap-narrow representation resembles the swap-wide in approaches without a learning model (cf. Section 5.7), we use swap-narrow for the random and hill climbing baselines.

Agents	**Random (%)** swap-narrow	**Hill Climbing (%)** swap-narrow	**PCGRL (%)** swap-wide
A vs. B	27.6	46.1	**80.4**
A vs. C	16.2	24.6	**56.5**
A vs. D1	28.8	46.6	**72.3**
A vs. D2	26.8	35.2	**57.9**
A vs. A (Table 5.3)	59.8	73.9	**91.5**

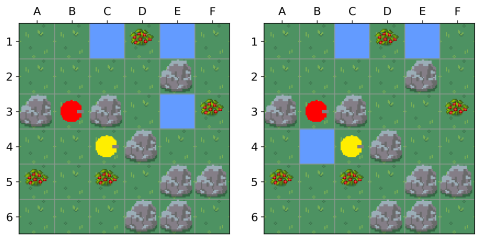
Generated samples

Figure 6.2 shows generated samples from different models and archetype setups. Sample 1 in Figure 6.2a shows the setup of the *Rock Agent* B (yellow) against a normal archetype (red). The model achieved balance by swapping the tiles D3 and C4. Since the yellow agent can move over rock tiles, it can reach the area at the bottom right before the other player. If it could not move over rocks, this level would not be balanced. Sample 2 shows the setup with the *Handicap*



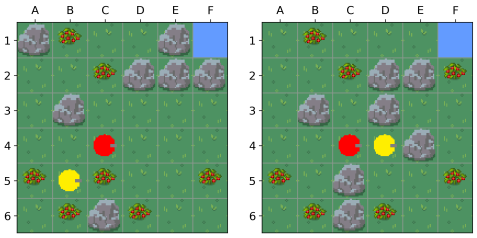
Imbalanced, 1.0 Balanced, 0.5

(a) Sample 1: A vs. B



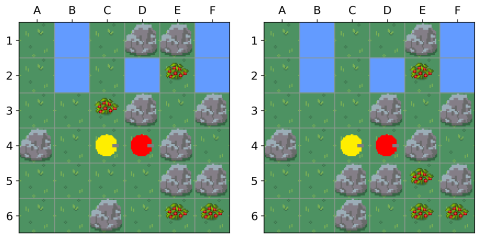
Imbalanced, 0.0 Balanced, 0.5

(b) Sample 2: A vs. C



Imbalanced, 1.0 Balanced, 0.5

(c) Sample 3: A vs. D2



Imbalanced, 0.3 Balanced, 0.5

(d) Sample 4: A vs. D2

Figure 6.2: Generated samples from different models and archetype setups in comparison with the initial imbalanced version. A player of archetype A (red) is paired with different archetypes (yellow).

Agent C (yellow), which can only move every second turn (Figure 6.2b). The model achieves balance by separating the two players from each other by swapping B4 and E3 tiles. This prevents them from stealing food resources from each other, mitigating the yellow player's handicap. The setup of the *Food Agent* D2 (yellow), which already wins with three collected food resources instead of five, is shown in Figure 6.2c. With several changes to the initial level, the model achieved balance by placing more resources on the side of the weaker agent (red). Also, the only water resource (tile A6) is now accessible to both players.

Sample 4 (Figure 6.2d) illustrates a limitation where the model achieves balance by excluding both players from access to food resources. This is a strategy that we see the model exploiting in certain cases to achieve a balanced state where both players technically win equally often – i.e., never — but this outcome is not intended. This is possible due to the Statistical Parity metric on which the reward function is based (cf. Section 4.2.2) assumes that draws are always balanced, even if both players lose. We will discuss this issue further in Section 6.2.3.

6.2.3 Levels without Winners: An Alignment Problem

We improved our method's performance in terms of different aspects and tested various applications in previous sections, but we observed cases where the RL models exploited an unintended loophole in the reward function and generated levels where none of the players can win (cf. Section 6.2.2). This is possible due to the reward function which assumes balance based on the underlying

Statistical Parity metric. According to that metric, also a game without winners is considered as fair and therefore balanced.

A common challenge in RL in general is the definition of a reward function which ensures that the agent learns a behavior with the *actual* intended goal(s). While the RL algorithm optimizes the policy in order to increase the reward by any means, the reward function itself is treated as a black box [Icarte et al., 2022]. This is also what Christian [2021] refers to as the *Alignment Problem* – the disconnection between the goals a human specifies for an Artificial Intelligence (AI) system and the behaviors the system actually learns. This problem is not limited to RL, but concerns all AI systems where a metric is optimized. A crucial step is therefore the evaluation of the system, especially if the system's behavior and goals are achieved as intended by humans.

Although, in our case, the reward is technically maximized and according to the Statistical Parity fairness is guaranteed – making the game theoretically balanced – the outcome is still not a game which we would consider enjoyable, since nobody can win. The Statistical Parity ensures balance, but it does not capture *all* intended nuances of game design, such as the fact that games without winners are also not fun to play.

6.3 Benchmarks on Transferability to another Environment: Balancing the City Game

We have introduced a method for balancing tile-based levels in Chapter 5 and have further improved and explored it in the previous sections of this chapter. In contrast to many related works (e.g., Lara-Cabrera et al. [2014] and Lanzi et al. [2014]), our goal was to develop a method that does not include any game-specific information in order to be transferable to other games. So far, however, we have not demonstrated this on an additional environment. Moreover, as we have discussed in Section 5.8, the intent of our swap-based action space for PCGRL is to avoid or at least to reduce states where the game is not playable and thus no simulations can be run in order to estimate balance. Using our swap-based approach, the Feast & Forage environment is always playable. However, there may be games where playability depends on the adjacencies of the tiles for instance. For this reason, we will experiment in this section with cases where playability is not always guaranteed, even when using our swap-based approach.

6.3.1 The City Game Environment

To evaluate the transferability of our method, we introduce the *City Environment* as an additional testbed. It is narratively inspired by urban planning and mainly focuses on achieving balance through the spatial placement of tiles. The environment meets the same requirements: it is for two players, competitive, tile-based, and it also includes probabilistic elements. Like for Feast & Forage, we set the size to a grid of 6×6. Since road users in inner cities differ, especially in means of transportation, we adopt an asymmetric setting with different player archetypes as introduced in Section 6.2.2 by balancing the environment

for a pedestrian and a car driver, each with distinct movement speeds and tile accessibility. Each round, both players can choose from the same five actions: to move up, down, left, right, or to do nothing. The key difference between the both archetypes – and also from the ones in Feast & Forage – is that the car driver can move up to two tiles each round, whereas the pedestrian can only move one. The car driver, however, needs more turns to cross certain tiles, or even cannot cross some tiles at all and is additionally hindered by traffic jams. This setup is not to be equated with the *Handicap* Archetype C from Section 6.2.2, as the number of possible moves here depends on the specific tile(s) the driver agent traverses in a round, potentially resulting in one or two moves depending on the tile costs. The archetype C in contrast, just skips each second round. In Section 6.3.2 we will experiment with different settings and will then give a more detailed descriptions on the abilities of each archetype. Table 6.5 provides an overview of the available tiles and their sprites[14]; their functionality within the environment is explained in the following paragraph.

Players take their turns simultaneously. To win, a player must visit three locations located in the city center, represented by flag tiles. In the context of the narrative, this represents locations a citizen must visit in order to complete everyday tasks, such as grocery shopping or attending a doctor's appointment. A location is considered visited when a player moves the first time onto the respective flag tile. The player who has completed all three jobs first wins the game. As a result of the discussion on fairness in Section 4.1, we allow a range in which we consider the game as fair, allowing for a draw if the second player finishes equally or just one step later. The order in which the flags are visited is irrelevant. Depending on the setting, the car driver may be unable to cross park tiles, which can lead to unplayable levels where a flag is inaccessible due to being surrounded by parks.

A key aspect is the unpredictability of inner-city traffic, and thus each turn, street tiles have a 25 % chance of turning into traffic jam tiles. Starting from the next turn, each traffic jam tile has a 50 % chance of clearing and transitioning back to a street tile. Following the narrative of a pedestrian and a car driver in an inner city, the car driver is typically faster — except when caught in a traffic jam, where this advantage is repealed, or when there are shortcuts through parks for pedestrians for instance.

A sample level is given in Figure 6.3. While the car driver (C3) has a higher range per turn, it is hindered to reach the flag in D1, since it cannot cross the park tile and thus has to take a detour through the traffic jam in E3 and E2. The pedestrian (B4), in contrast, can reach the flag by just walking through the park in D2. Balance can therefore spatially be influenced by strategically placing different tile types throughout the environment.

6.3.2 Experimental Setup

To ensure comparability across various experiments and baselines, we create a shared test set of 500 levels for evaluation for all experiments. Even though the

[14]The sprites are from the *Tiny Battle* tileset by Kenney, available at `https://kenney.nl/assets/tiny-battle` (CC0 1.0).

Name	Resource	Sprite
Street	No	
Park	No	
Building	No	
Target	Yes	
Traffic Jam	No	
Player 1: Pedestrian	No	
Player 2: Car Driver	No	

Table 6.5: Overview of the tile types in the city game environment.

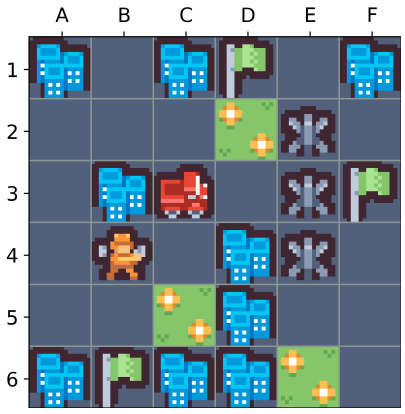


Figure 6.3: The city game: An example level.

Table 6.6: Different tile cost setups of archetypes for experiments with the city environment.

Setup		Action Costs per Tile					Unpl.
No.	**Player Arch.**	**Street**	**Park**	**Building**	**Target**	**Traffic Jam**	**State**
S1	Pedestrian	1	1	1	1	1	No
	Car Driver	1	2	2	1	2	
S2	Pedestrian	1	1	1	1	1	Yes
	Car Driver	1	-	2	1	2	
S3	Pedestrian	1	1	1	1	1	Yes
	Car Driver	1	-	-	1	2	

win condition differs from Feast & Forage, we can use the same architecture (Section 5.2), reward function (Equation 4.4), and experimental setup as we only use the information which player has won the game or if it is a draw. To determine the appropriate number of simulations for estimating the balance, we follow the same approach presented in Section 5.2.2. This approach involves selecting a sufficient number of simulations to minimize scattering of results while keeping computational effort minimal. Figure C.1 in the appendix displays the data from simulations with different values of n ranging from 2 to 40. With this approach, we determine that 30 is a suitable number. We also use PPO as the RL algorithm with the same configuration as before. Since it performed best in the evaluation and comparison to baselines in Section 5.7, we use the swap-wide representation for all PCGRL models. Accordingly, we use swap-narrow for the hill climbing and random baselines.

We evaluate our method using asymmetric archetypes across three different settings, referred to as S1, S2, and S3, for the game simulations. The settings differ in how action costs are configured for the car driver agent to traverse specific tiles. An overview of the configuration is given in Table 6.6. The heuristic that

Table 6.7: Results of the evaluation of asymmetric level balancing with our swap-based PCGRL method in the city environment across three different setups (cf. Table 6.6). For PCGRL we use the improved definition of the observation and action space of the swap-wide representation as defined in Section 5.6. We further compare PCGRL with the same swap-narrow baselines as in Section 5.7 and also report the proportion of levels each method could balance ($b = 0.5$) out of an evaluation set. In addition, we list the levels which are in a playable state in the end (Play.).

Setup	Method	Balanced	Init. Balanced	Play.	Init. Play.
S1	Random	53.3	25.0	100	100
	Hill Climbing	58.0	24.8	100	100
	PCGRL	**96.8**	25.6	100	100
S2	Random	47.6	22.2	91.0	91.2
	Hill Climbing	50.4	22.2	93.6	91.2
	PCGRL	**87.6**	21.2	**96.8**	91.2
S3	Random	16.7	6.4	42.5	39.6
	Hill Climbing	18.3	7.2	47.2	39.6
	PCGRL	**84.6**	6.4	**90.4**	39.6

defines the behavior of the agents is described in Algorithm 5 in the appendix. It is based on using the A* algorithm to find the shortest path to the nearest target that has not yet been visited. Depending on the archetype, it can be configured differently by providing the tile costs. Similarly to the heuristic used in the context of Feast & Forage (Algorithm 3), it implements a function that returns the next action for an agent based on the current game state when called. Depending on the archetype, however, this function may be called multiple times by the environment. For example, the driver agent can perform two actions each turn.

In S1, both agents can traverse all tiles, but the car driver must spend two actions to cross parks, buildings, or traffic jams, while the pedestrian requires only one action for any tile. There can be no unplayable state in this configuration. In contrast in S2, the car driver cannot cross any parks, and in S3, neither parks nor building tiles. In these configurations, the driver agent may be unable to reach flag tiles to finish the game, which can result in unplayable states in S2 and S3. We compare each setting to the same random and hill climbing baselines as described in Section 5.7.

6.3.3 Results

The results of using our PCGRL method with the improved action and observation space introduced in Section 5.6, along with a random and the hill climbing baseline is given in Table 6.7. We tested three different configurations, each with a higher level of difficulty in balancing, because the car driver is more restricted in which tiles he can access. The results reflect this as for instance the proportion of initially balanced levels decreases from S1 with 24.0 % to only 6.4 % in S3.

This is also related to the proportion of levels that are initially playable, where a similar pattern can be observed.

For all three configurations of the asymmetric agent archetypes, our PCGRL approach can balance significantly larger proportions of levels in comparison to the baselines. While in S2 still most of levels are initially playable, they are not for S3. For S3, PCGRL improved the proportion of playable levels after balancing more than the baseline methods, demonstrating its ability to also handle environments where playability is not always guaranteed. We also notice that the hill climbing approach is only slightly better than a completely random approach in this environment, indicating that learning a trajectory of actions is beneficial. Exploring balancing in an additional, asymmetric environment provides further evidence that our method is effective and easily transferable.

6.4 Discussion

In this chapter, we explored additional applications of our method for level balancing from Chapter 5. Our contributions include new insights that enhance the understanding of the previously introduced approach in order to strengthen our answer to RQ 1 (quantification of balance) and RQ 2 (accelerating automated level balancing). This involves investigating its applicability to imbalanced levels and asymmetric player archetype setups, as well as demonstrating its easy transferability to another environment.

In general, a PCG method should not also be fast at inference but also generate content that is diverse [Togelius et al., 2011a]. According to Zakaria et al. [2022], the original PCGRL has been reported to yield better content diversities than other PCG approaches for Sokoban levels. Based on matrix comparisons, we evaluated the diversity of our best performing configuration, the swap-wide PCGRL approach (cf. Section 5.7). We could also confirm that all generated levels in a sample of 1000 levels were unique. It must be said, however, that this was a quantitative evaluation. This does not include cases where individual tiles are different but do actually have no effect on the gameplay, resulting in an essentially equal experience for the players. A qualitative evaluation would require comparing each level in the sample of 1000 with all 999 others, which leads to an immense manual effort. Therefore, we also examined a randomly selected subset of ten levels manually, already resulting in 55 direct comparisons. We found no levels that were too similar. While this gives an additional qualitative impression of level diversity, a sample size of ten is, however, far too small to be considered a robust result. For the sake of the high manual effort in combination with good results from a small-case qualitative study, we rely on the quantitative evaluation, as also Zakaria et al. [2022] did in their evaluation of content diversity.

In addition to evaluating the ability to learn a trajectory for balancing, we also evaluated the ability of our method to generate levels with different degrees of imbalance based on a given configuration. We used both the improved action and observation spaces from Section 5.6. The results (Table 6.1) showed that our PCGRL approach based on the swap-wide representation improved the proportion of levels for all degrees of imbalance. We were also able to demon-

strate that it outperformed the hill climbing approaches that we had previously used as baselines in Section 5.7. This indicates that training a model is more efficient for inference than a model-free approach. While for b in $[0.1, 0.4]$ the results are similar or slightly better, for the most imbalanced configuration $b = 0.0$ PCGRL using the swap-wide representations reaches a comparable quality as for $b = 0.5$. In both cases, PCGRL clearly outperformed the model-free baselines, suggesting that learning in advance indeed is beneficial. We thus draw the overall conclusion that the model can learn best to create either balance or maximal imbalance, and it is more difficult for the model to learn a particular degree of imbalance. Indeed, creating maximal imbalance is much easier — e.g., by systematically excluding a distinct player from resources — than to achieve a specific degree of imbalance. As much as we aim to reduce fluctuations caused by probabilistic game elements through multiple simulations, some randomness still remains. Especially in the case for $b = 0.4$ which is very close to balance, it is hard to distribute the resources in such a way that one particular player wins 40 % and the other 60 % of the games. In this environment, that would mean to create a resource distribution whose respawn probability slightly — and only slightly — favors a selected player. Given the level and so the fixed available tiles for balancing, this can be challenging and sometimes even impossible for specific degrees of imbalance. Perhaps the degrees of imbalance chosen were also too fine-grained.

In Chapter 5, we evaluated our method on balance for agents controlled by the same heuristic. In Section 6.2.2, we extended our exploration to different heuristics for both agents, representing asymmetric player archetypes with varying abilities. These disparities create an unfair setup, leading to initial imbalances, and as in previous experiments, the model's task is to achieve balance entirely through level design. Our results showed that balancing also with different heuristic setups with varying strengths is feasible and can balance a larger proportion of levels compared to both random search and hill climbing baselines. Furthermore, we observe that the difficulty of learning to balance increases with larger initial disparities in strength between the player archetypes, highlighting the challenge of dealing with highly imbalanced starting conditions. This was evidenced by decreased performance and increased time to reach a particular reward threshold.

A limitation of the RL approach is that it can exploit a strategy where neither player can win. While this technically ensures balance (both players win equally often), it is not the intended outcome.

In Section 6.3, we explored our method from Chapter 5 regarding two additional things: First, one of our goals is a domain-independent method combined with a balance estimation that can be applied to other competitive two-player tile-based games. While we demonstrated feasibility and superior performance over baselines, we only tested it in one environment. In this chapter we therefore investigated how well our method generalizes to another environment inspired by urban planning, aiming to ensure equal accessibility to resources for a pedestrian and a driver. Like in Section 6.2.2, the player agent setup was asymmetric.

Second, the intent of our swapping-based action space definition for the Markov Decision Process (MDP) for balancing is that it only changes the po-

sition of tiles, not like the MDP definition of the original PCGRL, completely exchanging tile types with others. Since we evaluate the balance of each level state simulation-driven, exchanging a player's starting position with a different tile would result in an unplayable state. As a consequence, we cannot simulate the game, and the balance cannot be estimated. In the previous Chapter 5 we could successfully demonstrate that swapping-tiles to improve a level's balance as a fine-tuning task in the context of PCG is beneficial. However, unplayable states can also occur when a game has other constraints, such as when certain tiles must be adjacent to a certain other tile, not just a certain number of specific tile types must be present. We further explored this in the city environment, where unplayable states can occur when a player is blocked in such a way that he can never access a flag. For a better comparison, we approached this by experimenting with the three different experimental setups S1, S2, and S3. While in S1 all player archetypes could cross all tile types, in S3 the car driver cannot cross parking and building tiles, leading to possibly unplayable game states in this setup.

To estimate a suitable number of simulations, we followed the same approach as in Section 5.2.2. For the city environment, however, the number of required simulations is much higher (30) than for Feast & Forage (14). This can be explained by the fact that randomness has a greater impact in the city environment due to the 25 % chance of traffic jams occurring each round and the 50 % chance of traffic jams disappearing (cf. Table 6.5).

The results overall demonstrated that our PCGRL approach could balance a significantly larger proportion of levels on a shared evaluation set than the same baselines as used in Section 5.7. For S1, where all levels are always playable, PCGRL achieved a balanced proportion of 96.8 % in contrast to the hill climbing baseline (58.0 %). The performance disparity is even larger for S3 where PCGRL achieved a proportion of balanced levels of 84.6 % in comparison to hill climbing with only 18.3 %. In addition, PCGRL could also significantly increase the proportion of levels which are playable after balancing to 90.4 %. In previous experiments, all levels were initially playable (e.g., Feast & Forage and S1), but in S3, only 39.6 % were initially playable, and in comparison, the hill climbing baseline could only improve it to 47.2 %. This also shows that learning a trajectory whose level changes affect the overall balance is advantageous over search-based model-free approaches. Moreover, we have shown that our method introduced in Chapter 5 is easily transferable to another game without the need to adapt the MDP.

6.5 Conclusion

Applying the method introduced in Chapter 5, we explored and evaluated it across three different aspects in this chapter. In doing so, we extend our contribution C 2 to further strengthen our response to accelerating automated level balancing (RQ 2). In addition, we demonstrated the applicability of our balance estimation C 1 in another environment.

We demonstrated the applicability to extended use cases, such as unbalancing levels. The results indicate that the method can also be used to intentionally

unbalance levels. However, it is easier to produce either balanced or maximally imbalanced levels than to achieve a specific degree of imbalance. Moreover, we demonstrated that our method for generating balanced levels can also produce asymmetric levels for asymmetric player archetypes, not just for symmetric setups. By evaluating the method in another environment, we demonstrated its easy transferability. In this context, we also showed that the method can handle environments where unplayable states may occur while performing significantly better than the baselines.

Chapter 7

Procedural Generation and Balancing of Game Economies

This section is adapted from the author's publications:

- **Florian Rupp** and Kai Eckert. G-PCGRL: Procedural Graph Data Generation via Reinforcement Learning. *2024 IEEE Conference on Games (CoG)*, Milan, Italy, 2024, pp. 1–8, doi: 10.1109/CoG60054.2024.10645633 .
- **Florian Rupp** and Kai Eckert. GEEvo: Game Economy Generation and Balancing with Evolutionary Algorithms. *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, pp. 1–8, doi: 10.1109/CEC60901.2024.10612054 .

This chapter is about the procedural generation and balancing of graph-based game economies, addressing research question RQ 4. As explained earlier (cf. Section 1.1), balancing a game consists of several layers that need to be coordinated with each other. In Chapter 5, we proposed a method for balancing a game through its level design, focusing on the spatial placement of e.g., resources in relation to players' positions. Then, in Chapter 6, we highlighted different aspects of the automated balancing, such as unbalancing, but also a balancing for asymmetric player archetypes.

Alongside levels, a game's *economy* is a key factor in ensuring a balanced and engaging experience on a macro level. In this context, we will deliver two contributions: First, we present G-PCGRL (Graph Procedural Content Generation (PCG) via Reinforcement Learning (RL)). We build directly upon results from Chapter 5 and 6 and propose a Markov Decision Process (MDP) for graph data generation based on a set of constraints. This is our contribution C 4. Second, we propose Game Economy Evolution (GEEvo), a framework with two evolutionary algorithms to generate and balance graph-based games economies. This is our contribution C 5.

7.1 Overview and Motivation

A game's balance is, among other things, heavily influenced by the design and configuration of its internal economy which defines how virtual resources are

created and can be transitioned to other resources (cf. Section 1.1.3). In other words, it defines elements such as the action costs for moving a pawn, directly influencing the player's decision-making within the broader game context. To enrich the game and to increase its replay value, many games involve probabilistic elements, such as dice rolls or card drawings. Virtual resources are hereby not limited to health points or currencies, but also include the players' time, i.e., turns are a valuable resource that need to be balanced as well. In this way, the design of such an economy has a tremendous impact on how players progress through the game. Even if a game shines in all other aspects, if its economy is not well designed, players won't be engaged and will stop playing it. That being said, a game's economy determines the incentive and motivation for players to engage in certain behaviors and why they choose a certain strategy [Schreiber and Romero, 2021].

A game economy must therefore be understood as a system, where already small changes can drastically impact or even break the whole experience. For this reason, balancing a game's economy requires a lot of manual fine-tuning, as well as a deep understanding of the particular game, and experience in balancing games in general. Klint and van Rozen [2013] introduced a notation to formulate a game economy as directed graph, where connected nodes define, based on a node's type, a relation configured through the weight of the connecting edge. The nodes and edges are considered here only as particular functional components, completely detached from the narrative elements they represent to the players. The commercial tool *Machinations*[15] implements this idea and offers users an interface to model and simulate such an economy.

There is, however, little research on game economies and graph data in games at all, yet graph data in games is omnipresent. Only Rogers et al. [2023] investigate the generation of game economies targeting different complexity levels, but their work is limited to crafting systems and does not address other narrative setting or a broader macro-level context.

To fill this gap, this chapter introduces graph data as a relevant data domain for PCG in games, and introduces two methods for generating such data using game economies as an example, as well as a method for balancing a game economy. As a first step, we build on the results from Chapter 5 and transfer the PCGRL (PCG via RL) framework by Khalifa et al. [2020] from the game level domain to the generation of graph data by introducing an adapted version of the MDP, we name G-PCGRL (Section 7.3). To the best of our knowledge, no work has been published yet on generating graph data for games with RL. In this context, we consider the integer matrix, which in PCGRL represents a game level, as the adjacency matrix of a graph. While PCGRL evaluates the validity of a level if it is playable, we evaluate the validity of a graph if it satisfies a set of multiple constraints. These constraints are oriented on the definitions of game economies by Klint and van Rozen [2013].

The results indicate that G-PCGRL is able to generate valid graphs within a short computational window in comparison to baselines, but has limited scalability for increasing graph sizes. To address this shortcoming, we introduce the GEEvo (Game Economy Evolution) framework (Section 7.4) and use an evolu-

[15]Online available at: `https://machinations.io/`

tionary algorithm to generate graph data. Other search-based approaches could be applied here, such as hill climbing or simulated annealing. However, evolutionary algorithms are more robust and less likely to get stuck in local optima [Eiben and Smith, 2015]. This is also a reason, why evolutionary algorithms are a widely used search-based approach for PCG for games [Togelius et al., 2024]. For more information see Section 2.3 on search-based optimization and Section 2.1 on PCG in the background chapter.

Using an evolutionary algorithm, we are able to generate larger valid graphs with more complex constraints, however, it requires a much higher computational window to explore the search space in comparison to G-PCGRL using RL. In addition, these algorithms are highly dependent on randomness.

With the GEEvo framework, we propose two evolutionary algorithms: (1) to generate valid game economies, and (2) to balance a game economy within a given constraint. A first question to be answered here is, what it actually takes to balance an economy and how we can quantify this. To estimate and gather information, we use, as in previous chapters, a simulation-based evaluation function (cf. Yannakakis and Togelius [2011]). In Chapters 5 and 6, we used agent-based simulations to sample from the game's win-rate distribution, which is not applicable for game economies that define abstract systems. The simulations we will conduct in the context of GEEvo can be understood as sampling from the distribution of possible states of the game's economic system.

7.2 Related Work: Graph Data Generation

This section provides an overview of related work on graph data generation, particularly in the context of this chapter. We separate two parts: graph data generation in the context of PCG for games and also in a more general context. Several approaches have been proposed for graph data generation, differing in both the methods used and the domains they target.

Graph rewriting systems, such as graph grammars, are widely used to generate graph-based structures. Graph grammars provide a flexible way to model complex structures and are used in the context of games to generate levels, as in platformers or dungeon layouts [Valls-Vargas et al., 2017; Hauck and Aranha, 2020; Gutierrez and Schrum, 2020], rules [Cook et al., 2013], or graphs as the basis for narratives [Alvarez and Font, 2022]. Whereas graph grammars apply rules iteratively, with G-PCGRL we focus on the learning of a set of constraints in a general way from which a graph can be constructed.

Rogers et al. [2023] generate graph-based game economies using an evolutionary algorithm. The authors thereby focus on the generation of economies with different perceived complexity levels and proof their results alongside with a user study. In their implementation, the economy exists solely within the narrative context of a crafting system, where nodes form a tree structure and each resource is always passed on to a subsequent conversion process. In contrast to Rogers et al., we focus with GEEvo also on the *balancing* of the economy and allow for generating more complex economies including probabilities or supporting loops within the graph.

The generation of graph data is not limited to game-related content. It is also widely used in the field of chemistry, where a common problem domain is the optimization of the properties of molecular graphs. Elton et al. [2019] provide a detailed survey on how to apply deep learning for graph-based molecular design, Leguy et al. [2020] generate molecules with an evolutionary algorithm, and Kwon et al. [2021] apply an evolutionary algorithm combined with recurrent neural networks. Similar like we will do in G-PCGRL, Zhou et al. [2019] formulated the problem of graph data generation as an MDP to apply RL. The molecule graph generation is a sequential task where the agent's action space is to add atoms and to create or remove bonds. For G-PCGRL, however, we formulate the problem differently and manipulate the graph's adjacency matrix as opposed to a sequential task. You et al. [2018] combine a Graph Neuronal Network (GNN) with RL to generate molecules with an optimized drug-likeliness. The GNN is used to create node embeddings based on existing data, the molecules are then generated with RL, where the agent's action space is to connect atoms (nodes) and subgraphs through link prediction. The key distinction for G-PCGRL and GEEvo from chemistry-related graph data generation is that molecular graph generation uses real-world data. The constraints for generating new molecules are derived or learned from this data. In contrast, this work addresses a scenario where no data exists, only abstract concepts which can be formulated as a set of constraints.

7.3 G-PCGRL: Graph Procedural Content Generation via Reinforcement Learning

With this section, we aim to automate the generation of game-related graph-based structures. Therefore, we explore PCGRL also in the context of the previous Chapters 5 and 6 to transfer findings for procedural graph data generation using RL. We contribute G-PCGRL (C 4), a novel method for PCG for graph data with RL. Once trained, a RL model can generate content quickly and is less dependent on randomness compared to evolutionary methods [Khalifa et al., 2020]. The generation of graph data including RL is highly researched in others fields like chemistry in the context of molecules [You et al., 2018; Leguy et al., 2020; Elton et al., 2019]. To generate new realistic molecules, the work there incorporates existing data. When creating games, however, this is in most cases not applicable since there is no data to learn from.

The PCGRL framework has been introduced for the generation of game levels which we have used and extended in the previous Chapters 5 and 6. In PCGRL, the RL agent learns a policy π to alter an integer-represented tile-based level – for this section, we frame this as the adjacency matrix of a graph. By entering parameters $\mathcal{P}$ into the initialization process, trained models can be further controlled in terms of the size and number of different node types of the graph to be generated (Figure 7.1). The validity of a graph is verified with a set of constraints $\mathcal{C}_{graph}$. Training the generator is selecting a distribution from the set of all possible distributions of graphs D_π. Following the previous notations on content generation with PCGRL (cf. Equations 5.1 and 5.3), generating graphs

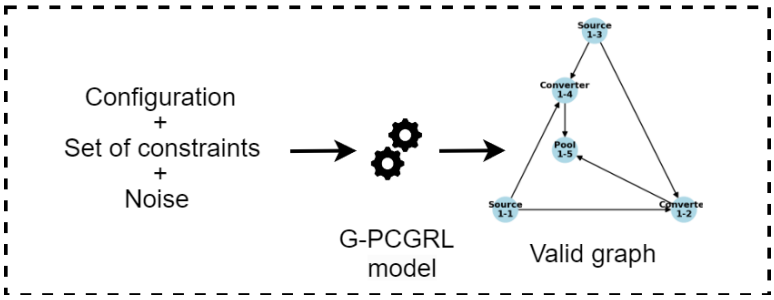


Figure 7.1: Graph generation with G-PCGRL: A G-PCGRL model is controllable through a given configuration to generate a valid graph from random noise according to a set of constraints on which it has been trained.

$\mathcal{G}$ can be described formally as:

$$PCG^{RL}_{graph} : \mathcal{P} \times \{\mathcal{C}_{graph}\} \rightarrow \mathcal{D}_{\pi}(\mathcal{G})\,. \tag{7.1}$$

Likewise as when balancing levels, a graph G is generated by rolling out the trained policy π_ϕ with parameters ϕ in the graph environment, which can be understood as sampling from the distribution $d_{\pi_\phi} \in \mathcal{D}_\pi$ defined by the learned policy under a given input parameter $p \in \mathcal{P}$:

$$G \sim d_{\pi_\phi}(\cdot \mid p)\,. \tag{7.2}$$

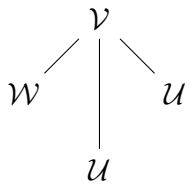
Classification within the PCG taxonomy: According to the PCG taxonomy by Yannakakis and Togelius [2025b] (described in Section 2.1), G-PCGRL differs from the method for balanced level generation introduced in Chapter 5 and can be classified as follows. In terms of content, it is *functional*, *non-spatial*, and *necessary*, but it can also be used for optional content. In terms of its method, it is *stochastic*, *generate-and-test*, *trained*, *controllable*, and *iterative*. In terms of its role, G-PCGRL can generate content both at *runtime* and offline. It operates *autonomously*, and is *experience-agnostic*, as no player or player experience is modeled.

7.3.1 Graph Representations

We introduce the *graph-narrow* and the *graph-wide* representations, both inspired by the corresponding ones in the PCGRL paper [Khalifa et al., 2020]. Like in Section 5.3, we further extend PCGRL's representations in order to adapt the MDP to fit our problem domain. In PCGRL the RL agent modifies an integer matrix representing a tile-based game level. For G-PCGRL, this matrix is considered to be the adjacency matrix of a graph (Figure 7.2). A simple example of such an adjacency matrix $\mathbf{M}$ is given in Figure 7.2a. In this thesis, we extend the conventional concept of an adjacency matrix to represent the node types on the matrix's diagonal. The different types of nodes are represented with the set $\mathcal{N}$. Different nodes are encoded as $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$ for instance. The information as to whether two nodes are connected by an edge is a binary information and is represented by a 1 (connected) or 0 (no connection). To allow controllability in terms of also

$$\mathbf{M} = \begin{bmatrix} \mathcal{U} & 0 & 0 & 0 & 0 \\ 1 & \mathcal{V} & 0 & 0 & 0 \\ 0 & 1 & \mathcal{W} & 0 & 0 \\ 0 & 1 & 0 & \mathcal{U} & 0 \\ 0 & 0 & 0 & 0 & \mathcal{E} \end{bmatrix}$$

(a) An adjacency matrix $\mathbf{M}$.



(b) The graph visualization of $\mathbf{M}$.

Figure 7.2: An extended adjacency matrix $\mathbf{M}$ (left) and its representation as a graph (right). The nodes are represented on the matrix's diagonal and encoded depending on the node type. Edges between nodes are represented as 1, 0 for no connection. $\mathbf{M}$ is the parameter $p \in \mathcal{P}$ in Equation 7.1.

generating graphs smaller than the size of $\mathbf{M}$, we pad $\mathbf{M}$ with additional symbols of type $\mathcal{E}$ (empty). Since we are dealing with undirected graphs and infer the direction from the domain (cf. Section 7.3.2), no information is assigned to the upper right area of the adjacency matrix, resulting in a triangular shape of the action space.

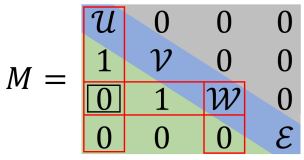
To formulate the problem of graph data generation as an MDP, we need to define the 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ represents the set of states and $\mathcal{A}$ the set of actions. P is the state transition probability function and R is the reward function, defining the dynamics and rewards of the environment. In PCGRL, $\mathcal{S}$ (also called the observation space) and $\mathcal{A}$ are modeled with different *representations* (cf. Section 4.4).

For all representations, the action space is the prediction of whether two nodes are connected or not, in other words the area below the adjacency matrix's diagonal. The diagonal cannot be modified by the agent, it is randomly created or can be predefined to allow controllability. Since all values in the matrix represent labels, we apply a one-hot-encoding in all representations as the final transformation for the observations as in PCGRL.

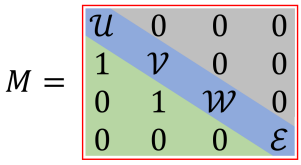
Graph-narrow representation

Like in the PCGRL narrow representation, we implement the position selection through the environment. Since it is important that the agent is given the opportunity to potentially modify any position of the adjacency matrix, we use the narrow sequential mode. The episode terminates when either a complete iteration over the action space has been performed, a maximum number of changes (cf. change percentage, [Khalifa et al., 2020]) or iterations have been exceeded, or the graph is valid (cf. Section 7.3.3).

In the context of a graph, a position represents the state of an edge between two nodes. Using this representation, the action space $\mathcal{A}$ is minimal and contains only two actions: toggling the state of an edge or leave it unchanged. $\mathcal{A}$ is then $\mathcal{A} = 2$. To encode the information about which edge between which nodes has been selected for modification, an appropriate observation is required. Therefore, we model this as two vectors, each representing a selected node and its connections. This is comparable to the cropping approach in the PCGRL narrow representation. The resulting matrix $\mathbf{S} \in \mathcal{S}$ in a configuration with $|\mathcal{N}|$ differ-



(a) The adjacency matrix in the graph-narrow representation: A position, i.e., an edge (black frame), is selected by the environment. The observation consists of the two affected nodes and their connections (red frames).



(b) The adjacency matrix in the graph-wide representation: the agent is free to select any position in the matrix that is in its action space (green). The observation space is the entire matrix (red frame).

Figure 7.3: Visualization of the action and observation spaces for the graph-narrow (left) and graph-wide (right) representation. The action space is highlighted with a green area, the observation space with red frames. The diagonal (blue) is static for both representations and can be configured at initialization or is generated randomly. Since we are only dealing with undirected graphs so far, the gray area is not used, but is important for defining the shape of a square matrix.

ent node types is then: $\mathbf{S} \in (\{0,1\} \cup \mathcal{N})^{2\times n\times|\mathcal{N}|}$. An example matrix is shown in Figure 7.3a. In this example, the position marked with a black frame $(0,2)$ is selected by the environment; the resulting observation space is marked with red frames. Here it is the two vectors representing the nodes $\mathcal{U}$ and $\mathcal{W}$ with all their edges. The advantage of the graph-narrow representation is its small action space, however, its cropped observation space and the limited selection mechanism can be a disadvantage, as there is less information available to the agent.

Graph-wide representation

As in the PCGRL wide representation, the agent is given full control in the graph-wide representation. For each time step, the agent predicts a position on the adjacency matrix, i.e., an edge, and whether this edge state should be toggled or not. Full observation of the adjacency matrix provides the agent with maximum information, but significantly increases the action and observation space. The size of the action space $|\mathcal{A}|$ for a graph with n nodes can be calculated using the triangular formula (Equation 7.3) with $n-1$, since the diagonal is not part of $\mathcal{A}$. $|\mathcal{A}|$ is then $|\mathcal{A}| = \mathrm{Tri}(n-1)$.

$$\mathrm{Tri}(x) = \frac{x \cdot (x+1)}{2} \tag{7.3}$$

The RL agent in G-PCGRL predicts an action a as a discrete value. To map this action to a 2D position in the matrix, we use the discriminant of the square polynomial (cf. triangular root, Equation 7.4) and round it up.

The y-position is then calculated with $\text{Tri}^{-1}(a)$:

$$y = \text{Tri}^{-1}(a) = -\left\lceil \frac{-1 + \sqrt{1 + 8a}}{2} \right\rceil . \tag{7.4}$$

The x-position is calculated from the resulting y-position with:

$$x = a - \text{Tri}(y - 1) - 1 . \tag{7.5}$$

The observation space is like in the original wide or the swap-wide representation the full matrix one-hot encoded. The resulting matrix for $\mathbf{S}$ in a configuration with $|\mathcal{N}|$ different node types is then: $\mathbf{S} \in (\{0, 1\} \cup \mathcal{N})^{n \times n \times |\mathcal{N}|}$. The episode ends when the graph is either valid (see Section 7.3.3), or a maximum number of changes or iterations is exceeded.

7.3.2 Declaration of Sets of Constraints

With a set of constraints $\mathcal{C} = \{\varphi_1, ..., \varphi_n\}$, we define multiple constraints φ as abstract concepts to define when a graph is valid. It thereby defines the search space for the RL agent consisting of the set of all possible graphs $\mathcal{G}$ (cf. Equation 7.1). These constraints determine which node types must be connected and, conversely, which must not be connected. Only if all constraints are satisfied, a graph G is valid:

$$G_{\text{valid}} \in \{G \in \mathcal{G} \mid \forall \varphi \in \mathcal{C}, \varphi(G)\} . \tag{7.6}$$

To keep the syntax simple, human-readable, easily editable, and extendable, we choose a JSON-like syntax, as described in the example in Figure 7.4. For each node type, we declare the set of node types, each of which must be connected to a node of that type with a direct edge. The number of connected valid node types must be ≥ 1. By default, direct connections to node types that are not included in the set of allowed edges of a node type are not allowed. Although only this type of constraints is supported, it can also be considered as a simple and lightweight ontology.

Depending on the node size of a graph and the number of nodes of the same type, a set of constraints can allow for multiple different graphs. Figure 7.2b shows an example of a valid graph of the set of constraints in Figure 7.4a. In a literal sense, these constraints require that a node of the symbolic type $\mathcal{U}$ must be connected to at least one node of type $\mathcal{V}$, $\mathcal{V}$ must be connected to at least one $\mathcal{U}$ and $\mathcal{W}$, and $\mathcal{W}$ must be connected to at least one $\mathcal{V}$. A more concrete use case in the specific narrative context of a game economy setting is given in Figure 7.4b, where we substitute the symbolic node types with entities of a typical game economy setting. We stick to the node naming for game economies established by Klint and van Rozen [2013] and we will also use it along with a set of constraints in the context of our GEEvo framework which we will introduce in Section 7.4. Since our primary focus here is on learning to generate valid graphs and the narrative remains yet secondary, we use a subset of the node types intended for later use in the GEEvo context.

$\mathcal{U}$ represents a node of type `Source`, which is an entity where resources are added to the economy, such as a mine or a tree that can be farmed. $\mathcal{V}$ is a `Converter` where one or more resources from `Source` nodes are converted into a new resource (e.g., an item). This abstract design logic is thereby enforced by the set of constraints. The newly transitioned resources are held then in a `Pool` node. In this narrative setting, we can infer the directions from the domain: `Source` always is directed to `Converter` and `Converter` to `Pool`. In the game Minecraft [2011], for instance, the process of crafting (Converter) torches (Pool) from wood (Source) and coal (Source) can be illustrated this way. We will discuss this in more detail in the context of GEEvo in the next Section 4.5 (also cf. Figure 4.4). This concept is not unique to Minecraft and can be found in many other games [Schreiber and Romero, 2021].

$\mathcal{U}$: [$\mathcal{V}$]	Source: [Converter]
$\mathcal{V}$: [$\mathcal{U}$, $\mathcal{W}$]	Converter: [Source, Pool]
$\mathcal{W}$: [$\mathcal{V}$]	Pool: [Converter]

(a) Notation with generalized symbols.

(b) Notation for a game economy setting (cf. Klint and van Rozen [2013]).

Figure 7.4: The set of constraints $\mathcal{C}_1$ for graph data generation as an example, consisting of four rules and three different node types. The same constraints are written with generalized symbols (left) and in the narrative context of a game economy use case (right). G-PCGRL can be trained with an arbitrary set of constraints, we will experiment with several ones that are listed in the appendix (Figure C.2).

7.3.3 Reward Design and Graph Validation

An appropriate reward design is crucial for the successful use of RL. s in PCGRL, we use an intermediate reward to reward the agent at each time step t according to the change it has made from $t-1$ to the validity of the graph G_t in the context of the given set of constraints with a validity function:

$$\text{valid}(G_t) = \sum_{\varphi \in \mathcal{C}} \mathbf{1}_{\{\varphi(G_t)=\text{false}\}} \,. \tag{7.7}$$

It returns the sum of the number of all constraints that are not satisfied. If all given constraints $\mathcal{C}$ are satisfied for every node $v \in \mathcal{V}_{G_t}$, the graph is assigned a score of 0 and considered valid. If an agent's action removed an incorrect edge or created a missing one, the reward for G_t in the context of G_{t-1}) is positive, otherwise it is negative. There is no reward (value 0) if the agent's action did not affect the validity. Since in both representations only the creation or removal of edges between nodes can be performed, there are exactly these five possible reward states. If G_t is valid, we add an additional α to reward the agent. If the graph is invalid, α is zero. The reward at t is defined as:

$$r_t = \text{valid}(G_{t-1}) - \text{valid}(G_t) + \alpha \,. \tag{7.8}$$

Table 7.1: Overview of parameters for experimental setup.

Parameter	Values
Representations	graph-narrow, graph-wide, PCGRL-wide
Graph sizes up to	4, 5, 6, 7, 8, 9, 10
Number of node types	2, 3
Sets of constraints	$\mathcal{C}_1$ (Figure 7.4), $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, $\mathcal{C}_5$

The sets of constraints 2–5 are listed in the appendix in Figure C.2.

7.3.4 Experimental Setup

To evaluate our method, we train multiple controllable models with different configurations for the graph-narrow and graph-wide representations and compare the performance with the original PCGRL wide representation and search-based approaches. The evaluation is done in several steps: First, we describe the experimental setup. Second, we evaluate the overall performance in Section 7.3.5. Finally, we demonstrate the ability of G-PCGRL to generate appropriate graphs for two different game use cases: the generation of economy structures and skill trees (Section 7.3.7).

We test different sets of constraint configurations (e.g., the number of contained constraints), the number of node types, and the graph size (the number of nodes) for all representations. The latter refers to a graph size up to a maximum size, as all models are trained to be controllable to also generate arbitrarily smaller graphs. An overview of the different parameters and their values is given in Table 7.1. This results in 105 trained models, which will be compared and evaluated in this section. All models are trained on a specific set of constraints but are controllable in terms of the size of the graph and the exact number of node types. To train the models in a controllable manner, we ensure that all possible configurations are sampled uniformly during training. To compute metrics for comparison, we generate a sample of 500 graphs from each model.

Like in previous experiments with PCGRL in Chapters 5 and 6, we use Proximal Policy Optimization (PPO) [Schulman et al., 2017] with a multi-layer perceptron architecture for the feature extractor and the value function. We configure both with three fully connected layers of sizes 128, 256, and 128. We use a step size of 125 and 10 parallelized environments. The graph-narrow representation models are all trained for 500k steps, resulting in 400 policy updates. For more information on the setup of PPO see Section 5.4 and Equation 5.5. All graph-wide and PCGRL wide models are trained for 1.5 million steps (1200 policy updates) due to preliminary investigations have shown that the larger action space requires more training steps to converge.

7.3.5 Performance Overall

We measure the overall performance of a G-PCGRL model by estimating the average proportion of validly generated graphs from the sample of 500 per model. The average validity for all models per representation and per maximum graph

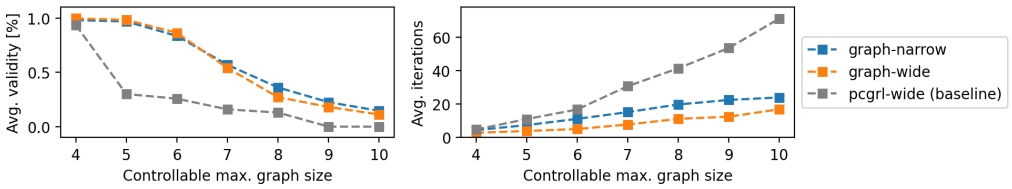


Figure 7.5: The overall performance of the controllable G-PCGRL models compared to the different representations based on the maximum controllable graph size from a sample of 500 each. Performance is measured as the percentage of valid graphs generated (left). The average iterations per maximum graph size are shown in the right subplot. We compare both newly introduced representations to the original PCGRL-wide as a baseline (gray).

size is shown in Figure 7.5 (left). Both graph representations show a similar performance, but the PCGRL wide as a baseline is significantly worse. For smaller graph sizes, the mean validities for both graph representations are at their maximum, however, as the graph size increases, the performance gets worse. Figure 7.5 (right) compares the average iterations of all models per representation and per maximum graph size. Iterations are the number of actions taken by an agent within an episode before termination. An episode is over when the graph is valid, or when a maximum number of changes or total iterations is exceeded. The larger the graph size, the more iterations were required by the agents in both representations. The graph-wide representation, however, requires fewer iterations than the graph-narrow representation. The PCGRL wide baseline requires comparatively many more iterations than the graph representations.

7.3.6 Execution Time and Comparison to Baselines

A major advantage of using RL for PCG is its remarkably fast inference speed. We evaluate this by contrasting it with a random search and an evolutionary algorithm. The experimental setup is as follows: each method is given the same graph configuration and a set of constraints to generate a graph. Execution time[16] is measured as the time required for each method to produce a valid graph from the given combination of configuration and set of constraints. For each method, set of constraints, and graph size, 100 graphs are generated.

The random search method operates by randomly selecting edges within the permissible search space between nodes until the specified constraints are satisfied. This approach doesn't involve learning from previous runs or using prior knowledge. Our implementation of the evolutionary algorithm is based on that described by Rogers et al. [2023] to generate game economy graphs of varying complexity. We also randomly select nodes for crossover, replacing them with their connections between individuals, while mutations occur at a rate of 5 %, randomly replacing a node's connections. An individual's fitness is also determined by the cumulative sum of constraints that are not met (cf. Equation 7.7). The algorithm terminates when a valid graph is found.

[16]We use a 2.6 GHz AMD EPYC family 23 model 1 processor for all experiments.

Setup		Runtime		
Constraints	**Size**	**G-PCGRL**	**Evol. algorithm**	**Random search**
$\mathcal{C}_1$	5	**6.6**	42.2	96.6
	6	**9.4**	59.7	378.1
	7	**5.6**	109.1	2509.7
$\mathcal{C}_2$	5	**7.6**	49.7	457.3
	6	**10.8**	107.6	6811.9
	7	**5.4**	143.1	16112.2
$\mathcal{C}_3$	5	**6.3**	26.1	37.7
	6	**6.3**	45.4	147.1
	7	**3.5**	67.3	750.8
$\mathcal{C}_4$	5	5.6	30.6	**2.8**
	6	**7.0**	47.9	11.2
	7	**7.5**	92.7	100.7
$\mathcal{C}_5$	5	5.8	**0.6**	1.8
	6	7.2	**0.8**	8.6
	7	9.6	**3.1**	24.1

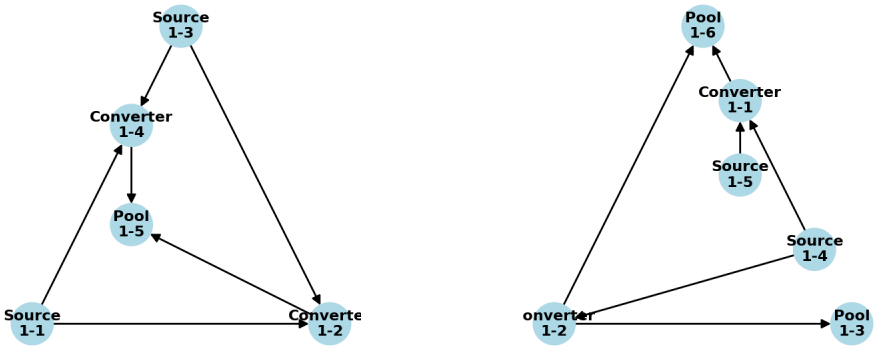
Table 7.2: Comparison of median runtimes across various setups and baselines (in ms). For all G-PCGRL runs, we used models with the graph-wide representations. The set of constraints are defined in Figure 7.4 ($\mathcal{C}_1$) and C.2 ($\mathcal{C}_{2-5}$) in the Appendix.

The results (Table 7.2) indicate that G-PCGRL exhibits notably superior speed in producing valid graphs with the specified combination of configuration and constraints for sets 1, 2, and 3 across various graph sizes. For set 4, G-PCGRL only performs better at sizes 6 and 7. For set 5, G-PCGRL is slower than the evolutionary algorithm and the random search for the size 5, and slower than the evolutionary algorithm for sizes of 6 and 7.

7.3.7 Generated Samples

This section provides insights into generated samples in a game economy and skill tree setting. We show how to control a graph-wide model to output different graphs with varying sizes and different numbers of node types. The model is trained with the set of constraints 1 (Figure 7.4) and is controllable for graphs with a maximum size of six with three different node types. A description of the game economy setting is given in Section 7.3.2.

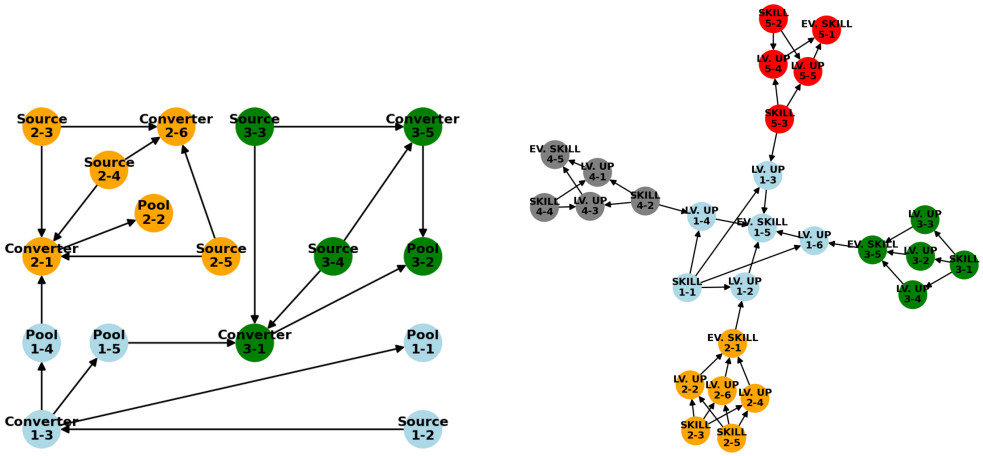
Figure 7.6 shows two samples generated from the model with different configurations. Figure 7.6a shows a graph configured to have a size of five: two source nodes, two converter nodes, and one pool node. In this economy, an item (pool) can be crafted from two source resources, with two conversion processes each requiring both resources. Sample 7.6b adds another source node, resulting in a graph of size six. The model now designs the conversion process so that one source is required for both, but the other two sources are required for one each.



(a) Sample 1 configurations: A graph of size 5 with 2 source nodes, 2 converter nodes, and 1 pool node.

(b) Sample 2 configurations: A graph of size 6 with 3 source nodes, 2 converter nodes, and 1 pool node.

Figure 7.6: Generated samples in a game economy setting from one model controlled by different configurations. The configuration controls the graph size and the number of each node type. The model was trained with the set of constraints $\mathcal{C}_1$ given in Figure 7.4, including three rules and three different node types.



(a) An initial economy graph (blue) extended with additional subgraphs (green and orange).

(b) A skill tree built from one initial tree (blue) and concatenated with multiple subgraphs.

Figure 7.7: The generation of larger graphs by iteratively concatenating the outputs of the same model, each controlled with different configurations. We show this for two domains: game economies (a) and skill trees (b). First, we generate an initial graph (blue). Second, we generate multiple graphs (green, orange, red, and gray), each with different configurations, and concatenate them to the initial graph.

Expanding graphs through concatenation

In this section, we explore the possibilities of generating large graphs of arbitrary size by concatenating multiple generated subgraphs. Thus, outputs from the same model, but each controlled with a *different* configuration, increase the diversity of content. The resulting concatenated graph is not a multipartite graph, since there are edges between nodes within the same subgraphs. For better visualization, we color each subgraph and name the nodes in the schema `subgraphIndex-nodeIndex`. Figure 7.7 shows generated samples of two different domains: an economy graph and a skill tree. For both we use the set of constraints from Figure 7.4. First, we generate an initial graph (blue) and expand it with subgraphs. In the context of a game economy (Figure 7.7a), items may not be the final product, they may also be the input for another conversion process. Therefore, we connect a randomly selected `Converter` node to a `Pool` node from the initial graph. This results in a more complex crafting path for e.g., Pool 3–2, which now also requires Pool 1–5 and Source 1–2.

In Figure 7.7b, we use G-PCGRL to generate a skill tree from subgraphs. In the context of a skill tree node type $\mathcal{U}$ stands for `Skill`, $\mathcal{V}$ for `Lv.up` (level up), and $\mathcal{W}$ for `Ev.skill` (evolved skill). This implements the abstract concept of having basic skills that can be enhanced by leveling up, leading to the development of more advanced abilities that can be further improved. The leveling path of skills can vary depending on the player's preferences and choices. For example, a player could start with the basic skill 5–3 and level it up to skill 1–5 or the evolved skill 5–1. The latter could also be achieved by choosing a different path and leveling up skill 5–2 first.

7.3.8 Discussion & Limitations

The results of the experiments with G-PCGRL showed promising results regarding the feasibility of procedurally generating graph-based data from a set of constraints by manipulating the adjacency matrix with RL. There are, however, several things that need to be discussed.

We presented two different representations, graph-narrow and graph-wide, inspired by the respective PCGRL representations. Apart from minor differences, both appear to perform equally well in terms of the proportion of valid graphs generated in a sample of 500. We trained multiple controllable models with different sets of constraints, node types, and graph sizes. As graph sizes increase, the validity metrics for both representations become worse. This is due to the fact that increasing the maximum graph size nonlinearly increases the complexity and thus the search space for the RL. Furthermore, the models are trained to be controllable *up to* the given size. This also increases the search space and requires more training iterations to learn the controllable configurations. Both are the reasons for the observed results. Training the models to be uncontrollable would improve these results due to the less complex search space. In our opinion, however, it is more beneficial to be able to generate graphs with different configurations using the same model. Therefore, to address the performance loss with increasing graph size, we proposed to concatenate subgraphs generated from the same model but with different configurations. In comparison, the

PCGRL wide baseline performs significantly worse. After all, PCGRL was not designed to generate graph data, but it shows that the graph representations bring improvements to solving this problem.

We compared the execution times of G-PCGRL with an evolutionary algorithm and a random search approach across various sets of constraints and graph sizes. Notably, G-PCGRL outperforms both alternative methods in terms of fast content generation, which is particularly evident for the example sets 1, 2, and 3. Unlike the evolutionary algorithm and the random search, the execution time of our method is solely dependent on the changes made (cf. Figure 7.5, right), showcasing superior robustness compared to the other approaches being strongly dependent on randomness. Additionally, we examined the execution speed for constraint sets 4 and 5. In particular, set 5 allows for a large number of valid graphs within its search space. Consequently, random-based approaches demonstrate faster execution times for comparably simple constraints compared to our method, primarily due to the wide range of solutions within the search space.

On average, the graph-wide representation requires significantly fewer iterations to generate valid graphs than the graph-narrow (Figure 7.5). This is due to its ability to see the full adjacency matrix and then predict where to add or remove an edge. With this capability, it is possible to create a valid graph more accurately and therefore faster. On the other hand, the complexity of the model and the number of training steps required are increased. However, the faster generation combined with the controllability adds the greatest value and we thus recommend using the graph-wide representation.

Creating a valid graph according to a set of constraint can also be approached with a constraint satisfaction problem (CSP) solver. Such methods, however, assume explicit constraints and typically operate by *searching* for feasible assignments, which can be described as deterministic PCG (cf. Equation 2.1). In contrast, RL leverages the constraints in the reward and *learns* a policy that produces a distribution over valid solutions (cf. Equation 7.1). This increases diversity and enables the possibility for optimizing for additional objectives beyond validity. Moreover, since the focus here is on learning the constraints from feedback rather than searching for valid solutions, G-PCGRL would also be applicable when constraints are not (fully) known or fixed, as only feedback on the quality of a current candidate is required.

We have shown how the model can learn to generate graphs based on a set of constraints, and we have experimented with different sets and rule combinations. A limitation is that currently only one type of constraint is supported: if specified in a set of a node type, at least one node of a type must be connected. Future work will extend this to implement additional types of constraints, such as the ability to explicitly exclude node types or require a certain number of edges for a particular node type. Finally, the reward function motivates the agent to create missing edges and remove incorrect ones. This tempts the agent to create graphs with as many edges as possible to increase the reward, potentially reducing the diversity of content created.

In summary, we see this work as a foundational examination, where we apply PCGRL to the generation of graph data in games in general. We see a

high potential for further research in the procedural generation of graph data for games.

7.4 GEEvo: Game Economy Evolution

In the previous section, we explored G-PCGRL, a RL-based approach for generating graph data based on a set of constraints. While G-PCGRL handles a range of graph sizes and outperforms baselines in computational speed, it shares scalability limitations similar to those discussed in Chapter 5 and 6, as well as by Earle et al. [2024].

In this section, we address these limitations by using an evolutionary algorithm as the PCG method. Whereas G-PCGRL focused primarily on graph structure generation, we now shift our focus to the generation and balancing of game economies. With GEEvo, we introduce a framework to generate and balance graph-based game economies simulation-driven. Based on our findings in Chapter 5, we split up the generation and balancing into distinct units to address the PCG and balancing problem separately. An overview of the GEEvo process is given in Figure 7.8.

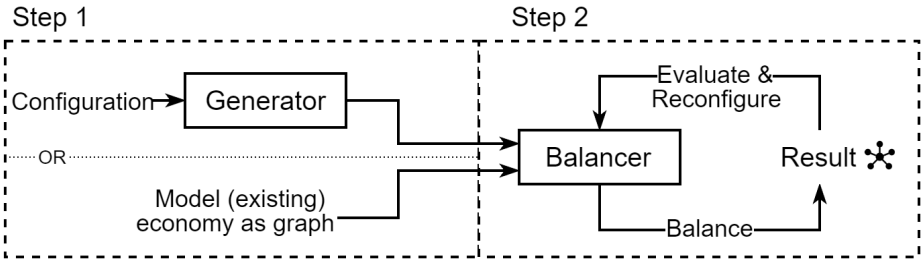


Figure 7.8: The process of GEEvo is two-step: First, a game designer models an existing game economy using our simulation framework or creates one with the generator. Second, the designer sets an objective based on which the balancer then optimizes the economy graph's weights. In an iterative process the designer evaluates the weights found and, if needed, may reconfigure the balancer. This may involve specifying static weights for e.g., a particular narrative context or enhancing the influence of probabilistic elements within the economy.

GEEvo consists of two evolutionary algorithms: the *generator* — a controllable evolutionary algorithm designed to generate random but valid economies, and the *balancer* for optimizing the resource flow by adjusting edge weights in the economy graph. Simulations are conducted using a lightweight framework inspired by Machinations[17], which leverages the domain-specific language introduced in Klint and van Rozen [2013]. With this approach, an economy can be flexibly modeled as a graph with nodes representing functional components. The simulation framework and the process of its execution is defined and described in the foundational concepts chapter in Section 4.5. With GEEvo, a designer e.g., has the capability to model, simulate, and balance the pace at which

[17]Machinations: `https://machinations.io/`

players progress towards a particular achievement, like crafting the mighty sword. To assess our method, we evaluate the general performance on a set of generated economies (Section 7.4.3) and conduct a case study in balancing the damage output of fictitious economies of a mage and an archer (Section 7.4.4).

7.4.1 The Generator: Evolutionary Generation of Game Economies

The generator creates valid game economy graphs within the framework outlined in Section 4.5. A valid economy graph must be weakly connected and adhere to the constraints $\mathcal{C}_{economy}$ in Table 4.3. The generator's task thereby is to connect nodes with edges, meeting all constraints. It operates by defining a population of individuals and iteratively optimizing them through mutations over multiple generations. The generator is designed for controllability, allowing users to specify the number and types of vertices in the generated graph via parameters from the set $\mathcal{P}$. For instance, it can generate an economy with three sources, two random gates, one converter, and four pools. The execution of the algorithm stops if a valid graph has been found or a maximum of allowed steps is exceeded. The configuration space $\mathcal{P}$ and $\mathcal{C}_{economy}$ form the set of distributions $\mathcal{D}$ over the set of content (economy graphs) $\mathcal{G}$. It is defined as:

$$PCG^{EA}_{economy} : \mathcal{P} \times \{\mathcal{C}_{economy}\} \to \mathcal{D}(\mathcal{G}) \,. \tag{7.9}$$

The evolutionary algorithm's internal configuration Θ, such as the crossover and mutation, creates a selected distribution $d_\Theta \in \mathcal{D}$ to generate a graph G from a given configuration $p \in \mathcal{P}$. This is defined as:

$$G \sim d_\Theta(\cdot \mid p) \,. \tag{7.10}$$

Classification within the PCG taxonomy: According to the PCG taxonomy by Yannakakis and Togelius [2025b] (described in Section 2.1), GEEvo's generator can be classified as follows: In terms of content, it is *functional*, *non-spatial*, and *necessary*, but it can also be used for optional content. In terms of its method, it is *stochastic*, *generate-and-test*, *authored*, *controllable*, and *iterative*. In terms of its role, it is designed to generate content *offline* to operate in a *mixed-initiative* setting to support designers. Lastly, it is *experience-agnostic*, since no player or player experience is modeled in any way.

Initialization and Population: During initialization, all vertices defined in the external configuration are initialized depending on their type. The population consists of a configurable number of individuals, with a single individual representing the edge list of the graph. After initialization, the edge lists of all individuals are empty and will be filled iteratively in the execution.

Mutations

Mutations are the driving force to evolve the graph. Since the performance is sufficient, we do not implement a crossover to explore the search space more efficiently. In each generation, for each individual, two vertices are randomly selected to add an edge. If this edge is allowed according to the constraints, it

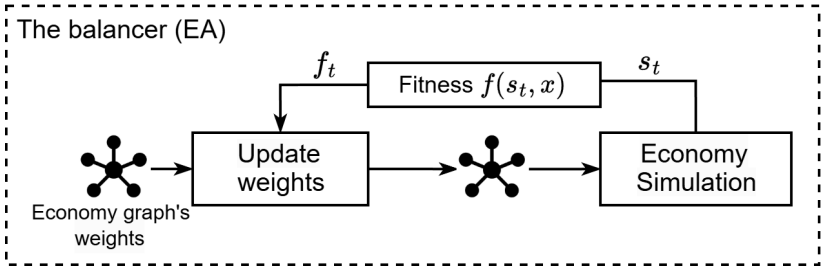


Figure 7.9: The structure of the balancer in detail: The balancer iteratively optimizes an economy's weights toward a balancing objective x. Therefore, it adjusts the weights based on the fitness f_t per time step t in relation to x through crossovers and mutations. f_t is calculated based on the result s_t of multiple simulation runs of the economy.

is added to the individual, otherwise it is not. This simple and greedy approach may get stuck, since at some point the graph is still not valid and no valid edges are allowed anymore. To address this shortcoming, a second mutation, may in each generation, remove a previously created edge from a random individual with a certain probability.

Fitness function

The fitness function embeds the constraints to ensure the validity of the graph G with its vertices V. It creates the graph based on the created edges and sums up the number of dissatisfied constraints for each vertex $v \in V$. Therefore, we use the same Equation 7.7 as for validating graphs in the previous Section 7.3.3 in the context of G-PCGRL. The function is defined in the interval $[\,0, |\,\mathcal{C}\,|\,]$, where 0 represents the maximum fitness of an individual. To achieve best fitness the function must therefore be minimized.

7.4.2 The Balancer: Evolutionary Balancing of Game Economies

The balancer optimizes the weights of one or multiple economies towards a balancing objective which is expressed by parameterizing its fitness function. An overview of the balancer's internal structure is given in Figure 7.9. Utilizing crossovers and mutations, the balancer optimizes its population, consisting of multiple individuals. Following crossover and mutations, the population undergoes sorting based on fitness, retaining only individuals up to the population size for the subsequent generation. The algorithm terminates either when the fitness is at its best for a single individual, or the maximum number of generations is exceeded. In some narrative settings, a designer wants to keep specific values. Therefore, the balancer can be set up with static weights that remain unalterable throughout the balancing process. An example on the application is given in the case in Section 7.4.4.

Individuals and population: A population consists of n individuals where each represents all weights of an economy. It can be thought of as a list where

each weight's index is mapped to the same index of the graph's edge list. In case of balancing two economies at once, an individual consists of the two weights lists of the respective economies. At the initialization of an individual, its weights are set randomly where all values must be greater than 0.

Crossover: In each generation all individuals of the current population are paired randomly for crossover. For each pair, we iterate over both individuals' k and l weights w_{ki}, w_{li} simultaneously and apply one of four randomly selected operations, each with equal probability. The four operations for the new weight at index i are choose w_{ki} or w_{li} directly, $w_{ki} + w_{li}$ or, $w_{ki} - w_{li}$. After crossover, two individuals each produce a new child individual.

Mutations For each mutation a random individual is selected from the population and a random weight is chosen. This weight is then modified by either adding or subtracting the random number by 0.5 each. If subtracting would yield a value < 0 it is set to 1.

Fitness functions

With the fitness function we define the goal towards which an economy should be balanced. All fitness is computed by observing the state $s_t^{p_j}$ (the amount of contained resources) of one or multiple pool nodes $p_j \in \mathcal{P}, \mathcal{P} \subset \mathcal{V}$ at a selected time step $t, t \leq n$ within the $[1, n]$ steps of a simulated economy. Like in the previous chapters in the context of estimating the balance of game levels (cf. Section 5.2.2), we will run each simulation with the same weights m-times (see Section 7.4.3) in order to mitigate randomness.

Balancing a resource to an absolute value: This function (Equation 7.11) is motivated by adapting the weights of an economy so that a selected p_j equals a given value x after a fixed number of time steps n. One possible use case is to balance the economy to be capable of producing a distinct amount x of a resource (p_j) within a given time period n. In a particular narrative context, such a resource can e.g., be coal, damage points, or time units. Since $s_t^{p_j}$ is based on stochastic simulations, we add an additional parameter α as a threshold value to the average so that the algorithm can also terminate at values close to the maximum fitness. Preliminary tests have shown that in practice it is in most cases impossible to achieve the specified value identically, but it is possible to get very close to it. In addition, as we have discussed earlier in the context of game level balancing, balance can already be assumed when a value close to the desired value is reached (cf. 4.1). Moreover, as we will later explore in our case study in Section 7.4.4, the configuration of this parameter is further important to control the influence of randomness in an economy when balancing. Equation 7.11 is defined in the interval $[0, 1 + \alpha]$, where $1 + \alpha$ represents the maximum fitness of an individual. The auxiliary function $prop$ (Equation 7.12) computes the proportion between x and $s_t^{p_j}$, i.e., how far the two values deviate from each other.

$$f_1(s_t^{p_j}, x) = \alpha + \frac{1}{m}\sum_{i=1}^{m} \text{prop}(s_{ti}^{p_j}, x), \quad p_j \in \mathcal{P} \tag{7.11}$$

$$\text{prop}(s_t^{p_j}, x) = \begin{cases} \min(\frac{x}{s_t^{p_j}}, \frac{s_t^{p_j}}{x}), & s_t^{p_j}, x \neq 0 \\ 0, & s_t^{p_j} \cdot x = 0 \quad \text{and} \quad s_t^{p_j} + x \neq 0 \\ 1, & s_t^{p_j}, x = 0 \end{cases} \tag{7.12}$$

To balance two nodes $s_t^{p_j}$ and $s_t^{p_k}$ within the same economy instead of a single absolute value we can use Equation 7.11, but with a different parametrization of f_1 by passing $s_t^{p_j}$ and $s_t^{p_k}$ directly as parameters: $f_1(s_t^{p_j}, s_t^{p_k})$, $p_j, p_k \in \mathcal{P}$.

Balancing two nodes of different economies to the same value: Function f_2 (Equation 7.13) parametrizes Equation 7.11 to balance two nodes of two different economies θ and ϕ. We will apply this function in the case study in Section 7.4.4 to balance the dealt damage of a mage and an archer class within the same time period.

$$f_2(s_t^{p_j}, s_t^{p_k}) = f_1(s_t^{p_j}, s_t^{p_k}), \quad p_j \in \mathcal{P}_\theta,\ p_k \in \mathcal{P}_\phi \tag{7.13}$$

7.4.3 Evaluation of the Generator and Balancer

The evaluation is twofold: First, we investigate on the general performance of the generator and second, on the balancer. Evaluating the economy balancer is also twofold: We first evaluate its general capability of balancing economies for arbitrary configurations and then test it in a case study. We use for all experiments an AMD EPYC family 23 model 1 processor with 2.6 GHz. One core is assigned per execution of a single graph.

Evaluation of Game Economy Generation

To evaluate the generator, we create a set of 200 economy graphs with a number of nodes in the range of 5 to 20. Also, the distribution of different node types is randomly varied. Within 50k iterations, the algorithm could generate valid graphs according to the constraints and the external configuration in 97 % of cases. The generated dataset therefore consists of 194 graphs. The median number of iterations required to complete is 641 (90 % quantile: 9354), the median running time is 25 ms (90 % quantile: 296 ms). We use this set of economy graphs for the evaluation of the balancer with different balancing objectives.

Evaluation of Game Economy Balancing

The evaluation of economy balancing consists of two parts: We first evaluate the balancer's general capability of balancing economies for arbitrary configurations and then test it in a case study.

Table 7.3: Results for balancing the generated dataset of economy graphs using fitness function 1 (Equation 7.11) towards different values of α.

	$\alpha = 0.05$	$\alpha = 0.01$	$\alpha = 0.0$
Balanced (%)	93.3	83	58.8
Improved (%)	77.3	88.7	94.8
Initial balanced (%)	27.3	8.8	2.5
Median generations	1	7	196
Median execution time (s)	18.4	66	703.2

General performance with different balancing objectives To assess the balancer's overall performance, we apply it to the previously generated dataset, focusing on balancing a randomly selected pool of an economy to a specific value after a defined number of simulation steps. To test the algorithm's adaptability to varied values, both the number of simulation steps and the specific target value are randomly chosen within the intervals $[10, 30]$ and $[20, 100]$, respectively. An example would be balancing the pool for torches in Figure 4.4a to a value of 28 after 16 simulation steps. Therefore, we employ the fitness function (Equation 7.11) introduced earlier. Given the probabilistic nature of random gates, achieving maximum fitness is often impossible; hence, we compare the results for different values of α. The timeout for the algorithm to stop is set to 500 iterations. The size of the population is set to 20 and we run ten simulations with the same weights each generation for each individual. This results in 300 simulation runs per generation alone. The computational effort involved in carrying out the simulation is acceptable. Conducting ten simulations per economy and generation consistently results in a computing time of less than 284 ms.

Table 7.3 presents the results, indicating a significant variation in the proportion of balanced economies based on α. As the threshold α increases, so does the proportion of balanced economies, including those that were initially balanced. Additionally, the median execution times and generations are dependent on α, with higher values of α leading to solutions being found in fewer generations, thereby reducing overall computation time.

7.4.4 Case Study

Many game genres, such as MOBA (Multiplayer Online Battle Arena), offer players the option of choosing from different characters, each of which is assigned to a specific class. While each character has a unique game design, characters of the same class have a similar play style. The different character designs offer players various strategies to win the game. However, in order for a game to be balanced, it is necessary to ensure that different strategies, if played well, are viable to win the game [Schreiber and Romero, 2021].

In this case study, we examine GEEvo for its ability to balance two different economies, using two popular classes as an example: a mage and an archer. For each class, our goal is to achieve a comparable maximum damage output within a specified time frame. Therefore, we use Equation 7.13. Both economy graphs

are shown in Figure 7.10. The mage's game design is based on casting two different abilities (Figure 7.10a). Each ability has a specific mana[18] cost, a cooldown[19], and a value for the damage it deals. In our setting mana is generated at each time step via a pool node. So, there are seven values which need to be balanced accordingly: the cooldowns, mana costs, and damage values for both abilities; as well as the overall amount of mana regenerated per time step. In contrast, the archer is designed to perform attacks that have a cooldown based on its attack speed (Figure 7.10b). Each attack has a chance to deal additional damage (cf. critical damage). In this economy five values can be adjusted for balancing: the probabilities for a normal attack and critical damage, the damage values for both, and the attack speed. Given the narrative context for modeling cooldown of abilities for both economies, we configure these weights to be static, allowing us to manually set the value to one (cf. Section 7.4.2).

We experiment with two different values for the fitness threshold α, each time with the same seed. To mitigate randomness, we run ten simulations with a length of 30 time steps per generation and a population size of ten. The algorithm terminates after two generations with a total computation time of 1.4 seconds for $\alpha = 0.05$. For $\alpha = 0.01$ the algorithm terminates after six generations in 16.6 seconds.

Results of the found attributes (weights) are displayed in Table 7.4. For both runs, the balancer finds a solution and terminates within the permitted number of generations. The weights found for the mage are both comparable, whereas those for the archer differ mainly for the probabilistic values for a critical hit. With $\alpha = 0.05$, a critical hit (A2) with a change of 12 % would cause three damage, a normal attack only one damage. For $\alpha = 0.01$, the probabilities for a critical hit are irrelevant, as the balancer has equalized the damage for both cases. The detailed discussion, also in the context of the influence of the parameter α, will follow in Section 7.4.5.

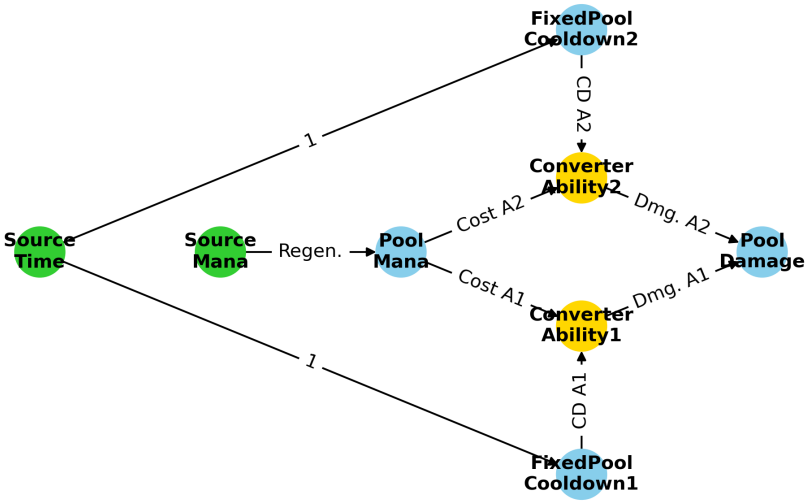
7.4.5 Discussion & Limitations

The results of our experiments showed that the generation and value optimization for balancing graph-based game economies with the proposed framework is feasible. There are, however, several points that need to be discussed.

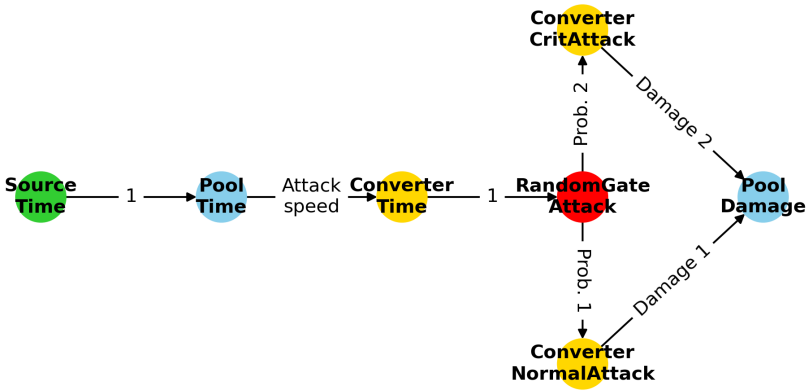
The GEEvo generator is controllable in terms of the number and types of nodes and generated valid graphs in terms of the given constraints, showing an average validity of 97 %. A median execution time of 25 ms indicates fast performance. In comparison to Rogers et al. [2023], our implementation is also able to construct game economy graphs that do not only represent tree structures and therefore allow loops or contain probabilistics, for instance. This allows for greater precision and flexibility in modeling economies [Schreiber and Romero, 2021; Klint and van Rozen, 2013]. So far we only focused on the validity of generated graphs in relation to the node types. One approach for future work is therefore to focus on creating interesting or differently complex economies.

[18] Mana is a fictional resource commonly used in video games as a cost to cast abilities.

[19] Serving as a balancing mechanism in video games, cooldowns define the time interval during which an ability is disabled after use. Here the unit is simulation steps.



(a) The mage's economy graph. Seven values can be balanced accordingly: the cooldowns (CD), mana costs (Cost A1, A2), and damage values (Dmg.) for both abilities (A1 and A2); as well as the overall amount of mana regenerated (Regen.) per time step.



(b) The archer's economy graph. Five values can be balanced accordingly: the overall attack speed, the probabilities (Prob.) to perform a normal or critical attack and the the damage values for each.

Figure 7.10: Two economy graphs for the case study to balance the damage dealing of a mage (a) and an archer (b). The values to be balanced are the weights on the edges. Fixed values are represented by absolute values.

We investigated the general performance of the balancer by applying it to each economy in the generated dataset to a randomly chosen target value with a randomly chosen simulation length. The results (Table 7.3) vary greatly dependent on the value of α. The best-balanced proportion yields $\alpha = 0.05$ with a proportion of 93.3 %. The configuration $\alpha = 0.0$ allows no margin for the sim-

Table 7.4: Results after balancing attributes for the mage and archer economy compared for two different values of α with the goal to deal the same damage within a given period of time. For a description of the abbreviations used for the particular economy attributes see the description of Figure 7.10.

Result α=0.05				Result α=0.01			
Mage	w	**Archer**	w	**Mage**	w	**Archer**	w
M. Reg.	3	A-Speed	2	M. Reg.	2	A-Speed	1
A1 CD	1	A1 Prob.	0.88	A1 CD	1	A1 Prob.	0.76
A1 M.	3	A1 Dmg.	1	A1 M.	3	A1 Dmg.	2
A1 Dmg.	3	A2 Prob.	0.12	A1 Dmg.	3	A2 Prob.	0.24
A2 CD	3	A2 Dmg.	3	A2 CD	2	A2 Dmg.	2
A2 M.	2			A2 M.	2		
A2 Dmg.	3			A2 Dmg.	3		
$\sum$ **Dmg.**	55	53.8$\pm$4.8		60		60$\pm$0	
Fitness		0.95 + α				1.0 + α	

ulated target values, thus the proportion is smaller. Also, the execution time and needed number of generations differ per α. Since the permissible margin of $\alpha = 0.05$ allows for greater scatter, the target balance can be achieved faster and therefore its median generation and execution time are much faster compared to smaller α. However, there is no setting where all economy graphs could be balanced. In cases where the balancer improved overall, but could not achieve the expected quality, the algorithm may have gotten stuck in local optima.

Another problem is the challenge of balancing randomly selected combinations of pools and values in combination with the distribution and networking of different node types. For instance, there are cases in which a certain target value cannot be mathematically achieved within the randomly chosen simulation length. This could be addressed by interpreting the balancing constraint as a value range. In particular this is beneficial for use cases where the actual value is not important, but the perfect balance is.

With the case study of two fictional economies (mage and archer class), we delve into how the balancer optimizes both to ensure equal damage output in the same time frame, addressing the objective in game balancing for diverse strategies and preventing a single dominant strategy from consistently prevailing. We compare the results of two runs with two different values of α, representing the threshold for values to consider as balanced. For both configurations, the balancer could find a solution within a short number of generations. A key finding here is that for a low value of α (0.01), the algorithm tries to minimize the fluctuation of values caused through the stochastic simulation. While the probabilities for normal and critical hits still differ, it equalizes the damage for both hits and thereby mitigates the randomness. At the one hand, it fits the balancing criteria, at the other it might not be a desirable solution since now the intention of the economy design is obsolete. To address this shortcoming, we

recommend using low values of alpha only if randomness should have no or little impact. In other words, α can be used not only to configure the precision to a specific value but also to modulate the stochastic impact.

Another solution for a game designer is to use static weights for e.g., one of the damage values to prevent the balancer from adjusting it. For this case study we used the parametrization of the fitness function in Equation 7.13 to balance two economies at once to the same value. In many cases, however, new game entities are to be integrated into an existing game ecosystem. Therefore, to not adjust the whole existing system, Equation 7.11 can be used directly to balance the newly introduced content to a value which fits into the ecosystem. Another point to mention is that, due to the recursive execution of the economy framework, the mage's economy implements the play style of a so-called spammer, using an ability whenever its cooldown is ready and enough mana available. Human players would also use other strategies, such as waiting for an ability that deals more damage even though another one is available. A future approach is thus the implementation of lightweight bots, each with different strategies for the same economy.

It was shown that the economy simulation framework is able to implement basic concepts of game economics on examples from the game Minecraft and two fictitious character classes. However, it lacks components to e.g., influence edge weights based on values of pools as implemented in Micromachinations for instance. This would open further opportunities to study more complex economies and other balancing objectives such as the counteracting of positive feedback loops.

Another limitation is that this research is based on simulations on an abstract game. Since by design no real game is implemented and no humans for testing are involved, playtests are still required. It still depends on human players whether the generated economies with the computed weights are actually fun to play within the narrative setting chosen by the designer. Therefore, this research is intended to support the early stages of game design to find an interesting economy through generation and balancing initial values for the first player tests. Lastly, evolutionary computing relies heavily on randomness, particularly when combined with probabilistic simulation behavior. Hence, we attempt to mitigate this by evaluating GEEvo on a large sample of economies with a wide range of configuration values.

7.5 Conclusion

In this chapter, we targeted automated game balancing via PCG and simulations from the perspective of game economies. This adds another dimension beyond the accelerated automated level balancing examined in previous chapters in order to answer RQ 4. We identified a research gap in generating graph data for games and addressed it by introducing two PCG methods tailored to graph-based structures. As a common thread throughout this thesis, we avoided incorporating domain-specific information into our methods by design, also making the approaches presented in this chapter transferable to other domains or game settings.

In order to build upon and extend our previous results with PCGRL (Chapter 5 and 6), we introduced G-PCGRL, a controllable PCG method for learning to generate graph data from a given set of constraints with RL. Although G-PCGRL demonstrates faster graph generation compared to baselines in experiments, it lacks scalability for larger graphs. We addressed this shortcoming with a second approach and introduced GEEvo, a framework using evolutionary algorithms to generate and balance graph-based game economies.

In the early stages of game design, there are usually only abstract concepts. G-PCGRL embeds these concepts as a set of constraints and learns to generate graph-based content accordingly to support designers in this process. To frame the problem as an MDP, we introduced two new representations for graph data generation for the PCGRL framework. In addition, our method is controllable in terms of the output to be generated and performs better than the plain PCGRL on this task. Moreover, we demonstrated a superior generation speed for more complex sets of constraints when compared to both an evolutionary algorithm and a random search. Through experiments with different representations, graph sizes, and sets of constraints, we further demonstrated the feasibility of our method. However, since the action space does not grow linearly with the maximum controllable graph size, we observed a decrease in performance in terms of the proportion of valid graphs generated at larger sizes. To address this shortcoming in generating larger graphs, we proposed a recursive concatenation of generated graphs from the same model controlled with different configurations and introduced our second approach GEEvo.

GEEvo is a framework for generating and balancing graph-based game economies in a two-step process using evolutionary algorithms and simulations. By considering game economies from an abstract perspective, GEEvo is independent of a specific game or genre and is intended to support designers in the early stages of development. Building on our insights from Chapter 5, we separated the generation and balancing processes into separated components. This separation allows for greater flexibility — for example, enabling the generation of a new economy or the balancing of an existing one independently. In addition to a lightweight framework for simulating the economies for balancing inspired by the definition of game economies by Klint and van Rozen [2013], we presented a fitness function that can be parameterized differently to balance economies towards various objectives. For instance, this enables balancing selected elements within an economy to reach a specified target value after a defined simulation window, as well as simultaneously balancing selected elements of two independent economies at once. The results of benchmarking experiments show that the balancer can optimize the weights of the economies to arbitrary values and simulation lengths in most cases. We further evaluated GEEvo in a case study using fictional economies of two popular game character classes.

We conclude that representing game economies as graphs is suitable for generating such structures in the context of PCG. Although we have demonstrated feasibility in terms of both generation and balancing, further evaluation is needed to assess how humans perceive these economies.

Chapter 8

Conclusion, Implications, and Outlook

This thesis explores algorithmic approaches to automating the game balancing process in terms of two facets: game levels and economies. In doing so, we contribute to the academic fields of search-based optimization and deep Reinforcement Learning (RL), particularly in applications to Procedural Content Generation (PCG) and automated game balancing. Games stand out for their uniqueness, and related work therefore proposes highly tailored solutions for specific games. In contrast, we propose methods that can be used independently of a specific game. We thus introduce a data-driven method to reliably quantify the balance of a competitive two-player game using simulations (RQ 1). To accelerate level balancing and its generation, this work is the first to frame game balancing as both, a Markov Decision Process (MDP) and a PCG problem (RQ 2). In a study with human playtesters, we evaluated the artificially estimated balance and proved that it was perceived as intended in most scenarios (RQ 3). Beyond levels, we introduce two approaches for generating game economies and one for balancing them, governing game balance on an abstract, macro level (RQ 4).

In Section 8.1, we conclude by summarizing our findings in response to the research questions posed in Section 1.2 and outlining the implications of this thesis in Section 8.2. Section 8.3 discusses limitations, Section 8.4 ethical considerations, and Section 8.5 directions for future work.

8.1 Summary

RQ 1 **Quantification of balance:** *What is a reliable, data-driven foundation that enables automated, game-independent measurement and quantification of balance in competitive, two-player game levels?*

To establish a data-driven foundation for expressing the balance of a competitive two-player game level numerically, we combine two components: (1) the concept of an existing fairness metric, Statistical Parity [Dwork et al., 2012], which is commonly applied in fair machine learning research, and (2) data collected from multiple runs of an agent-based game simulation. Using the Statistical Parity as a basis, we express the balance of a two-player game within the fixed interval

$[0, 1]$, where 0.5 indicates balance, whereas 0 and 1 indicate maximum imbalance against a particular player. In contrast to related works, our approach is by design completely independent of the game itself. We only take into account *how often* each player wins, without using game-specific information, such as health or victory points. A game-independent solution is superior, since it can be easily applied to other competitive, two-player settings. In the context of RQ 2, we show that this approach can be seamlessly transferred to a different environment and used with distinct heuristics for each player in the simulations as well.

Information about which agent(s) win(s) is collected using multiple heuristic, agent-based simulations of the game. Since games often contain probabilistic elements, the simulations must be run multiple times with an identical setup in order to isolate the balance from randomness. This can be considered as sampling from the win rate distribution of a game. Although this approach is flexible in terms of agent and appliance configuration, executing the simulations is computationally intensive. To reduce this effort, we propose an approach to determine the appropriate number of simulations, minimizing their number while maximizing the robustness of the estimation. This number is, however, game-dependent and should therefore be evaluated for each environment accordingly, since the probabilistic elements of a game, in combination with the heuristics used to control agents in simulations, influence the number of simulations required. This is our contribution C 1.

RQ 2 Accelerating automated level balancing: *How can automated game level balancing be accelerated while maintaining content quality, diversity, and asymmetries?*

To accelerate automated level balancing at inference time, we formulate level balancing as both, a PCG problem and an MDP to apply RL. Our method balances levels entirely through tile placement, such as by balancing the spatial availability of resources. Once the model is trained, inference requires less steps in comparison to baselines. This reduces the computational costs caused by game simulations (cf. RQ 1). To learn a policy for level balancing, we adapt and extend the PCGRL (PCG via RL) framework by Khalifa et al. [2020] with swapping-based action space representations.

Overall, our method can balance given imbalanced but playable levels in 91.5 % of cases in our evaluation environment. To simplify the process for RL, we introduce an architecture with separated units, enabling balancing to be treated as a fine-tuning step after level generation. This allows generation and balancing to be addressed independently. Moreover, this approach allows for the balancing of human-created levels. This is our contribution C 2.

To thoroughly address this research question, we explore our method through an additional series of application-driven investigations. Collectively, these investigations strengthen our contribution C 2. We improved the method iteratively, such as in terms of tile representation and the size of the action space. We also experimented with different types of representations based on the action space of PCGRL. We report that our swap-wide representation gives the best

results in comparison to those introduced with PCGRL. However, swap-wide has limited scaling capabilities due to its non-linear dependence of its action space on the level size. In comparison to local, greedy, search-based approaches that do not learn from experience, such as random search and hill climbing, we show that RL requires fewer steps to optimize the balance of a given level. Thus, the generation requires less computational time by avoiding unnecessary simulations in production. In general, the original PCGRL approaches outperform their hill climbing analogues, and PCGRL swap-wide outperforms hill climbing in general. Thus, we conclude that good performance arises from two factors: (1) using the swap-based action space representation pattern in combination with (2) using RL.

In addition, we examine our method of balancing levels at various degrees of imbalance. While the results indicate that this is generally possible, the approach performs best when the goal is either perfect balance or maximal imbalance. Moreover, we demonstrate that our method can be applied to asymmetric agent archetype setups where multiple strategies must be maintained balanced entirely through level adaptation. This demonstrates that both our architecture and the balance estimation framework from RQ 1 are flexible and robust, supporting the use of different heuristics, including distinct ones for each player simultaneously. Since the agents then have different strengths, the initial setups are more imbalanced than when both are the same. We find that learning to balance game levels becomes more difficult to learn as the initial inequity of the archetypes increases. Furthermore, we demonstrate the transferability and the application of our method to another environment and show that it can also handle environments where playability is not always guaranteed.

Finally, we observe an alignment problem with using the Statistical Parity metric to express balance. Some levels generated by the models happen to be balanced, but no player can win. While this is permitted according to the metric, it is not intended. The metric also exploits scenarios in which both agents follow the exact same path, resulting always in a draw. These cases indicate limitations of the Statistical Parity metric.

RQ 3 **Human perception of balance:** *What is the human perception and evaluation of the artificially quantified balance of game levels through simulations?*

An empirical evaluation with human playtesters, supported by hypothesis testing, shows that our level balancing approach improved human balance perception in most cases, compared to the initial imbalanced version. This provides evidence that the heuristics used in the agent-based simulations appropriately replicate human behavior for balance simulation. For humans it is difficult to evaluate and express the balance of a game, also partly due to the subjective nature of perception. To address this shortcoming, we used a comparative survey design under blind conditions, in which participants were asked to compare a balanced and an imbalanced version of a level after playing them.

Moreover, the balance of a game level is multifaceted, consisting of various functional components. Thus, we asked participants about their perceptions of different aspects of the game. Our findings show that the perceived differences

in balance depend on the level but correspond to the particular changes made by the RL model for balancing. This is our contribution C 3.

RQ 4 Automated game economy generation and balancing: *How can automation techniques be integrated into the generation and balancing of game economies?*

Game economies are powerful, abstract concepts that greatly influence a game's overall balance. Following the existing formal representation of game economies as graphs by Klint and van Rozen [2013], we identify graph data as an underrepresented area of game content in PCG research in general.

Building on previous results from accelerating automated level balancing (RQ 2), we propose an MDP for controllable graph data generation (G-PCGRL, Graph PCG via RL). G-PCGRL extends the PCGRL framework in combination with results from RQ 2 to generate graph data by *learning* a policy to manipulate the adjacency matrix of a graph to fulfill a given set of constraints. Therefore, we introduce new graph representations for the narrow and wide action space representation and demonstrate that they are better suited to this problem domain than the existing representations. In addition, we show that G-PCGRL can learn to generate valid and diverse graphs for different sets of constraints and graph sizes. Moreover, G-PCGRL is controllable in terms of the graph size and the node types. In comparison to random search and an existing evolutionary approach for game economy generation [Rogers et al., 2023], G-PCGRL is superior in computational speed, in particular when increasing the graph size. However, due to the dependence of the RL's action and observation spaces on the graph size, scaling up to generate larger graphs is limited. With G-PCGRL, we propose a fast, controllable, and reliable method for graph data PCG. Scalability may be slightly improved by omitting controllability, but controllability is a better deal in the context of PCG. G-PCGRL forms our contribution C 4.

To address G-PCGRL's shortcoming in generating larger graphs and to explore the automated balancing of graph-based game economies, we adapt the approach to a purely search-based approach: the GEEvo, (Game Economy Evolution) framework. In addition, GEEvo incorporates insights from level balancing, and also separates content generation and balancing into different units using two evolutionary algorithms. Likewise when balancing levels, we simulate the generated content by executing the economy over several time steps to evaluate the balancing state. We demonstrate that GEEvo is able to generate game economies of different sizes and can balance economies of arbitrary configurations in most cases (93.3 %).

While examining the balancing in a case study, we find that it is important to configure the algorithm for balancing to not produce a "perfect" balance, but to produce a balance that is within an acceptable range (cf. padding in balancing in Chapter 6). This is particularly important in order to avoid weakening or even mitigating any probabilistic mechanics that a designer may have intended. This is consistent with a finding in level balancing, where the RL model, in some cases, also attempts to eliminate all probabilistic elements by keeping players

away from food tiles to reduce any potential risk in ensuring a reliable balance. While GEEvo can generate larger graphs with more complex constraints, it requires more computational effort compared to G-PCGRL. Since G-PCGRL models are trained in advance, GEEvo's evolutionary algorithm must start from scratch each time it explores the search space. GEEvo is our contribution C 5.

In summary, we conclude that automation techniques can be integrated into game economy generation and balancing by: (1) modeling economies as graph structures, (2) simulating the economies, (3) applying search-based optimization approaches, but also (4) allowing space for randomness in the balancing process.

The Feast & Forage environment: Although several environments for PCG or balancing have been introduced by related work, we have identified a lack of environments for level balancing directly via PCG. To bridge this gap, we introduce the *Feast & Forage* game environment, which can be used to apply, explore, test, and evaluate automated balancing of competitive, tile-based, two-player game levels via PCG. Based on the existing NMMO (Neural Massively Multiplayer Online) environment [Suarez et al., 2019], Feast & Forage is lightweight enough to run experiments with billions of simulated game rounds, and complex enough that it is not easy to be balanced asymmetrically. In addition, it contains a probabilistic mechanic that can cause different outcomes, even when the agents are deterministic. The heuristics and rules can easily be adapted and extended to e.g., investigate asymmetries in player archetypes or to add other functional components, making the environment a well-suited playground for automated game balance research. Therefore, it can serve as a versatile testbed for future research. Along with this thesis, we have developed a human-playable prototype of the game in the context of our contributions to accelerating balanced level generation (C 2) and its empirical evaluation (C 3). The environment is published as open source (see Table A.1). Feast & Forage forms our contribution C 6.

Finally, we expect that recent advances in automated content creation, such as images [Rombach et al., 2022], audio [Oord et al., 2016], and text [Chang et al., 2024], to impact game creation processes, including PCG. These advances will accelerate asset generation and can therefore contribute to a potential new era of interactive content, which is generated on demand and is tailored to players' progress, or even to their personalities (see also the considerations in Section 8.4). However, prompt-controlled foundational models cannot contribute directly to game balancing automation at the current state of the art in 2025. This is due to their limited reasoning ability, which is necessary for actually understanding the rules and mechanics of a game. Moreover, it is up to humans to decide whether they perceive the game as balanced and therefore as fair — and everyone is different.

8.2 Implications

We will now summarize and discuss the implications of this thesis for researchers studying automated game balancing and fairness, as well as the PCGRL framework.

8.2.1 Automated Game Balancing

Randomness is optimized out We proposed two methods for automating game balance that are independent of a specific game. Based on our adaptations of the PCGRL framework, we introduced an architecture that frames level balancing as a PCG problem. We demonstrated its applicability in two tile-based game environments. With GEEvo, we proposed a framework for generating and balancing graph-based game economies. This framework can be used to balance games on an abstract, macro level and can be easily transferred to various types of games.

A major finding of this thesis is that we observed automated approaches bypassing probabilistic game mechanics to stabilize the balance, for both balancing facets (Chapters 6 and 7). While probabilistics play an important role in game design in order to create an entertaining and replayable experience, they introduce uncertainty by counteracting deterministic results. Wether the approach is purely search-based or involves a model for learning to balance, randomness is attempted to be "optimized out". While this is reasonable from an optimization perspective, it is not the result that a game designer would want. For this reason, we included configurable parameters to introduce a small bias, in order to leave room for randomness. This implies that the goal is not to automatically optimize balance by ensuring exactly equal win rates for all players, but rather to ensure that win rates are *very close* to equal.

8.2.2 PCGRL

Content generation beyond "just" levels This thesis has used, adapted, and extended the PCGRL framework by Khalifa et al. [2020] and goes far beyond PCGRL's original goal of generating tile-based game levels (Chapters 5, 6, and 7). Our findings highlight that manipulating an integer matrix using RL in order to generate content can also be employed as a fine-tuning process for level balancing or generating graph data. For both of these applications, we have introduced new representations of the MDP that are tailored to the respective use case and problem domain. In addition, we have shown that these representations outperform the original PCGRL ones. Our swap-based action space representation pattern addresses problems requiring fine-tuning of existing content, while our graph representations manipulate a graph's adjacency matrix. The latter approach is controllable by design.

Moreover, in the context of level balancing, we have demonstrated that PCGRL can learn from a simulation-driven reward function, as well as constraint-checking reward functions.

8.2.3 Balance and Fairness in Games

Games must be balanced, not fair Based on existing definitions of the terms *balance* and *fairness* from various disciplines, we emphasize that balance refers to a state of equilibrium, whereas fairness is also an ethical and social concept that inherently involves the human perspective (Section 4.1). According to Schreiber and Romero [2021], winning a game should feel earned, and thus, players' decisions must directly impact the game's outcome. If all players were identical clones, winning or losing would solely depend on the game's probabilistic elements. To ensure this, the game must be balanced first to provide the same conditions for all players across all facets of its system. But human players are never identical. From a design perspective, a game should thus not strive to be fair, as it is exactly the differences between human players that should influence the game's outcome.

Therefore, we emphasize that: (1) balance must be carefully maintained to provide a stable foundation that isolates skill from randomness, and (2) it is the interplay between humans and the balanced system that makes games truly interesting and engaging by incorporating the social and ethical dimensions that give games their depth and meaning. Returning to the opening quote by Oscar Wilde on page 1, it is therefore a good thing that also games are, by design, never fair.

8.3 Limitations

Human evaluation remains necessary: Throughout this thesis, we have introduced various PCG methods for automating balanced game level and economy generation. These methods can reduce the need for human manual work, such as playtesting. However, the balance is artificially estimated. Therefore, we agree with Volz et al. [2018], that considering human perception of balance is essential and thus a (final) human evaluation will always remain necessary to evaluate how humans perceive the intended balance.

To address this to some extent, we conducted an empirical study with human playtesters to evaluate our level balancing approach. Human testing is costly and time-consuming, letting us test a subset of levels only, whereas our method could generate thousands. We did not evaluate the human perception of game economy balance Within our GEEvo framework, we did not evaluate human perception of game economy balance, as it is intentionally held abstract and does not support the direct creation of a playable prototype for user testing. For this reason, our testing is limited to evaluating the performance of the proposed algorithms using metrics based on simulated balance.

Lastly, we proposed methods to optimize balance through creating a mathematical equilibrium of even win rates for all players. By design, we did not aim to generate games which are fair, since fairness includes players' subjective backgrounds, which create the differences that isolate skill from randomness (cf. Section 8.2.3).

Scalability: We have introduced RL-based content generators for level balancing and graph data generation. One limitation of these approaches is their limited scalability for the level and graph sizes that can be generated, mainly caused by the exponentially growing action space dependent on the size.

For game levels, we addressed this issue by further optimizing e.g., the action space of the MDP to improve performance. We achieved good results for generating levels up to a size of 6×6 and limit the method to that size. Larger levels may be possible with significantly more computing resources in combination with different configurations of the RL algorithm. As this work focuses on using PCG to generate balanced levels simulation-driven, the challenge of improving scaling lies more in solving a foundational RL problem than in solving a PCG problem directly. Therefore, we exclude it from the scope of this work. Since graph data generation with G-PCGRL does not require simulations to evaluate the content, computational reduction is not an argument. Thus, we introduced a second method (GEEvo) which has improved scalability at the expense of computational complexity.

Complexity: We evaluated our RL architecture for balancing tile-based levels through level design in two environments. These environments are both research environments with a limited complexity and are thus, of course, not intended for commercial sale or serious entertainment. The contributions of this thesis are focused on the design, implementation, and study of the algorithmic approaches for generation and balancing rather than game design. Since our method was developed independently of a specific game, it can easily be transferred to other, more complex, competitive, two-player tile-based games. However, the computational cost increases depending on the complexity of the game and the heuristics used to estimation balance through the simulations.

We adapted PCGRL to learn to generate graph data based on a given set of constraints (G-PCGRL). However, the types of constraints that can be used are limited. Similarly to our level balancing approach, G-PCGRL is not dependent on specific types of constraints. This makes it easy to extend to include other, more complex constraints.

We followed the existing formal definition of game economies established by Klint and van Rozen [2013] and also kept our framework for generating and balancing game economies (GEEvo) abstract. This enables the design and study of the flow of a game's virtual resources at a macro level while ensuring our framework is easily applicable to various types of games. We showed how our evolutionary algorithm can create balance in abstract economic simulations. However, we did not test it in a real game setting. This would help to evaluate whether the simulated balance appears balanced to human players.

Our simulation framework is limited to a list of functional components. For instance, it is currently not possible to create economies with positive feedback loops, which are a popular engagement mechanic in game design [Schreiber and Romero, 2021]. In addition, functional components, such as the converter, behave in a simple, rule-based manner, without any deeper logic or intelligent behavior. Therefore, executing economies to simulate specific cases or more complex behavior is, as implemented in this thesis, not possible. We focused on

the algorithmic implementation, so we intentionally kept the simulation framework lightweight. Like the other approaches introduced in this thesis, the simulation framework is independent of the algorithms and can therefore easily be extended to address the aforementioned limitations without affecting the design of the introduced algorithmic approaches. We see potential in further developing this in future work (cf. Section 8.5).

8.4 Ethical Considerations: Critical Thoughts on Automating Game Balance for other Purposes

This thesis contributed to the study of game balancing by exploring methods to automate and improve this process. However, game balancing can be exploited for monetization purposes, which can further encourage addictive behavior in certain players. Automating this process can help to achieve this goal. Many publishers already use data-driven insights from thousands of players [Seif El-Nasr et al., 2013], and this practice will only increase in the future. In their position paper, Seif El-Nasr and Kleinman [2020] discussed ethical considerations of data-driven game development in general. In this final section, we will focus on these concerns in the context of game balancing, particularly its misuse for monetization strategies and the associated risk of gaming disorders.

Monetization: The digital gaming industry, especially the mobile gaming sector, knows how to best balance a game to maximize not only entertainment, but also income. As we have mentioned several times throughout this thesis, balancing is important for incentivizing players to engage in certain behaviors and with the game itself. Publishers know that this can also be used to nudge players to spend money on the game, whether for purely cosmetic content, to speed up in-game advancement (pay-to-progress) or for upgrades that enhance the player's virtual power in the game (pay-to-win). While computer games used to be sold as a finished software, modern games are continuously developed after release to keep players on board – and to generate more revenue [Nieborg, 2016].

Many games offer microtransactions – small real-money purchases – or gambling elements, such as loot boxes and Gacha mechanics, that allow players to spend real money inside the game. A small percentage of players is particularly susceptible to this and generates most of the revenue, essentially paying for all the other players. This practice is known as *whaling*. Game designers can identify psychological weaknesses and balance the game to encourage this group of players to spend even more, a concept known as a "dark pattern". Close et al. [2021] showed that game publishers disproportionately profit from moderate and high-risk gamblers. While all players are affected by these elements, children and adolescents are particularly vulnerable and must be protected.

Legislators are trying to address this issue. For example, in a resolution from 2022, the European Parliament emphasized the need for greater transparency regarding loot boxes and their effect on winning the game, e.g., how monetization

is tied to the game's balance.[20] The German legislation tries to protect young players via the Youth Protection Act (Jugendschutzgesetz), which requires labels for games with (random) in-game purchases and to take this into consideration when assigning age ratings to games.[21] Furthermore, parental controls and spending limits must be implemented.[22] In our opinion, however, legislators are not doing enough to restrict manipulative strategies implemented through clever game balancing. In our view, the situation is likely to become even more challenging in the future.

Internet Gaming Disorder: Addiction to video games is officially known as *Internet Gaming Disorder (IGD)* [Darvesh et al., 2020] and is listed in the International Classification of Diseases of the WHO.[23] While designed for engagement, well-balanced game play can be too effective for certain players, resulting in addictive behaviors up to a gaming disorder. For an IGD diagnosis five out of nine criterias must be met such as the use of gaming as escape or relief from a negative mood, or the loss of interest in other activities, among others [Petry et al., 2014].

The reward-driven mechanics of video games, which are primarily balanced by the game's economy, can contribute considerably to triggering addiction. These mechanics are what make progress or achieving victory possible and, most important, that it feels *earned*. This is also related to the previously mentioned psychological incentives to encourage spending real money on gambling-like mechanics with intentional game balancing.

With that in mind, why not use automated game balancing strategies to improve game systems that enhance how players perceive rewards and their effect on their psyche? What if we combined such a system with a data-driven system of millions of players to better target specific groups? Or, we could use available personal data to fully customize the entire game for each player, targeting individual vulnerabilities with personalized PCG to gradually paying out rewards to create a psychological addiction.

While this is, of course, a dystopian scenario, the technological foundation for such systems is close at hand. On the one hand, further customizing game content through PCG and automated balancing can provide unprecedented ways of entertainment. On the other hand, as we saw in the previous two paragraphs, there are also drawbacks, in particularly for especially vulnerable player groups. Therefore, it is up to society in combination with legislators to decide how to address this issue in the future.

[20] European Parliament, resolution of 10 November 2022 on esports and video games (2022/2027(INI)), Official Journal of the European Union, C 161, 5 May 2023, pp.2-9.

[21] § 10b (3) JuSchG – Jugendschutzgesetz of 23 July 2002 (BGBl. I, S. 2730), as last amended by Art. 12 of the Act of 6 May 2024 (BGBl. 2024 I No. 149).

[22] Gesetz zum Staatsvertrag über den Schutz der Menschenwürde und den Jugendschutz in Rundfunk und Telemedien (Jugendmedienschutz-Staatsvertrag – JMStV – of 10 to 27 September 2002), as enacted in Baden-Württemberg by the Act of 4 February 2003 (GBl. 2003 S. 93), last amended by the Act of 25 July 2024 (GBl. 2024 No. 67).

[23] International Classification of Diseases of the WHO: `https://www.who.int/standards/classifications/frequently-asked-questions/gaming-disorder`

8.5 Future Work

While this thesis contributed several methods exploring various use cases for automated game balancing, this is certainly not the end. Thus, we outline several areas for future research.

Incorporating narratives: Our focus was on maintaining a mathematical equilibrium of win rates to ensure balance. However, in the context of an entire game, this must also be connected to the game's narrative. Due to recent advances of Large Language Models (LLMs), future work may thus combine balancing methods such like ours in combination with an LLM. PCGRLLM [Baek et al., 2025a], a very recent work that combines PCGRL with an LLM to create more tailored and fine-grained reward functions, could be a promising approach to extending our level balancing method with PCG. For instance, while still ensuring equal win rates, specific categories of levels could be generated, such as where players always have few or many resources to collect. This becomes particularly interesting when using asymmetric player archetype setups.

In the context of game economies and our GEEvo framework, LLMs can also be beneficial. While we focused solely on the abstract modeling of functional components, LLMs could provide a global economic narrative based on a GEEvo-generated and balanced economy by assigning labels and descriptions to the nodes. One possible approach could be to embed GEEvo in a system called like "Narrative to Economy". Incorporating narratives further motivates players to engage in specific behaviors, adding depth to the game.

Transfer learning: We have proposed two RL methods: one for level balancing and one for generating graph data. We observed a high computational effort for the training of the Proximal Policy Optimization (PPO) models for level balancing due to simulation-driven rewards and limited scalability due to a large, growing action space.

In our opinion, the use of transfer learning, also in the context of the entire PCGRL, can be a promising future research direction to reduce scalability and computational effort problems. For example, scalability to larger level sizes might be improved by initially training on smaller levels and gradually increasing the level size. This curriculum-based approach could also prove beneficial in the context of G-PCGRL. In addition, we could train models with fewer simulations in the early training phase and gradually increase the number of simulations as training progresses.

Smarter economy simulations: In the context of GEEvo, we developed a lightweight game economy simulation framework. However, the different functional components always trigger based on a limited set of static rules. Future work may address this by extending this functionality to investigate how game economies behave when executing more complex actions. Using scripts or smaller trained models could help control this behavior more precisely.

Appendix A

Resources

Table A.1 provides an overview of all created resources in the context of this thesis which are publicly available on Github. Column one contains the number of the belonging chapter and column two a short description. The third column describes the type of the resource such as code, data, or a question catalog. A link where the resource(s) are available is given in column four.

Table A.1: Compilation of all created resources in the context of this thesis.

Chapter	Description	Type	Location
5	Simulation-driven level balancing, PCGRL	Code, Data, Models	`https://github.com/FlorianRupp/pcgrl-simulation-driven-balancing`
5.5	Empirical evaluation	Question Catalog, Data	`https://github.com/FlorianRupp/pcgrl-balancing-empirical-evaluation`
7.3	G-PCGRL	Code, Data, Models	`https://github.com/FlorianRupp/g-pcgrl`
7.4	GEEvo	Code	`https://github.com/FlorianRupp/GEEvo-game-economies`
5, 6	Feast & Forage Environment	Code, Models	`https://github.com/FlorianRupp/feast-and-forage-env`

Appendix B

Empirical Evaluation: Question Catalog

This appendix lists the questions for the empirical evaluation in Section 5.5. The catalog is divided into two parts: Questions which were asked of participants when playing level version 1 (Table B.1) and the questions which were asked of after playing level version 2 (Table B.2).

For each question, the following are listed: the number (column 1); the name of the item (column 2); the question text, as it was presented to the participants (column 3); and the available response options (column 4). The response options use a 5-point Likert scale, with the ordinal values given in brackets.

Table B.1: Questions catalog: Level version 1.

No.	Name	Question	(Value) Choices
Q 1.1	Movement Difficulty Player	How easy was it for you, in your opinion, to access berry bush (food) or water tiles?	(-2) Very inconvenient/ Impossible (-1) Slightly inconvenient (0) Adequate (1) Easy (2) Very easy
Q 1.2	Amount Food Player	How would you rate the amount of berry bush tiles (food) near your starting position?	(-2) Way too few / None (-1) Too few (0) Sufficient (1) Too much (2) Way too much
Q 1.3	Amount Water Player	How would you rate the amount of water tiles near your starting point?	(-2) Way too few / None (-1) Too few (0) Sufficient (1) Too much (2) Way too much
Q 1.4	Movement Difficulty Opponent	How easy was it for your opponent, in your opinion, to access berry bush (food) or water tiles?	(-2) Very inconvenient/ Impossible (-1) Slightly inconvenient (0) Adequate (1) Easy (2) Very easy
Q 1.5	Amount Food Opponent	Compared to your own starting position, how would you rate the availability of berry bush tiles (food) near your opponent's starting position?	(-2) Way too few / None (-1) Too few (0) Sufficient (1) Too much (2) Way too much
Q 1.6	Amount Water Opponent	Compared to your own starting position, how would you rate the availability of water tiles near your opponent's starting position?	(-2) Way too few / None (-1) Too few (0) Sufficient (1) Too much (2) Way too much

Table B.2: Questions catalog: Level version 2.

No.	Name	Question	(Value) Choices
Q 2.1	Movement Difficulty Player	Compared to the previous version, would you say, that you were able to access resource tiles more or less easily than before?	(-2) Way more difficult (-1) More difficult (0) Unchanged (1) Easier (2) Much easier
Q 2.2	Amount Food Player	Compared to the previous version, would you say that you had more or less berry bush tiles (food) near your starting position?	(-2) Way fewer (-1) Fewer (0) Same amount (1) More (2) Way more
Q 2.3	Amount Water Player	Compared to the previous version, would you say that you had more or less water tiles near your starting position?	(-2) Way fewer (-1) Fewer (0) Same amount (1) More (2) Way more
Q 2.4	Movement Difficulty Opponent	Compared to the previous version, would you say, that your opponent was able to access resource tiles more or less easily than before?	(-2) Way more difficult (-1) More difficult (0) Unchanged (1) Easier (2) Much easier
Q 2.5	Amount Food Opponent	Compared to the previous version, would you say that your opponent had more or less berry bush tiles (food) near its starting position?	(-2) Way fewer (-1) Fewer (0) Same amount (1) More (2) Way more
Q 2.6	Amount Water Opponent	Compared to the previous version, would you say that your opponent had more or less water tiles near its starting position?	(-2) Way fewer (-1) Fewer (0) Same amount (1) More (2) Way more

Appendix C

Supplementary Materials

C.1 City Game Environment

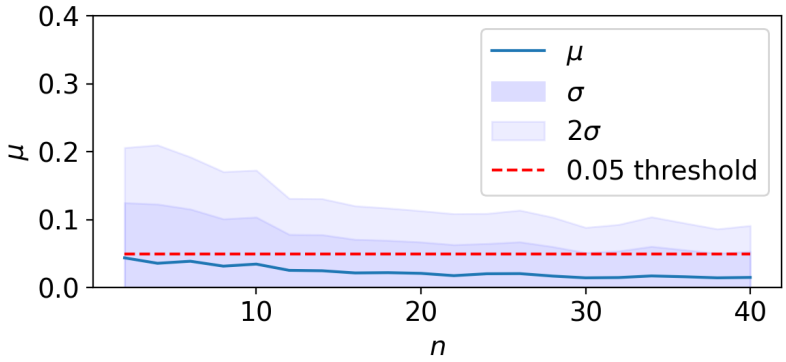


Figure C.1: Determination of a suitable number of simulations for the city environment in Section 6.3. We follow the same approach as described in the context for Feast & Forage in Section 5.2.2. The suitable number of simulations is therefore 30.

Algorithm 5 Heuristic Agent for the City Environment.

```
1: procedure STEP(gameState, tileCosts, targets)
2:     init selectedTarget ← CLOSESTREACHABLETARGET(targets, tileCosts,
                                                    gameState)
3:     action ← FINDSHORTESTPATHTO(selectedTarget, tileCosts, gameState)
4:     if NEXTPOSITION(action, tileCosts, gameState) ∈ targets then
5:         targets ← targets \ {gameState.agent.position}
6:     end if
7:     return action, targets
8: end procedure
```

C.2 Additional Sets of Constraints

An overview of the additional sets of constraints used for the experiments in Section 7.3.4 in the context of G-PCGRL is given in Figure C.2. The sets differ in the rules themselves, the number of node types used, and the number of constraints. In total, five sets of constraints were used for the experiments for G-PCGRL in this thesis; set $\mathcal{C}_1$ is included in Figure 7.4.

$\mathcal{U}$:	$[\mathcal{V}]$
$\mathcal{V}$:	$[\mathcal{U}]$

(a) Set of constraints $\mathcal{C}_2$: two rules with two node types.

$\mathcal{U}$:	$[\mathcal{V}]$
$\mathcal{V}$:	$[\mathcal{U}, \mathcal{V}]$

(b) Set of constraints $\mathcal{C}_3$: three rules with two node types.

$\mathcal{U}$:	$[\mathcal{U}]$
$\mathcal{V}$:	$[\mathcal{U}, \mathcal{V}, \mathcal{W}]$
$\mathcal{W}$:	$[\mathcal{V}]$

(c) Set of constraints $\mathcal{C}_4$: five rules with three node types.

$\mathcal{U}$:	$[\mathcal{V}, \mathcal{W}]$
$\mathcal{V}$:	$[\mathcal{U}, \mathcal{W}]$
$\mathcal{W}$:	$[\mathcal{U}, \mathcal{V}]$

(d) Set of constraints $\mathcal{C}_5$: six rules with three node types.

Figure C.2: Additional sets of constraints for experiments in the context of G-PCGRL.

Bibliography

Adams, Ernest [2014]. *Fundamentals of Game Design*. 3rd. Indianopolis, Indiana, USA: New Riders Publishing. ISBN: 0321929675.

Alvarez, Alberto and Jose Font [2022]. TropeTwist: Trope-based Narrative Structure Generation. In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. Athens, Greece: ACM, pp. 1–8. DOI: 10.1145/3555858.3563271.

Andrade, Gustavo, Geber Ramalho, Alex Gomes, and Vincent Corruble [2006]. Dynamic Game Balancing: an Evaluation of User Satisfaction. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 2. Marina del Rey, California, USA, pp. 3–8. DOI: 10.1609/aiide.v2i1.18739.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer [2002]. Finite-time Analysis of the Multiarmed Bandit Problem. In: *Machine Learning* 47.2, pp. 235–256. DOI: 10.1023/A:1013689704352.

Avedon, Elliott M and Brian Sutton-Smith [1971]. *The Study of Games*. New York, New York, USA: John Wiley & Sons. ISBN: 978-0471038399.

Awiszus, Maren, Frederik Schubert, and Bodo Rosenhahn [2020]. TOAD-GAN: Coherent Style Level Generation from a Single Example. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16, pp. 10–16. DOI: 10.48550/arXiv.2008.01531.

Awiszus, Maren, Frederik Schubert, and Bodo Rosenhahn [2021]. World-GAN: a Generative Model for Minecraft Worlds. In: *2021 IEEE Conference on Games (CoG)*. Copenhagen, Denmark. DOI: 10.1109/CoG52621.2021.9619133.

Baek, In-Chang, Sung-Hyun Kim, Sam Earle, Zehua Jiang, Noh Jin-Ha, Julian Togelius, et al. [2025a]. *PCGRLLM: Large Language Model-Driven Reward Design for Procedural Content Generation Reinforcement Learning*. arXiv:2502.10906 [cs]. DOI: 10.48550/arXiv.2502.10906.

Baek, In-Chang, Sung-Hyun Kim, Seo-Young Lee, Dong-Hyeon Kim, and Kyung-Joong Kim [2025b]. IPCGRL: Language-Instructed Reinforcement Learning for Procedural Level Generation. In: *2025 IEEE Conference on Games (CoG)*, pp. 1–8. DOI: 10.1109/CoG64752.2025.11114105.

Baek, In-Chang, Tae-Hwa Park, Jin-Ha Noh, Cheong-Mok Bae, and Kyung-Joong Kim [2024]. ChatPCG: Large Language Model-Driven Reward Design

for Procedural Content Generation. In: *2024 IEEE Conference on Games (CoG)*. Milan, Italy, pp. 1–4. DOI: 10.1109/CoG60054.2024.10645619.

Bamford, C. [2021]. Griddly: A platform for AI research in games. In: *Software Impacts* 8, p. 100066. DOI: 10.1016/j.simpa.2021.100066.

Barocas, Solon, Moritz Hardt, and Arvind Narayanan [2023]. *Fairness and Machine Learning: Limitations and Opportunities*. Cambridge, Massachusetts, USA: MIT Press. ISBN: 9780262048613.

Beau, Philipp and Sander Bakkes [2016]. Automated game balancing of asymmetric video games. In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. Santorini, Greece, pp. 1–8. DOI: 10.1109/CIG.2016.7860432.

Becker, Alexander and Daniel Görlich [2020]. What is Game Balancing? - An Examination of Concepts. In: *ParadigmPlus* 1, pp. 22–41. DOI: 10.55969/paradigmplus.v1n1a2.

Beukman, Michael, Branden Ingram, Ireton Liu, and Benjamin Rosman [2023]. Hierarchical WaveFunction Collapse. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 19. 1. Salt Lake City, Utah, USA, pp. 23–33. DOI: 10.1609/aiide.v19i1.27498.

Beyer, Marlene, Aleksandr Agureikin, Alexander Anokhin, Christoph Laenger, Felix Nolte, Jonas Winterberg, et al. [2016]. An integrated process for game balancing. In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. Santorini, Greece, pp. 1–8. DOI: 10.1109/CIG.2016.7860425.

Bishop, Christopher M and Nasser M Nasrabadi [2006]. *Pattern recognition and machine learning*. Vol. 4. 4. New York, New York, USA: Springer.

Blizzard Entertainment, Inc. [2010]. *Starcraft II*. Video Game [Windows, macOS]. Irvine, California, USA.

Blizzard Entertainment, Inc. [2012]. *Diablo III*. Video Game [Windows, macOS]. Irvine, California, USA.

Bontrager, Philip and Julian Togelius [2021]. Learning to Generate Levels From Nothing. In: *IEEE Conference on Games (CoG)*. Copenhagen, Denmark.

Braben D. Bell, I. [1984]. *Elite*. Video Game [Commodore 64, Nintendo Entertainment System]. Acornsoft Ltd., Cambridge, England.

Branke, J., B. Scheckenbach, M. Stein, K. Deb, and H. Schmeck [2009]. Portfolio optimization with an envelope-based multi-objective evolutionary algorithm. In: *European Journal of Operational Research* 199.3, pp. 684–693. DOI: 10.1016/j.ejor.2008.01.054.

Campbell, Murray, A. Joseph Hoane, and Feng-hsiung Hsu [2002]. Deep Blue. In: *Artificial Intelligence* 134.1, pp. 57–83. DOI: 10.1016/S0004-3702(01)00129-1.

Cardamone, Luigi, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi [2011]. Evolving Interesting Maps for a First Person Shooter. In: *Applications of Evolutionary Computation*. Berlin, Heidelberg: Springer, pp. 63–72. DOI: 10.1007/978-3-642-20525-5_7.

Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, et al. [2024]. A Survey on Evaluation of Large Language Models. In: *ACM Transactions on Intelligent Systems and Technology* 15.3, 39:1–39:45. DOI: 10.1145/3641289.

Charity, Megan, Ahmed Khalifa, and Julian Togelius [2020]. Baba is Y'all: Collaborative Mixed-Initiative Level Design. In: *2020 IEEE Conference on Games (CoG)*. ISSN: 2325-4289, pp. 542–549. DOI: 10.1109/CoG47356.2020.9231807.

Chouldechova, Alexandra [2017]. Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. In: *Big Data* 5.2, pp. 153–163. DOI: 10.1089/big.2016.0047.

Christian, Brian [2021]. *The alignment problem: How can machines learn human values?* London, UK: Atlantic Books. ISBN: 978-1-78649-433-7.

Clark, Daniel, Corey Konieczka, Adam Sadler, and Kevin Wilson [2012]. *Descent: Journeys in the Dark (Second Edition)*. Fantasy Flight Games, Roseville, Minnesota, USA.

Close, James, Stuart Gordon Spicer, Laura Louise Nicklin, Maria Uther, Joanne Lloyd, and Helen Lloyd [2021]. Secondary analysis of loot box data: Are high-spending "whales" wealthy gamers or problem gamblers? In: *Addictive Behaviors* 117, p. 106851. DOI: 10.1016/j.addbeh.2021.106851.

Cook, Michael, Simon Colton, Azalea Raad, and Jeremy Gow [2013]. Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design. In: *Applications of Evolutionary Computation*. Lecture Notes in Computer Science. Springer, pp. 284–293. DOI: 10.1007/978-3-642-37192-9_29.

Cooper, Seth [2022]. Sturgeon: Tile-Based Procedural Level Generation via Learned and Designed Constraints. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 18. 1. Pomona, California, USA, pp. 26–36. DOI: 10.1609/aiide.v18i1.21944.

Cooper, Seth, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, et al. [2010]. Predicting protein structures with a multiplayer online game. In: *Nature* 466.7307. Publisher: Nature Publishing Group, pp. 756–760. DOI: 10.1038/nature09304.

Crist, Walter [2019]. Playing against complexity: Board games as social strategy in Bronze Age Cyprus. In: *Journal of Anthropological Archaeology* 55, p. 101078. DOI: 10.1016/j.jaa.2019.101078.

Csikszentmihalyi, Mihaly [1990]. *Flow: The Psychology of Optimal Experience*. New York: Harper Collins Publishers.

Darrow, Charles [1935]. *Monopoly*. Board game. Hasbro, Inc., Pawtucket, Rhode Island, USA.

Darvesh, Nazia, Amruta Radhakrishnan, Chantelle C. Lachance, Vera Nincic, Jane P. Sharpe, Marco Ghassemi, et al. [2020]. Exploring the prevalence of gaming disorder and Internet gaming disorder: a rapid scoping review. In: *Systematic Reviews* 9.1, p. 68. DOI: 10.1186/s13643-020-01329-2.

De Kegel, Barbara and Mads Haahr [2020]. Procedural Puzzle Generation: A Survey. In: *IEEE Transactions on Games* 12.1, pp. 21–40. DOI: 10.1109/TG.2019.2917792.

Dijkstra, E. W. [1959]. A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1.1, pp. 269–271. DOI: 10.1007/BF01386390.

Dwork, Cynthia, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel [2012]. Fairness through awareness. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. New York, NY, USA: Association for Computing Machinery, pp. 214–226. DOI: 10.1145/2090236.2090255.

Earle, Sam, Maria Edwards, Ahmed Khalifa, Philip Bontrager, and Julian Togelius [2021]. Learning Controllable Content Generators. In: *2021 IEEE Conference on Games (CoG)*. Copenhagen, Denmark, pp. 1–9. DOI: 10.1109/CoG52621.2021.9619159.

Earle, Sam, Zehua Jiang, and Julian Togelius [2024]. Scaling, Control and Generalization in Reinforcement Learning Level Generators. In: *2024 IEEE Conference on Games (CoG)*. Milan, Italy, pp. 1–8. DOI: 10.1109/CoG60054.2024.10645598.

Eiben, A.E. and J.E. Smith [2015]. *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer. DOI: 10.1007/978-3-662-44874-8.

Elton, Daniel C., Zois Boukouvalas, Mark D. Fuge, and Peter W. Chung [2019]. Deep learning for molecular design—a review of the state of the art. In: *Molecular Systems Design & Engineering* 4.4, pp. 828–849. DOI: 10.1039/C9ME00039A.

Emanuel, Ezekiel J., Govind Persad, Ross Upshur, Beatriz Thome, Michael Parker, Aaron Glickman, et al. [2020]. Fair Allocation of Scarce Medical Resources in the Time of Covid-19. In: *New England Journal of Medicine* 382.21. Publisher: Massachusetts Medical Society, pp. 2049–2055. DOI: 10.1056/NEJMsb2005114.

Erikson, Erik H. [1977]. *Toys and Reasons: Stages in the Ritualization of Experience*. New York, NY, USA: W. W. Norton Company, Inc. ISBN: 978-0393336184.

Feldman, Michael, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian [2015]. Certifying and Removing Disparate Impact. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Association for Computing Machinery, pp. 259–268. DOI: 10.1145/2783258.2783311.

Floreano, Dario, Peter Dürr, and Claudio Mattiussi [2008]. Neuroevolution: from architectures to learning. In: *Evolutionary Intelligence* 1.1, pp. 47–62. DOI: 10.1007/s12065-007-0002-4.

Gallotta, Roberto, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, et al. [2024]. Large Language Models and Games: A Survey

and Roadmap. In: *IEEE Transactions on Games*, pp. 1–18. DOI: 10.1109/TG.2024.3461510.

Gilbert, Nigel [2008]. *Agent-based models*. Thousand Oaks, California, USA: Sage Publications, Inc. ISBN: 978-1-4129-4964-4.

Gisslén, Linus, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar [2021]. Adversarial Reinforcement Learning for Procedural Content Generation. In: *2021 IEEE Conference on Games (CoG)*. DOI: 10.1109/CoG52621.2021.9619053.

Golub, Gene H and Charles F Van Loan [2013]. *Matrix computations*. Baltimore, Maryland, USA: Johns Hopkins University Press. ISBN: 978-1-4214-0794-4.

Gómez-Bombarelli, Rafael, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, et al. [2018]. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. In: *ACS Central Science* 4.2. Publisher: American Chemical Society, pp. 268–276. DOI: 10.1021/acscentsci.7b00572.

Gottman, John M. [1986]. "The world of coordinated play: Same- and cross-sex friendship in young children". In: *Conversations of friends: Speculations on affective development*. Studies in emotion and social interaction. New York, NY, US: Cambridge University Press, pp. 139–191. ISBN: 978-0-521-26321-4.

Granic, Isabela, Adam Lobel, and Rutger C. M. E. Engels [2014]. The benefits of playing video games. In: *American Psychologist* 69.1. American Psychological Association, pp. 66–78. DOI: 10.1037/a0034857.

Gutierrez, Jake and Jacob Schrum [2020]. Generative Adversarial Network Rooms in Generative Graph Grammar Dungeons for The Legend of Zelda. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185631.

Guzdial, Matthew, Sam Snodgrass, and Adam Summerville [2025]. "Constraint-Based PCGML Approaches". In: *Procedural Content Generation via Machine Learning: An Overview*. Cham: Springer Nature Switzerland, pp. 57–72. DOI: 10.1007/978-3-031-84756-1_5.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine [2018]. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1861–1870.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael [1968]. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

Hastings, Erin Jonathan, Ratan K. Guha, and Kenneth O. Stanley [2009]. Automatic Content Generation in the Galactic Arms Race Video Game. In: *IEEE Transactions on Computational Intelligence and AI in Games* 1.4, pp. 245–263. DOI: 10.1109/TCIAIG.2009.2038365.

Hauck, Eduardo and Claus Aranha [2020]. Automatic Generation of Super Mario Levels via Graph Grammars. In: *2020 IEEE Conference on Games (CoG)*. IEEE, pp. 297–304. DOI: 10.1109/CoG47356.2020.9231526.

He, Haibo and Edwardo A. Garcia [2009]. Learning from Imbalanced Data. In: *IEEE Transactions on Knowledge and Data Engineering* 21.9. Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1263–1284. DOI: 10.1109/TKDE.2008.239.

Hernandez, Daniel, Charles Takashi Toyin Gbadamosi, James Goodman, and James Alfred Walker [2020]. Metagame Autobalancing for Competitive Multiplayer Games. In: *2020 IEEE Conference on Games (CoG)*. ISSN: 2325-4289, pp. 275–282. DOI: 10.1109/CoG47356.2020.9231762.

Holmgård, Christoffer, Michael Cerny Green, Antonios Liapis, and Julian Togelius [2019]. Automated Playtesting With Procedural Personas Through MCTS With Evolved Heuristics. In: *IEEE Transactions on Games* 11.4, pp. 352–362. DOI: 10.1109/TG.2018.2808198.

Hom, Vincent and Joe Marks [2007]. Automatic Design of Balanced Board Games. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 3.1. Number: 1, pp. 25–30. DOI: 10.1609/aiide.v3i1.18777.

Hoover, Amy K., Paul A. Szerlip, and Kenneth O. Stanley [2011]. Interactively evolving harmonies through functional scaffolding. In: *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*. Dublin, Ireland, pp. 387–394. DOI: 10.1145/2001576.2001630.

Hu, Chengpeng, Yunlong Zhao, Ziqi Wang, Haocheng Du, and Jialin Liu [2024]. Games for Artificial Intelligence Research: A Review and Perspectives. In: *IEEE Transactions on Artificial Intelligence* 5.12, pp. 5949–5968. DOI: 10.1109/TAI.2024.3410935.

Hudson Soft Co., Ltd. [1983]. *Bomberman*. Video game [NES, PC Engine]. Sapporo, Japan.

Icarte, Rodrigo Toro, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith [2022]. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. In: *Journal of Artificial Intelligence Research* 73, pp. 173–208. DOI: 10.1613/jair.1.12440.

Imabayashi, Hiroyuki [1982]. *Sokoban*. Video game [NEC PC-8801]. Thinking Rabbit Co., Ltd., Takarazuka, Japan.

Istamar, Ahsan Imam, Terence, Samuel Philip, and Hidayaturrahman [2023]. An Autonomous Player Agent for Game Balance Insight on an Educational Video Game. In: *2023 4th International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, pp. 269–273. DOI: 10.1109/AiDAS60501.2023.10284610.

Jaffe, Alexander, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic [2012]. Evaluating Competitive Game Balance with Restricted Play. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Inter-*

active Digital Entertainment 8.1. Number: 1, pp. 26–31. DOI: 10.1609/aiide.v8i1.12513.

Jiang, Zehua, Sam Earle, Michael Green, and Julian Togelius [2022]. Learning Controllable 3D Level Generators. In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. DOI: 10.1145/3555858.3563273.

Kallabis, Leonie, Timo Bertram, and Florian Rupp [2025]. Deceptive Game Design? Investigating the Impact of Visual Card Style on Player Perception. In: *2025 IEEE Conference on Games (CoG)*, pp. 1–8. DOI: 10.1109/CoG64752.2025.11114299.

Karth, Isaac and Adam M. Smith [2017]. WaveFunctionCollapse is constraint solving in the wild. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. Hyannis, Massachusetts, pp. 1–10. DOI: 10.1145/3102071.3110566.

Khalifa, Ahmed, Philip Bontrager, Sam Earle, and Julian Togelius [2020]. Pcgrl: Procedural Content Generation via Reinforcement Learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 16, pp. 95–101. DOI: 10.1609/aiide.v16i1.7416.

Khalifa, Ahmed, Roberto Gallotta, Matthew Barthet, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis [2025]. The Procedural Content Generation Benchmark: An Open-source Testbed for Generative Challenges in Games. In: *Proceedings of the 20th International Conference on the Foundations of Digital Games*. FDG '25. Association for Computing Machinery. DOI: 10.1145/3723498.3723794.

Khalifa, Ahmed, Julian Togelius, and Michael Cerny Green [2022]. Mutation Models: Learning to Generate Levels by Imitating Evolution. In: *FDG '22: Proceedings of the 17th International Conference on the Foundations of Digital Games*. ACM. DOI: 10.1145/3555858.3563267.

Klarkowski, Madison, Daniel Johnson, Peta Wyeth, Mitchell McEwan, Cody Phillips, and Simon Smith [2016]. Operationalising and Evaluating Sub-Optimal and Optimal Play Experiences through Challenge-Skill Manipulation. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. San Jose, California, USA, pp. 5583–5594. DOI: 10.1145/2858036.2858563.

Klint, Paul and Riemer van Rozen [2013]. Micro-Machinations. In: *Software Language Engineering*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 36–55. DOI: 10.1007/978-3-319-02654-1_3.

Koster, Raph [2013]. *Theory of fun for game design*. 2nd. Sebastopol, California, USA: O'Reilly Media, Inc. ISBN: 978-1449363215.

Kwon, Youngchun, Seokho Kang, Youn-Suk Choi, and Inkoo Kim [2021]. Evolutionary design of molecules based on deep learning and a genetic algorithm. In: *Scientific Reports* 11.1, p. 17304. DOI: 10.1038/s41598-021-96812-8.

Lanzi, Pier Luca, Daniele Loiacono, and Riccardo Stucchi [2014]. Evolving maps for match balancing in first person shooters. In: *2014 IEEE Conference on Computational Intelligence and Games*. DOI: 10.1109/CIG.2014.6932901.

Lara-Cabrera, Raúl, Carlos Cotta, and Antonio J. Fernández-Leiva [2014]. On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms. In: *Natural Computing* 13, pp. 157–168. DOI: 10.1007/s11047-014-9418-9.

Larian Studios [2023]. *Baldur's Gate 3*. Video game [Windows, macOS, PlayStation 5]. Ghent, Belgium.

Leguy, Jules, Thomas Cauchy, Marta Glavatskikh, Béatrice Duval, and Benoit Da Mota [2020]. EvoMol: a flexible and interpretable evolutionary algorithm for unbiased de novo molecular generation. In: *Journal of Cheminformatics* 12.1, p. 55. DOI: 10.1186/s13321-020-00458-z.

Liapis, Antonios, Georgios N. Yannakakis, and Julian Togelius [2013]. Sentient Sketchbook: Computer-Assisted Game Level Authoring. In: *Proceedings of the 8th International Conference on Foundations of Digital Games*, pp. 213–220. ISBN: 978-0-9913982-0-1.

Liu, Jialin, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius [2021]. Deep learning for procedural content generation. In: *Neural Computing and Applications* 33, pp. 19–37. DOI: 10.1007/s00521-020-05383-8.

Maguire, Angus [2016]. *Illustrating Equality VS Equity*. Available at: https://interactioninstitute.org/illustrating-equality-vs-equity/. Boston, Massachusetts, USA: Interaction Institute for Social Change.

Mahlmann, Tobias, Julian Togelius, and Georgios N. Yannakakis [2012]. Evolving card sets towards balancing dominion. In: *2012 IEEE Congress on Evolutionary Computation*. Brisbane, Australia: IEEE, pp. 1–8. DOI: 10.1109/CEC.2012.6256441.

Makhlouf, Karima, Sami Zhioua, and Catuscia Palamidessi [2021]. On the Applicability of Machine Learning Fairness Notions. In: *ACM SIGKDD Explorations Newsletter* 23.1, pp. 14–23. DOI: 10.1145/3468507.3468511.

Maleki, Mahdi Farrokhi and Richard Zhao [2024]. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 20.1. Number: 1, pp. 167–178. DOI: 10.1609/aiide.v20i1.31877.

Merino, Timothy, Roman Negri, Dipika Rajesh, M. Charity, and Julian Togelius [2023]. 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* 19.1. Number: 1, pp. 107–115. DOI: 10.1609/aiide.v19i1.27506.

Merrell, Paul and Dinesh Manocha [2011]. Model Synthesis: A General Procedural Modeling Algorithm. In: *IEEE Transactions on Visualization and Computer Graphics* 17.6, pp. 715–728. DOI: 10.1109/TVCG.2010.112.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, et al. [2015]. Human-level control through deep reinforcement learning. In: *Nature* 518.7540, pp. 529–533. DOI: 10.1038/nature14236.

Mojang Studios AB [2011]. *Minecraft*. Video game [Windows]. Stockholm, Sweden.

Morosan, Mihail and Riccardo Poli [2017]. Automated Game Balancing in Ms PacMan and StarCraft Using Evolutionary Algorithms. In: *Applications of Evolutionary Computation*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 377–392. DOI: 10.1007/978-3-319-55849-3_25.

Nagel, Thomas [1995]. *Equality and partiality*. Oxford, UK: Oxford University Press. ISBN: 978-0-19-509839-6.

Newzoo [2024]. *Global Games Market Report 2024*. https://newzoo.com. Available with a free Newzoo account. Accessed: 4.12.2024.

Nieborg, David [2016]. From premium to freemium: The political economy of the app. In: *Social, Casual and Mobile Games: The Changing Gaming Landscape*, pp. 225–240. DOI: 10.5040/9781501310591.ch-016.

Nintendo Co., Ltd. [1985]. *Super Mario Bros.* Video game [Nintendo Entertainment System]. Kyoto, Japan.

Nintendo Co., Ltd. [2001]. *Advance Wars*. Video game [Game Boy Advance]. Kyoto, Japan.

Oganov, Artem R. and Colin W. Glass [2006]. Crystal structure prediction using ab initio evolutionary techniques: Principles and applications. In: *The Journal of Chemical Physics* 124.24, p. 244704. DOI: 10.1063/1.2210932.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, et al. [2016]. *WaveNet: A Generative Model for Raw Audio*. DOI: 10.48550/arXiv.1609.03499.

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, and Pamela Mishkin et al. [2022]. Training language models to follow instructions with human feedback. In: *Advances in Neural Information Processing Systems* 35, pp. 27730–27744.

Oxford English Dictionary [2025]. *game, n.* https://doi.org/10.1093/OED/3374114774. II.6.a.

Perlin, Ken [1985]. An image synthesizer. In: *ACM SIGGRAPH Computer Graphics* 19.3, pp. 287–296. DOI: 10.1145/325165.325247.

Petry, Nancy M., Florian Rehbein, Douglas A. Gentile, Jeroen S. Lemmens, Hans-Jürgen Rumpf, Thomas Mößle, et al. [2014]. An international consensus for

assessing internet gaming disorder using the new DSM-5 approach. In: *Addiction* 109.9, pp. 1399–1406. DOI: 10.1111/add.12457.

Pfau, Johannes [2025]. Progression Balancing × Baldur's Gate 3: Insights, Terms and Tools for Multi-Dimensional Video Game Balance. In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM, pp. 1–12. DOI: 10.1145/3706598.3713162.

Pfau, Johannes, Antonios Liapis, Georg Volkmar, Georgios N. Yannakakis, and Rainer Malaka [2020]. Dungeons & Replicants: Automated Game Balancing via Deep Player Behavior Modeling. In: *IEEE Conference on Games (CoG)*. ISSN: 2325-4289, pp. 431–438. DOI: 10.1109/CoG47356.2020.9231958.

Pfau, Johannes, Antonios Liapis, Georgios N. Yannakakis, and Rainer Malaka [2023]. Dungeons & Replicants II: Automated Game Balancing Across Multiple Difficulty Dimensions via Deep Player Behavior Modeling. In: *IEEE Transactions on Games* 15.2, pp. 217–227. DOI: 10.1109/TG.2022.3167728.

Piaget, Jean [1951]. *Play, dreams and imitation in childhood*. Play, dreams and imitation in childhood. New York, NY, USA: W. W. Norton Company, Inc. ISBN: 978-1-315-00969-8.

Piepenbrink, Rolf and Rafael Bidarra [2025]. Non-Uniform Tile Wave Function Collapse. In: *2025 IEEE Conference on Games (CoG)*, pp. 1–8. DOI: 10.1109/CoG64752.2025.11114084.

Politowski, Cristiano, Fabio Petrillo, Ghizlane ElBoussaidi, Gabriel C. Ullmann, and Yann-Gaël Guéhéneuc [2023]. Assessing Video Game Balance using Autonomous Agents. In: *2023 IEEE/ACM 7th International Workshop on Games and Software Engineering (GAS)*, pp. 25–32. DOI: 10.1109/GAS59301.2023.00011.

Preuss, Mike, Thomas Pfeiffer, Vanessa Volz, and Nicolas Pflanzl [2018]. Integrated Balancing of an RTS Game: Case Study and Toolbox Refinement. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. ISSN: 2325-4289, pp. 1–8. DOI: 10.1109/CIG.2018.8490426.

Railsback, Steven F and Volker Grimm [2019]. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. 2nd edition. Princeton University Press. ISBN: 978-0-691-19082-2.

Rapoport, Anatol [1989]. "Prisoner's Dilemma". In: *Game Theory*. London: Palgrave Macmillan UK, pp. 199–204. DOI: 10.1007/978-1-349-20181-5_23.

Rawls, John [1971]. *A Theory of Justice: Original Edition*. Cambridge, Massachusetts, USA: Harvard University Press. DOI: 10.2307/j.ctvjf9z6v.

Riot Games, Inc. [2009]. *League of Legends*. Video game [Windows, macOS]. Los Angeles, California, USA.

Risi, Sebastian and Julian Togelius [2017]. Neuroevolution in Games: State of the Art and Open Challenges. In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.1, pp. 25–41. DOI: 10.1109/TCIAIG.2015.2494596.

Roemer, John E. and Alain Trannoy [2015]. "Chapter 4 - Equality of Opportunity". In: *Handbook of Income Distribution*. Vol. 2. Handbook of Income Distribution. Elsevier, pp. 217–300. DOI: 10.1016/B978-0-444-59428-0.00005-9.

Rogers, Katja, Vincent Le Claire, Julian Frommel, Regan Mandryk, and Lennart E. Nacke [2023]. Using Evolutionary Algorithms to Target Complexity Levels in Game Economies. In: *IEEE Transactions on Games* 15.1, pp. 56–66. DOI: 10.1109/TG.2023.3238163.

Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer [2022]. High-Resolution Image Synthesis with Latent Diffusion Models. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, pp. 10674–10685. DOI: 10.1109/CVPR52688.2022.01042.

Saleiro, Pedro, Benedict Kuester, Loren Hinkson, Jesse London, Abby Stevens, and Ari Anisfeld et al. [2018]. *Aequitas: A Bias and Fairness Audit Toolkit*. DOI: 10.48550/arXiv.1811.05577.

Salen, Katie and Eric Zimmerman [2003]. *Rules of play: Game design fundamentals*. Cambridge, Massachusetts, USA: MIT Press. ISBN: 978-0-262-24045-1.

Sandhu, Arunpreet, Zeyuan Chen, and Joshua McCoy [2019]. Enhancing wave function collapse with design-level constraints. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. San Luis Obispo California USA: ACM, pp. 1–9. DOI: 10.1145/3337722.3337752.

Schreiber, Ian and Brenda Romero [2021]. *Game Balance*. Boca Raton, Florida, USA: CRC Press. DOI: 10.1201/9781315156422.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz [2015]. Trust Region Policy Optimization. In: *Proceedings of Machine Learning Research*. Vol. 37. Lille, France: PMLR, pp. 1889–1897.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov [2017]. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347.

Schuur, Edward A. G., James Bockheim, Josep G. Canadell, Eugenie Euskirchen, Christopher B. Field, and Sergey V. Goryachkin et al. [2008]. Vulnerability of Permafrost Carbon to Climate Change: Implications for the Global Carbon Cycle. In: *BioScience* 58.8, pp. 701–714. DOI: 10.1641/B580807.

Seif El-Nasr, Magy, Anders Drachen, and Alessandro Canossa [2013]. *Game Analytics: Maximizing the Value of Player Data*. London, UK: Springer. DOI: 10.1007/978-1-4471-4769-5.

Seif El-Nasr, Magy and Erica Kleinman [2020]. Data-Driven Game Development: Ethical Considerations. In: *International Conference on the Foundations of Digital Games*. Bugibba Malta: ACM, pp. 1–10. DOI: 10.1145/3402942.3402964.

Sen, Amartya [1986]. "Chapter 22 Social choice theory". In: *Handbook of Mathematical Economics*. Vol. 3. Elsevier, pp. 1073–1181. DOI: 10.1016/S1573-4382(86)03004-7.

Shahid, Asad Ali, Loris Roveda, Dario Piga, and Francesco Braghin [2020]. Learning Continuous Control Actions for Robotic Grasping with Reinforcement Learning. In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. ISSN: 2577-1655, pp. 4066–4072. DOI: 10.1109/SMC42975.2020.9282951.

Shaker, Noor, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neill [2012]. Evolving levels for Super Mario Bros using grammatical evolution. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 304–311. DOI: 10.1109/CIG.2012.6374170.

Shaker, Noor, Julian Togelius, and Mark J. Nelson [2016]. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Cham: Springer International Publishing. DOI: 10.1007/978-3-319-42716-4.

Shu, Tianye, Jialin Liu, and Georgios N. Yannakakis [2021]. Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study. In: *2021 IEEE Conference on Games (CoG)*, pp. 1–9. DOI: 10.1109/CoG52621.2021.9619124.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, and Arthur Guez et al. [2018]. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. In: *Science* 362, pp. 1140–1144. DOI: 10.1126/science.aar6404.

Siper, Matthew, Ahmed Khalifa, and Julian Togelius [2022]. Path of Destruction: Learning an Iterative Level Generator Using a Small Dataset. In: *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 337–343. DOI: 10.1109/SSCI51031.2022.10022073.

Smith, Adam M. and Michael Mateas [2011]. Answer Set Programming for Procedural Content Generation: A Design Space Approach. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3, pp. 187–200. DOI: 10.1109/TCIAIG.2011.2158545.

Sorochan, Kynan and Matthew Guzdial [2022]. Generating Real-Time Strategy Game Units Using Search-Based Procedural Content Generation and Monte Carlo Tree Search. In: *Proc. of the AIIDE Workshop on Experimental AI in Games*.

Stanley, Kenneth O. and Risto Miikkulainen [2002]. Evolving Neural Networks through Augmenting Topologies. In: *Evolutionary Computation* 10.2, pp. 99–127. DOI: 10.1162/106365602320169811.

Stegmaier, Jamey [2016]. *Scythe*. Board game. Stonemaier Games, LLC, St. Louis, Missouri, USA.

Suarez, Joseph, Yilun Du, Phillip Isola, and Igor Mordatch [2019]. *Neural MMO: A Massively Multiagent Game Environment for Training and Evaluating Intelligent Agents*. DOI: 10.48550/arXiv.1903.00784.

Sudhakaran, Shyam, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi [2023]. Prompt-Guided Level Generation. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. New York, NY, USA, pp. 179–182. DOI: 10.1145/3583133.3590656.

Summerville, Adam, Shweta Philip, and Michael Mateas [2015]. MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 11.3, pp. 68–74. DOI: 10.1609/aiide.v11i3.12816.

Summerville, Adam, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, et al. [2018]. Procedural Content Generation via Machine Learning (PCGML). In: *IEEE Transactions on Games* 10.3. Conference Name: IEEE Transactions on Games, pp. 257–270. DOI: 10.1109/TG.2018.2846639.

Sutton, Richard S. and Andrew G. Barto [2018]. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press. ISBN: 978-0-262-03924-6.

Sutton-Smith, Brian [1997]. *The ambiguity of play*. The ambiguity of play. Cambridge, Massachusetts, USA: Harvard University Press. ISBN: 978-0-674-01733-7.

Teikari, Arvi [2019]. *Baba Is You*. Video game [Windows, macOS, Linux, Nintendo Switch]. Hempuli Oy, Helsinki, Finland.

Teuber, Klaus [1995]. *Die Siedler von Catan*. Board game. Kosmos GmbH Co. KG, Stuttgart, Germany.

Thurstone, Louis L. [1927]. A Law of Comparative Judgment. In: *Psychological review* 34, pp. 273–286.

Todd, Graham, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius [2023]. Level Generation Through Large Language Models. In: *Proceedings of the 18th International Conference on the Foundations of Digital Games*. FDG '23. New York, NY, USA, pp. 1–8. DOI: 10.1145/3582437.3587211.

Togelius, Julian [2019]. *Playing Smart: On Games, Intelligence, and Artificial Intelligence*. Cambridge, Massachusetts, USA: MIT Press. ISBN: 978-0-262-35015-0.

Togelius, Julian, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis [2011a]. What is procedural content generation? Mario on the borderline. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. PCGames '11. New York, NY, USA: Association for Computing Machinery, pp. 1–6. DOI: 10.1145/2000919.2000922.

Togelius, Julian, Ahmed Khalifa, Sam Earle, Michael Cerny Green, and Lisa Soros [2024]. "Evolutionary Machine Learning and Games". In: *Handbook of Evolutionary Machine Learning*. Genetic and Evolutionary Computation. Springer Nature, pp. 715–737. DOI: 10.1007/978-981-99-3814-8_25.

Togelius, Julian, Georgios N. Yannakakis, Kenneth Stanley, and Cameron Browne [2011b]. Search-Based Procedural Content Generation: A Taxonomy and Survey. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3, pp. 172–186. DOI: 10.1109/TCIAIG.2011.2148116.

Toy, Michael, Glenn Wichman, Ken Arnold, and Jon Lane [1980]. *Rogue*. Epyx, Inc., San Francisco, California, USA.

Vaccarino, Donald X. [2008]. *Dominion*. Board game. Rio Grande Games, Placitas, New Mexico, USA.

Valls-Vargas, Josep, Jichen Zhu, and Santiago Ontañón [2017]. Graph grammar-based controllable generation of puzzles for a learning game about parallel programming. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pp. 1–10. DOI: 10.1145/3102071.3102079.

Schneider, W. Joel and Kevin S. McGrew [2012]. "The Cattell-Horn-Carroll model of intelligence". In: *Contemporary intellectual assessment: Theories, tests, and issues, 3rd ed*. New York, NY, US: The Guilford Press, pp. 99–144. ISBN: 978-1-60918-995-2.

Vinyals, Oriol, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, and Junyoung Chung et al. [2019]. Grandmaster level in StarCraft II using multi-agent reinforcement learning. In: *Nature* 575, pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

Volz, Vanessa, Günter Rudolph, and Boris Naujoks [2016]. Demonstrating the Feasibility of Automatic Game Balancing. In: *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 269–276. DOI: 10.1145/2908812.2908913.

Volz, Vanessa, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi [2018]. 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. New York, NY, USA: Association for Computing Machinery, pp. 221–228. DOI: 10.1145/3205455.3205517.

von Neumann, John and Oskar Morgenstern [1944]. *Theory of Games and Economic Behavior*. Princeton, New Jersey, USA: Princeton University Press.

Vygotsky, L. S. [1978]. *Mind in Society: Development of Higher Psychological Processes*. Cambridge, Massachusetts, USA: Harvard University Press. DOI: 10.2307/j.ctvjf9vz4.

Wang, Ziqi, Jialin Liu, and Georgios N. Yannakakis [2022]. 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*. Athens, Greece: ACM, pp. 1–8. DOI: 10.1145/3555858.3563282.

Watkins, Christopher J. C. H. and Peter Dayan [1992]. Q-learning. In: *Machine Learning* 8.3, pp. 279–292. DOI: 10.1007/BF00992698.

Whitehead, Jim [2020]. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In: *Proceedings of the 15th International Conference on the Foundations of Digital Games*. FDG '20. Bugibba, Malta: Association for Computing Machinery. DOI: 10.1145/3402942.3409603.

Whitehead, Jim, Thomas Wessel, Blythe Chen, Raven Cruz-James, Luc Harnist, William Klunder, et al. [2025]. Conversational Interactions with Procedural Generators using Large Language Models. In: *Proceedings of the 20th International Conference on the Foundations of Digital Games*. Vienna & Graz Austria: ACM, pp. 1–7. DOI: 10.1145/3723498.3723788.

Wilde, Oscar [1899]. *An Ideal Husband*. Quote from Act II: "Life is never fair. And perhaps it is a good thing for most of us that it is not." London, UK: Leonard Smithers and Co.

Wube Software Ltd. [2020]. *Factorio*. Video game [Windows]. Prague, Czech Republic.

Yang, Hongyang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid [2020]. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. In: *SSRN Electronic Journal*. DOI: 10.2139/ssrn.3690996.

Yannakakis, Georgios N. and Julian Togelius [2011]. Experience-Driven Procedural Content Generation. In: *IEEE Transactions on Affective Computing* 2.3, pp. 147–161. DOI: 10.1109/T-AFFC.2011.6.

Yannakakis, Georgios N. and Julian Togelius [2025a]. *Artificial Intelligence and Games*. Cham: Springer Nature Switzerland. DOI: 10.1007/978-3-031-83347-2.

Yannakakis, Georgios N. and Julian Togelius [2025b]. "Procedural Content Generation". In: *Artificial Intelligence and Games*. Cham: Springer Nature Switzerland, pp. 235–251. DOI: 10.1007/978-3-031-83347-2_7.

You, Jiaxuan, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec [2018]. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In: *Advances in Neural Information Processing Systems*. Vol. 31.

Zakaria, Yahia, Magda Fayek, and Mayada Hadhoud [2022]. Procedural Level Generation for Sokoban via Deep Learning: An Experimental Study. In: *IEEE Transactions on Games* 15.1, pp. 108–120. DOI: 10.1109/TG.2022.3175795.

Zelada, Elías and Francisco J. Gutierrez [2023]. Dynamic Difficulty Adjustment of Video Games Using Biofeedback. In: *Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2022)*. Cham: Springer International Publishing, pp. 925–936. DOI: 10.1007/978-3-031-21333-5_91.

Zhou, Zhenpeng, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley [2019]. Optimization of Molecules via Deep Reinforcement Learning. In: *Scientific Reports* 9.1, p. 10752. DOI: 10.1038/s41598-019-47148-x.