Orchestrating the Generation of Game Facets via a Model of Gameplay

Konstantinos Daniel Karavolos
Institute of Digital Games
University of Malta

A thesis submitted for the degree of
Doctor of Philosophy

January, 2020

Abstract

Computer games are media that weave together many different facets. When the design of games is supported by the automatic creation of game content, a multidisciplinary approach would be expected. Yet, many approaches to procedural content generation tend to focus on a single facet at a time, assuming that a human game designer will guarantee a suitable context. The systems that do create multiple types of content usually rely on expensive game simulations to evaluate their quality and complementarity. However, the complexity of modern games increases the runtime of these simulations to such a degree that at some point this approach becomes infeasible.

This thesis proposes a framework for the procedural generation of the level and ruleset components of games via a model of gameplay that can act as a surrogate for expensive game simulations. By combining the level and ruleset components as input and gameplay outcomes as output, deep learning is used to construct a mapping between three different facets of a game. This thesis argues that the learned mapping enables the model to identify the synergies between these facets, which can be used to orchestrate the generation of both level and rules towards desired gameplay outcomes. The experiments support this by demonstrating the ability of a search-based generative approach that uses a surrogate model for quality evaluation to adapt players' character classes, levels, or both towards designer-specified targets in the domain of shooter games. The findings demonstrate that the proposed method of game facet orchestration can produce improved designs of both facets without the use of simulations and makes less changes to an initial design than traditional single-facet methods.

Acknowledgements

First of all, this work would not have been possible without the guidance of my supervisors Antonios Liapis and Georgios Yannakakis. I am very grateful that they have given me this opportunity. Their feedback and insights have made me a better researcher and greatly shaped the work that was done for this thesis. They have taught me that scientific texts should be written as stories with the research question as the main character. Thanks for providing me with this perspective. The protagonist of this thesis has taken me on a big adventure and I am glad to be able to share it.

A deep thank you goes to my former colleagues at the Hogeschool van Amsterdam for providing me with knowledge on game development and education and for cultivating the scientist that was inside me. A special thanks goes to Joris Dormans, Rafael Bidarra and Stefan Leijnen for igniting my enthousiasm for research into procedural content generation.

A big thank you goes to all my colleagues at the Institute of Digital Games, for their support, inspiration and making life in Malta more fun. I want to thank Phil Lopes for getting me started on my journey, and Jasper Schellekens for always lightening the mood and for introducing me to historical fencing. I thank David Melhart for sharing his wisdom on and off topic, and for teaching me that bowties are cool.

I want to thank Daniele Gravina for the time we spent together, for our discussions and brainstorms, our coffees and our beers. We shared an office, a social life and a house for the better part of these years in Malta and there was never so much as an echo of frustration. For this, I am deeply grateful.

Furthermore, I thank all of my friends in Groningen and Amsterdam for making the distance between Malta and The Netherlands seem smaller. A special thanks goes to Eric Jansen, who frequently shared his home when I was around and encouraged me to step out of my comfort zone. I am grateful to Nick Degens for frank reflections, probing questions, fruitful discussions and encouraging words. Without him I probably would not have started nor finished this PhD.

I would like to thank my mother, Marianne Rook, and René Vermeulen. For hosting me during the last phase of writing, and for everything else they have done to support me. René's listening ear and wise words have calmed me many times. I can hardly capture the incredible support of Marianne in words. Thanks for always believing in me and encouraging me to reach for the stars.

Finally, I would like to thank Laura Schomaker for her patience, her love and her faith in me. She had the courage to join me in this journey, even though it meant having a long distance relationship for a longer period than we had known each other before then. I am proud that we made it work and I look forward to the next phase of our life together.

Statement of Originality

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. (Konstantinos Daniel Karavolos)

To Marianne,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last thirty years, the video games industry has grown from a small niche market to a multi-billion dollar industry that produced the biggest entertainment release in history (Cherny, 2018). The industry is growing at an increasing pace; its global revenue has grown from 54 billion dollars in 2011 to almost 140 billion in 2018, a lot more than the 82 billion projected in 2016 (NewZoo, 2019; WePC, 2019). Needless to say, the financial stakes are high; with development budgets of millions of dollars and teams that can consist of more than a hundred people. Procedural content generation is a way for game development studios to stay competitive. Such software can automate laborious tasks for game designers during development or personalize player experiences in an autonomous fashion.

Due to the financial stakes and the plethora of games released each year, it has become a requirement for a successful game to test whether the intended player experience is achieved. Over the years, the methodology of user testing has grown from questionnaires and observing playtests (Bernhaupt, 2010) to include data-oriented approaches, such as game analytics (El-Nasr et al., 2016) and data-driven models of player experience (Melhart et al., 2019). It is, by now, common to even collect data from the game after shipping, to inform the design of a sequel, patches, or downloadable content. During development it is not always sustainable to go through the time consuming process of collecting data from players, especially in relation to procedural content generation. Playtesting tasks with a low quality threshold can be automated via artificial agents that simulate player behavior (Holmgård et al., 2015; Liapis et al., 2015a).

The rise of deep learning (LeCun et al., 2015) has increased the interest in developing game playing agents. Training agents through machine learning, instead of manually constructing them in code, should ultimately result in more natural behaviors. For example, deep reinforcement learning has been used to learn shooter behavior for *Battlefield* (EA, 2016) (EA, 2018), driving in *Watch Dogs* (Ubisoft, 2014) (Jacquier et al., 2018) and sword fighting in *For Honor* (Ubisoft, 2017) (Jacquier et al., 2018). These agents can not only be shipped with the game as non-playable characters or artificial opponents, but can also be employed during development to simulate a human playtest and reduce the time between game design and feedback (Edge, 2018). Indeed, the developers of *Candy Crush Saga* (King, 2012) specifically model player decisions via deep learning to give them more reliable feedback on the feasibility of levels (Gudmundsson et al., 2018). They report a drastically reduced feedback time; from a week to a couple of minutes. This affects the affordances of game designers, giving them more time to iterate on their design and focus on innovation and creativity.

A recent milestone in the area of artificial intelligence, i.e., the program *AlphaGo* (Silver et al., 2016) beating the world's best *Go* player Lee Sedol, features deep learning in a different role. In *AlphaGo*, actions are selected by a search algorithm, while deep learning is used to evaluate game states in terms of their utility towards winning the game. This is more in line with the traditional task of image classification for which deep learning is so well known (Russakovsky et al., 2015). Inspired by the successes of deep learning for image recognition, and game state evaluation in particular, this dissertation explores the use of deep learning for predicting human interpretable metrics of game content as a time efficient alternative to agent-based playtesting. Instead of playing a game, the learned model is used to create game content by guiding a search algorithm towards content with desirable gameplay properties.

This dissertation takes places in the context of procedural content generation, and specifically mixed-initiative design tools (Liapis et al., 2016). Such tools support game designers by pro-actively contributing to design, e.g., by filling gaps in the design (Smith et al., 2011) or by procedurally generating suggestions for alterations or alternatives (Liapis et al., 2013d; Alvarez et al., 2018). Most often, this content is generated based on heuristics for good design, which should enable good gameplay without actual playtesting, e.g., heuristic for good level design (Preuss et al., 2014). Algorithms that do playtest during generation, use artificial agents to simulate games (Gravina and Loiacono, 2015; Zook and Riedl, 2018) and have the additional benefit of being able to consider multiple types of content at the same time (Browne and Maire, 2010). Even though these agents are typically much simpler than the deep learning agents described above, simulations still do not scale to procedural generation for modern games due to the number of evaluations involved. At least, not in the context of the real-time feedback that is required by a mixed-initiative tool. This thesis hypothesizes that deep learning models can provide an alternative to expensive game simulations, even when the suitability of multiple game facets are to be considered in conjunction. The success of this approach can be a step towards an efficient form of game facet orchestration (Liapis et al., 2019b).

This thesis explores whether and how deep learning can be used to predict gameplay outcomes or other metrics based on a visual analysis of the initial game state. This could provide gameplay feedback without having to simulate a playtest. The model is connected to a procedural generator which provides suggestions to a game designer and judges whether the new content achieves the desired gameplay. The system searches based on the feedback of the model until it finds content that is deemed to satisfy the targets set by a designer.

1.1 Motivation

A popular analogy for the workflow with co-creative tools is that of two creators having a conversation with each other about what they are trying to create (Novick and Sutton, 1997). Unlike their computational counterparts, when two human game designers talk about a new game concept, they do not have to write down all the rules or play a prototype of that game in order to understand each other. They can make an estimate of the resulting gameplay based on only a few pieces of information by extrapolating from past experiences, or by using examples of games that they both know well. In a level for a shooter game, for example, it is easy for a designer or a veteran player to spot choke points or an unfair distribution of powerups (e.g., based on distance to the respective team bases) by looking at the top-down map.

Based on their own expert knowledge, several researchers have identified key game design patterns (Björk and Holopainen, 2004; Hullet and Whitehead, 2010). Moreover, many academic papers have formulated properties of game levels (Liapis et al., 2013c) or rulesets (Browne and Maire, 2010) in a quantitative fashion, often to use them as an objective to optimize towards (Togelius et al., 2011). However, it is arguably infeasible to formalize and accurately compute all possible metrics of level quality. Not only are some metrics difficult to capture, but the ways they impact game quality can be influenced by the nuances of the specific game's ruleset. Moreover, each game level can be evaluated in any number of dimensions, e.g., in terms of visuals such as symmetry and in terms of game-specific patterns such as exploration or balance (Liapis et al., 2013c). It is not straightforward how to combine these metrics in a way that the computer can understand or produce content that optimizes them.

Beyond patterns in a single facet of games (such as patterns for level design), a game constitutes a multi-faceted product and experience (Liapis et al., 2014b). Game designers must integrate a variety of creative domains, such as visuals, music and narrative, to bring about an intended experience to a player. Liapis et al. (2019b) have identified six different facets of games which require distinct creative input: rules, levels, visuals, audio, narrative and gameplay. When designing a new element for a game, a designer must consider its effect on players' experience, as well as how it matches with the other content. For example, switching from a game rule that allows players to regenerate health continuously to a rule that a player colliding with a specific powerup heals a preset number of hit points will have vast repercussions across facets. For starters, the number of hit points healed (which is a detail of the above rule) may greatly affect the priorities of players in seeking this powerup out, and may unbalance other parts of the ruleset such as the relative power of different weapons. Moreover, the powerup will need a mesh and textures to signify its function, while the game levels will need to include such powerups at critical locations which must also be carefully considered. Finding how one element of the game design (e.g., a visual asset, a level design choice, or a game rule) will influence the design of other game elements as well as the player's experience is a core challenge for a human designer — let alone a computer.

Procedural content generation (PCG) has largely side-stepped the issue by focusing on a single type of content at a time, and running the generative algorithm in a well-defined space that does not require knowledge about the rest of the game. For instance, a level generator for *StarCraft* (Blizzard, 1998) already has established what the possible resources, the units' statistics, or the textures of the game are. The generator thus must only consider how to distribute spatially game elements that are otherwise pre-authored (Togelius et al., 2013b). When the need for multiple generators does arise, their dependencies are carefully crafted. For example, *Dwarf Fortress* (Bay 12 Games, 2006) has many generators that contribute their content independently, while in *No Man's Sky* (Hello Games, 2016) the important information of one generator is passed to the next generator as a sequential pipeline.

Driven by a desire to instill a more human-like creative agency to computational game designers, we identify an important stepping stone in the automated assessment of game content less as parts and more as a whole (Bidarra et al., 2015). Being able to identify interrelations between, for instance, game rules and level structures without needing to produce and test the outcome can allow such a computational designer to reason in a more similar fashion to a human designer. In turn, this opens the possibility of a more intuitive dialog between a human and a computational designer where the artificial intelligence can suggest or undertake changes in one facet based on human input in the same or a different facet.

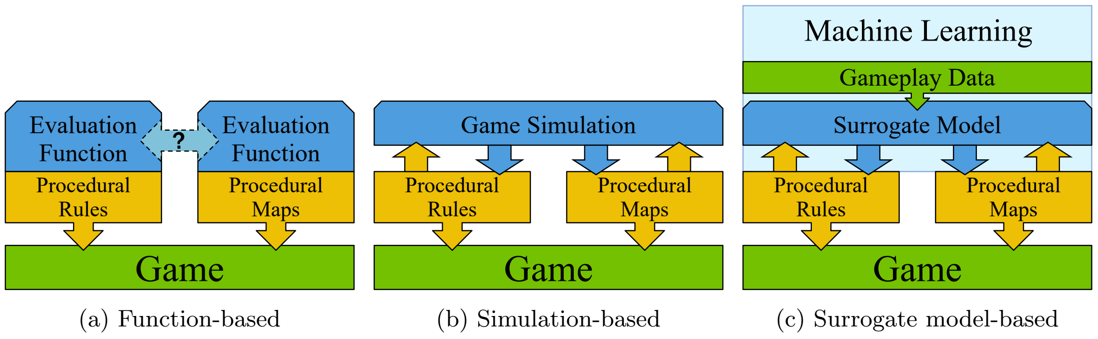


Figure 1.1: Approaches to evaluation functions in multi-facet content generation based on search-based algorithms. Facets can be evaluated by (a) exchanging information between fitness functions, e.g.,via a blackboard, (b) a simulation that contains all facets, (c) a model that predicts metrics of a combination of facets.

Inspired by the music domain, the process of procedurally harmonizing the output of multiple generators has been dubbed game facet *orchestration* (Liapis et al., 2019b), which enables analogies between multi-faceted procedural content generation and the task performed by a composer when directing the output of multiple instruments or the activity of musicians when improvising a song together. Such analogies can aid the design of systems that aim to solve this problem (see Section 2.4.3).

Design patterns aim to describe the relations between different facets of a game and their effect on player experience. Yet, it is hard to formalize the effects that other facets will have on a specific game facet that is being generated; let alone to do so in a way that guides the generator towards good content. When attempting this for multiple facets, the complexity grows exponentially. It is unclear how to design a system that orchestrates multiple facets based on heuristics which is simultaneously precise and robust (see Fig. 1.1a).

A solution can be found in the gameplay facet, which most closely approximates player experience and generally depends on a combination of other facets. We follow the definition of gameplay by (Liapis et al., 2019b), which is based on (Sicart, 2012, p.104): "gameplay, or the experience of a game, is the phenomenological process of an epistemic agent interacting with a formal system". In the context of this work, we require a quantification of this process to make the algorithms work. As such, we aim to measure gameplay via the playtraces of agents, i.e., by recording their actions, locations visited, kills scored against the opponent and the outcome in terms of one player's advantage and the duration of the match. While most experiments in this thesis focus on the two metrics of final outcomes (kill ratio and duration), Section 8.2.1 explores the possibility of predicting other metrics of gameplay, such as the average time spent in combat and the entropy of the heatmap.

The dynamic nature of this facet does not allow it to be computed from content alone without actually running the game. Although artificial agents generally do not capture the phenomenological process of human players, their interactions with the other facets in the game do display similar dynamics of play. As such, observations collected from game simulations are better at capturing the relations between multiple facets than heuristics and can replace them as an evaluation function (see Fig. 1.1b), which has led to research into evaluations of content via game simulations (Browne and Maire, 2010; Holmgård et al., 2015; Cook et al., 2016b). However, in this thesis we argue that simulation-based approaches for modern games are too computationally expensive to produce the desired content within reasonable time.

Towards realizing a computationally creative game designer, this thesis proposes an approach to *orchestrating* the generation of levels and rulesets by evaluating the joint artifacts with a model of gameplay. The harmonization of multi-faceted content emerges in a bottom-up fashion, resulting from the joint evaluation of the content produced by two separate generators (see Fig. 1.1c). This framework is evaluated within the context of competitive first-person shooters. The common practice of summarizing gameplay in the form of match statistics (e.g., headshots, damage dealt, kill/death ratio, match duration) makes this popular game genre a particularly suitable test case. A custom shooter game called *SuGAr* was developed specifically for this purpose. Nevertheless, the approach does not utilize any genre-specific information and is designed to easily extend to other game genres. Similar to a human designer's perception, the system uses the visual description of a game level as an image and learns to see the important level patterns that can affect the balance or quality of a playthrough. This model has been trained via deep learning on gameplay logs to automatically extract features and find an approximate mapping between game levels, rules and gameplay outcomes. The model does not offer the precision of a simulation, but its real-time feedback makes it a useful design tool.

The computational model is able to identify interrelations between the three facets without needing to produce and test the outcome, following a similar process to a human designer who drafts and corrects content without playtesting every iteration. The model can be used as a surrogate for actual game playthroughs, informing generators that adapt content of all contributing facets (in this case levels and rules). The ability of the framework to both predict gameplay outcomes from the game design specifications and to automatically create new designs renders it a computational creator, which can easily work alongside a human creator in a mixed-initiative design task (Yannakakis et al., 2014).

1.2 Concept

This thesis proposes to use a machine learned surrogate model to guide a search-based procedural content generator towards content with desirable gameplay. The advantage of such an approach is that the model can evaluate multiple facets of game content without having to recourse to expensive game simulations. The utilization of this approach consists of four phases, as visualized in Fig. 1.2.

First, the model requires an annotated dataset with examples of content and resulting gameplay (in the form of metrics). For this thesis, we create such a dataset by generating content using constructive PCG and game simulations with artificial agents to estimate the corresponding gameplay. The second phase consists of finding the appropriate input representation and a suitable model, and training that model via (supervised) machine learning to map game content to gameplay metrics using the annotations. Typically, this phase will occur offline, since many iterations are required to find an effective model and its parameters.

After a sufficiently precise model has been obtained, phase three consists of generating content based on the game designer's targets and constraints. A search-based algorithm generates content based on an initial design which has either been generated by another procedural system (e.g., the same generator that created the dataset) or provided by a human designer. Since the model has learned to map multiple facets to gameplay, the generated artifacts can be evaluated in relation to other content. The designer can choose

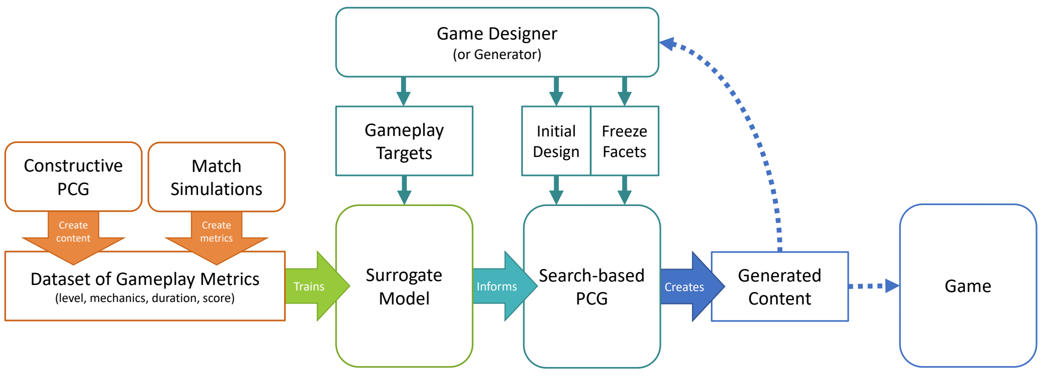


Figure 1.2: Proposed pipeline for surrogate-assisted facet orchestration. The setup process consists of three phases: creating a data set (orange), training a surrogate model (green) and generating content (cyan). The system's output (blue) can either be presented to the game designer, as a design tool, or put directly into the game, as an autonomous creator.

which facets can be adjusted and which are non-adjustable. Alternatively, multiple facets can be generated and evaluated simultaneously. Note that the search algorithm does not include a strategy for *model management*. Given the goal of providing real-time feedback, the evaluation of the content depends fully on the approximations of the model and does not include game simulations for true evaluations or online model updates.

The fourth phase depends on the context in which this system is used. It could be part of an autonomous computationally creative system; in which case the output is either inserted in the game or passed to a generative system upwards in the hierarchy. However, in this thesis we will assume the approach is part of a mixed-initiative game design tool. In that case the generated content will be presented to a game designer for evaluation.

The proposed framework is tested on *SuGAr*, a two-player, competitive first-person shooter. This genre was chosen for its relatively straightforward core gameplay (i.e., moving around and tagging opponents) and its common practice of displaying numerical summaries of gameplay at the end of a game. The gameplay facet is quantified as two outcomes: the genre-specific metric of kill balance between the two opponents and the duration of the game, which is a general game design target. The level facet is the arena in which the game takes place and includes the placement of powerups that affect the players' survivability. The rules facet is represented by the character classes of the two players. *Character classes* are a common way to group game mechanics in multiplayer shooter games such as *Team Fortress 2* (Valve, 2007) and *Battlefield: Bad Company* (Electronic Arts, 2008). Character classes offer players different gameplay options (e.g., scouting or area control) and may have a different survivability and movement speed as well as signature weapons that balance their affordances.

A computational model that maps levels and rules to gameplay outcomes is trained via deep learning on a rich corpus of simulated playthroughs. Moreover, this model is used to adapt either facet individually or orchestrate the generation of both. Experiments focus on the designer-guided aspect of this framework, allowing game designers to specify a map, the target character classes that this map is intended for, and the desired game balance and duration. The designer also chooses the content they would like to improve; the map, the character classes, or both. The system then adapts the content via variation operators to minimize the distance with the designer's desired gameplay outcomes, using the surrogate model to predict the evolving maps' gameplay properties.

1.3 Research Questions

This thesis aims to explore the possibilities of evaluating multiple game facets in relation to each other in order to improve procedural content generation. The central question of this thesis is:

"How can a computational model be used to orchestrate the procedural generation of multiple game facets?"

With the success of deep learning in mind, the scope of possible models was narrowed to the area of neural networks. Even with this constraint, the question is way too broad to be answered within the limits of a doctoral thesis. We therefore formulate the following four core research questions that will drive our experimentation in this thesis:

RQ1 How can deep learning be used to create a surrogate model of gameplay?

RQ2 Does multi-modal fusion of game facets improve the gameplay prediction of a neural network?

RQ3 Can a surrogate model replace simulation-based testing as an alternative and more efficient approach to content evaluation in search-based procedural content generation?

RQ4 How can a surrogate-assisted search-based algorithm be used to orchestrate multiple game facets with multiple gameplay targets?

1.4 Contributions

The goal of this thesis is to describe a computationally creative system that can operate along a human designer as part of a mixed-initiative game design tool and, as such, contribute to the state of the art in procedural content generation and computational creativity. By addressing the above mentioned research questions, this thesis contributes to various areas of existing research, as summarized below.

- Deep learning requires a large corpus of levels, rules and gameplay metrics. The dataset and an algorithm for generating more data are created specifically for this thesis.
- The introduction of a vision-based content evaluation approach that automatically extracts relevant features and fuses multi-modal input for predicting gameplay.
- The introduction of a surrogate-based game facet orchestration algorithm that generates three facets: levels, rules and gameplay.
- An analysis of the use of a deep neural network as a surrogate model in an evolutionary algorithm without model management.
- A comparison of surrogate-based single-objective and multi-objective evolutionary algorithms for PCG in terms of achieving a desired design goal.

1.5 Publications

The research efforts for addressing the key questions of the thesis have resulted in a number of peer-reviewed publications as listed below:

1. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. A Multi-Faceted Surrogate Model for Search-based Procedural Content Generation. In *IEEE Transactions on Games*, 2019.

2. Antonios Liapis, Daniel Karavolos, Konstantinos Makantasis, Konstantinos Sfikas and Georgios N. Yannakakis. Fusing Level and Ruleset Features for Multimodal Learning of Gameplay Outcomes. In *Proceedings of the IEEE Conference on Games*, 2019.

3. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. Pairing Character Classes in a Deathmatch Shooter Game via a Deep-Learning Surrogate Model. In *Proceedings of the FDG Workshop on Procedural Content Generation*, 2018.

4. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. Using a Surrogate Model of Gameplay for Automated Level Design. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2018

5. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. Learning the Patterns of Balance in a Multi-Player Shooter Game. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2017.

Additionally, the following papers were published during the research for this thesis. They are tangentially related to this work, but are not included in the thesis as they diverge too much from the main research question.

1. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. Evolving Missions to Create Game Spaces. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

2. Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis. Mission Evolution for Dungeon Levels. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

1.6 Thesis Outline

This thesis is organized as follows:

- **Chapter** 2 surveys relevant literature on game design, (search-based) procedural content generation and computational creativity. Special attention is paid to facet orchestration, PCG via machine learning, surrogate-assisted approaches and research on first-person shooter games.

- **Chapter** 3 provides details on the algorithms used in this thesis. It explains the basics of machine learning, deep learning, (multi-objective) evolutionary algorithms and covers concepts related to surrogate-assisted evolution.

- **Chapter** 4 outlines the methodology of this research. It reflects on an initial study into the feasibility of deep learning for predicting gameplay outcomes and introduces the system's pipeline, the game test bed, the components of the content generators and the representations of the game facets.
- **Chapter** 5 analyses the dataset that is created to train the surrogate model and provides descriptions of the constructive level generation process and the agent behaviors. Patterns in the dataset are explored via an expressivity analysis of the generators and correlations between the game facets and the gameplay metrics.
- **Chapter** 6 compares the proposed surrogate model, a convolutional neural network, with several baselines and model variations. The model architecture, training procedure and choice of dataset size are examined, as well as the performance of the model in a classification task. Finally, two techniques for probing neural networks are applied to get insight into the model and explore its explainability.
- **Chapter** 7 studies the viability of the proposed system for generating content with the desired gameplay. The experiments include adjusting one facet to another, orchestrating both facets, and simultaneously targeting multiple gameplay goals. Aside from achieving gameplay targets, the resulting content is analyzed in terms of variation, required changes, and improvement over initial designs.
- **Chapter** 8 summarizes the findings of this thesis and provides directions for future work. It discusses the limitations of the proposed approach and the performed experiments, and proposes extensions to address these limitations. In particular, it demonstrates extensibility of the surrogate model towards other gameplay targets and more domain knowledge, and provides suggestions for alternate processes and domains for surrogate-assisted facet orchestration.

Many contributions of this thesis are based on previous publications listed in Section 1.5. The proof of concept study in Chapter 4 is based on publication 5. Publications 1-4 each feature models trained on the data set that is detailed in Chapter 5 and report performance results of various surrogate models and baselines, which are aggregated in Chapter 6. The activation mapping for explaining model decisions in Chapter 6 is reported in publication 2. The single facet experiments in Chapter 7 on character classes and map generation are respectively reported in publications 3 and 4, while the facet orchestration experiments are published in publication 1. Chapter 8 discusses extensions regarding the input to the surrogate model, which are reported in publication 2.

1.7 Summary

This chapter has outlined the frame of reference for this thesis. It started broadly from the current needs in game development and narrowed down to the specific area of multi-faceted procedural content generation that is the direct context of this research. This thesis is motivated by the observation that game developers require fast feedback on the gameplay of their game and that progress in simulations for procedural content generation will not scale to the required response time in tools for modern games.

This thesis proposes to use deep learning to create a surrogate model of gameplay, which is to replace the costly simulations as the content evaluation method in search-based procedural content generation. This chapter has sketched a methodology for a surrogate-assisted procedural content generation process that can orchestrate multiple game facets via the evaluation of their gameplay. The research followed is be based on the case study of *SuGAr*, a competitive first-person shooter that was developed specifically for this research.

This chapter has described the research questions that drive this thesis as well as the contributions of this thesis towards the domain and tools of procedural content generation. The chapter concludes with a list of publications that result from work on this thesis. Chapter 2 outlines the context of this thesis by providing a literature review of the related domains.

Chapter 2

Related Work

This thesis proposes a method for generating game content via a model of gameplay. The approach can either be incorporated into a tool for game designers or be part of an autonomous computational content creator. This chapter surveys three major areas of related literature as well as game-related research that has been done with similar techniques. Section 2.1 highlights work on game design and game balancing which inspired this work and provides a perspective on evaluating the output of this research.

Section 2.2 provides some terminology for describing procedural content generation (PCG) systems and discusses other work in this field, in particular those that search a design space. The generative approach in this thesis is designed to work with any 2D perspective of a game level and any other facet(s) that can be captured by a vector of numbers. To demonstrate how it works, we chose to apply it to the domain of first-person shooter games and in particular, a competitive free-for-all multiplayer game mode. As such, we focus on describing generators of maps and weapons in Section 2.2.1. The algorithms in this thesis have been applied to games before in various forms. Section 2.2.2 describes other applications of machine learning for PCG, while Section 2.2.3 covers work that also suggest surrogate models for searching the design space. The type of model that we use for predicting gameplay outcomes has been used in the context of artificial game-playing agents, some of these successes are highlighted in Section 2.3.

In Section 2.4 we provide a perspective of computational creativity on this work and argue how this thesis can contribute to that field. In this thesis we argue that the use of a model for intermediate evaluations can be useful for coordinating information streams when combining multiple game facets or facets from multiple games. Section 2.4.2 covers tools that support the creativity of game designers, while Section 2.4.3 surveys other work that generates multiple game facets and outlines how they coordinate the generators that are involved.

2.1 Game Balancing

In this thesis we aim to find a model that can unify map design with rule design for a multiplayer first-person shooter (FPS) via observation of gameplay. Shooter games have a long history of customizable content, with *Doom* (id Software, 1993) being the first game to support modifications made by players (Shahrani, 2013). Since then, these topics have been widely studied and discussed, yet neither map design nor rule design for shooters (or any game genre) is a solved problem and experts suggest to create something playable as soon as possible to start a process of iterative improvement (Holloway, 2013; Fullerton, 2019). Indeed, commercial games are typically balanced via countless playtesting sessions and some studios even aim to playtest on a daily basis (Parker, 2012).

In an effort to formalize common game design knowledge, Björk and Holopainen (2004) have identified a set of frequently occurring *design patterns* in games. These design patterns describe solutions to common problems in game design along with the consequences of applying them and can be utilized in different stages of development. They can either be used to guide creative decisions during the design process or to analyze the interaction between existing pieces of content after they have been designed or generated. Hullet and Whitehead (2010) detail design patterns specifically for FPS games, such as choke points, arenas and sniper locations. As the latter name suggests, there are relations between areas in a map and the weapons or character roles that are part of the game. Indeed, Giusti et al. (2012) report design patterns for weapons, such as the sniping weapon, and relate these to patterns in level design. Similarly, Rivera et al. (2012) identify common roles of enemy non-playable characters (NPCs). These are related to weapon usage and their higher level categories would be suitable for player classes as well. NPC patterns are less closely related to maps, though their analysis of pattern usage does use map patterns to describe how NPCs affect the gameplay experience.

While the above mentioned patterns focus on single player games, Hullett (2012) concludes that they largely overlap with multiplayer shooters; though with slightly different effects on the gameplay as both sides of the conflict now contain players. Multiplayer games require design for counter-play and a balance between equal opportunities for success and rewards for successful play. An important aspect of this is the concept of *conflict location*, i.e., an area in a level that is designed to host encounters between opponents. These can either be carefully crafted from a combination of patterns (especially choke points and strongholds) and objectives, pickups and spawn locations, or even emerge from the shortest paths between spawn points and objectives of opposing teams (Güttler and Johansson, 2003).

Most people will agree that games should be balanced, especially multiplayer games. Yet, the term *balance* can have many meanings and even more ways to achieve it (Sylvester, 2013; Fullerton, 2019; Schell, 2008). In this thesis, we would ideally aim for balance as defined by (Sylvester, 2013), which is based on *fairness* and *strategic depth*. A game is fair when no player as an inherent advantage at the start of play and a game has strategic depth when a player, in pursuit of a goal, can choose from multiple strategies that meaningfully affect gameplay. The latter of which resembles the notion of counter-play in a multiplayer setting by (Hullett, 2012). Strategic depth is hard to quantify and typically requires extensive play testing. Zook and Riedl (2018) attempt to simulate this by comparing agents with different quantities of computational resources. This approach was not feasible with the available agents in this thesis. Therefore, we only evaluate fairness, i.e., whether all players have an equal chance of victory, which is a common definition of balance in the area of automated game balancing (Volz et al., 2016; Beyer et al., 2016; Preuss et al., 2018; Hom and Marks, 2007).

Game levels have often been parsed in terms of their geometric or path properties to estimate a modicum of balance between two or more competing players (Beyer et al., 2016; Liapis et al., 2013c; Liapis, 2018). These heuristics observe the level structure alone, and are expected to be accurate only if the two players have the same in-game characteristics (e.g., available units, tech tree, or weapon power) and the same play style or skill. Yet, a more objective way of assessing game balance is through agent-based simulations or human playtesting. While play traces from actual players are preferred, these are generally not available in the required quantities. In those cases, artificial agents have been used in game simulations to assess each and every candidate design that is encountered in the design space; this approach is computationally expensive and forms the bottleneck in simulation-based evaluations for search-based PCG (Togelius et al., 2011). Such agents are often simplistic (including those used in this thesis), but agents may use more general game-playing algorithms such as Monte-Carlo Tree Search (Zook and Riedl, 2018) or may have different goals depending on the play persona they are trying to emulate (Liapis et al., 2015a). Preuss et al. (2018) describe how an automated approach to game balancing based on simulated agents can be incorporated in the iterative process of game design.

One of the core assumptions in this thesis is that a dataset of generated maps and weapons for a shooter game will contain frequently occurring patterns that can steer gameplay, which can be extracted via machine learning and be used to inform the generation of new content. We take the approach of balancing via procedural generation, i.e., by adjusting the game content but not the premise of the game. It is assumed that the players are of equal skill and that the balance problem originates from the content. If they are not equally skilled, there are other options available to balance multiplayer shooters, such as match making (i.e., only pairing players of equal skill), giving weaker players more map information, asymmetric roles (e.g., healers that rely less on aim) and different levels of aim assistance (Vicencio-Moreira et al., 2015).

2.2 Procedural Content Generation

Procedural content generation (PCG) has been used in the game industry for almost 40 years, and it has been an active field of academic study for over a decade. Early applications in games, such as *Rogue* (Toy and Wichman 1980) and *Elite* (Acornsoft 1984), were the result of a need to circumvent memory constraints. Algorithmically creating game environments during runtime, from low-dimensional input such as random numbers, facilitated the desired complexity without needing to store everything on disk. Nowadays, computational resources are abundant and reasons for generating content have shifted towards increasing the replay value of a game, supporting game designers with tooling and (academically) exploring the creative potential of computing systems.

With its long history, it is not surprising that a wide variety of algorithmic techniques have been devised. Several taxonomies have been proposed, distinguishing the algorithms involved, the role of the generator and the type of content that is being generated (Yannakakis and Togelius, 2018, Ch. 4). Following the taxonomy of (Togelius et al., 2011; Shaker et al., 2016b), we can distinguish three major categories: constructive, generate-and-test and search-based.

Constructive algorithms create content at once without validating it before presenting it to a user. This typically involves carefully crafted rules, procedures and/or constraints which guarantee that the algorithm never creates broken content. Examples include fractals/noise for terrain generation, generative grammars and constraint-solving algorithms such as answer set programming and the recently introduced wave function collapse.

Generate-and-test algorithms contain a test mechanism that follows the constructive procedures. This evaluation checks the validity of the content according to some criteria, e.g., whether there is a path from start to end, whether are all doors unlockable, etc. If the test fails, the content is discarded and the process is restarted until the output is valid.

There is a fundamental tension between increasing the expressivity of these algorithms and potentially failing the test, as more diversity can lead to content of varying quality. Since the algorithm starts from scratch, it is possible to have consecutive failures and even to get stuck in an infinite generate-and-test loop.

Search-based algorithms represent a special kind of generate-and-test approach that uses a more fine-grained evaluation function to guide generation. Instead of accepting or rejecting the content, this function provides a numerical evaluation. New candidate output is generated with the aim of increasing the value. Successive comparison of the values of different candidates results in a gradient towards good content. As such, these algorithms search the design space by iteratively improving upon potential output. Many heuristic or stochastic search algorithms can meet these conditions, yet the most common implementation by far is an evolutionary algorithm. Details of artificial evolution are provided in Section 3.2, as this is one of the algorithms used in this thesis. The next section gives an overview of existing work on game content generation with search-based algorithms.

2.2.1 Search-based Procedural Content Generation

For the purpose of orchestrating the generation of multiple game facets search-based algorithms are particularly suitable, as they allow the generative process to be guided via evaluations of the candidate content. These evaluations can be based on the explicit choices of the designer (Cardamone et al., 2011a) or the player (Hastings et al., 2009; Ølsted et al., 2015), a computational model of preference (Liapis et al., 2012, 2013b), game simulations (Togelius and Schmidhuber, 2008; Browne and Maire, 2010) or static fitness functions (Liapis et al., 2013c; Sorenson et al., 2011). When there is no direct interaction with a player or a designer, the main trade-off is between using slow simulations that produce a complete picture of the gameplay or fast, heuristic approximations. This thesis aims to combine the speed of heuristics and the complexity of simulations by using a computational model as evaluation function. While this is a relatively new phenomenon, it has been used for parameter tuning in *TopTrumps* (Volz et al., 2016), and *Ms. Pac-Man*, *StarCraft* and *TORCS* (Morosan and Poli, 2018; Morosan, 2019). This will be described in more detail in Section 2.2.3) .

While content evaluations are usually specific to the application, there have been attempts to create general evaluation functions. In particular, Liapis et al. (2013c) designed map evaluations based on distances between players' initial positions and resource locations which can be used for multiple genres, such as real-time strategy games, dungeons and shooters. These functions will be used in this thesis to explore the variation of the generated data set in Section5.2. Preuss et al. (2014) extend on this approach by adding the objective of diversity in the candidate content. This fits into a larger trend of using quality-diversity algorithms for PCG (Gravina et al., 2019).

Quality-diversity algorithms have been applied to the domain of shooter games as well. Gravina et al. (2016) use the notion of surprise to create a weapon generator for *Unreal Tournament III* (Epic Games, 2007) that outputs unexpected, yet balanced and usable weapons. Other work on weapon generation by Gravina and Loiacono (2015), describes multi-objective approaches to combining balance with respectively other game design goals and minimizing the changes made to the original weapon. Both works aim to achieve a balanced match by equalizing the ratio of kills obtained by each player; as in this thesis. However, instead of computing a distance to a designer selected value, they encode this by maximizing the entropy of this value as in (Lanzi et al., 2014). McDuffee and Pantaleev

(2013) evolve weapons based on how long they are equipped and the number of kills obtained with them. A weapon's ten parameters not only affect its gameplay but also its graphical representation in the game. Hastings et al. (2009) evolve neural networks that produce bullet patterns for guns in a top-down arcade shooter. The runtime evaluation and generation is interactive and based on player usage. While applicable to first-person shooters, the approach is an interesting combination of the algorithms applied in this thesis.

The first search-based approach to map generation for FPS games was described by Cardamone et al. (2011b). They use a simulation-based fitness to maximize combat time between players and add a term to promote open spaces. They explore four representations in which the evolutionary algorithm either has to remove walls in a grid of walled off tiles (*grid*), place walls on an empty canvas (*all-white*), dig space out of a canvas of wall blocks (*all-black*) or evolve the parameters of a digging agent (*random-digger*). While this thesis also evolves maps, the manipulation of the map occurs at a higher level than individual tiles. The dataset for model training does depend on a similar digging agent and Section 8.2.1 explores how well the model can learn to predict combat time.

Extending the work of Cardamone et al., Lanzi et al. (2014) aim to balance matches between two players with different weapon types by evolving a balanced map, which is similar to our map generation experiment in Section 7.4.2. They evolve maps with the all-black representation but instead of combat time, they maximize the entropy of the score ratio. This value is maximal when both players have an equal score, so it is a similar - but less linear - metric as the one used to measure balance in this thesis.

Cachia et al. (2015) generate maps with two floors for death matches between groups of players. The first floor is represented by all-black and the second by random-digger. Generation is split into two stages, one for geometry and one for objects; reasoning that proper object placement requires a finished architecture. The first stage is evaluated via both of the above mentioned simulation-based metrics, while the second uses a combination of the general map metrics of (Liapis et al., 2013c).

In contrast to the above mentioned approaches, Bhojan and Wong (2016) use smart initialization (based on constructive PCG), a small population size, a high level approach to variation and a static fitness function in order to generate maps at a reasonable runtime time scale. While ideas concerning smart initialization and higher level mutations and the goal of fast generalization are similar to this thesis, the implementation is very different. Aside from the model-based fitness in this thesis that contrasts with their static evaluation, their search algorithm, for example, relies on mutations that apply a pass of a constructive algorithm to an area of the map, while this thesis performs more traditional (lower level) mutations for fine-tuning and crossover for large variation.

Other goals for shooter map generation include fostering team play interactive evolution (Ølsted et al., 2015), inducing 'fight or flight' behavior by evaluating the internal states of an agent (Loiacono and Arnaboldi, 2017) and facilitating designer control via seeding, post-processing and constraint-based fitnesses (Liapis, 2018).

2.2.2 PCG via Machine Learning

So far, machine learned models are primarily used to directly manipulate game content without evaluating its gameplay (Summerville et al., 2017). For instance, neural networks have mostly been used to learn level patterns which are then applied directly to the level. For example, a recurrent neural network that predicts sequences of tiles is used to create levels for *Super Mario Bros.* (Nintendo, 1985) in (Summerville and Mateas, 2016); a convolutional neural network (CNN) is used to place resources on a pre-made *Starcraft II* (Blizzard, 2010) map (Lee et al., 2016); autoencoders were trained to learn patterns in *Super Mario Bros.* levels, taking advantage of the encoding-decoding sequence to repair broken segments (Jain et al., 2016). Other work has used GANs to generate levels for DOOM (id Software, 1993) by splitting the representation into six feature maps, although it is not clear whether the algorithm produces valid, playable levels (Giacomello et al., 2018). Finally, CNNs have been used to predict various characteristics (difficulty, enjoyment and aesthetics) of *Super Mario Bros.* levels based on player annotations (Guzdial et al., 2016), but these networks were not used for content generation. The framework proposed in this thesis uses its learned model indirectly (as a surrogate model to guide search) rather than directly.

An interesting exception to the trend is (Volz et al., 2018), which describes a two-phased approach for creating *Super Mario Bros.* levels. Compared to this thesis, it inverses the roles of the search algorithm and the model. In the training phase, a convolutional generative adversarial network is trained to learn patterns of existing levels by mimicking them. In the generation phase, a simulation-based algorithm searches the latent space of the generative network for playable levels with the desired gameplay.

2.2.3 Surrogate-assisted PCG

Surrogate models for fitness evaluations have been widely used in evolutionary algorithms (Jin, 2011; Jin et al., 2018). Although game simulations can be as prohibitive as simulations in other domains, the application to games is relatively new. Probably the first work in this area, by Volz et al. (2016), demonstrates that a surrogate model based on a multi-objective perspective of the fitness can outperform a single objective approach when creating a deck of cards for the game *Top Trumps*. In this case, the surrogate is a function that evaluates the hypervolume of the search space that is dominated by an individual.

Morosan (2019) compares several surrogate models based on machine learning (i.e., neural networks, decision trees and k-nearest neighbors) for fine-tuning parameters of a *Ms.Pac-man* agent, a racing car in *TORCS* and unit parameters in *StarCraft*. The models have varying degrees of success in the different games and struggle especially in the *StarCraft* setting. Most interesting is that Morosan (2019) takes an opposite approach to machine learning as this work. The models are trained online, during evolution, while using the original fitness function (game simulations) until the model obtains a sufficient accuracy on a validation set. Once the model is in place, individuals that are predicted to have a high fitness are passed to the simulation in order to compute the true fitness. This online approach minimizes failures based on wrong approximations, but also defeats the purpose of using a surrogate model. Indeed, results show that all of the surrogate-assisted approaches have longer run times than using pure simulation. It would be interesting to see if this changes with the use of pretrained models.

Volz (2019) observes that games can be expensive fitness evaluations for search-based PCG and proposes an algorithm that is particularly lazy in its fitness evaluations. For the most part, the confidence intervals of the surrogate model (or even noisy simulations) are used to create rankings between individuals. The true evaluations are only used during survivor selection on lower rank individuals if the population cannot be filled with the predicted top rank. They apply the algorithm with mixed success to deck generation for *TopTrumps* and evolving Mario levels via the latent space of GANs (as in (Volz et al., 2018)) and note that model validation during generation requires further work.

So far, surrogate-assisted PCG has mostly been applied to automated game balancing via parameter tuning and there are still a lot of possibilities to explore. While Volz (2019) evolves latent vectors that are decoded into 2D levels, this thesis is the first approach that results in a 3D level. Compared to the other surrogate models, this thesis is the first to propose a vision-based model that looks at the initial game state for its predictions and the first that compares shallow and deep neural networks, the first that performs multi-modal fusion and also the first that outputs multiple, human-interpretable values. While other work has sought to handle the uncertainty of a surrogate model, this thesis has a separate training phase to obtain a robust model and instead explores how a model can explain its decisions.

2.3 Neural Networks for Game State Evaluation

Famous for their classification performance in natural images, convolutional neural networks (CNNs) can learn to extract patterns from any type of spatially arranged input and be applied to a wide variety of tasks (LeCun et al., 2015). The successes of CNNs in other fields have inspired the application to games. Perhaps the first big success in games was a Deep Q-Network (DQN), a CNN combined with reinforcement learning, for playing *Atari* arcade games based only on screen input (Mnih et al., 2015b). The authors claimed human-level performance and for some games even exceeding it, although this has recently been disputed (Toromanoff et al., 2019). Nevertheless, it has inspired the creation of many combinations of deep neural networks with reinforcement learning, which have been applied to games in several genres (Justesen et al., 2019).

In the area of first-person shooters, DQNs have been shown to perform human-like behaviors based only on screen input, in a set of simple scenarios in VizDoom, a research platfrom based on *Doom* (id Software, 1993) (Kempka et al., 2016). Since then, bots have been competing in the annual Visual Doom AI Competition[1]. Interestingly, the winner of the 2016 competition fuses its multi-modal input streams in a way that similar to this thesis (Dosovitskiy and Koltun, 2017). It uses two information streams in the network to process multiple prediction goals, including future game state information such as health or ammo; such an architecture has been explored for this thesis, but was not required for the complexity of this task.

Another big landmark was the defeat of Go champion Lee Sedol by the program *AlphaGo* (Silver et al., 2016). The algorithm combines MCTS with two CNNs: a policy network that predicts the best moves and a value network that predicts the outcome of the game. Similar approaches have been applied to RTS games with various degrees of success. Yang and Ontañón (2018) and Stanescu et al. (2016) demonstrate a CNN that generalizes well to larger maps and beats the (static) baselines at the cost of more computing power, which could probably be resolved by using a GPU. Barriga et al. (2017) combine the value network with a policy network that predicts the outcome of MCTS for choosing tactical behaviors. While the playing strength is slightly weaker, this algorithm plays much faster and eliminates the need of a forward model. The function of these value networks, and to some degree the latter policy network, is very similar to that of the surrogate model in this thesis; in the sense that they both have to summarize many time steps of gameplay into a value. The main difference is that these networks are used continuously during play, instead of once, and therefore typically receive temporal inputs.

[1]http://vizdoom.cs.put.edu.pl/

Extending this to evaluating human play is outside of the scope of this thesis, as player modeling is a field of its own (Yannakakis and Togelius, 2018, Ch. 5). Yet, in relation to balance prediction, it should be noted that neural networks have been successfully used for match making by predicting match balance and player enjoyment in *Ghost Recon Phantoms* (Ubisoft, 2014) based on an embedding of match summaries and player attributes (Delalleau et al., 2012).

2.4 Computational Game Creativity

The field of computational creativity (CC) brings together work on autonomous content creators and mixed-initiative tools for any aspect of the creative domain. An important binding factor is a degree of creative control that goes beyond 'mere procedural generation'. One of the fundamental questions is what it means to be creative and how creativity can be assessed in computational systems (Colton, 2008). Equally relevant is the question how software can foster human creativity by striving towards mixed-initiative co-creativity (Yannakakis et al., 2014). Liapis et al. (2014b) argue that the multi-faceted nature of games, which are ultimately evaluated via an interactive play experience, make games the perfect application domain for CC research. At the same time, games and PCG can benefit from considering the creative capacities of the algorithms involved. Interesting in regard to intentionality is work by Khalifa et al. (2019) on various algorithms for creating segments of *Super Mario Bros.* levels that require specific mechanics, which -if put in a sequence- is a step towards more purposefully generating certain gameplay experiences. In this thesis, we aim to use a model of gameplay to instill the generative software with a holistic perspective of game content by connecting the multiple facets into one evaluation. This, as well, could be a stepping stone towards more intentional game content generation; either as part of a design tool or a system that combines multiple game facets autonomously.

2.4.1 Computational Creativity and PCG

The application of CC theory to games has so far been limited and is in fact mentioned as one of the challenges for PCG (Togelius et al., 2013a). Similarly, Guzdial et al. (2018) evaluated various machine learning algorithms from a co-creativity perspective and conclude that the current approaches to level generation via machine learning are insufficient for supporting a wide variety of users in their needs. Computational creativity concepts that have been applied to PCG systems concern the perspectives of multiple forms of creativity and conceptual blending. Dormans and Leijnen (2013) explore how the notions of usefulness and novelty can transform a grammar-based algorithm from having combinatorial creativity into showing exploratory creativity. Similarly, Liapis et al. (2013a) explored these notions to create a system with transformational creativity based on alternated phases of exploration and transformation. On the other hand, Guzdial and Riedl (2018) apply techniques based on combinatorial creativity to increase the expressivity of a machine learning-based level generator.

Gow and Corneli (2015) propose a theoretical framework for generating new games via a formal model of conceptual blending. They describe how this could work for mechanics and sprites, while advising that levels should be blended via specialized generators instead of blending the level artifacts directly. This approach of blending level generators was successfully applied by Guzdial and Riedl (2016) to the domain of Mario levels with generators trained on the styles of various worlds. Though with the current focus on single facets, we cannot blend concepts that are very dissimilar. For example, Sarkar and Cooper (2018) generate platformer levels by blending the output of two deep neural networks trained on data from two different games. The resulting diagonal levels are visually interesting, but also unplayable, because the system does not take into account the traversal mechanics. The source games use horizontal and vertical movement respectively, so the resulting blend cannot be played using the mechanics of either one. In order to take conceptual blending to the next level, it is necessary to take into account multiple game facets, especially the resulting gameplay.

2.4.2 Mixed-initiative Game Design Tools

In contrast to regular authoring tools, mixed-initiative systems enable a shared control of the generative process between human and computer. Both creators pro-actively contribute to the design, although the two initiatives do not have to contribute to the same degree (Yannakakis et al., 2014). The goal of such a system is to not only alleviate work load, but also to inspire the human designer and foster creativity, e.g., by analyzing content and providing suggestions.

In the domain of game design, these tools have mostly been developed for level creation. *Tanagra* (Smith et al., 2011) assists in the design of 2D platformer levels by providing real-time suggestions that are guaranteed to be playable; the system and the designer can manipulate both the rhythm beats that create the level geometry and the geometry itself. The designer can freeze elements in the level, forcing the generator to take these fixed elements into account when adjusting the content. *Ropossom* (Shaker et al., 2013) creates partial or full designs for levels of the physics puzzle game *Cut the Rope* (Zeptolab, 2010). The tool incorporates a reasoning agent for checking playability, which ensures the generation of playable content and gives the designer feedback on the actions that are required to solve the level. Alternatively, *Evolutionary Dungeon Designer*(Alvarez et al., 2018) assists in creating dungeon levels by indicating or placing design patterns and providing suggestions based on symmetry, patterns or novelty.

While the tools above design the level content directly, another popular approach is to manipulate content at different layers of abstraction, starting with an abstract sketch before the details of the level are created. *Sentient Sketchbook* (Liapis et al., 2013d) uses the sketch representation to analyze the level based on general design patterns (Liapis et al., 2013c) and provides suggestions to the designer based on novelty and a model of what they might find visually appealing (Liapis et al., 2014a). The designer typically operates on the sketch, while the computer transforms it to the final representation via constructive PCG. On the other hand, Karavolos et al. (2015) describe a mixed-initiative waterfall-like pipeline based on an iterative refinement of level sketches in which the designer and the computer take turns in working towards the final representation. Whereas *Sketchaworld* (Smelik et al., 2010) converts sketches of multiple two dimensional layers into 3D environments and lets users manipulate the content at various levels of detail in no specific order, while it ensures consistency and preserves manual changes at the lower level.

An interesting tool that combines multiple game facets is *Cicero* Machado et al. (2018), which can support the design of any game that can be described in the Video Game Description Language (Schaul, 2013). To support level design, Cicero deploys various general game-playing agents for automated playtesting and gameplay visualization. Aside from such level-based features, the tool can search a database of game descriptions to suggest mechanics from other games based on their similarity to the sprites that are in the current game. An extension to this tool can even suggest new sprites and accompanying mechanics, in the form of collision rules, following a similar procedure (Machado et al., 2019).

2.4.3 Game Facet Orchestration

Procedural content generators usually focus on one facet of games (usually levels), a key challenge along the path of fully autonomous game generation is the *orchestration* of the multitude of creative facets of a game. Based on the survey of Liapis et al. (2019b), games consist of six facets (visuals, audio, narrative, levels, rules and gameplay) and designing content of one facet must account for properties of the other facets. In an analogy with music creation, they distinguish between processes based on a *composer*, which coordinates the generators in a top-down fashion, and those based on *improvisation*, in which harmonization emerges bottom-up from the interactions between generators.

A typical top-down approach is taken by *Sonancia* (Lopes et al., 2016), which generates levels for horror games. A designer or generator creates a curve of tension progression through the level, which is given to a search-based level generator that aims to match that progression by placing monsters in rooms. Once the level is finished, it is given to an audio generator that places background sounds that suit a room's tension value.

The *Game-O-Matic* allows players to input micro-rhetorics to create simple arcade games(Treanor et al., 2012). These tiny narratives, in the form of noun-verb-noun relations, create a concept map. The verbs (which are selected from an existing list) are converted into game mechanics. The nouns create the visuals via a web search, which are placed in a 2D level. While this system is more of an assembler than a generator, different combinations of mechanics and visuals can dramatically change the interpretation of the resulting gameplay.

Hoover et al. (2015) created *AudioInSpace*, a space shoot 'em up game that incorporates an interactive evolution of weapons similar to *Galactic Arms Race* (Hastings et al., 2009). In this game, the neural networks that produce the bullet patterns receive pitch information of the game's audio and the firing position on screen. This determines a bullet's trajectory, speed and color. However, the bullet information and the player's firing behavior are input to another neural network, which generates the audio (pitch and duration). These networks are co-evolved during gameplay. The result is an interesting fusion of facets (audio, visuals, rules and gameplay) that have a reciprocal effect on each other.

Perhaps the most well-known multi-faceted game content generator is *ANGELINA* Cook et al. (2016a,b), being the only generative system that competed with humans in a game jam. In the effort of creating a fully-automated game designer, there have been several versions over the years and in various stages it has orchestrated a combination of level designs, rules, audio and visuals. The core of the system has largely remained the same and relies on cooperative coevolution, which maintains multiple facets as different species that evolve separately, but share information for evaluation purposes. Due to the large independence of the species, multiple facets can be generated concurrently and the system is easily extended by adding more species. The first version of *ANGELINA* (Cook and Colton, 2011) generates the rules, levels and NPC positions for simple arcade games, using static fitness functions on each facet separately. At each step of the process, each generator incorporates the best design of the other generators into the evaluation of their design. Without actual game simulations, however, it was difficult to achieve playable content. As such, later versions included game simulations as part of the evaluation.

Compared to static fitness functions, simulation-based evaluation is especially interesting for facet orchestration and has shown promise for orchestrating multiple game facets, as each facet may affect the simulation differently. Simulations do not only inform the generator about playability, but can also be used to describe features of gameplay. Gameplay usually emerges from the interaction between several facets of a game and can therefore be used to evaluate a combination of designs based on the same metrics. In the seminal work of Browne and Maire (2010), on a system called *Ludi*, both rule sets and board layouts were optimized based on metrics that processed the progress of gameplay between two agents with the same decision-making criteria. Browne and Maire computed 57 metrics, many of which related to game progression: these included the duration of the game (in moves), the uncertainty of the game and its drama (i.e. how easy one player could reclaim the lead towards victory). When assessing game rules, Togelius and Schmidhuber (2008) measured the learnability of the game by observing the average score of a population of controllers learning to play the game across all generations. In the genre of first-person shooters, Lanzi et al. (2014) evolve levels towards score balance between players, while Gravina et al. (2016) use the heatmaps of players' deaths from a simulation in order to create surprising weapons that break heatmap patterns. If applied to the same game, these generators can be orchestrated by evaluating the same simulation, despite the fact that they do not compute the same metrics from it. This thesis extracts two simple but established metrics from the simulations: the score ratio at the end of the simulation, similar to Lanzi et al. (2014), and the duration of the match, similar to Browne and Maire (2010).

Based on the typology of orchestration processes in (Liapis et al., 2019b), the approach of this thesis lies in the bottom-up side of the spectrum. This thesis merges two generators that generate content concurrently and while they evolve the two facets in one genotype without communication or mutual influence, there is no top-down pipeline of generators passing down output. The surrogate model performs a holistic evaluation with regards to a target, but does not create the plan or the input for the generators. Therefore, facet generation is concurrent, similar to Ludi, rather than via a generative pipeline favored by Sonancia. With regards to interactivity, this thesis clearly lies in the non-interactive area of the spectrum. While the pipeline is designed with mixing initiatives in mind, the only true interaction between the system and a designer is in giving the initial input and setting the gameplay targets of the output.

2.5 Summary

This chapter has provided background on the fields that are touched upon by this thesis. The facet orchestration approach of this thesis is aimed at supporting game designers and is based on extracting patterns from data. This chapter has briefly outlined the concept of game design patterns, a formalization of game design knowledge, and how they are applied to shooter games. Furthermore, it discusses the notion of game balance and how we can try to pursue this via algorithmic means. This research is informed by and contributes to a large body of knowledge on procedural content generation. This chapter provides a short categorization of the field before diving into search-based procedural content generation and earlier work on generating maps and weapons for first-person shooters. Since this work merges the generation of multiple facets, this chapter surveys other works that have combined facets and created complete games, using the recently introduced taxonomy of game facet orchestration as a lens for comparison. Additionally, generating multiple game facets is part of a creative task, either performed by autonomous system or a human designer. As such, this chapter also covers work on computational creativity for games and tools that support game designers. The algorithms employed are machine learning and surrogate-assisted evolution. These algorithms can be used for any domain and applications to procedural content generation are relatively new. This chapter has positioned this thesis in relation to other approaches using these techniques without describing specifically how they work. In contrast, Chapter 3 will provide details on these algorithms regardless of the application domain.

Chapter 3

Algorithms

This chapter provides context for the algorithms used to create the multi-faceted procedural content generation approach in this thesis. As outlined in Section 1.2, it consists of two components: a model that can predict gameplay and a procedural content generation algorithm that creates game content based on the feedback of the model. Section 4.2.1 will describe how the algorithms described in this chapter fit into the proposed pipeline.

The model of gameplay is obtained via machine learning, a field of artificial intelligence that aims to learn functions by processing data. In particular, we propose to use a deep neural network, a type of algorithm that is inspired by the human brain. Section 3.1 explains how these algorithms work. Recent advancements in this field have been labeled as deep learning. Section 3.1.3 describes the innovations that have made deep neural networks so successful in recent years. The proposed neural network model in this thesis is based on these innovations, while the more traditional models in Section 3.1.1 are used as benchmarks.

The game content is generated via a search-based algorithm (Togelius et al., 2011), which means that it searches the design space for good designs according to some evaluation function. The successes of this approach have been described in Section 2.2.1. A particularly suitable category of algorithms for this form of PCG are evolutionary algorithms. Section 3.2 describes a typical evolutionary algorithm, followed by a background on using multiple evaluation functions simultaneously in Section 3.2.1 and approximating the evaluation function with a model in Section 3.2.2.

3.1 Machine Learning

Machine learning is a field in Artificial Intelligence that is concerned with making the computer "learn" a mapping between inputs and outputs by finding patterns in data. Traditionally, there are three main branches of machine learning based on the type of data available. We speak of *supervised learning* if the desired outputs are known, *unsupervised learning* if they are not and *reinforcement learning* if the output is learned by acting in an environment that gives feedback. In this thesis, a mapping is learned between game content as input and gameplay as output. The latter is observed via game simulations before training the model. As such, this thesis will apply algorithms for supervised learning. The goal of supervised learning is typically to find a mapping from an input x to a target variable t. This function is learned from labeled data in order to make predictions about t when new data is observed. While t can be used for *classification*, by taking the form of discrete classes, the majority of our models will aim to predict one or more continuous variables, i.e., to perform a *regression* task.

3.1.1 Linear Regression & Feed-forward Networks

The simplest model for regression is a linear combination of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + ... + w_D x_D \tag{3.1}$$

where $\mathbf{x}$ is the input vector with a number of D dimensions and $\mathbf{w}$ are the weights associated to each input, including the *bias* parameter w_0 which compensates for any fixed offset in the data. Though successful because of its simplicity, the fact that it is a linear model puts significant constraints on the functions that it can approximate. It is possible to extend this type of model with fixed non-linear transformations of the input vector (so called *basis functions*). This would make the model a non-linear function of $\mathbf{x}$ while remaining linear with respect to $\mathbf{w}$, but this introduces new constraints that can be inconvenient (Bishop, 2006, p. 172).

Instead, we focus on another class of non-linear models: feed-forward neural networks. Inspired by the brain, these models implement the concept of a network of interconnected nodes that perform simple operations, while complex function fitting properties arise from the connections between these nodes. In their simplest form, a single neuron, these models transform the output of linear regression with a non-linear function:

$$y(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{i=1}^{D} w_i x_i + w_0\right) \tag{3.2}$$

where $\sigma(z)$ is typically the logistic sigmoid function, defined by

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

Note that the summation in Eq. (3.2) is equal to the dot-product of $\mathbf{w}$ and $\mathbf{x}$. Indeed, by using matrix multiplication we can easily generalize Eq. (3.2) to a single-layer network that predicts K outputs:

$$\mathbf{y}(\mathbf{x}, W, \mathbf{b}) = \sigma\left(W \cdot \mathbf{x} + \mathbf{b}\right) \tag{3.4}$$

where b is a vector of bias parameters for each output k and the weights for each output k are combined into a single matrix $W \in \mathbb{R}^{K \times D}$. In the context of neural networks, the non-linear transformation in these equations is also known as an *activation function*. Traditional alternatives for the logistic sigmoid include the hyperbolic tangent for regression and the step-function for binary classification. In fact, the first neuron-inspired model, the *perceptron* (Rosenblatt, 1962) employed a step-function, since it was used for classification tasks. Given their close resemblance to the original perceptron, this thesis will use that name for this class of models.

Note that the single-layer network is not a network of interconnected neurons, but in fact an independent neuron for each output. The true strength of artificial neural networks with regards to function approximation lies in stacking multiple layers on top of each other, thus nesting the non-linear transformations of Eq. (3.4). Each of these layers functions as the generalized perceptron with multiple outputs, which explains the name *multilayer perceptron* (MLP) for this class of models. Before the rise of deep learning, an MLP rarely had more than two layers. A typical two-layer network for predicting K output variables is depicted in Fig. 3.1 and can be written as

$$\mathbf{y}(\mathbf{x}, W_1, W_2, \mathbf{b_1}, \mathbf{b_2}) = \sigma\left(W_2 \cdot \sigma\left(W_1 \cdot \mathbf{x} + \mathbf{b_1}\right) + \mathbf{b_2}\right) \tag{3.5}$$

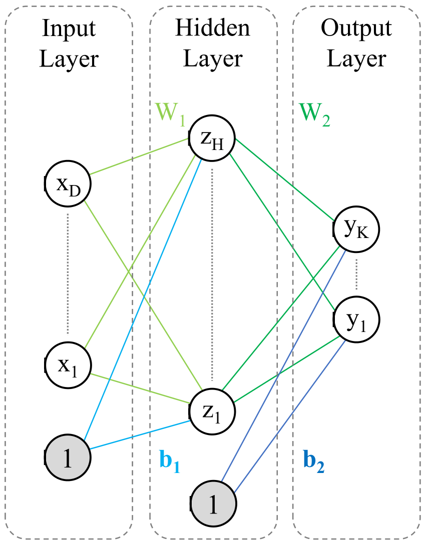


Figure 3.1: Schematic of a two-layer neural network with D inputs and H intermediate neurons and K outputs. Information flows from left to right. The weights are represented by the links between the nodes, while the biases are denoted by links coming from additional inputs that always have a value of one.

where each layer has their own weight matrix and bias vector, which are $W_1 \in \mathbb{R}^{M\times D}$, $\mathbf{b_1} \in \mathbb{R}^M$ for the first (hidden) layer, and $W_2 \in \mathbb{R}^{K\times M}$, $\mathbf{b_2} \in \mathbb{R}^K$ for the second (output) layer.

3.1.2 Network Training

The parameters of the network are found via *gradient descent.* First, the parameters are initialized at small, non-zero random values. This is followed by a forward pass, i.e., the network makes predictions on the training set. Then, the error of these predictions are computed and the gradient of this error is used to adjust the weights such that the error is minimized. For a regression task, we typically use a sum-of-squares error function. Given a set of labeled training data of size N with input vectors $\mathbf{x}$ and target vectors $\mathbf{t}$, this function is given by

$$E(\theta) = \frac{1}{2}\sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x}_n, \theta) - \mathbf{t}_n||^2 \tag{3.6}$$

where θ denotes the network parameters. The gradient of this error function with respect to each of the weights is evaluated via *backpropagation* (Werbos, 1974), which performs a backward pass through the network from output to input in order to compute the contribution of each weight and bias parameter towards the obtained error value. The weights are updated with a step $0 < \eta < 1$ in the direction that decreases the error; this step size η is called the *learning rate.* When using large datasets, computing time and memory are saved by iterating over small batches of data instead of evaluating the gradient of the whole training set. Another benefit of this variant of the algorithm, called *stochastic gradient descent*, is that it is more likely to escape from local minima, since a stationary point with respect to the error function on the whole dataset is rarely a stationary point for each batch of data.

Classification

When training a neural network for a classification task, much of the above remains the same. The differences result from treating the K outputs not as separate variables, but as one probability distribution over K classes. The network then predicts the class that has the highest probability. In order to transform the output to a probability distribution, the activation function of the final layer in Eq.(3.5) is chosen to be the identity function. The output of the network is then passed through a *softmax function*[1]:

$$s(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.7}$$

where $\mathbf{z}$ is the output vector of the network. Obviously, the target vector $\mathbf{t}$ should also be a probability distribution. This is obtained by converting each class label $t \in \mathbb{N}$ into a binary vector with a one-hot encoding of that label. The error function should reflect the comparison between probability distributions. For this we use the cross-entropy function:

$$E(\theta) = -\sum_{n=1}^{N} \mathbf{t}_n \log\left(s(y(x_n, \theta))\right) \tag{3.8}$$

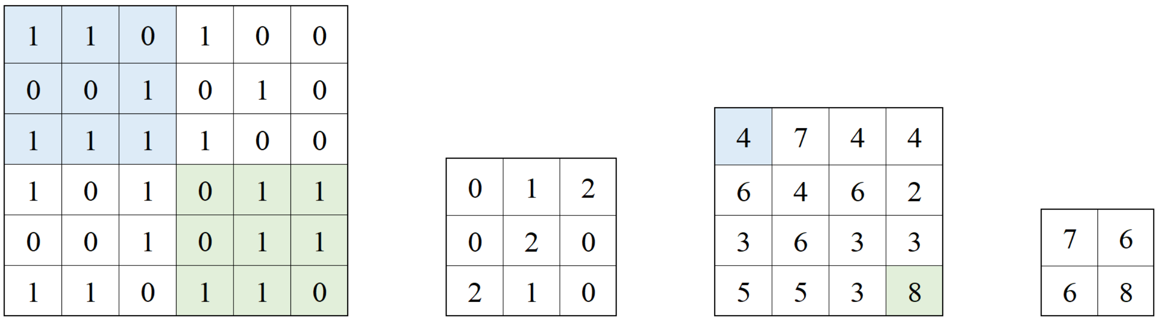
3.1.3 Deep Learning

Deep learning is the name of the subfield of machine learning dedicated to the training of artificial neural networks and originates from merging the terms machine learning and deep neural network. It refers to the recent advancements that have allowed for the training of networks with many hidden layers. A deep neural network contains multiple layers of artificial neurons and stands in contrast to the traditional networks with a single hidden layer (MLPs). These deep networks have led to a revolution in the field of machine learning, especially in computer vision and natural language processing (LeCun et al., 2015; Schmidhuber, 2015).

In recent years, convolutional neural networks (CNNs) have become the dominant machine learning approach to image processing due to their success in computer vision tasks (Russakovsky et al., 2015; Gu et al., 2018). CNNs are networks with at least one layer that applies the mathematical operation called *convolution* instead of general matrix multiplication (Goodfellow et al., 2016). This multiplication operation moves a *kernel* over an input with a certain *stride*, while multiplying the two at each location. An example of this is shown in Fig. 3.2. In the context of CNNs, these kernels are the nodes of the layer. The output of convolution still passes through an activation function like a normal fully-connected layer. The result of this operation with one node resembles the output of a fully-connected layer where each node shares the same parameters. While a convolutional layer usually contains more than one node per layer, a CNN is typically a lot more efficient with its parameters than a fully-connected network.

The intuition behind using convolution is that we do not need to know pixel perfect locations of features like eyes and ears to detect a face in an input image. To increase the

[1]This is considered to be the multiclass generalization of the logistic sigmoid of Eq. (3.3)(Bishop, 2006, p.198).



1	1	0	1	0	0
0	0	1	0	1	0
1	1	1	1	0	0
1	0	1	0	1	1
0	0	1	0	1	1
1	1	0	1	1	0

(a) input

0	1	2
0	2	0
2	1	0

(b) kernel

4	7	4	4
6	4	6	2
3	6	3	3
5	5	3	8

(c) output

7	6
6	8

(d) pooled

Figure 3.2: An example of convolution and max-pooling. The kernel (b) is moved over the input (a) with a stride of one from the blue position to the green position to produce output (c), while max-pooling applied to (c) results in (d).

location invariance and simultaneously reduce the size of the output, the output is often passed to a pooling function. Most common is the use of *max-pooling*, which outputs the maximum value in a 2×2 area; as shown in Fig. 3.2d.

Convolutional layers can be understood as a set of filters that are moved over the input to (learn to) detect certain features. As such, the output is typically called a *feature map*. The first layer of these filters often learns to detect edges or spots of different color. Higher layers learn compositions of lower-level features; generally, the more convolutional layers a network has, the more complex features it can detect.

One of the problems of traditional MLPs is that the error signal shrinks to such a degree during backpropagation through consecutive layers that earlier layers become untrainable. This primarily results from the use of sigmoids and hyperbolic tangents as activation functions. For example, the maximum gradient of the sigmoid is 0.25; so during backpropagation at least 75% of the error signal is lost between layers. This *vanishing gradient problem* prevented the construction of deeper networks. A major breakthrough resulted from the use of the rectified linear unit (ReLU)(Nair and Hinton, 2010) as activation function: $g(z) = max(0, z)$. This function encodes the degree of presence of particular features. The gradient of this function is either 1 (if $z > 0$) or 0 (if $z \leq 0$)[2]. In other words, when the node is not active, it did not contribute to the error and no gradient is passed during backpropagation; but when a node is active ($z = 0$), the error signal is passed through the layer without loss of strength.

While being efficient to compute, inducing sparsity in the network, and solving the vanishing gradient between layers, it turns out that the ReLU has a problem of its own: nodes can be pushed into a state of inactivity for all inputs, from which they cannot escape because the gradient will always be zero. Since then, several related functions have been proposed as alternative, such as the Leaky ReLU(Maas et al., 2013) and the Exponential Linear Unit (ELU)(Clevert et al., 2016). Both functions solve the "dying neuron" problem by encoding the absence of a feature, thus ensuring that there is a gradient when $z < 0$. The former adds a small negative linear segment to the ReLU function:

$$g(z) = max(0, z) + \alpha\, min(z, 0) \tag{3.9}$$

[2]Technically, it is non-differentiable at zero, but in practice this is not a problem and the value is often manually set to zero in code.

where α determines the slope of the negative segment, for which Maas et al. (2013) recommend $\alpha = 0.01$. The latter adds an exponential segment that plateaus the negative activation:

$$g(z) = max(0, z) + min(\alpha(e^z - 1), 0) \tag{3.10}$$

where α controls the value to which an ELU saturates, for which Clevert et al. (2016) recommend $\alpha = 1$. The saturation in the negative segment creates a robust deactivation state, making the ELU less sensitive to noise than other ReLU alternatives. Indeed, Clevert et al. (2016) have demonstrated faster training times and higher accuracies on certain computer vision tasks when using ELU compared to Leaky ReLU and ReLU. Despite theoretical and practical improvements, the original ReLU has an impressive track record and so far remains the most popular activation function.

Aside from activation functions, the vanishing gradient problem can be addressed via the optimization algorithm. Initial suggestions improved on stochastic gradient descent by gradually reducing the learning rate over time or by adding a *momentum* term to the loss function. The classical form of momentum bases the majority of the weight update on the previous update vector (typically 90%) while 10% is based on the current batch (Qian, 1999). This makes the weight update more robust to deviations between batches and continues updating the weights when the current error landscape is flat. Recent increases in computing power have led to more complex optimization algorithms that adapt the learning rate per parameter. The recently proposed *Adaptive Moment Estimation* (Adam) (Kingma and Ba, 2015) has quickly become the default optimization algorithm for most problems; it requires little hyperparameter tuning, is computationally efficient and performs particularly well in noisy and non-stationary problems. Adam computes moving averages of both the gradient and the squared gradient, which are exponentially decayed over time. These moving averages are estimates of the first and second moment of the gradient (hence the name), which are the mean and the uncentered variance . Essentially, each parameter is updated based on its own momentum with a learning rate that is scaled by the variance of its gradient. The decay rates are controlled by two parameters, which have recommended values close to one so as to give a lot of weight to the momentum term in the update.

The introduction of deeper networks resulted in the ability to fit more complex functions, but with it came the increased risk of overfitting to the training data. As such, there is an increased need for regularization. The simplest form is to add a *parameter norm* penalty term to the loss function. This penalty depends on the size of the parameters, which is typically computed by the norm of the weight parameters of a layer. The most common variant is the $L2$-norm:

$$p(\theta) = \frac{\lambda}{2}||W||^2 \tag{3.11}$$

where $0 < \lambda < 1$ determines the weight associated to this penalty compared to the error. At every update, this penalty pushes the values of W to zero, which is why this regularization is also called *weight decay.* Recent innovations in this area include operations on the output of a layer during training, such as randomly switching off a percentage of (fully-connected) neurons, which is called *dropout* (Srivastava et al., 2014), and *batch normalization* (Ioffe and Szegedy, 2015), an algorithm that learns to transform the output of a layer to a distribution with zero mean and unit variance for each batch of training data. Intuitively, the former method aims to make neurons more independent of each other, while the latter does the

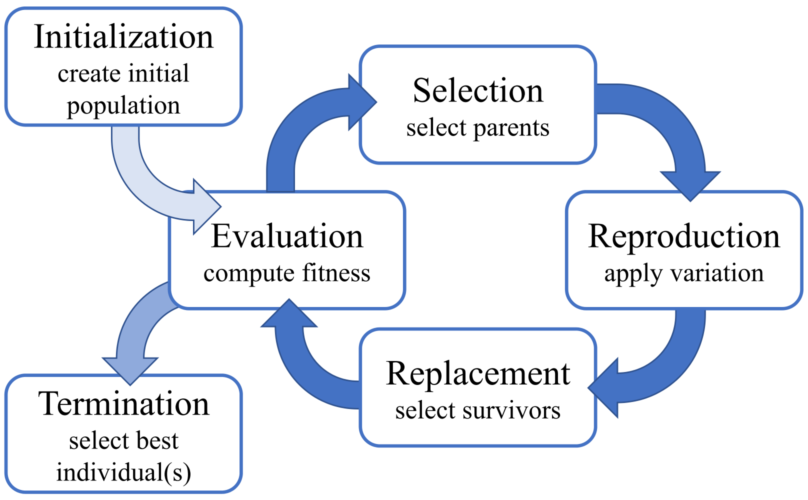


Figure 3.3: Schematic of an evolutionary algorithm

same for complete layers. While each of these methods improves the generalization capabilities of a network, the later methods are subsequently designed to replace the previous and they are typically not used in conjunction with each other.

The interested reader can find more details on neural networks and other forms of machine learning in (Bishop, 2006), while Goodfellow et al. (2016) cover the basics of neural networks up to recent trends in deep learning. Focusing on CNNs, an overview of the latest developments in deep learning can be found in (Gu et al., 2018).

3.2 Evolutionary Computing

An *evolutionary algorithm* (EA) is a population-based optimization algorithm that is inspired by the process of natural evolution. The fundamental metaphor is that of a population of individuals that compete for survival and reproduction in an environment with finite resources. From a computational perspective, these algorithms stochastically search the solution space of a certain problem for a global optimum. While Alan Turing (1969) noted the potential of evolutionary search as early as 1948, the first digital implementations were created in the 1960s and 1970s by various labs around the world (Fogel et al., 1966; Rechenberg, 1973; Holland, 1975). It was not until the 1990s that these variants merged together into the field of what is now known as *evolutionary computing*. While the variety of algorithms has only increased since then, this section will describe the components of a generic evolutionary algorithm (Eiben and Smith, 2003).

The core of the algorithm is the loop depicted in Fig. 3.3: initialize a population of (random) candidate solutions and evaluate each individual, select parents for reproduction based on their evaluation, apply variation operators, replace (a part of) the population with offspring to create the next generation and return to step two. This loop is repeated until a satisfactory solution is found or stopped after a predefined number of generations. After stopping the evolutionary loop, there is a *population* of potential solutions to the problem. Usually the individual with the best evaluation score is selected as the final solution. This score is called *fitness*, in analogy to 'survival of the fittest'.

Perhaps the most important component of an EA is the *representation* of an individual.

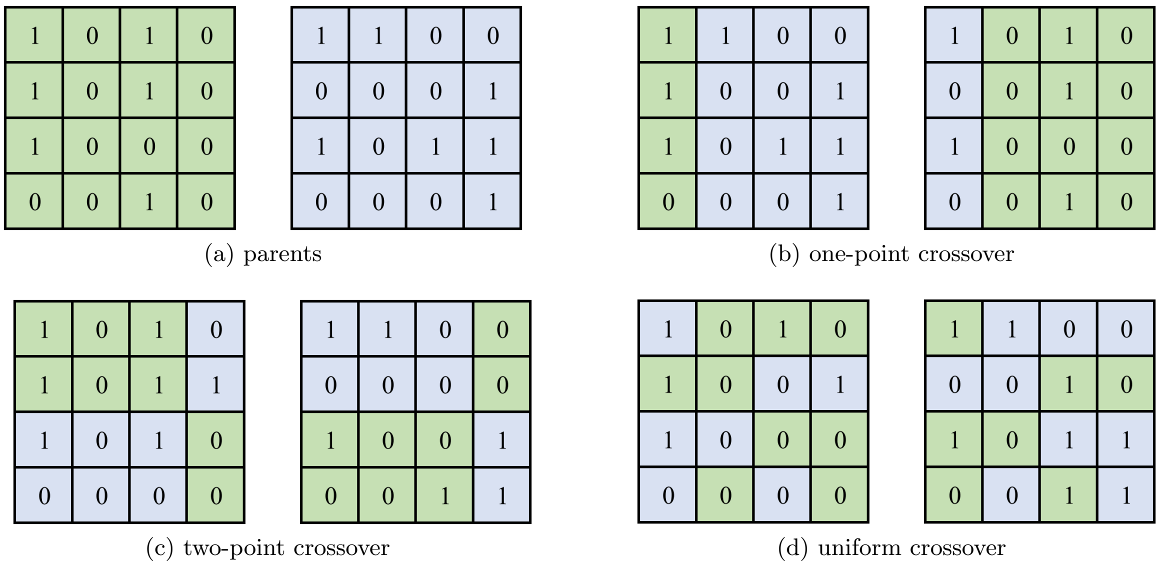


Figure 3.4: An example of offspring resulting from three different crossover operations applied to the parents.

This defines the potential solution to the optimization problem, and by extension the whole search space. Similar to the natural world, an individual typically consists of a *phenotype* and a *genotype*. That is to say, a solution in the original problem context that is evaluated in terms of its fitness and a low-level encoding that is modified by the variation operators of the algorithm. These elements respectively define the solution space and the search space. For example, in this thesis one of the phenotypes will be a 3D game environment, while the corresponding genotype is a 2D array of strings.

The quality of an individual is computed via a *fitness function*. This function evaluates the phenotype with regards to how well it solves the problem and the algorithm typically strives to maximize the outcome[3]. As such, this is also called the *objective function*. As we have seen in Section 2.2.1, a fitness function is not always purely mathematical. It can also contain, or consists entirely of, values obtained from simulations. Being the main element of parent selection, it also defines what constitutes an improvement in the search space. While the *selection* of parents is based on fitness values, it is paramount that the selection process is stochastic. Exploration is an important property of the algorithm and with too little stochasticity, the search process might get stuck in a local optimum. The most common procedures are roulette wheel and tournament selection. When using *roulette wheel selection*, parents are selected via uniform sampling. Each individual's probability of being chosen is either proportionate to their fitness with respect to the sum of all fitnesses or to their rank in the population. *Tournament selection*, on the other hand, does not require global knowledge. For each choice, k individuals are selected randomly (with replacement) and their fitness is compared locally. This results in less selection pressure, since an individual only has to be better than $k-1$ others. Though this also means that the $k-1$ worst individuals will never be selected for reproduction.

Offspring is created by subjecting the parents to variation operators: *crossover* and

[3]This is by convention. Minimization problems, such as those in this thesis, can be rewritten into maximization problems and vice versa.

mutation. Depending on a crossover rate, each pair of parents can either produce children by a *recombination* procedure (which is called crossover, in analogy to the process in meiosis) or by creating copies of themselves. Typical crossover operators are depicted in Fig. 3.4. These procedures either select one or multiple points in the chromosome to swap genes between parents (*n-point crossover*) or sample from a uniform distribution to decide which gene is inherited by which child (*uniform crossover*). Crossover can be seen as making large steps in the search space by recombining genotype information of existing candidate solutions. Mutation, on the other hand, makes small steps by changing one gene at a time. During the reproduction phase, each gene of each individual has a small chance (or mutation rate) of randomly changing its value. While the exact operation is very much dependent on the problem and the chosen representation, mutation should only make small changes to an individual so as not to step over the global optimum. Mutations are important for exploring the search space, as they add new information to the gene pool and can make changes cannot be obtained via recombination (e.g., changing a 1 into a 2 in Fig. 3.4). Yet, too much exploration can make the system unstable, because the changes might as well have negative effects on the fitness. A rule of thumb is to set the mutation rate such that the expected number of changed genes per individual is one. Whether crossover is a helpful component of an EA is a subject of ongoing debate, since mutations could potentially handle all of the steps in the search process. Regardless, when both operations are combined, mutation is applied after crossover.

Unlike populations in nature, the population size in an EA is usually kept fixed by synchronizing the births and deaths of individuals. The *replacement* of parents by their offspring can follow various schemes either based on fitness or age. The fitness-based scheme selects the new population from the combined pool of parents and children based on whichever individuals have the best fitness. Most common are the age-based schemes in which offspring replace their parents regardless of fitness. These methods differ in the fraction of the population that is being replaced. On one side of the spectrum, there is *steady state* replacement, in which a small number of parents (usually one) are replaced by offspring. On the other side is *generational* replacement, which dictates that the new population consists entirely of the new generation. However, generational replacement is often combined with copying a few of the best individuals of the previous generation to the next, regardless of the reproduction phase or other replacement rules. This *elitism* prevents the stochasticity of the search process to negatively affect the population by ensuring that the best individual(s) will not become strictly worse. Simultaneously, this process makes the algorithm more susceptible to local optima; so the number of elites should be carefully tuned.

This section has described the core elements of a typical evolutionary algorithm. While it mentions several of the involved decisions and variables, there are many more options. For details, the interested reader is referred to (Eiben and Smith, 2003). Aside from the algorithm's design, the parameter values of particular components can have a huge effect on the success or failure of finding a desirable solution as well as computation time. More information on the offline tuning and online adaption of parameters can be found in (Eiben and Smith, 2011; Karafotias et al., 2015).

3.2.1 Multi-Objective Evolutionary Algorithms

The quality of a solution cannot always be described by a single criterion. For example, when evolving an agent that plays a shooter game, the quality of such an agent can depend on many aspects: survival time, score, enemies killed, area traversed, number of health packs used, et cetera. A common strategy for such a situation is to aggregate these multiple *objectives* into a single function by creating a (weighted) sum. However, these objectives might conflict with each other and in such cases, optimizing for one objective might reduce the score in another, e.g., maximizing survival time might induce an agent to hide in cover without confronting opponents and by trying to kill more enemies, the agent will have to risk reducing its survival time or use more health packs. While searching for a good behavior, one such solution might not be strictly better or worse than the other. Yet this trade-off between objectives cannot be reflected well in an objective function that outputs a single value.

Multi-objective evolutionary algorithms (MOEAs) (Deb, 2001) address this problem by treating the multiple aspects of fitness as separate objectives. The goal of these algorithms is to find a set of solutions that are ideally better, but at least not strictly worse, than others. An individual is said to (Pareto) *dominate* another if it obtains a better score for at least one objective and is not worse with respect to all other objectives. The algorithm ideally finds one solution that dominates all others. However, if a multi-objective formulation is necessary, there is an inherent trade-off to be made between objectives. In such cases, the best solution we can obtain is a *Pareto Optimal*, or *Pareto non-dominated*, individual: a solution that cannot be improved upon in one objective without simultaneously decreasing at least one other objective. The aim of MOEAs is then to find the set of Pareto optimal solutions, also known as the *Pareto Front*, and let the user decide which of these solutions should be adopted.

The potential of EAs for solving multi-objective problems was recognized fairly early in their development and since then many approaches have been proposed, such as the aggregated fitness function mentioned above, (domination) rank-based fitnesses and keeping an archive of Pareto optimal solutions during search (Fonseca and Fleming, 1995; Coello et al., 2007). One of the most popular algorithms is Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002); reasons for this include that it is relatively fast, it contains elitism while preserving diversity and it does not require setting additional parameters. The algorithm sorts individuals based on their non-domination rank, i.e., non-dominated individuals are assigned rank 1, those that are only dominated by rank 1 are assigned rank 2, etc. This creates multiple fronts of non-domination. The ranking is used as fitness[4] for parent and survivor selection. Parents are drawn via binary tournaments, while survivors are deterministically selected from the combined pool of parents and children. When individuals of the same rank are compared, the tie is broken by computing a distance to neighbors with a similar fitness. This *crowding distance* is an estimate of the size of the largest cuboid that encloses the point of an individual in the search space without enclosing any other point of the same rank in the population. The evolutionary loop is typically continued until a predetermined number of generations is reached.

[4]Note that this results in a minimization problem regardless of the original fitness function.

3.2.2 Surrogate-assisted Evolutionary Algorithms

Evolutionary algorithms typically require a large number of evaluations before finding an acceptable solution. In many problem domains these calculations are not trivial, especially if there is a simulation involved. In cases where the computational cost becomes prohibitive it can be practical to use a model, or *surrogate*, to approximate the fitness function; thus trading accuracy for speed. The field of surrogate-assisted evolutionary computing involves many types of models, e.g., based on polynomial regression, Kriging interpolation or Bayesian optimization (Jin, 2005). In fact, any machine learning algorithm could be used to learn a mapping from genotype to fitness value. In relation to this work, notable applications of neural networks as surrogate models include modeling designer preference in a mixed-initiative music composition task (Biles et al., 1996) and game balance evaluations for fine-tuning parameters of game mechanics (Morosan and Poli, 2018).

Regardless of the model type, it is recommended to use a surrogate in conjunction with the true evaluation if it is available. While exploring the solution space, the EA will likely end up in areas that are unknown to the model, which will negatively affect its accuracy. Without the true fitness function to adjust these evaluations, the model might lead the EA to a *false optimum* (Jin et al., 2000). That is, an optimum that is perceived as such by the model, but which is not an optimum in the true solution space. There are several strategies for *model management*, or *evolution control*, which can be categorized into three types: using the true fitness function every couple of generations (generation-based), evaluating either the best or random individuals every generation (individual-based), or keeping separate populations for both evaluation types (population-based). In the context of procedural content generation, Volz (2019) proposes a surrogate-assisted EA that adaptively invokes the true fitness function during survivor selection based on the uncertainty of the approximations. More details on the combination of surrogates and true fitness evaluations can be found in (Jin, 2011), which provides an interesting overview of evolution control and surrogate-assisted evolutionary computation in general.

3.3 Summary

This chapter has provided the core concepts of the algorithms used for surrogate-based multi-faceted procedural content generation. The first section explained the training of neural networks to learn to approximate a function based on data. The first part elaborated on linear regression and multi-layer perceptrons, while the second part explained convolutional neural networks and briefly touched on the recent innovations that have made deep learning successful. The models in this section will be trained to map game facets to gameplay outcomes. The second section detailed the workings of a typical evolutionary algorithm, which is used to generate game content by searching the design space. A description of the main loop and all its elements is followed by additions that are required for solving multi-dimensional problems. In particular, the popular NSGA-II algorithm which is used in this thesis to generate content that meets multiple design objectives. The chapter closes with a short background on surrogate-assisted evolutionary algorithms, which is the field that applies machine learning to approximate quality evaluations in evolutionary computing when computing the exact values become too prohibitive. Chapter 4 will describe how these algorithms are integrated into the pipeline proposed by this thesis.

Chapter 4

Methodology

In Chapter 3, we have seen how neural networks can be used for making predictions about previously unseen inputs and how evolutionary algorithms can search a space of potential solutions. Additionally, Chapter 3 describes how an evolutionary algorithm can use a neural network as a *surrogate model* to approximate the quality measure of a potential solution. This chapter builds on the concept of combining both algorithms and proposes a pipeline for using it to create content for games.

The chapter starts with an experiment that aims to evaluate the validity of using a neural network for predicting metrics of gameplay. The methodology of this experiment is outlined in Section 4.1. Section 4.1.3 provides directions for the methodology of the dissertation based on this proof of concept: the results of that initial experimentation are included in Appendix A.

The chapter continues with the final concept of the proposed framework and the pipeline needed to get all the elements in place in Section 4.2.1. This is followed by a description of the game that was developed as a case study for the framework in Section 4.2.2. Section 4.2.3 details how the predictions of the model were used to guide the search for good game content. Based on the taxonomy of Liapis et al. (2014b), the types of content created in this thesis can be identified as rules and levels. The specific implementation of these facets is outlined in Section 4.2.4 and Section 4.2.5. Section 4.2.6 describes how the generative process is shaped when the generation of the two facets is orchestrated by the model. The generative process uses abstractions of the game content that makes it easy to manipulate these facets. Section 4.2.7 describes how these abstractions are converted into actual game content when the generation is done.

4.1 Proof of Concept

A working surrogate model is a prerequisite for the proposed facet orchestration process. The proof of concept experiment tests the validity of using a neural network to make gameplay predictions based on a map and the players' attributes. As described in Chapter 2, neural networks have been used for attributing values to a game state during the game. Yet, to our knowledge, there is no previous work on whether they can predict a property of gameplay of a finished game based on an initial state, other than the utility value for a game-playing agent.

This experiment focuses on one of the most common contests in the genre of first person shooters: team deathmatch. The goal of this game mode is to try to get more kills than the opposing team. Aside from being common, it is assumed that teams of agents produce richer data than one-vs-one or free-for-all battles as it is easier for coordinated behaviors, e.g., flanking or covering fire, to emerge. The agents of one team are all given the same weapon so as to be able to attribute the advantage in a game to a specific set of weapon features.

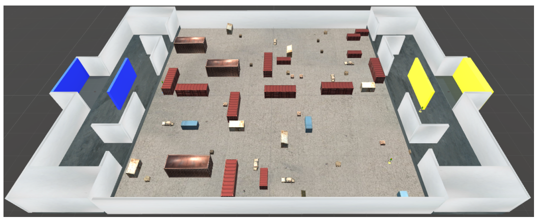
A first-person shooter game, centered around one arena level, was created specifically for the purpose of this experiment using the *Unity 3D* game engine. The game is played by artificial players in order to generate 50,000 datapoints. From these simulations we collect the maps, the weapons of both teams and the kill ratio of the first team compared to the total number of kills in the game. Supervised learning is applied to create a model that can classify the balance between two teams based on the weapons and a top-down view of the map. The experiment compares four types of models: logistic regression and multiple forms of neural networks.

4.1.1 The Team Deathmatch Testbed

In this proof of concept, a match consists of a battle between two opposing teams of three players. Each player has unlimited lives, but a match ends after a total of 20 kills. The balance is measured by the ratio of kills of each team. A match is balanced when it resulted in a draw, or when the difference in kills between the two teams was marginal.

The map of the game consists of three areas, two spawn areas on opposite edges of the level and a square *arena* in the middle, where the fighting will take place (see Fig. 4.1a). At the start of the game, two teams of 3 players start at their respective spawn areas, or *bases*. Every time a player is killed, they re-spawn at their team's base. Each base has three paths to the central arena, a ledge and an invisible wall that blocks players and projectiles in the arena from passing through, respectively; this prevents a team from being pinned down inside their base. The central arena is a flat square area, bordered by walls, and the only area that is procedurally filled with objects. Such objects are a variety of 'block' objects found in modern warehouses, and provide cover from enemy fire. Two types of object are identified: small objects which come up to chest height and allow for players to shoot opponents while taking (some) cover from them, and large objects which can completely block line of fire and line of sight from enemies. Large objects and small objects have fundamentally different tactical properties, and so are represented differently in the system and handled differently by the artificial intelligence controlling the players of each team.

In order for the level to be used as input for machine learning, it is represented as a 100 by 100 pixel image (see Fig. 4.1b), ignoring the teams' bases as they are symmetrical


(a) In-game view

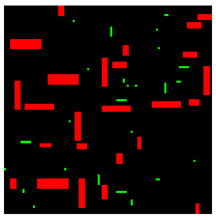

(b) Map

Figure 4.1: Example level, with team bases on the left (first team) and right (second team) of the arena, and the arena's map representation used as input for machine learning.

and no combat occurs within them. Each pixel value of this representation determines whether that tile is occupied[1] by a large object (red pixel), a small object (green pixel) or is empty (black pixel). The chosen image resolution allows for even the smallest objects to be ‘captured’.

4.1.2 Representation

The above mentioned game contains five weapons that are stereotypical for shooter games. Each weapon is represented by 20 parameters, including damage per bullet, number of bullets per volley, explosion size, reload time, accuracy, etc. All members of the same team use the same weapon, therefore the total weapon parameters used as inputs are 40 (20 per team). Weapon parameters are normalized to $[0, 1]$ through min-max normalization across the five weapons.

As shown in Fig. 4.1b, levels are stored in a 100 by 100 pixel image with three colors (red, green, black) which represent the type of objects in the level (large, small, and no objects respectively). When used by the network, the image is converted into a one-hot encoding (10 for red, 01 for green, 00 for empty), which results in a $2 \times 100 \times 100$ binary input to represent the level. This allows the network to clearly differentiate between a large object which blocks line of sight, a small object which provides cover, and the lack of either of them. As noted earlier, the inputs describe only the generated arena and ignore the teams’ hand-designed bases as no combat is allowed in those areas.

The kill balance prediction task is formulated as a classification problem with three classes based on the kill ratio of team one. If the ratio is in the range of $0.4 - 0.6$, the match is considered balanced. While above 0.6 or below 0.4 should be respectively classified as an advantage or a disadvantage for team one. The models are trained on the categorical cross-entropy between the predicted and the true class, using a one-hot encoding.

4.1.3 Reflection

The experiments for this proof of concept mainly consist of training various neural network models to classify a match as either balanced or favoring team 1 or team 2. A convolutional neural network that fuses the weapons and the map is demonstrated to be the best model, compared to several baselines. The details of the experiment and the results of the evaluation are described in Appendix A. The validation accuracy of the convolutional neural network (in Table A.1) is judged to be high enough to continue with this type of model for predicting metrics of gameplay. Moreover, a comparison between models with varying inputs demonstrates the benefits of fusing both input modalities. Yet, the results also provide grounds for changing the methodology of the experiment.

The insights provided by the proof of concept shaped the final version of the game environment, the game playing agents, the inputs and outputs of the surrogate model, the dataset, and the proposed pipeline. The major take aways that have been implemented are: reducing the number of players per team from three to one; using random generation instead of existing classes for training the model and increasing the sophistication of the level generation algorithm that creates the dataset.

A valid comment on this work is that it is unrealistic to assume that three players of a team will carry the same weapon, as strategic players will want to cover each other’s

[1]Whether a tile is occupied by an object is determined by performing a downwards raycast at the center of the tile.

weaknesses. However, it would be hard to gauge the model's gameplay predictions of multiple weapons per team, as this is a daunting task even for a human designer. To address this, the focus has shifted to a one-versus-one death match. This is a more realistic game scenario in terms of comparing two weapons and it simultaneously reduces the complexity of the gameplay predictions because there are less players involved. Additionally, it is arguably more worthwhile to understand the relations between these game facets and the gameplay of single players before moving on to the complexity of group behaviors.

Preliminary experiments on using fixed weapon classes for generation have shown that the model learns a simplified version of the weapon parameter space. When trying to improve the projected win rate of the shotgun, the model recommends a complete overhaul into a strong weapon, like the sniper rifle, instead of tuning a few parameters. Since the model has only seen five combinations of parameter values, it cannot be expected to extrapolate its predictions to the entire parameter space. To remedy this in the final pipeline, the model was shown the complete class parameter space by generating them randomly for the training data.

Placing blocks in an environment is as simple as level design gets. Though the levels in this game provided various degrees of cover in different areas, it is hard to attribute any meaning or purpose to these areas. Adding more purposeful patterns to the maps is likely to boost the performance of the model, as it can benefit more from its pattern recognition capabilities. Increasing the sophistication of the level design in the dataset includes the addition of powerups, as it is expected that changing the gameplay of a match via the map requires more than just a change in geometry. The ideal level generator would have intentional areas and design patterns. However, this seems to require the very model that we are trying to train. As such, the new and final approach we will follow in this thesis builds a dataset of random paths and areas, which are more likely to contain emergent patterns. Although it is hard to attribute any intentionality to a computational system, the goal is for the proposed surrogate-based generator to apply the learned patterns to certain areas of the map in order to change the gameplay towards the desired targets.

4.2 Final Concept

As motivated in Chapter 1, this thesis proposes a framework for orchestrating the generation of levels and rule sets by evaluating the joint artifacts with a model of gameplay. This model is trained via deep learning on gameplay logs to automatically extract features and find an approximate mapping between game levels, rules and gameplay outcomes. The computational model is able to identify interrelations between the three facets without needing to produce and test the outcome. The model can be used as a surrogate for actual game playthroughs, informing generators that adapt content of the contributing facets (in this case levels and rules). In this way, the framework can adapt the facets individually or orchestrate the generation of both. Section 4.2.1 has outlined the general pipeline for setting up the framework. This section will provide details on each of its elements, starting with a detailed view of the pipeline that is used as case study.

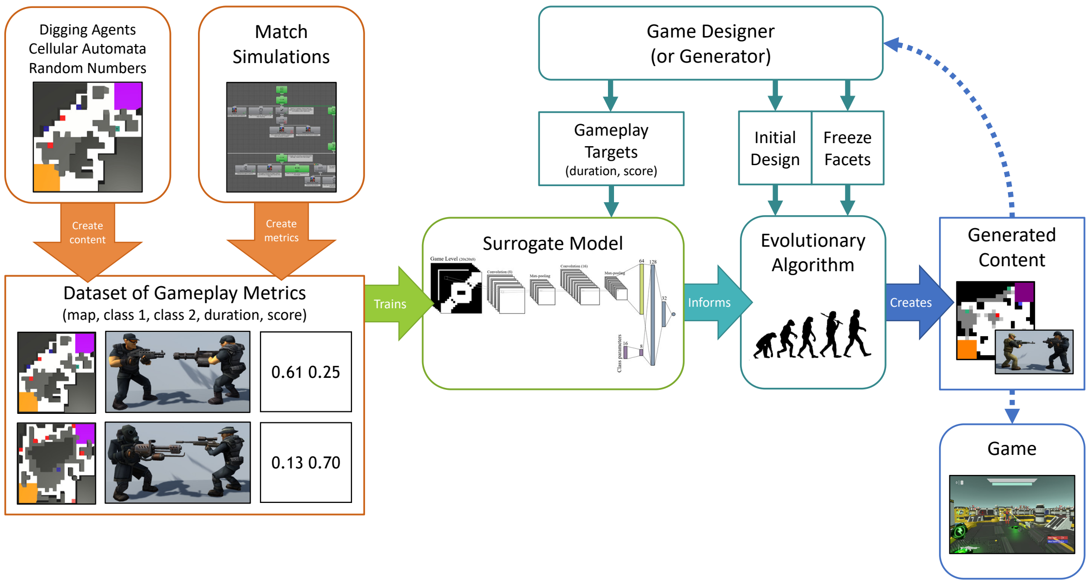


Figure 4.2: An overview of the pipeline to setup and use the framework. It consists of four phases: data collection (orange) , machine learning (green), facet orchestration (cyan) and utilization of the output (blue).

4.2.1 Pipeline

A surrogate-assisted procedural content generator requires a functioning model, which in turn requires a dataset. Setting up such a system thus consists of three phases: data collection, machine learning and facet orchestration (see Fig. 4.2). Additionally, there is a fourth phase that amounts to using the created content; either by putting it in the game or sending it to a designer for evaluation.

In the first phase, game content is generated via constructive algorithms. The map generator utilizes a combination of digging agents and cellular automata, while the weapon parameters are based on random number generation. Games are simulated using artificial agents in order to record gameplay outcomes for pairs of maps and classes. Chapter 5 describes the agents' behavior and the constructive algorithms used for creating the dataset.

The second phase consists of preprocessing the data to make it suitable for machine learning and training a surrogate model on this preprocessed dataset. Chapter 6 describes the preprocessing steps and compares various models for predicting the gameplay outcomes. Specifically, the models are trained to predict the kill ratio of player one and the duration of the game, which are defined in Section 4.2.3. Note that when data balancing is applied, a separate model is trained for each gameplay outcome.

With a trained model in place, phase three amounts to creating content by orchestrating the level and ruleset generators. The game facet orchestration process proposed in this thesis is bottom-up, i.e., there is no "composer" that manages multiple generators (Liapis et al., 2019b). The levels and rules are instead generated in parallel with an evolutionary algorithm by putting both facets into the same genotype and using unique genetic operators for each facet. With this approach, the generated artifacts can be adjusted to a specific instance of the other facet. This results in a higher chance of achieving the exact gameplay and promotes a diverse set of solutions, in contrast to two populations of generally good maps and rules. The details of facet orchestration can be found in the sections below.

The whole process can be automated by using the generators that created the dataset; thus taking a step towards complete game generation. Perhaps a more interesting challenge is to work towards a co-creative process with a human designer. The system is defined in such a way that it could be integrated into a mixed-initiative design tool. The designer can specify the gameplay targets, the initial design and which facets to adjust; levels, character classes or both. Indeed, the experiments in Chapter 7 are aimed at showing the performance of this system in a task of adapting an initial design towards designer specified gameplay.

The proposed framework is tested on a two-player competitive first-person shooter, which was chosen for its relatively straightforward gameplay of moving around and tagging opponents. The level facet is the map of the environment in which the battle takes place and includes the placement of powerups that affect the players' survivability. The rules facet is represented by the character classes of the two players. *Character classes* offer players different gameplay options (e.g., scouting or area control) and may have a different survivability and movement speed as well as signature weapons that balance their affordances. They are a common way to group game mechanics in multiplayer shooter games, such as *Team Fortress 2* (Valve, 2007). Indeed, *Team Fortress 2* will be used as a reference for character classes throughout this dissertation. The game and the representation of these two facets are described in the sections below.

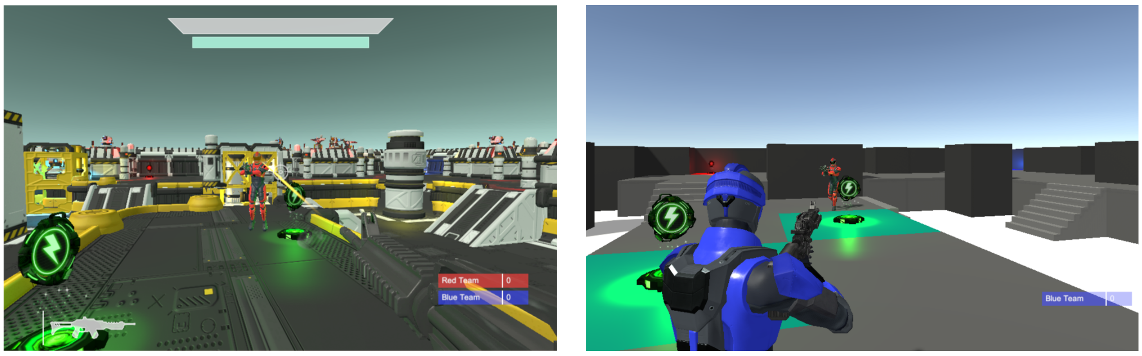
4.2.2 Description of the SuGAr game

The game used for the experiments is called SuGAr[2] and was developed specifically for this dissertation. SuGAr is implemented in the *Unity 2017* game engine, and is based on an existing toolkit (Opsive, 2012); see Fig. 4.3 for a screenshot. SuGAR is a first-person shooter with a single game mode, a deathmatch between two players. The goal of a deathmatch is for one player[3] to kill the other player's avatar more times than vice versa. A session in a deathmatch game finishes usually after a specific time has elapsed or after a specific number of kills has been reached by one or both players. The framework in this experiment considers matches to be complete when the combined number of kills on either side reaches a total of 20; a time limit of 600 seconds is also in place, but results from matches that timed out are ignored. The score balance is measured by the kill ratio (KR) of the player starting at the bottom-left of the map, i.e., the ratio of the kills scored by that player with respect to the total number of kills in the match.

In this game, the two competing players start in opposite sides of a game level (the characteristics of which are described below). When a player is killed, they respawn after five seconds at their original spawn location. The players are assumed to be on an equal skill level, and each of them controls an avatar that belongs to a specific character class. While game parameters for character classes and their weapons are described below, an important parameter is the hit points (HP): if any player drops to 0 HP, they are killed and the match comes closer to ending.

[2]SuGAr stands for Surrogate-based Generative Arena.

[3]Using arcade game terms, we identify "player" in this thesis as the virtual avatar, controlled by a human, which acts within the game. Evidently, no human players are hurt in these shooter games.



(a) Player perspective (b) Agent simulation

Figure 4.3: A screenshot of deathmatch playthrough in the 3D first-person shooter game SuGAr, depicting (a) the player's view and (b) a match simulated by agents in a block-version of the same level.

4.2.3 Facet Generation

Following the search-based PCG paradigm (Togelius et al., 2011), every generator in this thesis consists of a population of individuals and assigns a fitness score to each individual; the best individuals based on this fitness score are preferred for mutation and recombination, and their offspring replace the least fit individuals of the previous population. Particularly for the proposed framework, fitness is assigned based on the output of a surrogate model, using the parameters of the evolving individual as input. The goal of every EA in this framework is to adjust the initial designs (levels, classes, or both) to bring them closer to a designer-specified target gameplay outcome.

An important outcome of a playthrough is the winner of the match and the degree of winning; both are measured via a *kill ratio* (KR) as the number of kills of player 1 divided by the total number of kills of both players (i.e., 20 in these simulations). If $KR \approx 0.5$ then the matchup was balanced, while KR near 1 shows a clear advantage for player 1 and KR near 0 show a clear advantage for player 2. This can be used to tune the *fairness* of a game, i.e., whether both players have an equal chance of winning at the start of the match (see Section 2.1). The duration of the match (t) is another intuitive metric, as a designer may wish for some matches to have a certain approximate duration. If a game is too short, players might not have the time to develop meaningful strategies, while a game risks boring the players if it goes on for too long. This is an important feature of gameplay that often requires tuning (Schell, 2008, Ch. 11).

The simplest approach to assess proximity to a target gameplay outcome is to aggregate the distance between the outcomes of the current individual and the target values. For the single-objective evolutionary algorithm (SO-EA), the goal is to minimize the mean squared error between the model's predictions for individual m and the desired outcomes:

$$F(\boldsymbol{m}) = \frac{1}{2}\big((p_t(\boldsymbol{m}) - d_t)^2 + (p_{KR}(\boldsymbol{m}) - d_{KR})^2\big) \tag{4.1}$$

where $\boldsymbol{m}$ is the individual being evaluated (i.e., a vector of class and/or level parameters); $\boldsymbol{p}(\boldsymbol{m}) = \{p_{KR}, p_t\}$ is the two-dimensional vector of gameplay outcomes as predicted by the surrogate model, i.e., predicted kill ratio (p_{KR}) and predicted duration (p_t); $\boldsymbol{d} = \{d_{KR}, d_t\}$

is the two-dimensional vector of the desired gameplay outcomes specified by the designer, i.e., desired kill ratio (d_{KR}) and desired duration (d_t).

The experiments in this thesis employ a generational evolutionary algorithm with a population size of 100. In each generation, the fittest 10% of the population is copied from the previous generation unchanged (applying elitism), while the remaining 90% is chosen via tournament selection of size 5, and then recombined and mutated. Each gene of each individual has a 20% chance of being mutated; each pair of individuals has a 80% chance of producing offspring via recombination. These values were chosen based on a grid search of at least 100 runs with each generator, based on overall best performance across facets. The mutation and recombination operators depend on the content generated and are described in Sections 4.2.4, 4.2.5 and 4.2.6.

The fitness function described in Eq. (4.1) treats both gameplay outcomes as equal and attempts to optimize them simultaneously. There is no guarantee that the two gameplay outcomes are not conflicting, however: for instance, longer matches may be less balanced (assuming long duration and balanced kill ratio are the target outcomes). In order to test the effect of combining both gameplay outcomes into a single target, the experiments will compare the described single-objective algorithm with a multi-objective evolutionary algorithm (MO-EA) that aims to minimize the two components of Eq. (4.1) separately:

$$F_{KR}(\boldsymbol{m}) = (p_{KR}(\boldsymbol{m}) - d_{KR})^2 \tag{4.2}$$

$$F_t(\boldsymbol{m}) = (p_t(\boldsymbol{m}) - d_t)^2 \tag{4.3}$$

The MO-EA attempts to minimize both objectives of Eq. (4.2) and (4.3), using the popular NSGA-II algorithm (Deb et al., 2002) for creating the next generation. NSGA-II maintains a diverse Pareto front of non-dominated individuals and performs a binary tournament selection based on non-domination ranking and distance to other individuals (see Section 3.2.1). The other operators and hyperparameters (e.g., population size) of the MO-EA are the same as those of the SO-EA described above.

Characterization Metrics

Aside from the performance of the generative algorithm, we are interested in evaluating the created artifacts. The character classes will be evaluated in terms of Euclidean distance from the TF2 classes (Table 4.1). This is computed separately for the evolved classes of both players.

$$D(\boldsymbol{c_n}) = dist(\boldsymbol{\hat{c}_n}, \boldsymbol{\hat{o}_n}) \tag{4.4}$$

where $\boldsymbol{\hat{c}_n}$ is the normalized vector of class parameters of player $\boldsymbol{n}$ of the evolved match; $\boldsymbol{\hat{o}_n}$ is the normalized vector of class parameters for player $\boldsymbol{n}$ that was initially given to the generator; $dist(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. The evolved classes are also evaluated by the average difference per parameter compared to the initial two classes. In that case, we compute the *change* of each evolved class of player $\boldsymbol{n}$ with respect to the initial class vector of that player, using the same notation as above:

$$C(\boldsymbol{c_n}) = \boldsymbol{\hat{c}_n} - \boldsymbol{\hat{o}_n} \tag{4.5}$$

The evolved maps are characterized by the average change per tile type compared to the initial map. Each tile n in a map can have two types, a geometry type $(0, 1, W)$ and

Table 4.1: Normalized parameters for TF2 classes (raw values in parentheses).

	Heavy	Pyro[4]	Scout	Soldier	Sniper	Spy
HP	1.00 (300)	0.29 (175)	0.00 (125)	0.43 (200)	0.00 (125)	0.00 (125)
Speed	0.00 (77)	0.41 (100)	1.00 (133)	0.05 (80)	0.41 (100)	0.54 (107)
Damage	0.00 (36)	0.82 (130)	0.21 (60)	0.47 (90)	1.00 (150)	0.04 (40)
Accuracy	0.60 (60%)	0.30 (80%)	0.80 (80%)	1.00 (100%)	1.00 (100%)	0.95 (95%)
Clip size	1.00 (200)	1.00 (200)	0.03 (6)	0.02 (4)	0.12 (25)	0.03 (6)
Fire Rate	1.00 (40)	0.04 (2.3)	0.03 (1.6)	0.02 (1.25)	0.00 (0.67)	0.03 (1.72)
Bullets	0.00 (1)	1.00 (10)	1.00 (10)	0.00 (1)	0.00 (1)	0.00 (1)
Range	0.00 (short)	0.00 (short)	0.00 (short)	2.00 (long)	2.00 (long)	1.00 (med.)

pickup type (A, D, H). For a given tile type t in the set of all types T, we count the change in occurrence in the evolved map $\mathbf{m}$ as follows:

$$I(t,n) = \begin{cases} 1, & \text{if } t \in n \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

$$M(t,\mathbf{m},\mathbf{k}) = \sum\nolimits_{n \in N_{\mathbf{m}}} I(t,n) - \sum\nolimits_{n \in N_{\mathbf{k}}} I(t,n) \tag{4.7}$$

where $N_{\mathbf{m}}$ and $N_{\mathbf{k}}$ are respectively the set of tiles in $\mathbf{m}$ and its initial map $\mathbf{k}$.

4.2.4 Character Classes (Game Rules Facet)

Represention

The character class of each avatar is represented by eight parameters. Two of these parameters are specific to the character, i.e. hit points (*HP*) and movement speed (*Speed*), while the other six define the characteristics of their weapon: *Damage* per shot, *Accuracy* (i.e., the size of the cone in which bullets are fired), *Rate of Fire*, *Clip Size*, the number of *Bullets per Shot* and weapon *Range*. The damage per shot is the total damage a shot can do. In other words, adding a bullet to a shot will not increase the total damage, but decrease the damage of each single bullet. As noted earlier, the class parameters are inspired by *Team Fortress 2* (Valve, 2007), and based on parameter values obtained from the official *Team Fortress 2* (TF2) wiki (Valve, 2018). More specifically, the design space of the class parameters is based on the minimal and maximal values of a set of TF2 classes[5]. Special classes (the Spy, Medic and Engineer) were excluded from this set, because their role is not suitable for a one-vs-one deathmatch. The Demoman was excluded because his primary weapon is a grenade launcher, which is not implemented in *SuGAR*. The resulting minimum and maximum values that were used to determine the value range of each class parameter are described in Table 4.1.

Two additions were made to the TF2 parameter data; annotated based on the intuition of a TF2 veteran player. The first is *Accuracy*, which exists in the game as the spread of the bullets. It had to be annotated because this data was not found on the wiki page. The second is that of *Range*, to discern when AI agents should shoot. This encodes the human intuition of knowing at which distance to use a weapon. Usually this distance depends on the effective range of the weapon (based on, e.g., the bullet spread and damage drop over

[4]Pyro's damage over time was ignored.

[5]The values were extrapolated where it was deemed beneficial for the resulting design space.

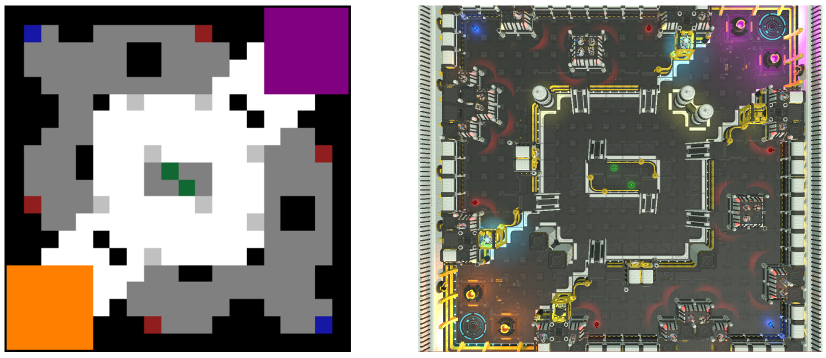

(a) Level sketch (b) Top-down view of a level

Figure 4.4: A level sketch and its in-game view. In Fig. 4.4a, orange and purple areas are the bases of player 1 and 2 respectively; black tiles are impassable walls; white and gray tiles are the ground and first floor; light gray tiles are stairs from the ground floor to the first floor; red tiles are healing locations, blue and turquoise tiles are armor and double damage powerups respectively.

distance) and the accuracy of the player. This parameter can take three values: *Short*, *Medium*, and *Long* range (represented as 0, 1 and 2 respectively). The effect of these values will be described in Section 5.1.2.

Several experiments make use of the class definitions of TF2 as initial seeds or as a comparison for the generated character classes. The parameter values of each class used in the experiments, as collected from (Valve, 2018), as well as the assigned range parameter are described in Table 4.1. The values used in the experiments are computed by normalizing these with min-max normalization (except *Range*, which remains 0, 1 or 2).

Variation Operators

The character class generator operates on the 16 class parameters described above (8 per player). Each parameter has an equal chance of being mutated. Except for *Range*, class parameters are mutated by adding a random number sampled from a normal distribution ($\mu = 0$, $\sigma = 0.1$). The mutation of *Range* randomly changes it into one of the other two possible values. Recombination is implemented as a standard one-point crossover.

4.2.5 Maps (Level Facet)

Represention

The levels in the game consist of a grid of 20×20 tiles. Each tile may be an impassable, tall wall or a passable tile on the ground or first floor. There are no tunnels or bridges. Stairs connect the ground floor with the first floor and players can drop from any ledge to go down from the first floor to the ground floor. Passable tiles can contain one of three types of powerups typically seen in shooter games: a *health pack* (increases HP up to a maximum), *armor* (offers additional HP which is depleted first) and a *damage boost* (player's bullets temporarily deal double damage). An example level is shown in Figure 4.4a: this simple representation will be used throughout the thesis, while the level can be transformed into a 3D detailed mesh as shown in Fig. 4.4b and Fig. 4.3. The spawn point of the first player (P1) is always in the bottom left corner, while the second player (P2) always spawns in the top right corner.

Variation Operators

The map generator of the proposed framework employs a hierarchical representation of two layers. While the level consists of 20×20 tiles, the gene represents this as a 4×4 grid of cells that each contain 5×5 tiles (see Fig. 4.5).

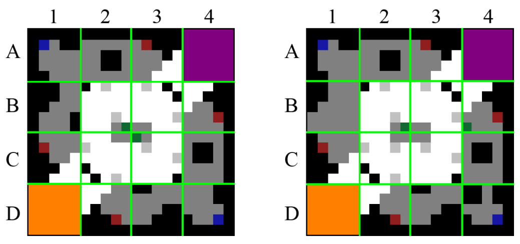
The upper layer acts as the set of genes for recombination and mutation, while the lower layer contains the actual pixel values. Recombination is implemented by randomly picking a cell from either parent at each position of the cell grid. When applying mutation, each cell has an equal chance of being mutated by one of the following variants:

- *Move Pickup:* If a cell contains one or more pickups, one of these pickups will be moved to a random location in a neighboring cell.
- *Grow Cell:* One of the following operators are chosen at random: either all ground tiles that are adjacent to a first floor tile transform into first floor tiles, or all first floor tiles adjacent to second floor tile transform into second floor tiles.
- *Erode Cell:* Opposite of *Grow Cell*, either first floor tiles adjacent to ground tiles are transformed into ground tiles or second floor tiles adjacent to first floor tiles are transformed into first floor tiles.
- *Place Stairs:* Add a stair to a random ground floor tile which is adjacent to only one first floor tile (remaining adjacent tiles must be empty ground floor tiles.
- *Place Block:* A 3×3 block of first floor tiles and a stair is created if there is enough space on the ground floor. Any pickups in this area are moved to the first floor.
- *Dig Hole:* Within a 5×5 block of first floor tiles, the central 3×3 first floor tiles are transformed into ground tiles with a stair directed inwards. Any pickups in this area are moved to the ground floor.

A mutation example is shown in Fig. 4.5. If a mutation is not applicable, (e.g., if a cell does not contain pickups for the first variant), the algorithm tries another mutation variant until either a mutation is applied or all mutations have been tried.

After mutation and recombination, each map is analyzed in terms of traversability in order to prevent infeasible maps from being created. The following constraints are enforced: (a) bases should always be reachable from ground floor tiles, (b) each pickup must be reachable, (c) each first floor tile should be connected to at least one stair, (d) there should be no holes in an area of first floor tiles without a stair to climb out of the hole and (e) a stair should always lead to a first floor. A naive constructive algorithm repairs unreachable areas and the stair placements (without changing players' bases). If this is impossible, the individual receives a fitness penalty equal to the number of tiles that violate these constraints. Note that the fitness functions described in Section 4.2.3 stay within the $0 - 1$ range, so a map that violates these constraints will always have a lower fitness[6] than a valid map.

[6]Remember, we are trying to minimize the fitness value.



(a) Parent (b) Offspring

Figure 4.5: An example of a level and its offspring when only mutation is applied. The levels are broken into cells: in cell B1 the *Erode Cell* mutation removed some walls; in C2 the *Place Stairs* mutation added a stair; in C3 the *Move Powerup* mutation moved a double damage powerup to B4; in D4 the *Place Block* mutation has added a 3×3 'chunk' of walls.

4.2.6 Combining Classes and Maps

When generating both facets at the same time, the classes and the level are treated as two components of one individual's gene. Mutation has a 50% chance of applying the respective mutation operations to either the class or the level component. Recombination is implemented by swapping the level component between the two parents, resulting in two offspring with the classes of one parent and the level of the other parent.

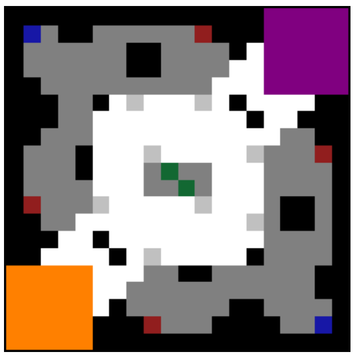
4.2.7 Converting to a Playable Game

Once the map and the classes are generated, the resulting data structure is parsed by *SuGAr*. The class parameters are converted back into the original, unnormalized range of Table 4.1. There are two versions for converting the map into a fully playable, three dimensional level (see Fig. 4.3). One is a three dimensional cube version of the map sketch, which is used for the agent simulations during data collection and performance evaluation. The cube level is created by placing cubes with the same colors as the map sketches, as well as stairs and powerup meshes, at their respective location.

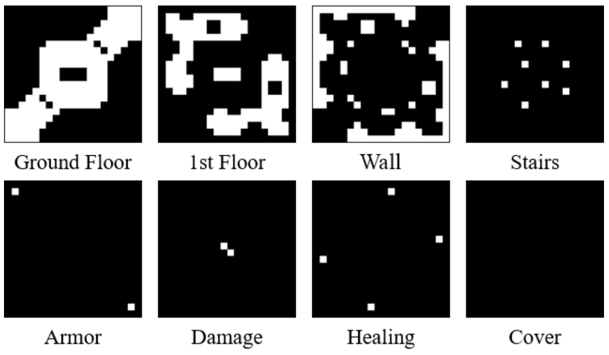
The other version is meant for human play and uses 3D meshes for all elements of the level. This is created by matching the two dimensional map representation to a list of predefined meshes covering every possible tile situation. These meshes are designed a priori by a human designer and for some tile situations there are multiple mesh options for increased map variability. Since meshes are more odd-shaped than blocks, they might cover more space or leave holes that the cubes did not have. The meshes are designed to take approximately the same space as the cubes and to block sight lines in a way that a human cannot take advantage of the space between the meshes.

4.2.8 Model Input

A level is a 20×20 grid of tiles. Each of these tiles can take one of 10 values, a combination of the elevation level of the tile and which pickup it contains, if any. This representation is fed to the models as a variation on the one-hot encoding. The result is a $20 \times 20 \times 8$ binary data structure, where each of the eight layers indicates a tile type and a one in that layer indicates the presence of that type at that location in the 20×20 grid. Any location in the grid can have at most two ones, e.g., a ground floor with a health pack. Figure 4.6 shows a level sketch and the resulting eight binary layers. This representation leverages



(a) Level sketch



(b) CNN inputs

Figure 4.6: A level sketch and its transformation into CNN inputs.

the capability of convolutional layers to process multiple channels of a 2D image, typically RGB-values. Linear regression, perceptrons and multi-layer perceptrons on the other hand, can only receive input vectors. They receive as input a flattened version of the same values as the CNN, i.e., a 3200×1 binary vector.

During the development of the model, we have experimented with adding extra information to the model input. One such example is giving pickups additional influence to the area around each pickup, or increasing its saliency if you will. This has been implemented in (Karavolos et al., 2017) by giving the (3×3) Moore neighborhoods of pickups the same "activation" value as the pickup, provided that they are passable tiles. Follow-up experiments included giving this neighborhood $0.5\times$ or $0.25\times$ the normal activation as the pick-up. None of these representations yielded significant changes in the accuracy of the model.

Another type of potentially useful information for the model is that of direction. The players always spawn at the same spot on the bottom-left or top-right corner of the map. Paths or pickups that are close to either corner can therefore be considered as "controlled" by either player. This human design intuition is for example captured by the general fitness functions of Liapis et al. (2013c). In an earlier stage of this work, a left-to-right color gradient was added to the map representation of Section 4.1.2. Yet, this did not result in any performance increase of the model. Based on the positive results in object detection and generative modeling (Liu et al., 2018), we tried an extension of this concept: adding a gradient from left to right and from top to bottom as additional channels to every feature map. Each convolution operation is then conditioned on where it is on the map, because it is multiplied by the coordinates of each pixel. Since this can be applied independent of input or domain, Liu et al. (2018) have proposed to integrate this into the convolution operation and call it *CoordConv*. In the best case, you get specialized activations for specific parts of the map. While in the worst case there is no effect, as the weights to the coordinates become zero. Although the domain of levels seems suitable for this operation, no significant change in performance was observed when using CoordConv instead of normal convolutions. This might mean that the locations of the features as obtained with the regular feature maps is enough for these modeling tasks. This resonates with the results obtained by Liu et al. (2018) on the MNIST benchmark for image classification, which did not produce a significantly different accuracy. Though perhaps the efficacy would change if the model is applied to gameplay outcomes that are more closely tied to location, such as heatmaps or entropy of heatmaps.

4.3 Summary

This chapter presented the concept of the proposed surrogate-based game facet orchestration framework and the pipeline to setup the system. The framework is tested on a competitive first-person shooter called *SuGAr*, which was developed specifically for this thesis. This chapter laid out the rules of this game and described how the generated facets, levels and rules, are represented in the game and in the framework. Moreover, we described how the feedback of the surrogate model can be used to orchestrate the generative process. The surrogate model learns to map levels and rules to gameplay via deep learning, which requires a corpus of labeled gameplay data from the game. The chapter describes how the maps are presented to the model and outlines the considerations for alternative representations.

The properties of the gameplay corpus and the procedure for creating it are detailed in Chapter 5, while considerations for the architecture of the model and the procedure for training the model are provided in Chapter 6. The proposed framework is designed to be a co-creative partner in a mixed-initiative design tool. As such, it is aimed at adapting an initial design towards desired gameplay values. While the evaluation function for the generated designs is described in this chapter, the framework's performance towards creating desirable content is evaluated in Chapter 7.

Chapter 5

Gameplay Data

Chapter 4 proposed a framework for multi-faceted content generation based on a model of gameplay. The case study for this framework is the generation of maps and character classes in *SuGAr*, a custom-made first-person shooter that is played between two players. The model in the framework guides the generation of both facets to a desirable gameplay outcome. The mapping between these facets is learned via deep learning, which requires a dataset of labeled data.

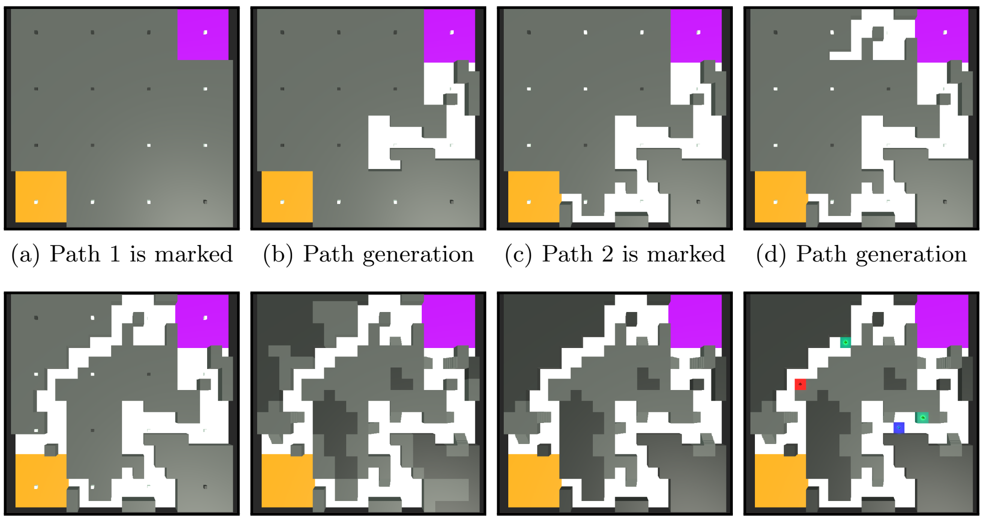
This chapter describes the properties of this dataset, which was obtained by simulating matches between artificial players. Section 5.1 describes how the data was obtained, including an outline of the agent behaviors and the constructive method used for level generation. The dataset is analyzed via the expressivity of its generators in Section 5.2. Focusing on the gameplay outcomes, Section 5.3 describes what the dataset looks like and how it was preprocessed for machine learning.

5.1 Data Collection

In order to obtain the rich and expressive dataset required for deep learning, 10^5 pairs of maps and classes were procedurally generated and evaluated in simulated matches between the artificial agents described in Section 5.1.2. A generated level and pair of character classes was simulated twice, swapping the classes of player 1 and player 2 in the second match. If either match did not result in a total of 20 kills within the time limit of 600 seconds, it was ignored and a new map-class combination was generated. This brings the total size of the dataset to $2 \cdot 10^5$ data points (matches). To generate the 8 character class parameters described in Section 4.2.4, a random real value within $[0, 1]$ was assigned to each parameter except for *Range*; which was randomly assigned a *Short*, *Medium* or *Long* value. With two competing classes, this results in 16 parameters. Level generation is based on a constructive approach to ensure the existence of enough meaningful patterns while avoiding the manual labor of handcrafting a dataset of this size.

5.1.1 Level Generation

To generate the levels for the corpus, we use a constructive approach which combines random digging agents and generative grammars (Shaker et al., 2016a). The final level consists of 20×20 tiles, broken into 16 cells of 5×5 tiles each. Two paths are created between the bottom-left and top-right corners of the level, first at the cell level and then at the tile level. The paths are initially split between ground-floor tiles and wall tiles, but first-floor tiles are added via a random transformation of some wall tiles followed by an iteration of cellular automata as per (Shaker et al., 2016a). Among all possible stair case locations (between a ground-floor and first-floor tile), a stair case is placed with a 20% chance. Each unreachable first floor tile is transformed into a wall, guaranteeing that all first-floor tiles are reachable. Finally, there is a 33% chance that a cell will have a pickup. This pickup is randomly assigned a type and randomly placed on a ground or first floor tile within that cell.

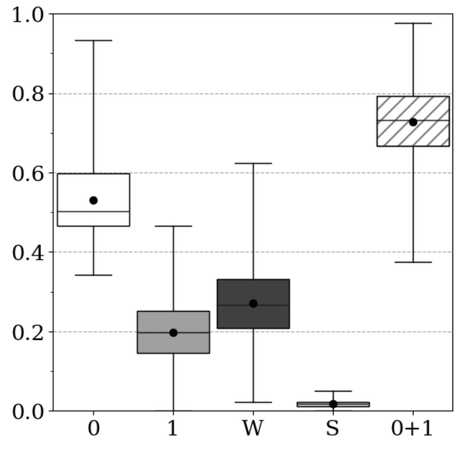

(a) Path 1 is marked (b) Path generation (c) Path 2 is marked (d) Path generation

(e) Ground finished (f) Floor 1 generated (g) Unreachable areas removed (h) Pickups placed

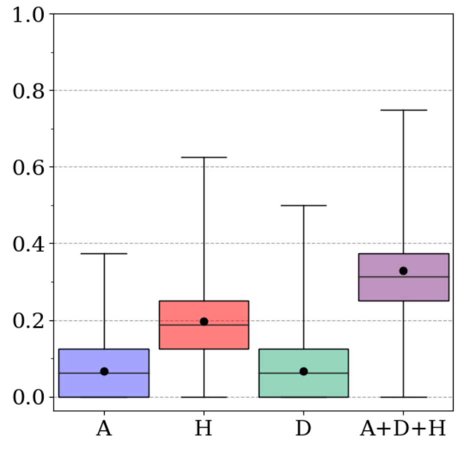
Figure 5.1: Stages of the 3D block level during constructive level generation.

5.1.2 Agent behavior

To simulate the gameplay, we use artificial agents controlled by behavior trees that were adapted from (Opsive, 2016). The agents are mostly reaction-based, they have no predictive model and very little memory. This was compensated by somewhat unrealistic perceptions compared to humans. The agents can detect the opponent through both sight and hearing. Sight is based on a cone of vision, which is blocked by objects. Hearing occurs in a wide circle around the agent that is not blocked by objects. Hearing is the encoding of a human player's prediction of where the opponent will appear based on intuition and map knowledge. Map knowledge is simulated by allowing the agents to perceive all pickups within a wide radius around them, regardless of walls. In descending order of priority, agents (a) search for healing items if their health is low, (b) pick up nearby powerups, (c) attack or chase the opponent if one is detected, and (d) search for distant powerups. When an opponent is detected they will be chased or shot, depending on the range of the weapon and whether they are in sight or not. If an opponent was detected but not currently in sight, the agent will go to the last known position and look around for 5 seconds before continuing with other behaviors. While their underlying logic is straightforward, the agents' in-game behavior was deemed complex enough to act as an approximation to human gameplay. In particular that of the more arcade-like FPS games, such as *Quake* (1996) or *Unreal Tournament III* (2007).



(a) Geometry



(b) Resources

Figure 5.2: Distribution of the ratio per level for each of the tile types: ground (0), first floor (1), walls (W), stairs (S), armor (A), health pack (H) and double damage (D). There are two aggregated values: '0+1', which encompasses all walkable tiles, and 'A+D+H', which combines all pickup tiles. Note that the pickups are normalized by the number of cells because there can only be one per cell. The black dot marks the mean and the error bars indicate the minimum and the maximum values of the data.

5.2 Expressivity Analysis

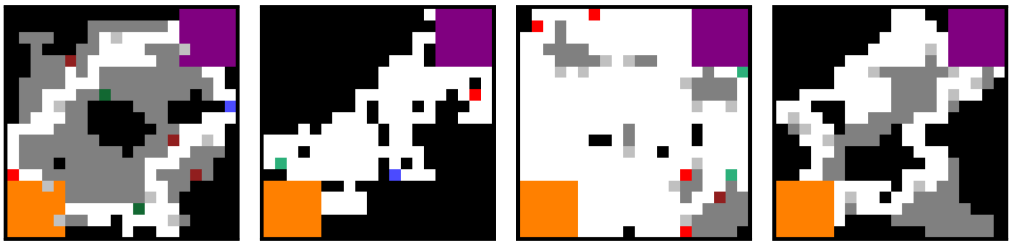
Ideally, the variation of content in the dataset is on the sweet spot of being rich enough for machine learning to model the whole design space (instead of a few straightforward correlations), while being constrained enough for common patterns to reoccur. In this section we will explore the expressivity of the level generator based on the composition of the levels in the dataset as well as metrics inspired on the notions of exploration and safety (Björk and Holopainen, 2004). The former gives us more insight into the layout of the levels, while the latter informs us about the gameplay balance between the two players.

5.2.1 Level Composition

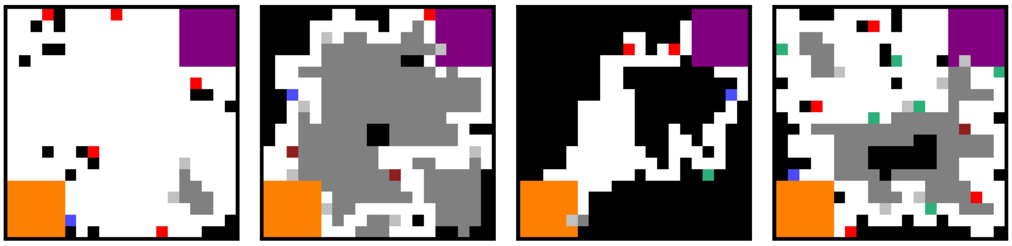
The most straightforward way to analyze a set of levels is to describe them in terms of their building blocks, in this case the tiles of the map sketches. The distributions of the different tile types are shown in Fig 5.2, while Fig 5.3 depicts several maps at the edges of these distributions. On average, approximately 70% of a level is traversable. The ground floor ratio ranges from 34-93%, while the first floor ratio ranges from 0-47%. The pickups occur in 33% of the cells as expected. However, Fig. 5.2b shows that the health packs on average occur almost three times as often as the other pickups. This is an unintended effect of an older version of the level generator that treated the health pack (a normal pickup) as a different entity than the armor and the double damage pickups (both powerups)[1]. The level generator allows the game designer to set a different spawn rate and to switch off the exclusivity per cell of the health pack with regards to the other pickups. The increased

[1]which in itself is a result of drawing inspiration from multiple shooter games with different types of pickups, e.g., some games even consider weapons to be pickups, while other games only have health pack pickups.

health pack frequency is a result of the generator spawning the health packs before the armor and double damage powerups.

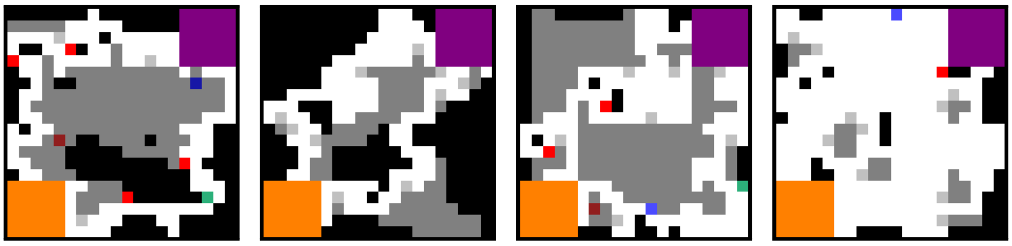


(a) Ground (34%) (b) First Floor (0%) (c) Walls (2%) (d) Pickups (0%)

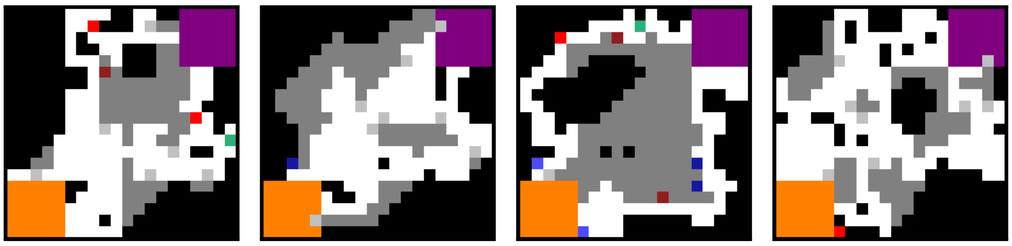


(e) Ground (93%) (f) First Floor (47%) (g) Walls (62%) (h) Pickups (75%)

Figure 5.3: Maps in the dataset that display extremes in tile ratios. The top row (a-d) minimizes the presence of the mentioned tiles, whereas the bottom row (e-f) maximizes their presence.

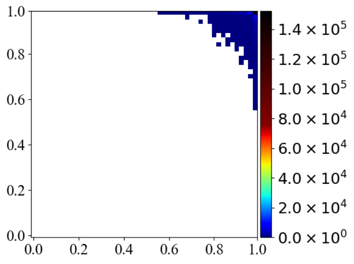


(a) $f_e(B_1, B_2) = 0.56$ (b) $f_e(B_1, P) = 0$ (c) $f_a(B_1) = 0.10$ (d) $f_s(B_1, P) = 0$

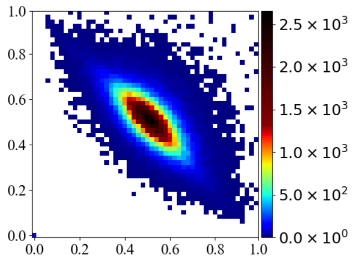


(e) $f_e(B_1, B_2) = 1$ (f) $f_e(B_1, P) = 1$ (g) $f_a(B_1) = 0.63$ (h) $f_s(B_1, P) = 0.75$

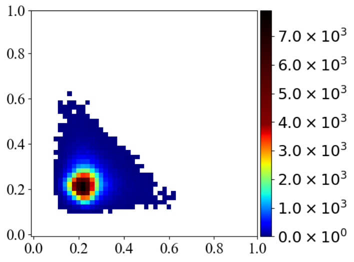
Figure 5.4: Maps in the dataset that display extremes in the map metrics. The top row (a-d) minimizes the metric, whereas the bottom row (e-f) maximizes it. B_1 and B_2 are the set of tiles in the bases of player 1 and 2 respectively. P is the set of pickup tiles of any type.



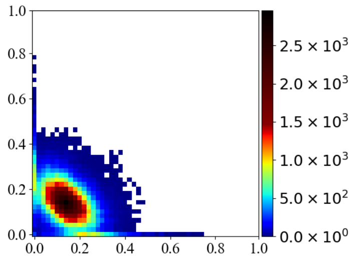
(a) $e(b_1, b_2)$ and $e(b_2, b_1)$



(b) $e(b_1, p)$ and $e(b_2, p)$



(c) $a(b_1)$ and $a(b_2)$



(d) $s(b_1, p)$ and $s(b_2, p)$

Figure 5.5: Expressive range of the level generator based on game level metrics. The x-axis and the y-axis respectively show the values for the bases of both players (b_1 and b_2). The variable p is resolved by any type of pickup tile.

5.2.2 Game Level Metrics

In order to analyze the design of the generated levels, we computed a set of metrics based on the general game level metrics as defined by (Liapis et al., 2013c). There are three groups of metrics, based on the following definitions:

Exploration, as $f_e(S_N, S_M)$, is based on a flood-fill algorithm that starts at a tile in the set S_N and counts the number of passable tiles that were filled until a tile of set S_M was encountered. In this dissertation, the exploration distance as well as the shortest distance are computed between the players' bases and between the bases and the pickups.

Area Control, as $f_a(S_N)$, evaluates the number of passable tiles much closer to one tile the set S_N than other tiles in the same set. This is computed both for the players' bases as well as the different types of pickups.

Safety (or *strategic resource control*), as $f_s(S_N, S_M)$, evaluates whether tiles of set S_N are significantly closer to tiles in set S_M than other tiles in the set S_N. Within the context of this game, it is used to compute whether pickups are much closer to the players' bases than other pickups and whether the base of one player is generally much closer to the pickups than that of the other.

The specific metrics and their correlations with the gameplay outcomes are described in Appendix B. Fig. 5.5 depicts the distribution of the maps in the dataset based on a selection of these metrics, while Fig. 5.4 shows maps on the edges of these distributions. The exploration value between the bases of both players (Fig. 5.5a) is generally quite high (80%-100% of the map) and has little spread. This reflects the fact that the bases are always placed in the opposite corners of the map. Lower exploration values typically occur due to non-traversable areas. Large asymmetries occur when one player does not have access to an elevated area, e.g., player one in Fig. 5.4a will reach the base of player two before exploring the first floor. The exploration values from either base to a pickup on the other hand, show a wide spread (Fig. 5.5b), indicating a diverse placement of pickups in the map. The majority of the maps have values around 0.5 for both players, resulting from maps with pickups in the center of the map as well as maps that have pickups close to both bases (where high and low exploration values cancel each other out). The edges of the range show that there are also maps that only have pickups close to one of the bases, giving the model opportunity to learn about uneven resource placement. Note that there is a set of maps with zero exploration to pickups for both players. These maps do not have any pickups, as can be seen from Fig. 5.4b.

The area control for both players is symmetric, though big differences between players do occur. Fig. 5.5c indicates that a large majority of the maps have a value around 0.25. This means that players typically only control their own corner, which is a desirable average. Large deviations seem to occur mostly because of stair placement and thus access to the first floor. For example in Fig. 5.4c, both large first floor areas are controlled by player two because the stairs are very close to their base. The high area control for player one in Fig. 5.4g has a similar cause. The only stair in the map is next to their base, giving them control over the entire first floor area.

Like the previous metrics, the distribution of resource control is symmetric and the majority of the maps have a balanced resource control with a value of 0.2 for both players. However, Fig. 5.5d shows that there are also large discrepancies between players. In many maps one of the players does not control any of the pickups; as showcased in both Fig. 5.4d and Fig. 5.4h. Although not desirable from a balanced map design perspective, it does offer the model data to learn about unbalanced maps.

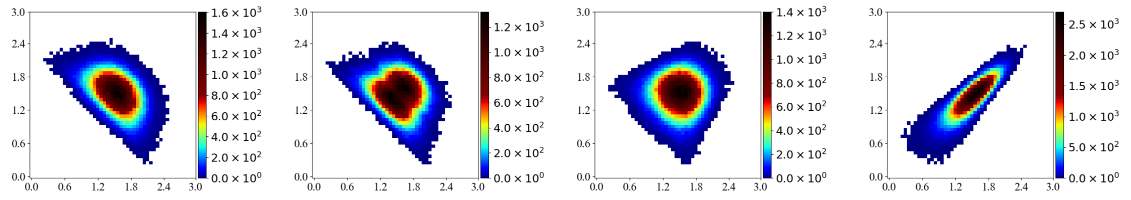

(a) Heavy and Sniper (b) Heavy and Scout (c) Sniper and Scout (d) Sniper and Soldier

Figure 5.6: Expressivity of the class generator based on the distances to TF2 classes. The class names indicate the reference classes for the distance metric on the x-axis and y-axis, respectively.

5.2.3 Classes

The distribution of the generated classes is explored by computing the Euclidean distance between each of the 10^5 generated classes to the archetypical TF2 classes (using Eq.(4.4)). The maximum distance between any two classes is obtained when each parameter has a maximal difference. The latter is one due to normalization, which makes the maximal distance $\sqrt{8} \approx 2.83$. Figure 5.6 depicts the comparison of several of these distances. We can immediately see that the distributions are quite symmetric. Fig. 5.6a shows that the generated weapons can either be close to the Heavy or close to the Sniper, but not both at the same time. The majority of the generated classes have a medium distance to both the Heavy and the Sniper. A similar distribution, though with more large distances to both, is observed when comparing the classes to the Sniper and the Scout. On the other hand, in

Fig. 5.6d we can see a very different distribution. Typically, when a generated class is close to the Sniper, it is also close to the Solder. As such, it is difficult to generate classes that are close to the Sniper, but not to the Soldier.

5.3 Properties of the dataset

Two gameplay outcomes were computed from each simulated match: the kill ratio (KR) of the first player (P1) with respect to the total number of kills, and the duration of the match in seconds. The distribution of these two gameplay metrics is shown in Figure 5.7. Match durations range from 150 to 600 seconds, broadly following a skewed bell curve. Most matches lasted approximately 300 seconds (mean of 323, standard deviation of 80), with few matches lasting less than 200 seconds (2%) or more than 500 seconds (4%). For the purpose of simplifying the target function for the model, game duration was normalized into the range of [0, 1] based on min-max normalization (from 150 to 600 sec). Kill ratio is almost uniformly distributed between matches with a clear advantage for P1 (KR=1), a clear disadvantage for P1 (KR=0) and balanced matches (KR=0.5). Despite the decreasing number of data points towards the edges, this distribution is an indication of a rich dataset with positive and negative examples of a balanced match.

Table 5.1: Significant Pearson correlations ($\alpha = 0.0005$) between the gameplay outcomes and the class parameters of the players P1 and P2. Strong correlations (≥ 0.10) are indicated in bold.

	Duration			KR		
	P1	P2	P1 - P2	P1	P2	P1 - P2
HP	0.05	0.04	0.01	**0.14**	**-0.14**	**0.20**
Speed						
Damage	**-0.10**	**-0.12**	0.01	**0.27**	**-0.27**	**0.38**
Accuracy	**-0.23**	**-0.27**	0.03	**0.56**	**-0.55**	**0.79**
Clip Size						
Fire Rate						
Bullets	-0.03	-0.03		0.02	-0.02	0.03
Range	-0.02	-0.01		0.01	-0.01	0.01

5.3.1 Class Parameters

In order to reveal any linear relationships between the class parameters and the gameplay outcomes, we compute the Pearson correlation between these values. The significant correlations are shown in Table 5.1. The accuracy, damage and health points seem to have the strongest relation with the gameplay outcomes. For duration, the accuracy and damage of both players have a negative correlation, while HP has a positive correlation Intuitively, these relations make sense; as the accuracy or damage of either player increases, it becomes easier to kill the opponent and the duration of the match decreases. On the other hand, having more HP makes players harder to kill, resulting in less frequent kills and longer matches.

For KR, the strongest correlation is again found with accuracy, followed by damage and HP. We find that the class parameters of player one correlate positively, while those of player two correlate negatively at a similar scale. This indicates that these properties are positive attributes; having more of it increases the score of that player. The symmetry indicates that it is perhaps the difference in parameter values that determines the power balance between the two players. Indeed, subtracting the values of player two from player one yields stronger correlations than either alone. These strong correlations might explain why the baseline models in Chapter 6 perform unexpectedly well when predicting the KR value. In Section 6.7.2 we will see to which degree these class parameters are important for the surrogate model's predictions of specific matches.

Table 5.2: Significant Pearson correlations ($\alpha = 0.0005$) between game duration and tile ratios of ground-floor tiles (0), first-floor tiles (1), walls (W), stairs (S), armor powerups (A), double damage powerups (D) and healthpacks (H). The tile ratios are computed in various areas of the map: bottom left (BL), bottom right (BR), top left(TL), top right (TR), central (C) and the whole map (WM). Strong correlations (≥ 0.10) are indicated in bold.

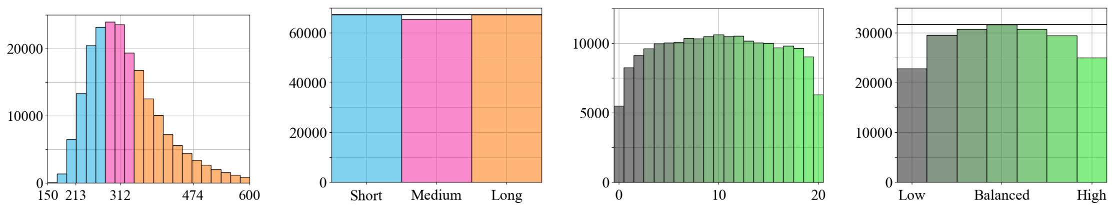
Tile	BL	BR	TL	TR	C	WM
0	**-0.24**	0.02	**0.10**	**-0.30**	**-0.48**	**-0.13**
1	**0.18**	0.05	-0.02	**0.24**	**0.35**	**0.17**
W	**0.14**	-0.06	-0.09	**0.15**	**0.33**	-0.02
A	0.05	**0.12**	0.08		-0.05	**0.13**
H	-0.06	0.03	0.09	-0.05	**-0.14**	
D	0.01	0.08	0.05	-0.04	**-0.10**	0.05
S	-0.07	0.02	0.03	-0.06	**-0.26**	-0.05

5.3.2 Map Features

To get more insight into the relation between features of the map and the two gameplay outcomes, we computed the Pearson correlation between these values and a set of 90 map features. This set of map features consists of the tile ratios of the map, the distances between the bases and the pickups, and the above mentioned *Safety*, *Exploration* and *Area Control* metrics of all pickups (separate and combined) with respect to both player's bases. The tile ratios are not only extracted from the map as a whole, but also from subareas of the map. The intuition behind this is that the composition of some areas, such as the center, have a larger predictive value than other areas. These subareas are 2×2 cells (a total of 100 tiles) in the corners of the map and the 2×2 central cells (see Fig. B.1).

Though many of the 90 map features have a significant correlation with KR, none of the correlations are larger than 0.10 and only 9 of them are larger than 0.05. This confirms that the score balance between the two players largely relies on the classes they use. As such, there is little value in analyzing these correlations in detail. The metrics with the strongest correlations are the average *Safety* value of the double damage powerups with respect to base 2 and base 1 (respectively $r = -0.09$ and $r = 0.08$). Although it seems intuitive that the placement of the damage bonus would affect the balance between the two players, it is surprising that this correlation is stronger than the average *Safety* value of all pickups combined (respectively $r = 0.05$ and $r = -0.05$ for base 1 and base 2). A table of all correlations is depicted in Appendix B.

The map features correlate much more strongly with game duration than with KR. The strongest correlations are found between duration and the ratio of geometry tiles in the central area (see Table 5.2). The ground and stair tiles are associated with short durations, while the first floor and walls are associated with long durations. Other areas with strong correlations between composition and duration are the bottom left and top right corner, where the players' bases are located. The observed associations are similar to the central area. The ground floors are associated with short durations, while the first floors and to some degree the walls are associated with longer durations. As can be expected, having cover around the spawn area increases the duration of the game, while too much open space leaves the newly spawned player vulnerable and shortens the match. Considering the whole map, the strongest correlation is found with the ratio of first-floor tiles. The weakest is that with the health packs, where no correlation was observed. On the other hand, the ratio of health packs in the center of the map has a relatively strong correlation. The latter seems a more intuitive relation between healthpacks and game duration. Perhaps the lack of correlation originates from a relatively stable number of generated health packs per map. More surprising is the fact that the number of health packs in the center have a *negative* correlation to duration. Indeed, one would expect that the presence of health packs extend the duration of the game. An explanation could be that health packs in the center of the map allow the winner of a fight to easily recover their health without having to search a remote area of the map, thus keeping the players relatively close to each other. The other pickups show a similar effect of reducing the duration if they are in the center while increasing the duration when counted over the whole map.



(a) Duration Distribution (b) Balanced Duration (c) P1 Kill Distribution (d) Balanced Kill Ratio

Figure 5.7: A visualization of the original gameplay outcomes (a,c) and the bins used for oversampling (b,d). In (b) and (d) the horizontal black line indicates the size of all bins after oversampling.

5.3.3 Balanced Data

The distribution of the duration targets is skewed towards the mean of the data. Performing machine learning on this might bias the model towards the mean. As such, a more uniformly distributed dataset for each outcome was obtained by oversampling the data with infrequent target values. The bins required for counting occurrences were created by artificially discretizing the outcome values in an ad-hoc manner. The duration targets (t) are normalized via min-max normalization and split into three equally sized bins with edges at $t = 0.28$ and $t = 0.43$. With this discretization, the first bin contains the lower third of the dataset's

t (shortest matches) and the third bin contains the upper third of the dataset's t (longest matches). After oversampling for balancing bins in t, the training set and validation set sizes are respectively 182.7×10^3 and 20.2×10^3 (see Fig. 5.7b). The KR targets have 20 possible values, which are split into 7 bins. Treating a difference of one kill as insignificant, the discretized space is based on treating KR=0.5 ± 0.05 as one bin, with equally sized bins up and down in the range of $[0, 1]$. After oversampling for balancing bins in KR, the new training set and validation set sizes are respectively 199.1×10^3 and 22.1×10^3 (see Fig. 5.7d).

5.4 Summary

This chapter has outlined the dataset that is used to obtain the surrogate model via machine learning. It describes the generative processes that were used to create the character classes and the levels, and elaborates on the expressive range of the generator. The generated levels are analyzed in terms of composition as well as several general map metrics introduced by (Liapis et al., 2013c), e.g., the *Safety* of resources and *Area Control* of the players' bases. Furthermore, it touches upon the agent behaviors that were used for simulating gameplay, and reports the distributions of the score and duration outcomes that were collected from the simulations as well has how the dataset was balanced for a reduced bias of the model.

As a stepping stone towards the mapping that is learned by the models in Chapter 6, this chapter addresses the relations between the intended inputs and outputs of the model by analyzing Pearson correlations between the character classes, the maps and the observed gameplay outcomes. This analysis shows that there are connections between all inputs and outputs. Although it seems that the class parameters correlate more strongly with score, while the map features correlate more strongly with game duration.

The next chapter explores which type of machine learning models are good at predicting the gameplay outcomes and whether both input dimensions are required for the best performance. In an attempt to explain the decisions of these black-box methods, Chapter 6 also applies probing algorithms to the best model in order to uncover how the features of specific inputs influence its decisions.

Chapter 6

Creating a Model of Gameplay

The main contribution of this thesis is a PCG framework that evaluates content by predicting aspects of gameplay with a computational model. As mentioned in Chapter 4, the selected model is a convolutional neural network (CNN). This chapter describes the architecture variations of that model, which have been applied as surrogate model in the content generation experiments in Chapter 7, and compares the selected model to various alternatives. The CNN architectures are described in Section 6.1, while the training procedure for all models in this chapter are outlined in Section 6.2. A description of the baselines and the primary comparison between the models can be found in Section 6.3. Throughout this chapter significant findings are based on a two-sided two-sample t-test with $alpha = 0.05$ unless otherwise specified. Findings of the CNNs that are significantly different from the multi-layer perceptron baseline are printed in bold in their respective tables.

The input representation of the models are detailed in Section 4.2.8 and the characteristics of the dataset as a whole have been described in Chapter 5. Nevertheless, it is worthwhile to repeat that each model receives two inputs: 16 class parameters and $20 \times 20 \times 8$ binary map features, which indicate 8 types of tiles on a 20 by 20 map. The baseline models can only receive vectors, so those models receive a flat vector of the same 3216 values. While the models generally receive both map and class features, Section 6.4 compares the use of a single facet input to having both inputs.

Throughout this thesis predicting gameplay outcomes is generally framed as a regression task, despite successful early experiments with classification (see Appendix A). Arguably, classification is an easier task for a model, because there are less options for predicting a category than for a floating point value. However, this property is a downside when using the output of the model for evolution. A category is a less fine-grained feedback value on the individuals and thus less of a gradient to follow for the evolutionary algorithm. Preliminary experiments confirmed this by showing hit-or-miss results. Either the initial population already had an individual with the desired classifications or the desired classification was not found in 100 runs with 100 individuals. Section 6.3 compares the various CNN architectures with three baselines, while framing gameplay prediction as a regression task. Classification results of the final CNN and the baselines are reported in Section 6.6 as an alternate way of comparing the models.

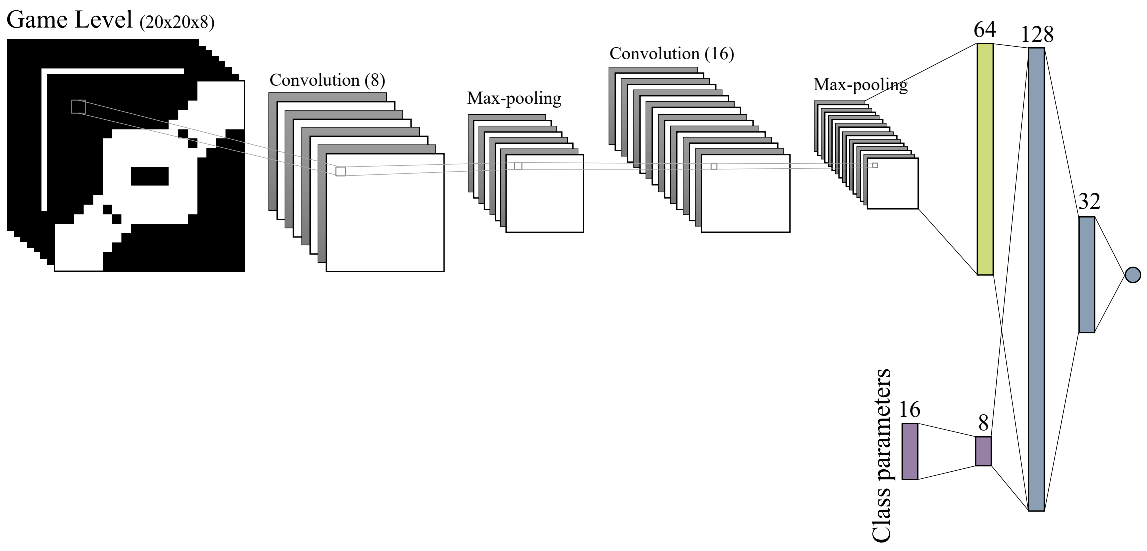


Figure 6.1: The architecture of CNN3, which is used for the facet orchestration experiments. The model can be seen as fusing two information streams into a two-layered decision maker.

6.1 Model Architecture

The surrogate model proposed in this thesis is based on the concept of having specialized feature extractors per facet and then feeding the joined features into a decision making architecture. The top-down view of the map is analyzed via convolutions, while the class parameters are passed through a normal fully-connected layer. The resulting features are flattened and concatenated into a single, flat feature vector, which is passed to one or more fully-connected layers and then an output layer. An example of this architecture can be found in Fig. 6.1.

Three variants of such an architecture are used in the experiments in this thesis, one for the level tuning experiments performed on the unbalanced dataset, one for the character class tuning experiments on the unbalanced dataset and one for the facet orchestration experiments on the balanced dataset. Each of these architectures was the best performing architecture on the dataset based on elaborate preliminary experiments at different stages of the research. The variations are fueled by shifting popular practices in the Deep Learning community, e.g., the increased popularity of batch normalization (Ioffe and Szegedy, 2015).

Each variant processes the 16 character class parameters the same way, by passing it to a fully-connected layer of 8 nodes, and each variant processes the level via two blocks of convolution and max-pooling. The main differences are layer and filter sizes and the non-linearity that is applied to the activations. The hyperparameters of these three variants are shown in Table 6.1.

Note that size of the output layer depends on the prediction task. For the regression task on the unbalanced dataset, one model was trained to predict both gameplay metrics, resulting in an output size of two. For the regression task on the balanced dataset and the classification task, a separate model was trained for each gameplay metric, resulting in an output size of one or the number of classes respectively.

Table 6.1: The hyper parameters of the CNN architecture variations used for character class tuning (CNN1), level tuning (CNN2) and facet orchestration (CNN3).

Hyperparameter	CNN 1	CNN 2	CNN 3
Activation function	ELU	ReLU	ELU
Batch normalization	no	no	yes
Convolution kernel size	5×5	3×3	5×5
Layer sizes, convolution	16, 32	8, 16	8, 16
Layer sizes, decision	128	32	128, 32
Learning rate decay	no	no	yes
Zero padding	yes	yes	no

6.2 Training Procedure

Each model was implemented in Keras (Chollet et al., 2015) and trained for up to 100 epochs on 90% of the data, while 10% was used for validation. Early stopping with a patience of 10 epochs was applied to prevent overfitting. All networks were trained with Adaptive Momentum Optimization (Kingma and Ba, 2015), or Adam, using the default parameters suggested in the paper ($\beta_1 = 0.9$, $\beta_2 = 0.999$) and a learning rate of $1e^{-4}$. Vanilla stochastic gradient descent can be very sensitive to learning rates that are either too large or too small, which can result in respectively skipping over the global optimum or getting stuck in a local optimum. Adam, on the other hand, is a variation of stochastic gradient descent that adjusts the size of the update per parameter based on a momentum term per parameter, which make it robust to variations in the learning rate. See Section 3.1.3 for more details.

The networks that were trained on the balanced dataset have an additional measure to improve the training process by making it more independent of the learning rate. Every three epochs without improvement on the validation set the learning rate was reduced by a factor 10, to a minimum of $1e^{-6}$. The intuition here is that the initial learning phase requires large steps to get to the most promising area of the search space and as training progresses, smaller steps are required to avoid skipping over the optimum. Consequently, when training on the balanced data set, the initial learning rate was set to $1e^{-3}$.

6.3 Model Selection

Both modeling duration (t) and kill ratio (KR) are regression tasks. Such tasks are typically evaluated via (a) the model's prediction error and (b) how much of the variance in the data is explained by the model. The former is computed by the mean absolute error, MAE_t and MAE_{KR} for duration and kill ratio, respectively. The latter is computed by the R^2 metric for these dimensions (R^2_t and R^2_{KR}). The typical value range of the R^2 metric is $[0, 1]$, where $R^2 = 0$ means that the predictions of the model are as good as predicting the mean and $R^2 = 1$ means that the variance of the predictions of the model have a perfect correlation with the variance of the ground truth values. In machine learning, this metric is also described as the 'goodness of fit' of the model. Negative R^2 values can occur, but indicate an unstable model; it would be preferable to predict the mean of the data in such a case.

Table 6.2: Number of trainable parameters per model.

Model	Parameters
Linear Regression	3,217
Perceptron	3,217
MLP (128 nodes)	411,905
CNN1	119,865
CNN2	15,009
CNN3	18,657

The CNNs will be compared to three more traditional statistical models: linear regression (LR), the perceptron (i.e., non-linear regression) and the multi-layer perceptron (MLP). Preliminary experiments with the MLP baseline included neural networks with up to two hidden layers and layer sizes between 2 and 1024 neurons. The MLP with a single hidden layer of 128 neurons performed best on validation sets of various datasets and targets. For the sake of brevity, this architecture is chosen to represent the MLP models.

These baseline models are considered simpler than the CNN, since these models existed before the CNN and rely only on normal multiplications and not on the relatively recent convolution operation (which was introduced to ANNs in (LeCun et al., 1998)). However, these statistical models can also be viewed from the perspective of trainable parameters. Models with less parameters generally require less data for training and are less prone to overfitting (Bishop, 2006). Despite being a baseline, the MLP with a single hidden layer actually has up to 27 times more trainable parameters than the CNNs, as can be seen in Table 6.2. As noted in Section 3.1.3, the parameter efficiency of the convolution operator is actually one of the major factors of the successes of CNNs.

6.3.1 Preliminary Experiment: Regression on the original dataset

As an initial experiment, the models are trained on both targets simultaneously. A task that most closely resembles how a surrogate model would ideally be used for facet orchestration, i.e., one model that predicts all gameplay outcomes. Indeed, the CNNs that are used for the experiments on generating a single facet that matches its context are trained on this task (see Section 7.2 and 7.3).

It is evident from Table 6.3 that most of the tested models struggle to find patterns between the input and the match duration ($0.49 < R^2_t < 0.59$). On the other hand, even simple models (such as linear regression and the perceptron) can fairly accurately predict the kill ratio ($0.82 < R^2_{KR} < 0.91$). This difference between accuracy of the two outputs is evident from R^2_t, as often the model can not explain the variance in the duration data (compared to high R^2_{KR} values). In general, the more complex models have both a lower error and a higher explained variance than the simpler ones. The CNN models can predict both of these outputs more accurately than the baseline models. All differences between the CNN2 and 3 and the MLP are significant with $p < 0.01$. Admittedly, the difference between CNN1 and the MLP is small and while the R^2_t is still significantly different ($p = 0.04$), the corresponding MAE_t is not ($p = 0.33$).

There is a likely explanation for the relatively good performance of the baselines, as there are significant Pearson correlations between the kill ratio of player one and 6 class

Table 6.3: Average Mean Absolute Error (MAE) and the R^2 values for duration (t) and kill ratio (KR) prediction on the initial validation set based on 10 runs.

Model	MAE_{KR}	MAE_t	R^2_{KR}	R^2_t
LR	$0.095 \pm 1 \cdot 10^{-5}$	$0.094 \pm 4 \cdot 10^{-5}$	$0.821 \pm 2 \cdot 10^{-5}$	$0.485 \pm 1 \cdot 10^{-4}$
Perceptron	$0.086 \pm 1 \cdot 10^{-5}$	$0.094 \pm 3 \cdot 10^{-5}$	$0.845 \pm 2 \cdot 10^{-5}$	$0.486 \pm 1 \cdot 10^{-4}$
MLP (128)	$0.074 \pm 9 \cdot 10^{-4}$	$0.090 \pm 4 \cdot 10^{-4}$	0.883 ± 0.003	0.517 ± 0.005
CNN1	$\mathbf{0.071} \pm 7 \cdot 10^{-4}$	$0.090 \pm 3 \cdot 10^{-4}$	$\mathbf{0.895} \pm 0.002$	$\mathbf{0.524} \pm 0.002$
CNN2	$\mathbf{0.068} \pm 7 \cdot 10^{-4}$	$\mathbf{0.084} \pm 3 \cdot 10^{-4}$	$\mathbf{0.901} \pm 0.002$	$\mathbf{0.577} \pm 0.001$
CNN3	$\mathbf{0.065} \pm 5 \cdot 10^{-4}$	$\mathbf{0.083} \pm 3 \cdot 10^{-4}$	$\mathbf{0.911} \pm 0.001$	$\mathbf{0.589} \pm 0.002$

parameters (accuracy, damage, and hit points of both players), and between duration and accuracy of both players (see Section 5.3). Perhaps the predictions are mostly based on the class parameters and only slightly moderated by the presence of certain tile types, such as the powerups. Another factor might be the skewed distribution of game duration in the dataset, which may allow the model to predict values near the mean of the data without a sufficient penalty. This could explain the the simultaneously low error (MAE_t) and low explainability (R^2_t).

Based on the results of this preliminary test, the best model is CNN3; it has a significantly better prediction of both targets than all of the other models ($p < 0.01$). Its KR predictions have a near perfect correlation with the ground truth, which is remarkable given the stochasticity of the data. This model has slightly more parameters than CNN2, but both are relatively small and the small but significant performance increase seems worth the trade-off. CNN3 is therefore the main candidate to serve as a model for orchestration. The next section explores whether this changes when trained on the balanced dataset.

6.3.2 Regression on the balanced dataset

In this section, the models are trained on each gameplay outcome separately, using the datasets that are balanced for each target (see Section 5.3.3). The results in Table 6.4 show that the performance of the models is quite similar to those that predict both values simultaneously. All models can accurately predict the kill ratio ($0.83 < R^2_{KR} < 0.91$), whereas they seem to struggle with duration ($0.49 < R^2_t < 0.61$). The more complex models have both a lower error and a higher explained variance than the simpler ones. The CNNs all differ significantly from the MLP baseline ($p < 0.01$), even CNN1. Interestingly, the errors of CNN1 and CNN2 are not significantly different on this task, unlike in the preliminary experiment. Despite this, the R^2_t of CNN2 is significantly higher ($p < 0.01$). Similar to the previous experiment, CNN3 significantly improves upon all of the other models with regards to all metrics. Interesting in this experiment is the MLP. Its performance on kill ratio has somewhat decreased, but it has improved in terms of duration prediction. Moreover, the seemingly small difference in MAE_t has a big impact on how well it models duration, as reflected by the large difference in R^2_t. Perhaps this is an indication that it is hard for the models to predict when the matches will be extremely long or short and that data balancing could still be improved.

In both experiments we can see the same trends. The small difference between the results in Table 6.3 and Table 6.4 is more likely to originate from the variation in data than

Table 6.4: Average Mean Absolute Error (MAE) and the R^2 values for duration (t) and kill ratio (KR) prediction on the balanced validation set based on 10 runs.

Model	MAE_{KR}	MAE_t	R^2_{KR}	R^2_t
LR	$0.096 \pm 1 \cdot 10^{-5}$	$0.094 \pm 3 \cdot 10^{-5}$	$0.832 \pm 3 \cdot 10^{-5}$	$0.491 \pm 1 \cdot 10^{-4}$
Perceptron	$0.086 \pm 3 \cdot 10^{-5}$	$0.094 \pm 2 \cdot 10^{-5}$	$0.856 \pm 5 \cdot 10^{-5}$	$0.493 \pm 1 \cdot 10^{-4}$
MLP (128)	$0.085 \pm 6 \cdot 10^{-4}$	$0.084 \pm 3 \cdot 10^{-4}$	0.856 ± 0.002	0.579 ± 0.002
CNN1	$\mathbf{0.069} \pm 4 \cdot 10^{-4}$	$\mathbf{0.084} \pm 3 \cdot 10^{-4}$	$\mathbf{0.905} \pm 0.001$	$\mathbf{0.587} \pm 0.002$
CNN2	$\mathbf{0.069} \pm 8 \cdot 10^{-4}$	$\mathbf{0.083} \pm 4 \cdot 10^{-4}$	$\mathbf{0.907} \pm 0.002$	$\mathbf{0.596} \pm 0.002$
CNN3	$\mathbf{0.067} \pm 0.001$	$\mathbf{0.082} \pm 2 \cdot 10^{-4}$	$\mathbf{0.912} \pm 0.002$	$\mathbf{0.606} \pm 0.001$

Table 6.5: Validation results when using only the maps as inputs. The results are an average of 10 independent runs and include the 95% confidence intervals.

Model	MAE_{KR}	MAE_t	R^2_{KR}	R^2_t
LR	$0.261 \pm 3 \cdot 10^{-5}$	$0.114 \pm 3 \cdot 10^{-5}$	$-0.015 \pm 2 \cdot 10^{-4}$	$0.316 \pm 1 \cdot 10^{-4}$
Perceptron	$0.261 \pm 5 \cdot 10^{-5}$	$0.114 \pm 2 \cdot 10^{-5}$	$-0.014 \pm 4 \cdot 10^{-4}$	$0.317 \pm 1 \cdot 10^{-5}$
MLP (128)	$0.259 \pm 1 \cdot 10^{-4}$	$0.105 \pm 8 \cdot 10^{-4}$	0.003 ± 0.001	0.391 ± 0.004
CNN3	$\mathbf{0.258} \pm 5 \cdot 10^{-5}$	$0.110 \pm 1 \cdot 10^{-4}$	$\mathbf{0.009} \pm 2 \cdot 10^{-4}$	0.355 ± 0.002

Table 6.6: Validation results when using only the classes as inputs. The results are an average of 10 independent runs and include the 95% confidence intervals.

Model	MAE_{KR}	MAE_t	R^2_{KR}	R^2_t
LR	$0.098 \pm 5 \cdot 10^{-6}$	$0.124 \pm 7 \cdot 10^{-5}$	$0.825 \pm 1 \cdot 10^{-5}$	$0.165 \pm 1 \cdot 10^{-5}$
Perceptron	$0.088 \pm 2 \cdot 10^{-5}$	$0.124 \pm 3 \cdot 10^{-5}$	$0.848 \pm 1 \cdot 10^{-5}$	$0.168 \pm 1 \cdot 10^{-4}$
MLP (128)	$0.088 \pm 1 \cdot 10^{-5}$	$0.124 \pm 5 \cdot 10^{-5}$	$0.848 \pm 2 \cdot 10^{-5}$	$0.169 \pm 1 \cdot 10^{-4}$
CNN3	$\mathbf{0.074} \pm 8 \cdot 10^{-4}$	$\mathbf{0.122} \pm 0.0002$	$\mathbf{0.896} \pm 0.0016$	$\mathbf{0.201} \pm 3 \cdot 10^{-4}$

from disparate learning dynamics. As such, we can conclude that using one model to predict both values does not seem to positively or negatively affect performance on either target. In both experiments CNN3 is the best model for both targets. There is a fair margin between performance of this model and the baselines. Although the margin with the MLP is not as large as with the other baselines, in both experiments the CNN3 is significantly better in predicting either target. Considering that CNN3 requires 22 times less parameters than the MLP and has significantly lower errors than the other CNNs, this is still the model of choice. CNN3 will be the main model for the rest of the chapter and for the main orchestration experiments in Chapter 7.

6.4 Sensitivity to input dimensions

The proof of concept experiment in Section 4.1 explored the effect of predicting the balance of a match based on a single input facet. The results showed that both the CNN and the MLP obtained the best accuracy when receiving both inputs. When these models received only classes, their performance degraded slightly, but not severely. However, predictions based only on maps were marginally better than random guessing. With a new game framework, a second gameplay dimension and a regression task instead of classification, it would be interesting to see whether the same effects emerge. The models are trained on the balanced data set according to the procedure described in Section 6.2. In order to test the effect of having just one facet as input, the second facet is replaced by an array of zeros with the same dimensions. The results of using only maps or only classes as input are described in Table 6.5 and Table 6.6, respectively.

Table 6.6 shows that classes are very important for predicting KR, as R^2_{KR} is only approximately 0.01 lower than that of the respective models with both inputs. Despite this seemingly small difference, all R^2 metrics are significantly lower (and the absolute errors significantly higher) compared to the respective models trained with both inputs. Note that in this case all models are effectively MLPs. Yet, the CNN3 model still has a significantly better performance than the baseline models. Moreover, the KR predictions of this "CNN3" model are significantly better than the MLP that receives both inputs. This is unexpected, as preliminary experiments on MLPs showed that models with two hidden layers did not have lower errors than those with a single hidden layer.

The ratio of the contribution of the class parameters to KR prediction might seem surprising at first. On the other hand, the maps were not generated with a purposeful bias. So it makes sense that in general the difference in, e.g., damage, will have a larger impact on KR than the map. Section 6.3.1 mentioned the high Pearson correlations between the targets and several class parameters of both players as evidence that class parameters are a major factor of the unexpected high performance of the baselines. These results provide additional support for that hypothesis for the KR target. Especially since we can see in Table 6.5 that using the map as the only input leads to $R^2_{KR} < 0.01$ for all models.

That hypothesis does not hold for duration prediction, as the MAE_t of all models that only receive class parameters are a lot higher than those that receive both inputs. Indeed, the resulting explained variance ($0.16 < R^2_t < 0.20$) seems little better than predicting the mean. The map is a more important input for predicting duration than the classes, despite a high correlation between duration and the accuracy parameter. This is corroborated by the significantly lower MAE_t for all map-only models in Table 6.5, compared to the class-only models in Table 6.6. Interestingly, when CNN3 receives only the map, its MAE_t is not lower than that of the MLP. Yet, this model is better on all fronts when receiving both inputs. This seems to indicate that the multi-modal CNN architecture of Fig. 6.1 is not just better at predicting the respective targets than a fully-connected network, but better at fusing the two types of input.

All in all, we can conclude that there are indeed interactions between both of the input modalities and both targets. The class parameters seem to be more informative for KR prediction, while the maps are more useful for duration. However, when the models receive both inputs, the error on both targets is lower than what would be expected based on this single input-target pairing. Given that the CNN is not always the best model based on a single input, the CNN with both inputs seems to model this interaction between modalities particularly well. This corresponds to the results described in Appendix A.

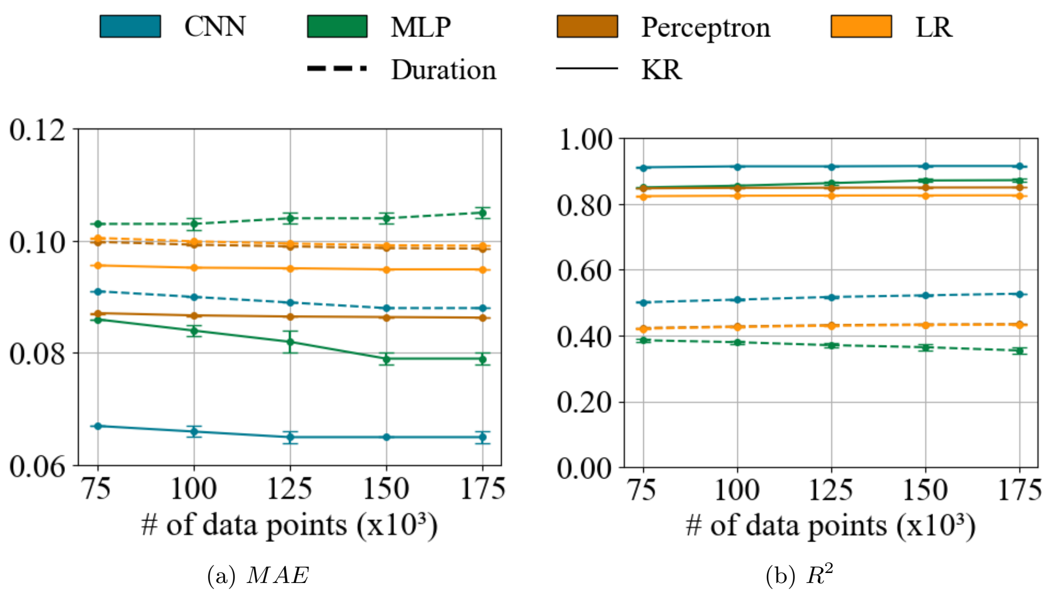


Figure 6.2: The mean absolute error and R^2 of models are trained on various dataset sizes. The results are obtained on a test set of $50{\cdot}10^3$ data points and are an average of 10 independent runs on different slices of the training set. The error bars indicate the 95% confidence intervals.

6.5 Sensitivity to dataset size

One of the drawbacks of deep learning is that it often requires large amounts of data to work. Obtaining a lot of gameplay data might be bottleneck for a game under development. It might be that the simpler models have a better performance when data is limited. In that light, the best performing model (CNN3) and the baselines are trained on varying amounts of data; in a range of 75-175 thousand data points. For each data set size, we create ten random subsamples of the original training set, resulting in ten independent runs per model. Each of these models is tested on a newly generated test set of $50 \cdot 10^3$ data points. The resulting absolute error and explained variance are shown in Fig. 6.2.

It is immediately clear that the lines are almost flat, indicating that the difference between training set sizes is very small. All models except the MLP slow a small increase in performance in either target dimension as more data is available. Instead, the performance of the MLP degrades, which is a sign that it is overfitting to the training set. The perceptron and linear regression have a very comparable performance on duration prediction and consistently have a lower error than the MLP. In this scenario, the CNN is still the model with the lowest error and highest explained variance (R^2) on both targets, regardless of the size of the training set. The R^2_{KR} is very high for all models; apparently not much data is needed to pick up on the strong linear correlations between the class parameters and KR. For duration prediction, the effect of having a larger data set is more profound. In fact, the difference in both MAE and R^2 between $150 \cdot 10^3$ and $175 \cdot 10^3$ is significant for all models, except the MLP. From this small test, we can conclude that models can be trained on (less than) half of the training set without much loss in predictive performance and that the CNN is still the best model for this task, even with smaller data sets.

Table 6.7: Cross-entropy loss (CE), F_1 score and accuracy (ACC) for duration (t) and kill ratio (KR) prediction. The results are an average of 10 independent runs and include the 95% confidence intervals. Note that KR is a 7-class problem, whereas t is a 3-class problem.

Model	CE_{KR}	CE_t	$F1_{KR}$	$F1_t$	ACC_{KR}	ACC_t
LR	1.23 ± 0.03	$0.81 \pm 2 \cdot 10^{-4}$	$0.48 \pm 9 \cdot 10^{-3}$	$0.62 \pm 3 \cdot 10^{-4}$	$0.49 \pm 8 \cdot 10^{-3}$	$0.62 \pm 3 \cdot 10^{-4}$
Perc.	1.20 ± 0.03	0.80 ± 0.01	$0.49 \pm 8 \cdot 10^{-3}$	$0.62 \pm 7 \cdot 10^{-3}$	$0.50 \pm 7 \cdot 10^{-3}$	$0.63 \pm 6 \cdot 10^{-3}$
MLP	$1.08 \pm 4 \cdot 10^{-3}$	$0.75 \pm 2 \cdot 10^{-3}$	$0.52 \pm 5 \cdot 10^{-3}$	$0.65 \pm 3 \cdot 10^{-3}$	$0.53 \pm 3 \cdot 10^{-3}$	$0.65 \pm 2 \cdot 10^{-3}$
CNN3	$\mathbf{0.90 \pm 4 \cdot 10^{-3}}$	$\mathbf{0.69 \pm 1 \cdot 10^{-3}}$	$\mathbf{0.63 \pm 3 \cdot 10^{-3}}$	$\mathbf{0.69 \pm 1 \cdot 10^{-3}}$	$\mathbf{0.63 \pm 3 \cdot 10^{-3}}$	$\mathbf{0.68 \pm 1 \cdot 10^{-3}}$

6.6 Classification

So far in this chapter, we have framed the task of gameplay prediction as a task of regression. But we could also frame it as a classification task, similar to the proof of concept experiment in Section 4.1. We can use the classes that were created for oversampling in Section 5.3.3 as targets for this task. Using the same training procedure as in Section 6.2, we can train CNN3 and the baseline models on the cross-entropy loss instead of the mean squared error. Cross-entropy describes the difference between the predicted and the true class distribution and is the preferred metric for classification problems (see Section 3.1.2).

Typical performance metrics for a classifier are accuracy and F_1 score. The accuracy measures which ratio of the test samples were given the correct classification. This metric has been criticized for giving distorted results on unbalanced data. For example, in binary classification it is easy to get 99.9% accuracy if one of the classes occurs only once in every thousand observations and the classifier is hard-coded to always return the other class. Even though we have balanced the dataset, we also report the F_1 score, which circumvents this problem by comparing more features of the returned predictions. The F_1 score is the harmonic mean of the *precision* and *recall* of a classifier. Precision (p) is the fraction of predicted positives that are actually positive, while recall (r) is the fraction of positives that were predicted as positive. In a binary classification task, this is computed by:

$$F_1 = 2 \cdot \frac{p \cdot r}{p + r} \tag{6.1}$$

In a multi-class problem, the F_1 score is an average of the score per class. In this experiment, averaging is done without weights, because the dataset is already balanced. For more information, the reader is referred to the documentation of Sci-kit learn (Pedregosa et al., 2011), which was used for the implementation.

As can be seen in Table 6.7, the difference between accuracy and F_1 score is quite small in this particular task. The same trends can be seen as in the regression task, i.e., the baselines are better as they increase in complexity and the CNN is better than the baselines. Although in this case the margin between the CNN and the MLP is small for time prediction instead of KR prediction. In general, the models have a higher accuracy on time than on KR. This is explained by the fact that time is much easier to predict correctly because it only has three classes, whereas the KR has seven. On the other hand, when compared to random predictions ($ACC_{KR} = 0.1428$, $ACC_t = 0.3333$), the increase in accuracy is larger for KR than for t.

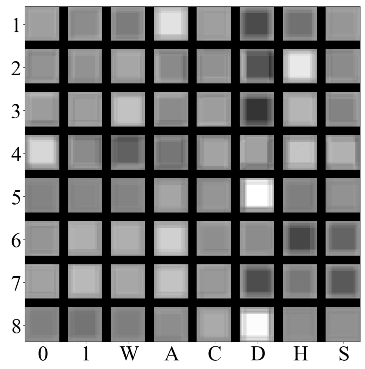
6.7 Inspecting the Model

As the success of machine learning applications grows, so does the call for explainable AI, especially in visualizing the inner workings of black-box machine learning methods (Simonyan et al., 2014; Nguyen et al., 2017; Erhan et al., 2009). Understanding the decisions of an AI system makes it more accessible and allows designers to better explore the creative potential of such systems (Zhu et al., 2018). One way of inspecting a CNN is by visualizing the patterns that the network is looking for, e.g., by creating the optimal image for each feature map that results from the activation of a convolutional node. This algorithm, called Activation Maximization (Erhan et al., 2009; Simonyan et al., 2014), focuses on general features that have been learned from the training set. Another option is to explain individual decisions, e.g., by visualizing which parts of the image had the most influence on the prediction of the model. A successful method in classification problems is called Gradient-weighted Class Activation Mapping (grad-CAM) (Selvaraju et al., 2017). These algorithms each increase the transparency of a machine learned model from a different perspective: interpreting the concepts it has learned and explaining the model's decisions by identifying the relevant input variables. Both methods will be applied to CNN3 in the sections below.

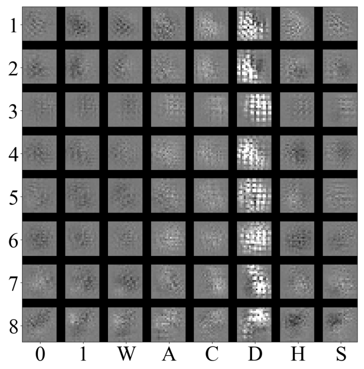
6.7.1 Activation Maximization

Activation Maximization became famous after a team at Google open sourced their code example and dubbed it *DeepDream*; they described the algorithm as the computer equivalent of looking at clouds and interpreting their random shapes (Mordvintsev et al., 2015a,b). The neural network is shown an image of clouds, after which the gradients of the activation of one or more output nodes are computed (activation step). The activation step can be applied to any node in the network. Typically these nodes are either in the convolutional layers or in the output layer, e.g., the node with the highest activation to amplify the classification of the model or a node that corresponds to a preselected class such as 'banana'. Next, the gradient is used to change the image in such a way that the activation will slightly increase (maximization step). The slightly changed image is shown again to the network, which is now reinforced in its prediction. These two steps are repeated for a number of times until the features become clear. After a while, the clouds will have changed into whatever the network has been trained to classify, e.g., faces, animals or buildings. Extra hallucinogenic effects are created by slightly zooming and/or rotating the image after each maximization step, concatenating these images into a video results in the "Deep Dreams" that made the algorithm famous.

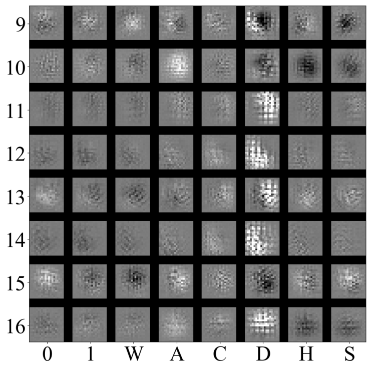
Aside from producing psychedelic algorithmic art, this algorithm becomes practical for model inspection when applied to white noise. Since there are no clear patterns in the input, the network will project the features that it "wishes" to see. Note that *DeepDream* applied the algorithm to nodes in the output layer, thus reinforcing specific classifications. For model inspection it is more useful to apply this to the convolutional layer, as this visualizes what shapes the feature detectors strongly respond to. Activation Maximization has been used to show that the first layers of well performing deep classifiers (e.g., AlexNet, OverFeat, and VGG16) learn detectors for edges and colors that resemble Gabor filters, a well-known type of detector used in manual feature extraction(Sermanet et al., 2014; Zeiler and Fergus, 2014; Chollet, 2016). While deeper into the network the patterns become more complex and show specialized nodes, e.g., for detecting eyes. The evidence that these models, which



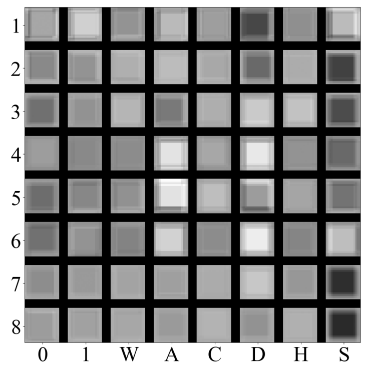
(a) L1 for KR



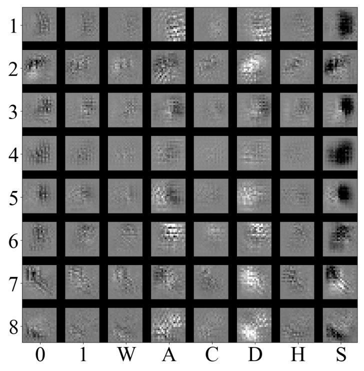
(b) L2 for KR (node 1-8)



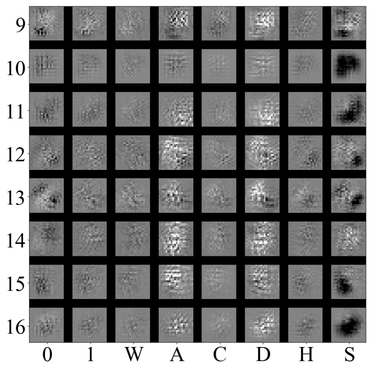
(c) L2 for KR (node 8-16)



(d) L1 for duration



(e) L2 for duration (node 1-8)



(f) L2 for duration (node 8-16)

Figure 6.3: Activation Maximization applied to the convolutional layers (L1 and L2) of CNN3, while maximizing KR (top) and duration (bottom). Each row is a convolutional kernel, while each column presents the maximized response to each tile type. Values are normalized from -1 (black) to 1 (white).

were so dominant in image classification competitions, learned both general and specialized feature detectors quickly led researchers to focus on *end-to-end* learning instead of manual image preprocessing.

Typically, this algorithm is applied to networks that have been trained on RGB-images, facilitating the visualization of all input dimensions simultaneously. The maps in this thesis have been shown as RGB-images, yet converting pixel activations in 8 dimensions to 3 color dimensions would imply relations between dimensions that do not exist. Therefore, each input dimension is treated separately by feeding the network 8 greyscale images. The results are shown in Fig. 6.3. Note that the cover input is empty in all levels in this thesis. The weights connected to this layer are untrained and thus the noise in this column can be seen as a random baseline for this visualization.

As can be seen from Fig. 6.3, the first layer of our CNN is mostly looking for power ups. Another interesting find are negative features, such as ‘not stairs’ (Fig. 6.3d, row 7 and 8). Though the patterns in the first layer are not as complex as in (Zeiler and Fergus, 2014) or (Chollet, 2016), there do exist neurons that respond to combinations of tiles, e.g., armor and damage (Fig. 6.3d, row 4) or health pack and not damage (Fig. 6.3a, row 2). The second layer has more intricate patterns than the first, yet the responses to the types of tiles are very similar. The focus lies on the powerups and even the 'not stairs'-feature (Fig. 6.3e, row 4) can be observed. By comparing the first two columns of Fig. 6.3b and 6.3c with those of Fig. 6.3e and 6.3f, we can see that the second layer of the model for duration seems to respond more strongly to floor levels than the model for KR.

The patterns found in these models are very simple compared to the literature. This could well be a result of the simple structure of the levels compared to natural images, which is perhaps amplified by the smaller input size. Indeed, one might wonder whether manually computing a (flat) summary of visual level features would yield similar results. Among other things, Section 8.2.2 discusses the replacement of either the original map or even the entire convolutional layer with handcrafted features. Comparing those results with the *end-to-end* model shows that a model based on 90 summary statistics is not better at KR prediction and strictly worse at predicting t, which is the gameplay outcome that is most influenced by the map. Additionally, (Liapis et al., 2019a) show in more detail that the automatic feature extraction from basic tiles is not easily replaced by manual features, even when these include two-dimensional feature maps.

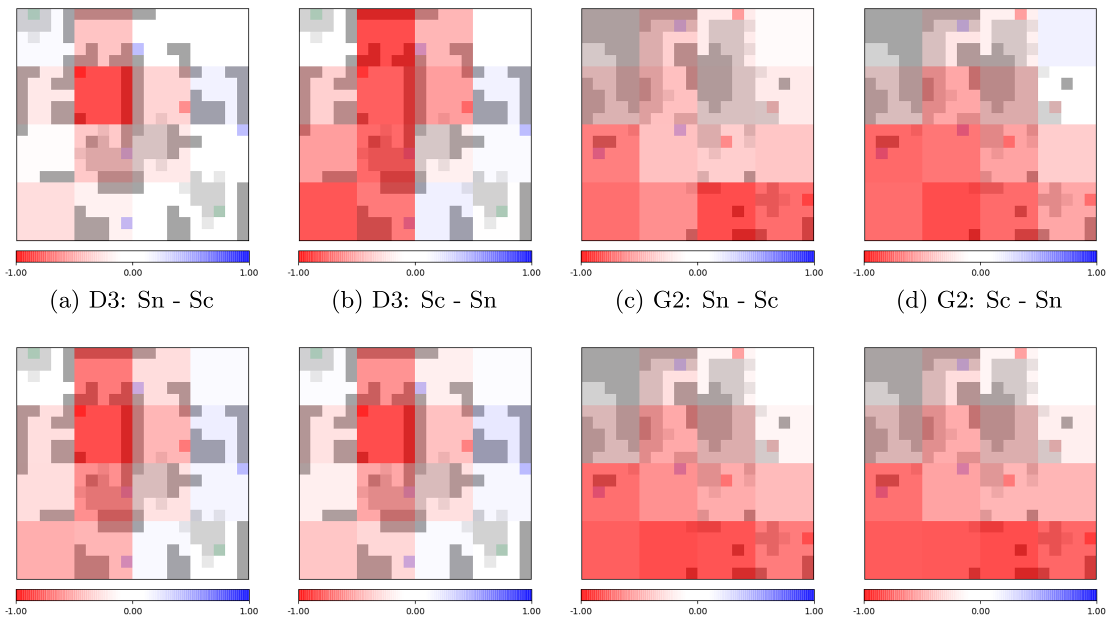
According to the visualization in this section, many nodes seem to respond to approximately the same tiles. Yet, replacing or augmenting these simple tile inputs does not automatically lead to a better accuracy. Given that knowledge, one might wonder whether these nodes are superfluous. But the results of preliminary experiments for hyperparameter tuning showed that networks with smaller convolutional layers had a larger error than CNN3. So perhaps the conclusion should be that, in this case, there is more to the network than can be visualized with this algorithm.

6.7.2 Gradient-weighted Activation Mapping

Gradient-weighted Class Activation Mapping (Selvaraju et al., 2017), or grad-CAM, consists of computing the gradient of an output node with respect to the nodes of a convolutional layer, given a particular input. By multiplying the input with the gradient, averaging over all nodes in the layer and normalizing the resulting values, we obtain a heatmap that shows how much each area of the input contributed to increasing the value of the output node. By selecting the most active output node, which corresponds to the class predicted by the model, this heatmap visualizes which areas of the image reinforce the decision of the model. While this approach is typically used in classification tasks, it also works for regression-based models. In that case, we are interested in the node that predicts a particular target (if there is more than one) and not in the most active node. Similarly, because we are dealing with a floating point prediction instead of a class, it becomes interesting to visualize not only the positive, but also the negative contributions to the output.

6.7.3 Applied to Maps

As an example of how grad-CAM can be applied to this particular problem, we examine the activation heatmaps for the generated level G2 of Fig. 7.2 and designed level D3 of Fig. 7.3 in two matchups with archetypal classes of Team Fortress 2: Sniper versus Scout, and Heavy versus Scout (see the parameters of each class in Table 4.1). The values at the last convolutional layer of CNN3 are presented, resulting in a 4×4 heatmap. Fig. 6.4 and Fig. 6.5 depict the heatmaps, respectively for duration and KR, as an overlay on top of the map sketch. Red areas in the activation maps show which parts of the level lead to a lower value in the gameplay outcome and blue areas show which parts lead to a higher value.



(a) D3: Sn - Sc (b) D3: Sc - Sn (c) G2: Sn - Sc (d) G2: Sc - Sn

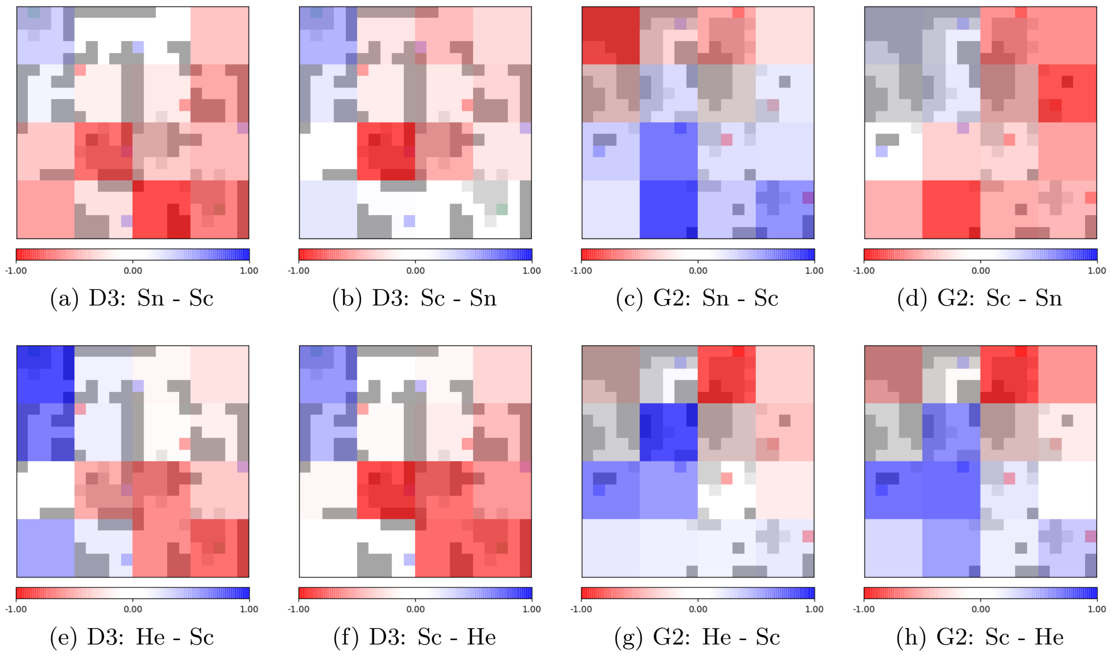
(e) D3: He - Sc (f) D3: Sc - He (g) G2: He - Sc (h) G2: Sc - He

Figure 6.4: Grad-CAM applied to CNN3 for duration prediction. The inputs are levels G2 of Fig.7.2 and D3 of Fig. 7.3, using as match Sniper vs Scout (top) and Heavy vs Scout (bottom).

From a machine learning perspective, these heatmaps can be used to validate the predictions of the model. It is obvious from Fig. 6.5 that the classes themselves can affect the level's heatmap, highlighting how the model acknowledges the interactions between the two facets. This is hard to observe from performance metrics alone. In Fig. 6.4 the effects of the classes are more subtle. These different scales of effect resonate with the observed correlations between the classes and the gameplay outcomes in the data set (see Section 5.3.1).

From a game design perspective, we can observe potential issues in match-ups between classes. For example, Fig. 6.5c shows that the kill ratio of player 1 (Sniper) is positively affected by the open areas of the bottom half of the level. While we can see in Fig. 6.5d that if player 1 plays as the Scout, the effect of the same area is negative. But if the opponent is changed to Heavy (Fig. 6.5h), the area has a positive effect again; perhaps indicating that the area can now be used by the Scout to maneuver. Aside from class specific pros and cons, general map issues can be spotted by comparing the heatmaps of different classes. From the heatmaps of D3 in Fig. 6.5, we can see that the damage pickup in the top-left generally has a positive effect on the score of player 1 regardless of the class. This is neutralized by the negative effect of the damage pickup in the bottom-right of the map. However, the armor pickup in the center has a negative effect for player 1, without an obvious advantage somewhere else in the map. In a mixed-initiative design tool, such elements could be flagged and shown to a designer as a potential balance issue.

Similar design tips can be given for duration. The heatmaps of G2 in Fig. 6.4 indicate that the open areas of G2 reduce the duration of the game regardless of the classes used. A mixed-initiative tool could either indicate this area as having potential for increasing duration or provide an alternate map based on mutations in that area. When combined with



(a) D3: Sn - Sc (b) D3: Sc - Sn (c) G2: Sn - Sc (d) G2: Sc - Sn

(e) D3: He - Sc (f) D3: Sc - He (g) G2: He - Sc (h) G2: Sc - He

Figure 6.5: Grad-CAM applied to CNN3 for score prediction. The inputs are levels G2 of Fig.7.2 and D3 of Fig. 7.3, using as match Sniper vs Scout (top) and Heavy vs Scout (bottom).

domain specific design knowledge, this technique could be used to give more sophisticated feedback to a game designer by linking the observed activations with design patterns. For example,the enclosed space in map D3 seems to be a choke point with a high impact on game duration (Fig. 6.4). Essentially, high activations in this visualization, whether they are positive or negative, can be used by a game design tool to indicate to a designer which part of the level to tweak.

6.7.4 Applied to Classes

So far, we have focused on visualizing the map, as an image is the more obvious application of grad-CAM. But the same principle can be applied to the class parameters by computing the gradient of the output with respect to the input layer. This produces a vector of activations. The activation vectors with respect to the class parameters in the scenarios of the previous section are detailed in Appendix C. The class parameter activations are perhaps even more straightforward to process and rationalize than the level heatmaps and can thus be a valuable tool for game designers.

By comparing different matchups across maps, this visualization can highlight the importance of a class parameter in general or for a specific class. For example, in D3 and G2 the accuracy is always the most important factor in the gameplay outcome predictions. In a match between Sniper and Scout, the activations indicate to the designer that an increase in match duration would require a drop in the accuracy of the Sniper, while a higher kill ratio for the Sniper would require reducing the accuracy of the Scout (as increasing the accuracy of the Sniper is impossible). When we look at a specific class, the HP of the Heavy stands out. For all Heavy vs Scout matchups, the HP of the Heavy is has a relatively large activation. Indicating that the survivability of this class is an important factor in both its kill ratio as well as in extending the match duration.

An experienced designer would easily find the above mentioned information without automated analysis. Yet, an algorithm that can extract information from the surrogate model, which is essentially a black box, does have value if it is extended properly and adjusted to the right context. For a fully automated game designer these simple patterns might already be valuable if extracted automatically and formatted in a useful way. It might provide actionable chunks of information; either for feeding it into another generative process (e.g., the importance of HP for the Heavy should be reflected in its visuals) or for explaining design decisions (e.g., tuning accuracy for a desired gameplay outcome). In the context of a mixed-initiative design tool, an analysis like this could provide useful feedback if it were more extensive in terms of maps and matchups. An analysis on a larger scale could explain the general trends in the relations between all classes or reveal to which degree multiple classes rely on the same parameters.

6.8 Summary

This chapter has introduced the architecture and the training procedure of the surrogate models that are used to predict the two gameplay outcomes that are used for facet orchestration, i.e., game duration (t) and score balance (KR). The latter is measured as the kill ratio of player one. It describes two experiments that were used for selecting the best model: the first consists of predicting both targets simultaneously from the original dataset, while the second consists of predicting the targets separately based on a dataset that was balanced for that particular target. Several models are compared in terms of their generalization performance on unseen data. Both experiments confirmed that a convolutional neural network (CNN) is a better model for this problem than three baseline models. The margin with the next-best model, i.e., the multilayer perceptron, is small but significant. In general, the KR seems an easier target for the models than game duration. This is illustrated by the best CNN, which has a near perfect fit of the balanced KR data ($R^2_{KR} \approx 0.91$) and a significantly lower fit for game duration ($R^2_t \approx 0.61$).

This chapter also addressed the question whether both inputs are necessary for modeling the gameplay targets. The results of experiments with single inputs showed that maps are important for duration prediction and character classes for KR prediction. Nevertheless, models that base their predictions on both inputs are more accurate at predicting both targets than those with a single input dimension. Furthermore, we demonstrated the diminishing returns of training on larger datasets. Even with less data, the CNN remained the best model. In fact, the CNN trained on half the data had a comparable error on KR to the one trained on the whole dataset. The gameplay outcome prediction was framed as a classification problem, yielding similar patterns as the regression problem and no clear benefits for using these models in a procedural content generation setting.

Finally, this chapter has explored two visualization methods in an attempt to explain how the CNN makes its decisions. Activation maximization was applied to show which features are learned by the CNN. This revealed a focus on powerups. Additionally, a slightly adjusted version of gradient-weighted class activation mapping was applied to both the maps and the class parameters to highlight which areas of the input are important in individual predictions. This latter technique shows promise for a game design tool that can explain the predictions of the model and indicate the most promising elements to change in a design.

Chapter 7 analyzes the application of multi-modal CNNs as surrogate models in evolutionary algorithms that search for good game content. The experiments consist of generating either maps or classes, and orchestrating the generation of both. Additionally, the chapter compares the use of an aggregated gameplay target with evaluating both gameplay targets separately via a multi-objective approach.

Chapter 7

Orchestration of Maps and Classes

The previous chapter has established the convolutional neural network as a surrogate model of gameplay. It demonstrated its predictive capabilities on the dataset of Chapter 5 and how it can be used to give feedback to game designers. This chapter explores the utilization of a CNN as a surrogate model to guide search-based procedural content generation. As outlined in Chapter 4, the general use-case of this thesis is for the computer to improve an initial design, which can either be created by a designer or by another generative algorithm. That chapter also detailed the genetic operators and the initial seeds used for searching the design space. In particular, Section 4.2.3 described the objective functions of both the single-objective (SO-EA) and multi-objective (MO-EA) evolutionary algorithms that are employed for the experiments in this chapter.

Preliminary tests have shown that the proposed algorithm is self-consistent and will indeed optimize for the objective function. However, the gameplay approximation made by the surrogate model might not perfectly fit the actual gameplay outcome for a particular match. In order to assess the actual improvement made by the system, we need metrics based on actual gameplay observations. Section 7.1 describes the used performance metrics based on the desired, actual and predicted improvement made on the initial design.

This chapter will explore the three possible modes of the generative approach: fine-tuning a match-up between two classes while keeping the map fixed in Section 7.2, adjusting a map to accommodate a fixed pair of classes in Section 7.3, and adjusting both the maps and the classes at the same time in Section 7.4. The first two experiments focus on characterizing the generated classes for a set of maps and the changes made to a map for a set of classes. The third experiment compares the three modes of the generative framework in terms of performance metrics, changes made to the initial design and the difference between using single-objective and multi-objective evolution for the search process.

7.1 Evaluation

The generative algorithm uses an objective function to create content that has desirable gameplay. The aim of the algorithm is to minimize the mean squared error between the gameplay predictions of the model and the gameplay targets set by a designer (see Section 4.2.3 for more details). This section describes the specific gameplay targets that were used in the experiments, an overview of the initial designs that were passed to the generative algorithm and metrics for evaluating the generated content.

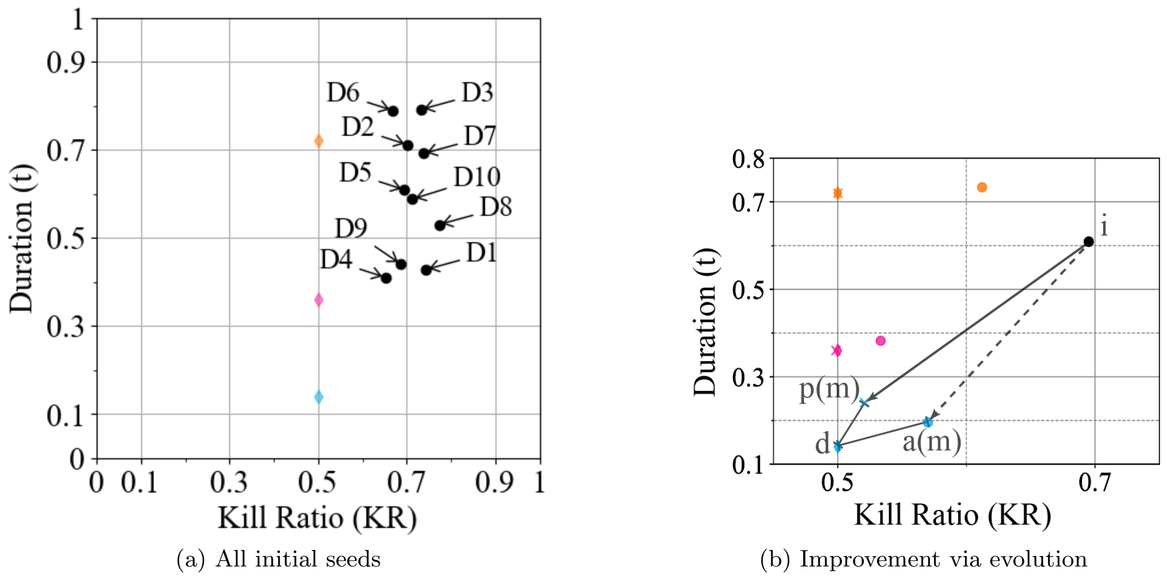


Figure 7.1: Initial state of the game design of Scout vs Heavy in terms of the gameplay dimensions, and evolution towards short matches. The black dots indicate the different levels, the diamonds indicate the desired gameplay values for short (blue), medium (magenta) and long (orange) matches.

7.1.1 Gameplay Targets

The generative system allows the designer to set two gameplay targets, one for the game duration and one for the score balance between the two players. In general, game designers aim to give opposing players an equal chance of winning the game, assuming they have equal skill. For players of unequal skill, a designer might introduce penalties or asymmetric design to rebalance the odds. The proposed generative system could be asked to give either player favorable odds. However, we cannot test whether this target achieves the goal of balancing for unequal skill, because the current game environment only has one type of simulated player. As such, each of the experiments is aimed at creating a balanced matchup, i.e., $d_{KR}=0.5$ in Eq. (4.1).

In order to test the expressivity of the system, as well as to identify how sensitive the generated results are to different designer priorities, this chapter uses three different durations as targets for evolution: *short*, *medium*, and *long*. For the experiments on the original dataset, these values are defined based on the edges and the center of the dataset, i.e., 200, 300 and 600 seconds. The experiments on the balanced dataset use duration targets based on the centers of the bins that were used for balancing. These centers are 213, 312 and 474 seconds. Throughout the chapter we use the term *short match* to identify a match with a map and/or pair of classes that was evolved for a short target time, regardless of how long the match actually lasted; similarly for medium matches and long matches.

7.1.2 Performance Metrics

The main goal of the experiments is to identify whether the surrogate-based facet generators can bring the different game elements closer to the intended gameplay outcomes. Towards this, we treat the two gameplay outcomes as a two-dimensional vector and use this to compute the Euclidean distances between the desired gameplay and respectively the gameplay predictions of the model (which is similar to the fitness function) in Eq. (7.1) and the gameplay outcomes obtained via game simulations in Eq. (7.2). Additionally, the *improvement* metric in Eq. (7.3) calculates the actual gameplay outcomes of the evolved content and compares it to the actual gameplay outcomes of the original content. The actual gameplay outcomes are calculated from 10-30 simulated playthroughs with AI agents, depending on the experiment. The multiple simulations allow us to minimize noise caused by simulation randomness, and to give us a statistical measure of each gameplay parameter through its mean and 95% confidence interval. A second goal is to observe whether the predictive model of gameplay outcomes matches the actual gameplay outcomes of evolved content. That is, we want to know the *prediction discrepancy* of the model. Towards this, the *error* metric in Eq. (7.4) compares the output of the surrogate model (which is used as part of the fitness calculation) with the actual gameplay outcome.

$$F(\boldsymbol{m}) = dist(\boldsymbol{p(m)}, \boldsymbol{d}) \tag{7.1}$$

$$A(\boldsymbol{m}) = dist(\boldsymbol{a(m)}, \boldsymbol{d}) \tag{7.2}$$

$$O(\boldsymbol{m}) = \frac{dist(\boldsymbol{d}, \boldsymbol{i}) - dist(\boldsymbol{d}, \boldsymbol{a(m)})}{dist(\boldsymbol{d}, \boldsymbol{i})} \tag{7.3}$$

$$E(\boldsymbol{m}) = dist(\boldsymbol{a(m)}, \boldsymbol{p(m)}) \tag{7.4}$$

where $dist(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; $\boldsymbol{m}$ is the individual being evaluated (i.e., a vector of class and/or level parameters); $\boldsymbol{p}(\boldsymbol{m}) = \{p_{KR}, p_t\}$ is the two-dimensional vector of gameplay outcomes as predicted by the surrogate model, i.e., predicted kill ratio (p_{KR}) and predicted duration (p_t); $\boldsymbol{d} = \{d_{KR}, d_t\}$ is the two-dimensional vector of the desired gameplay outcomes specified by the designer, i.e., desired kill ratio (d_{KR}) and desired duration (d_t); $\boldsymbol{a(m)} = \{a_{KR}, a_t\}$ is the two-dimensional vector of actual gameplay outcomes averaged from simulated playthroughs, i.e., actual kill ratio (a_{KR}) and actual duration (a_t); $\boldsymbol{i} = \{i_{KR}, i_t\}$ is the vector of actual gameplay outcomes of the initial class/level set, i.e., initial kill ratio (i_{KR}) and initial duration (i_t).

In order to evaluate the results of single-objective evolution, the fittest individual at the end of the evolutionary process is chosen and tested in simulated playthroughs. Unlike its single-objective counterpart, multi-objective evolution produces a large set of Pareto-optimal individuals that does not contain an objectively best individual. In the interest of computational effort and fairness in comparisons, only the best individual based on the single-objective fitness of Eq. (4.1) is tested in those experiments. Throughout the chapter, significant differences are established on a two-sided t-test assuming unequal variance, using a statistical significance threshold of $\alpha = 0.05$.

7.1.3 Initial Designs

The surrogate-based generative approach proposed in this thesis can be both applied as a standalone content generator or as part of a co-creative process in a mixed-initiative design tool. To demonstrate the applicability to both situations, experiments are performed on both procedurally generated maps and handcrafted maps. The maps depicted in Fig. 7.2 are created by the same level generator that was used to produce the dataset: they are similar (but not identical) to levels on which the surrogate model was trained. The maps in Fig. 7.3 were created by a human designer, and many of them feature a degree of symmetry that is unlikely to be exhibited by the generated levels. Moreover, the human-authored levels attempt to balance the distances from the bases to the powerups and feature explicit level patterns such as arenas, choke points and flanking routes (Hullet and Whitehead, 2010).

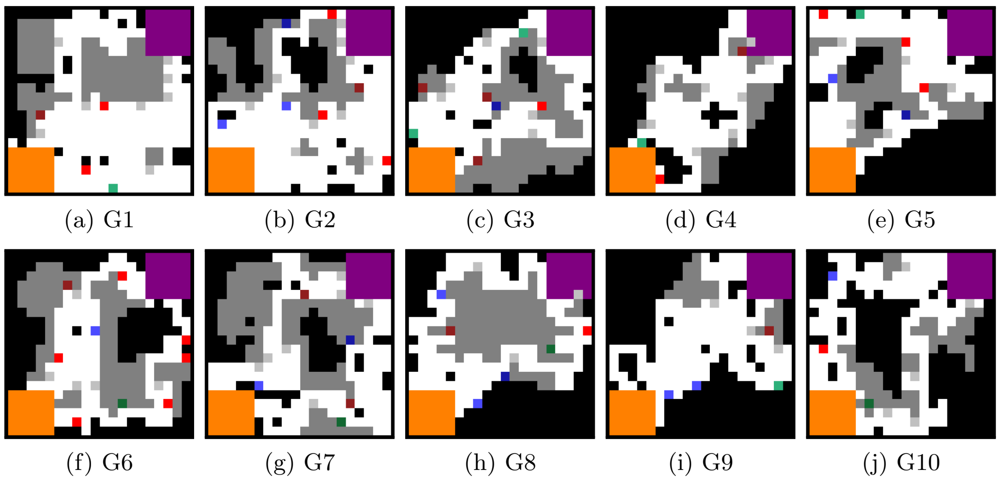


Figure 7.2: Procedurally generated game levels used as initial seeds for the generative process.

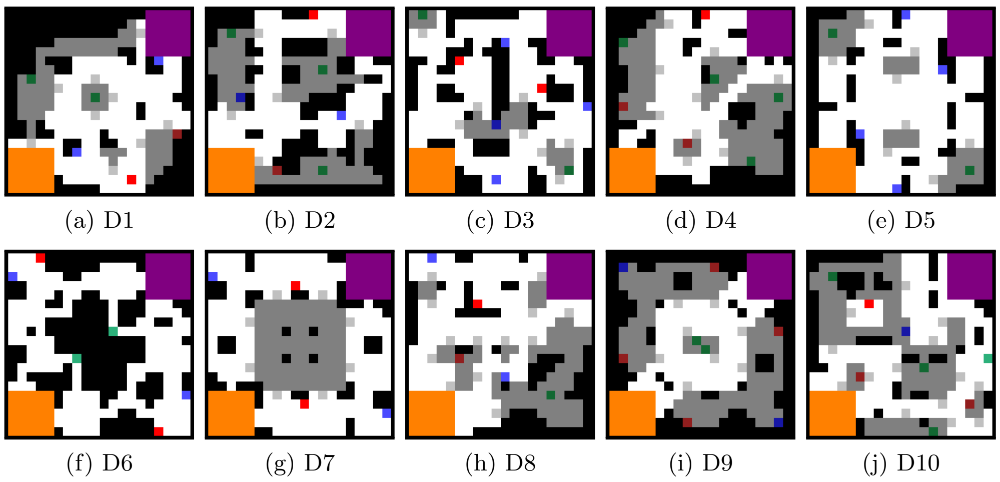


Figure 7.3: Handcrafted game levels used as initial seeds for the generative process.

The classes that are used as part of the initial design or as frame of reference in these experiments are based on the archetypical classes in shooters and named after their counterparts in *Team Fortress 2* (Valve, 2007). The parameters are listed in Table 4.1. In order to clarify the differences in terms of gameplay, the major trade-offs of the classes are listed in Table 7.1. The Heavy, Pyro and Scout all carry short ranged weapons; the Sniper and the Soldier carry a long ranged weapon. The cloaking and knifing abilities of the Spy are ignored, treating it as a regular, medium ranged class.

Table 7.1: Trade-offs between character classes in this thesis.

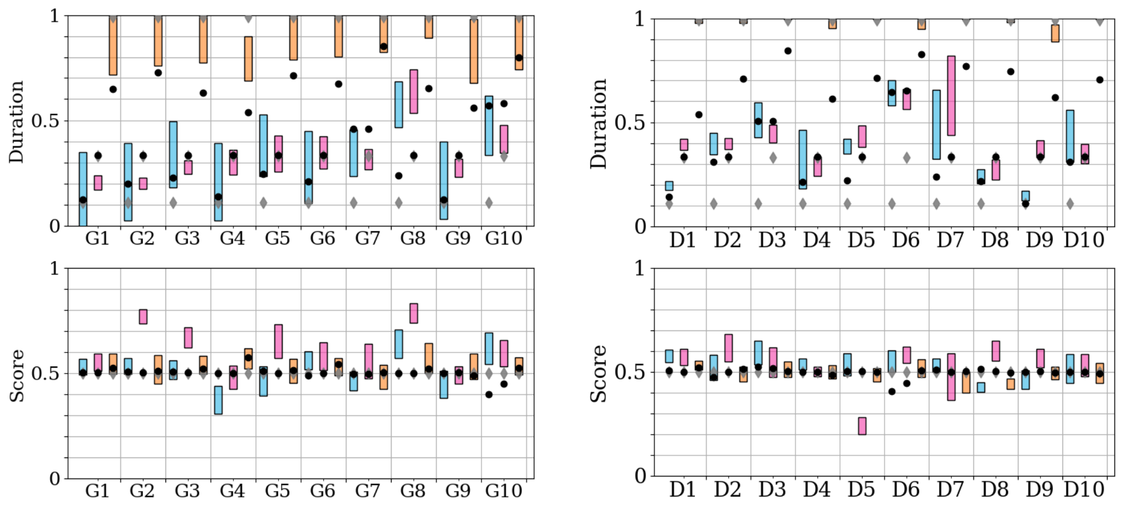
Class	**High**	**Low**
Heavy	health, rate of fire	speed, accuracy
Pyro	damage	accuracy
Scout	speed	health
Soldier	health, accuracy	rate of fire, clip size, speed
Spy	accuracy	clip size
Sniper	damage, accuracy	rate of fire, health

7.2 Generating Balanced Character Classes

The first experiment creates a matchup of two character classes appropriate for a specific level, i.e., changing the parameters of the character classes but not the level itself. Initial individuals have random parameters in a genotype of 16 values. In order to test a broad set of levels, we ran evolutionary algorithms on each of the 60 testbeds (all 20 levels of Section 7.1.3, with 3 desired durations each). Character class pairs are evolved for each provided level, and each desired score ($d_{KR} = 0.5$) and match duration (d_t). The normalized values for the desired durations (d_t) are respectively 0.11, 0.33, and 1. In each run, 100 individuals evolve for 100 generations, minimizing the fitness function of Section 4.2.3 based on the predictions of CNN1 of Table 6.1. To assess the algorithm, we perform 10 independent runs and select the best individuals at the end of each evolutionary run (i.e., for one level, d_t and d_{KR} tuple). We test the gameplay of each class pair by simulating 10 matches in the same level it was evolved for.

Results for the fittest character classes per run, compared to the ground truth (GT) collected from simulated matches, are shown in Fig. 7.4. It is evident that not all predictions made by the CNN fall into the confidence bounds of the ground truth from simulations. We consider the model's prediction accurate if its predicted gameplay metric falls within the confidence bounds of the 10 simulations' metrics. Among all 20 levels, the estimated duration is accurate in 21 of 60 cases (35% accuracy) and the estimated score is accurate in 36 of 60 cases (60% accuracy). Score balance is fairly consistently predicted in both generated levels (63% accuracy) and in designed levels (57% accuracy). The predictions in terms of match duration are less accurate: designed levels are more prone to inaccurate predictions (27% accuracy) than generated levels (43% accuracy). In short duration matches the predicted and ground truth durations often match (in 12 of 20 levels); accuracy for match duration drops with medium durations (35%) and more so with long durations (10%). Interestingly, score prediction is less accurate for medium matches (35%) than for short matches (55%), but predictions are exceptionally accurate in long matches (90%).

Besides the accuracy of the CNN model when evolving classes, it is important to investigate whether the resulting classes fulfill the broader goals of the designer, in terms of the actual duration and balance of the class matchups. Based on Fig. 7.4, it is clear that either predicted or GT values vary depending on the level. Notably, the differences in GT durations are not always perceptible between short and medium matches for the same level. Indeed, only in two of the tested maps are the GT durations of short matches significantly different than the GT durations of medium matches. Across all 20 levels, the class matchups for short matches last for 306 seconds on average while for medium matches the average is 319 seconds. Due to the small difference and high deviations from level to level, the increase



(a) Generated levels (b) Handcrafted levels

Figure 7.4: Predicted and actual gameplay parameters for generated class pairs. The actual gameplay parameters are displayed as the upper and lower 95% confidence bounds from 10 simulated matches per class pair. Class pairs are sorted by level and colored per target duration (blue for short, fuchsia for medium, orange for long). The black dots and gray diamonds respectively indicate the predicted (p_{KR}, p_t) values and the desired (d_{KR}, d_t) values.

in GT duration is not significant between short and medium matches. On the other hand, the GT duration of long matches is significantly different from GT durations of both short and medium matches in each of the 20 levels tested.

While classes evolved for a long match do indeed fight for longer in simulations, that does not mean that all such simulated matches last the intended 600 seconds. We find that the d_t (i.e., 200 sec, 300 sec, 600 sec) falls within the confidence bounds of GT duration in 25 out of 60 cases; the desired balance (0.5) is within the confidence bounds of the simulations' score ratio in 35 out of 60 cases. The origin of the map (generated or designed) seems to matter little, although designed maps match the d_t only in 8 out of 30 cases, and never accurately match the intended short duration (200 sec). Interestingly, classes evolved for medium matches are the worst at matching d_{KR} (7 of 20 cases). Despite differences from map to map, the classes evolved for medium matches in all 20 maps tend to favor player one (with an average score of 0.57, significantly higher than classes evolved for short and long matches).

Another way of evaluating the accuracy of the predictions made by the computational model is based on the error metric E in Eq. (7.4). Similarly, the distance metric A in Eq. (7.2) is an indicator of how close the resulting classes are to what the designer requested. These measures of model accuracy and goal satisfaction are shown in Table 7.2. A brief overview shows that the distance between the GT and predicted values increases as the desired duration increases, i.e., the prediction error is higher for longer matches. Yet the distance between GT and desired values decreases for longer matches. Interestingly, this results in the actual gameplay in long matches being closer to the desired gameplay than the model predicted. Though it would be intuitive to attribute this to the designed maps

Table 7.2: Euclidean distances between mean GT (a_t, a_{KR}), predicted (p_t, p_{KR}) and desired (d_t, d_{KR}) values of classes generated with a CNN, starting from a random initialization.

Duration	Maps	$dist(\boldsymbol{p}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{p})$
Short	Designed	0.182 ± 0.104	0.275 ± 0.091	0.121 ± 0.035
Short	Generated	0.144 ± 0.093	0.223 ± 0.082	0.139 ± 0.060
Short	Both	0.163 ± 0.068	0.249 ± 0.061	0.130 ± 0.034
Medium	Designed	0.050 ± 0.069	0.148 ± 0.063	0.128 ± 0.056
Medium	Generated	0.038 ± 0.053	0.154 ± 0.074	0.174 ± 0.071
Medium	Both	0.044 ± 0.042	0.151 ± 0.047	0.151 ± 0.045
Long	Designed	0.293 ± 0.060	0.031 ± 0.014	0.281 ± 0.056
Long	Generated	0.322 ± 0.061	0.113 ± 0.031	0.217 ± 0.049
Long	Both	0.307 ± 0.042	0.072 ± 0.025	0.249 ± 0.039
Total	Both	0.171 ± 0.041	0.157 ± 0.032	0.177 ± 0.026

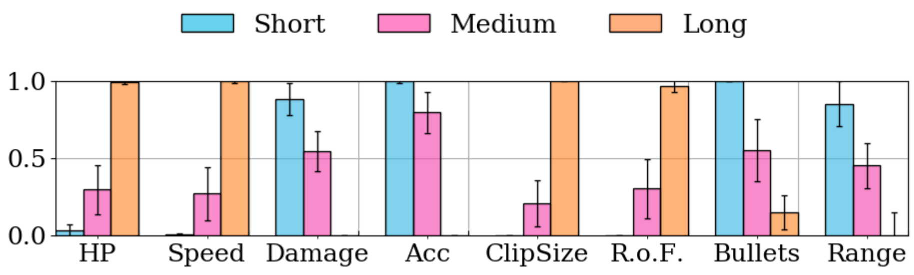
lying outside the distribution learned by the model, those matches generally have lower prediction errors than those on generated maps, except for long durations. At the same time, matches on designed maps are closer to the desired gameplay than those on generated maps, except for short matches. As can be seen from Fig. 7.4, the matches on designed maps generally take longer than predicted; regardless of the desired duration.

All in all, it seems that the designed maps might have a bias towards long matches, which highlights the importance of taking the initial gameplay into account. Nevertheless, the model seems to extend well to designed maps and it is interesting to see that the model manages to push the evolved classes in the right direction even without perfect predictions.

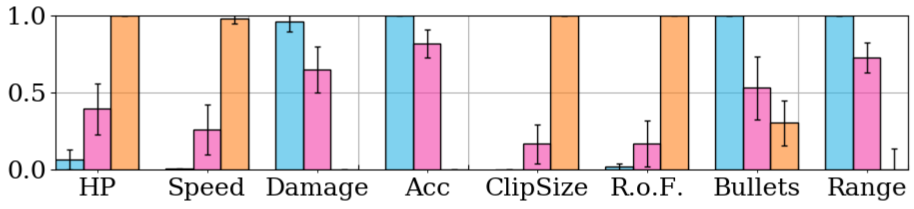
7.2.1 Patterns of the Evolved Classes

Aside from the accuracy of the model in terms of ground truth or desired gameplay outcomes, it is important to identify which type of classes are favored for each game level and for each match duration. Fig. 7.5 shows the average parameter values for each player's class across the 20 maps tested. We observe that there are significant differences between parameters in subsequent durations (short to medium, medium to long) in 30 of 32 cases. The patterns are consistent, e.g., hit points keep increasing with longer durations both when moving from short to medium and when moving from medium to long matches.

Considering the relationship between desired duration (d_t) and each player's class parameters, there are significant correlations with 9 of 16 class parameters (indicatively: speed, accuracy, clip size and range of both players). If we consider the mean GT duration of each map tested (a_t), there is significant correlation with all 16 parameters. Clearly, patterns learned by the surrogate model drive its choices for class selection. The patterns themselves make sense to a human designer as well: to make gameplay last longer, increasing the hit points of the players and lowering their weapons' damage and accuracy is intuitive. The only surprising trend is an increase in speed for longer matches, which is a consistent finding; we can hypothesize that the high speed coupled with opponents' low accuracy can be used as a secondary defensive mechanism, since a fast-moving player is more difficult to hit.



(b) Class parameters for player 1



(c) Class parameters for player 2

Figure 7.5: The average values for the generated classes' parameters with 95% confidence interval, colored per duration.

7.2.2 Similarities with *Team Fortress 2* classes

While the quantitative analysis has shown some important trends between the class parameters and durations, it is difficult to estimate in such a way how the generated classes can be played. Instead, we use the TF2 parameters of Table 4.1 to cluster generated classes based on their most similar game class. *Team Fortress 2* (Valve, 2007) has nine classes, each equipped with a signature weapon. We use five of these nine iconic classes (Sniper, Soldier, Scout, Heavy and Pyro) as the other classes have special class abilities such as healing or turrets that cannot be captured in our parameter mapping. In order to assess the difference between generated and TF2 classes, we use Eq. 4.4 to compute the distance in terms of parameters between the player's class and the TF2 class. In this way, we calculate the closest TF2 class. However, generated classes with a distance above 1.5 were deemed too different from any TF2 class and are classified as "undefined". The threshold was chosen on the one hand to avoid too many generated classes as undefined, while on the other hand to not classify generated classes as TF2 classes based on trivial similarities.

Figure 7.6a shows the distribution of classes per player and across players. The patterns here are more illustrative than Figure 7.5: for short matches, almost all classes evolved for either player are similar to long range classes (Sniper, Soldier). For long matches, the opposite is true and almost all classes resemble the Heavy class; this is likely due to the characteristic high HP count and survivability of the Heavy, but also due to the fact that it wields a mini-gun (a highly inaccurate, fast-firing short range weapon) which has similar characteristics with the trends in parameters for long durations in Fig. 7.6a. Interestingly, classes evolved for medium durations are the most diverse with a fair distribution between long-range and short-range classes, while 17% of generated classes for either player are too different to any TF2 class. While the differences between the distributions of TF2 classes for both players are not very pronounced, when evolving for medium duration in 65% of matches the two opponents are mapped to different TF2 classes (compared to 40% of short matches and 15% of long matches).

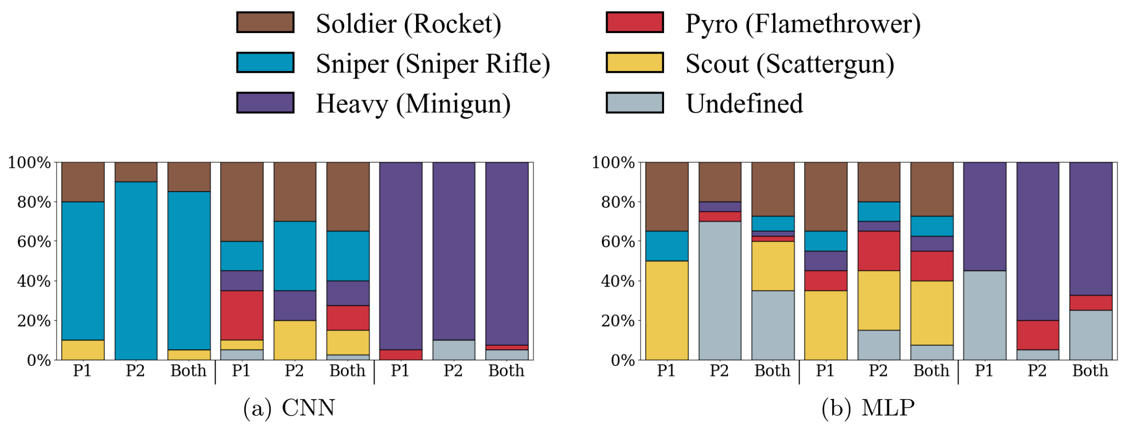


Figure 7.6: The distribution of classes per player and across players generated by both surrogate models, matched to TF2 classes. Results are grouped per target duration (left: short, center: medium, right: long).

7.2.3 Influence of Level Patterns

Beyond the performance of the genetic algorithm and its accuracy, it is worthwhile to investigate how the initial maps of Fig. 7.2 and Fig. 7.3 influence both the accuracy and the ground truth gameplay values of the evolved class pairs.

A simple qualitative analysis of Fig. 7.4 shows that accuracy (i.e., whether the prediction is within the 95% confidence bounds of 10 simulated matches) fluctuates from map to map. Taking into account all 3 intended match durations per map, we observe that predictions in some maps are inaccurate for both duration and score: maps G8 and D1 have no accurate duration predictions and only one accurate score prediction (both in the long match), while D8 has no accurate score predictions and only one accurate duration prediction (in the short match). Other maps as inputs result in inaccurate predictions in one gameplay dimension but are accurate in the other: maps G7, G9 and D10 have accurate score predictions in all 3 evolved class pairs but only one accurate duration prediction (long match for G7, short match for G9, medium match for D10).

An analysis of the ground truth gameplay values in Fig. 7.4 shows that in some levels evolution was unable to find classes for a particularly short duration (based on the ground truth of duration collected from 10 simulations): G8, D6 and D7 particularly stand out, as even short matches are around 400 seconds. It is possible that the surrogate model was not able to detect important visual patterns of the level to drive evolution to shorter durations. However, the small difference between predicted and actual durations for D3 or D6, as well as an inspection of the levels themselves suggests a more likely reason: G8 and D6 are very dense with winding corridors and choke points on the ground floor, which lowers the usefulness of long-range weapons while much of the time is spent exploring the level to find an opponent. On the other hand, D1 and D9 are very suitable for evolving classes towards all three durations, given the difference in actual and desired duration. A common feature in these maps is a damage bonus in the center. Perhaps this map feature works well in drawing the agents together, resulting in more time spent on combat instead of searching for resources. In that scenario weight lies more on the characteristics of the classes than on that of the map. In future work this might be investigated in more detail by performing experiments on maps with smaller differences, e.g., with and without such powerups.

In general, we can see from Fig. 7.4 that the ground truth durations on the designed maps have less variation (i.e., a smaller confidence interval) than the generated maps. This suggests that more purposefully designed maps can be tuned better towards specific gameplay when combined with procedural elements, which is promising for an orchestration approach to PCG.

7.2.4 Comparison with Classes Evolved by MLP

Due to the fairly similar validation accuracies of the best MLP with CNN1 in Table 6.3, it is worthwhile to investigate whether using an MLP as a surrogate model for evolving character classes would yield similar results. Using the same process for evolving and assessing the final content, the 20 maps of Section 7.1.3 were used as input to evolve 60 character class pairs for balanced short, medium and long matches based on predictions of the MLP. Ground truth values were established on the fittest individuals, and the core findings are compared with the results of the CNN1 surrogate model.

Classes evolved for the MLP model were accurate (i.e., within the GT confidence bounds) in 36 of 60 instances for predicting score and in 15 of 60 instances for predicting match duration. The score accuracy is the same as that of the CNN model; duration accuracy drops from 21 (CNN) to 15 (MLP), a relative decrease of 29%. This is not surprising, since the MLP had a lower R_t^2 value than the CNN, while both had high R_{KR}^2 values (see Table 6.3).

Classes evolved for the MLP model also have different patterns. Some of the class parameters have diverging trends: e.g., the speed of both players decreases in longer matches (contrary to CNN trends) and the rate of fire of both players does not increase in long matches. The distribution of MLP-based classes, matched to TF2 classes with the same threshold (1.5), is shown in Fig. 7.6b. The differences with Fig. 7.6a are clear and especially large in short matches. Overall, 23% of the classes evolved by the MLP do not match TF2 classes, compared to 5% for the CNN. In short matches this difference is even larger: 35% uncategorized instances versus 0% for the CNN. For the same target duration the CNN produces classes that match the Sniper in 80% of the instances, while 5% of the classes produced by the MLP do the same.

From this short analysis, it is clear that despite having similar accuracies, the MLP not only underperforms when attempting to predict duration in more cases than the CNN, but it also favors very different patterns in the evolved classes.

7.2.5 Summary

From the perspective of a regression task, it was evident when generating classes for specific levels that the predictions were not always matching the gameplay logs collected via simulations. Some level structures were more difficult to parse and resulted in less accurate gameplay predictions than others, mainly when predicting match duration (as expected from the lower R_t^2 values during model validation). The distribution of durations in the training set may be responsible for this bias, as it is skewed towards values near 300 seconds: the model may often estimate values near the mean duration without suffering from a high error. Another effect is that some maps seem to have a bias towards certain durations. Specifically in the designed maps we observed a potential bias towards long matches. We observed higher prediction errors but a lower distance towards the desired gameplay compared to the other duration targets. Apart from these inconsistencies when predicting duration, the model was shown equally capable of handling both handcrafted levels as well as generated levels that are similar to the ones it was trained on. This points to promising applications as a design tool.

The system seems consistent in applying certain design rules when targeting different durations: increasing the health and lowering the accuracy and damage as the target duration increases. As a result, in short matches the evolved classes tend to be similar to the Sniper, while in long matches the classes are predominantly similar to the Heavy. From a design tool perspective, it will be necessary to place additional constraints on the evolved individuals, as frequent suggestions to turn a class into a Heavy or a Sniper would soon be ignored by the user. Aside from placing constraints, more diverse results could also be obtained by alternating between the CNN and the MLP as surrogate model. The MLP favors very different designs than the CNN, although this comes at the cost of reduced duration accuracy.

7.3 Adjusting Maps for Balanced Matches

This experiment concerns the adjustment of a given map to accommodate desired gameplay between given classes. In order to test a broad set of distinctive class matchups, a set of five classes was chosen from *Team Fortress 2* (Valve, 2007): Heavy, Pyro, Scout, Spy[1], and Sniper. Matching all character classes against each other results in 25 matchup combinations (including five between avatars of the same class). In each run, 20 individuals evolve for 100 generations based on the predictions of CNN2 of Table 6.1. The initial population consists of 20 copies of the human authored map of Fig. 7.8a. Each combination of class matchup and intended duration (200s, 300s, 600s) is provided as additional input and intended output. Unless explicitly noted, all metrics are calculated from the average of 20 independent runs per matchup and intended duration. The actual gameplay outcomes are evaluated by averaging the results of 10 game simulations.

7.3.1 Starting Map

The initial population is seeded with the map shown in Fig.7.8a: its central area is symmetrical, as opposed to the edges. The map has four healthpacks on a diagonal between the two bases and two damage boosts on the first floor at the center of the map. P1 (orange) spawns near a damage boost and P2 (purple) spawns near armor. Based on simulations on the initial map (as part of the ground truth evaluations discussed in Section 7.3.2), the matches on this map on average last for 266 seconds, but can be as short as 203 seconds (Heavy versus Sniper) and as long as 484 seconds (Heavy versus Heavy). Judging by their average kill ratio against other classes, the map is most suitable for Snipers and least suitable for Heavies. This explains the short duration of the Sniper versus Heavy matchup, which ends in a hands-down defeat for the Heavy which manages a kill ratio of 0.2 (the worst in all matchups). All matches seem to give a slight advantage to P1 regardless of their class.

7.3.2 Improvements over the initial map

Based on the surrogate model's predictions, the Euclidean distance to the target balance and playtime decreases over the course of evolution, dropping to an average of 0.19 from

[1]The cloaking and knifing abilities of the Spy are ignored, treating it as a regular, medium ranged class.

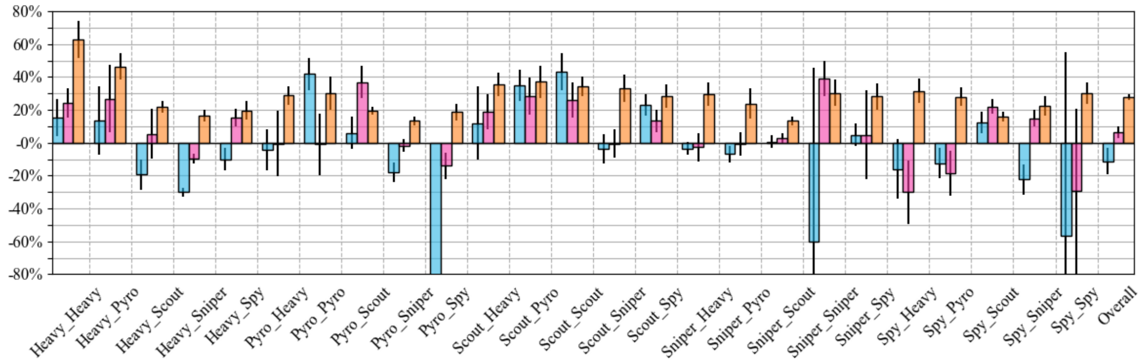


Figure 7.7: The improvements of the generated maps with respect to the initial map for each of the three durations: short (blue), medium (fuchsia) and long (orange). Results include the 95% confidence interval.

0.42 in the initial map. As expected, evolution is more challenging for the longer durations ($F=0.35$), which are rarely seen in the training set. The easiest duration to predict is the medium duration ($F=0.09$), which is very common among matches in the training set. The improvements per target duration per matchup are shown in Fig. 7.7, while the distance and improvement metrics aggregated per duration are described in Table 7.3

Observing the improvement of maps evolved for medium and short duration, we note some mixed results. On the one hand, maps evolved for medium duration show a minor improvement over the initial map, while maps evolved for short duration often have negative improvements (i.e., moving away from the desired values). It is important to note that for short durations specifically, more matchups have negative improvements (14) than positive (10). Many negative improvements were also observed for medium duration (7), but not more than positive (14). When evolving for long duration, on the other hand, evolution was far more successful in improving the maps compared to the initial state. A likely reason for this discrepancy between short and long durations is that the initial map was already favoring short or medium durations more than long durations; improving towards short durations was therefore more challenging for evolution.

Although maps for long durations were always improved over the initial map (in all 25 matchups), the model overestimated the predicted improvements. The high prediction error for long durations is primarily due to the fact that most matches in truth took less time than what was predicted. Similarly, for short durations the model overestimated how short the duration would be. This is somewhat expected from a regression model (especially one with a worse R_t^2 than R_{KR}^2), but it should be noted that in the majority of matchups the ground truth durations for maps evolved for short durations were shorter than those evolved for medium (16 out of 25 matchups) and maps evolved for long durations were longer than those evolved for medium (25 out of 25 matchups).

It is worthwhile to note that different class matchups have different performances across intended durations. The matchup with the highest improvement is Scout versus Scout for short durations ($O = 42\%$), Sniper versus Sniper for medium durations ($O = 38\%$), and Heavy versus Heavy for long durations ($O = 63\%$). Interestingly, the matchups with the highest improvements were predominantly symmetrical (i.e., both players have the same class). The largest negative improvement is Pyro vs Spy ($O = -189\%$), followed by Sniper versus Sniper ($O = -60\%$) and Spy versus Spy ($O = -56\%$); all evolved for short durations.

Table 7.3: Improvement and Euclidean distances between mean GT (a_t, a_{KR}), predicted (p_t, p_{KR}) and desired (d_t, d_{KR}) values of the generated maps; including the 95% confidence intervals.

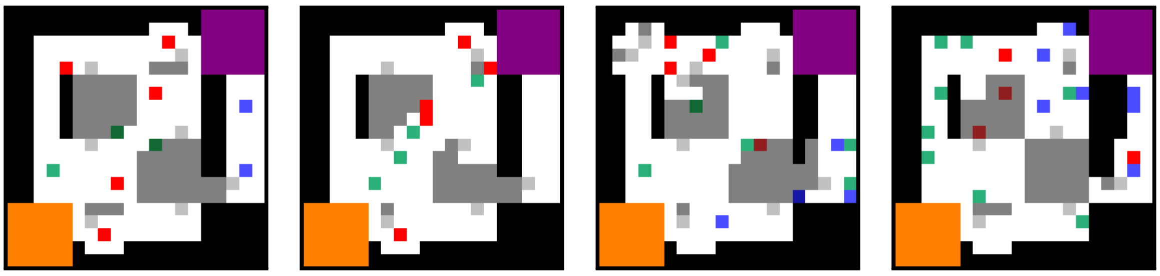
Duration	$dist(\boldsymbol{p}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{p})$	O
Short	0.133 ± 0.009	0.266 ± 0.013	0.159 ± 0.011	-0.113 ± 0.081
Medium	0.088 ± 0.009	0.215 ± 0.011	0.155 ± 0.008	0.063 ± 0.036
Long	0.343 ± 0.012	0.579 ± 0.017	0.283 ± 0.011	0.279 ± 0.017
Total	0.187 ± 0.005	0.352 ± 0.007	0.199 ± 0.006	0.078 ± 0.046

In all cases, the initial match was already fairly close to the desired gameplay, which made it hard for the system to find improvements. While the system did not manage to create any positive improvements for Pyro vs Spy, the results on the latter two matchups are skewed by respectively one and two catastrophic failures. These matches lasted 600 seconds and were more unbalanced in terms of score than the initial match. Although it is possible that these evolved classes were particularly bad, it is more likely that an AI error occurred while playing the game. If these data points were to be disregarded as outliers, the improvements would be a lot better: $O = -7.2\%$ for Sniper versus Sniper and $O = 25\%$ for Spy versus Spy. A similar "outlier" is found in the evolved classes of Spy versus Spy for medium duration, which explains the low improvement and large confidence interval in a similar fashion.

To assess which matchups improve more for different intended durations, the Kendall rank correlation coefficient (Kendall's τ) will be used. Kendall's τ coefficient provides a measure of similarity of the order of matchups in terms of improvement (through map evolution) from one intended duration to the next. A value of τ near 1 means that in both durations the maps evolved for the same matchup have a high or low improvement compared to others. A value of τ near -1 means that the order in terms of improvements is reversed, and a value near 0 means that there is no relationship between the rankings. We observe a fairly high rank correlation between class matchups improved for short and medium durations ($\tau = 0.56$), but less so for long durations ($\tau = 0.19$ from medium to long, $\tau = 0.31$ from short to long). Clearly, evolution favors different classes in longer games than in medium or shorter games. This is evident also from the fact that the most improved matchups for short and medium games were between fragile classes (Scout versus Scout, Sniper versus Sniper) while for long games the most improved matchup was between two highly survivable Heavies.

7.3.3 Analysis of level structures

Beyond the performance and accuracy of the genetic algorithm, it is worthwhile to investigate how the level itself changes to cater for one class over another, or to increase the duration of the playtime. In order to assess the plethora of level features which may or may not change in the evolving maps, a broad range of metrics were computed. Inspired in part by Liapis et al. (2013c), 31 metrics assess the structure of the level, and include simple enumeration (e.g., the total number of pickups, or the number of healthpacks on the first floor), pathfinding distance (e.g., distance from base of P1 to base of P2), or aggregated metrics (e.g., distance of one pickup to the nearest other pickup, averaged across all pickups). As an example, the most improved maps for each target duration are shown in Fig. 7.8.



(a) Initial
S: $\boldsymbol{i} = (0.60, 0.25)$
M: $\boldsymbol{i} = (0.53, 0.16)$
L: $\boldsymbol{i} = (0.54, 0.60)$

(b) Short
$\boldsymbol{a} = (0.49, 0.12)$
$O = 84\%$

(c) Medium
$\boldsymbol{a} = (0.47, 0.28)$
$O = 69\%$

(d) Long
$\boldsymbol{a} = (0.52, 0.99)$
$O = 95\%$

Figure 7.8: Most improved levels, starting from (a). The corresponding matchups are: Scout vs Scout (b), Sniper vs Sniper (c) and Heavy vs Heavy (d).

In terms of structure, the only major changes are observed in the long map; i.e., the closing of the flanking route on the right and the blocking of direct access on the diagonal from base to base. Clearly, this results in less shortcuts, more walking around and possibly even getting stuck on the right side of the map. Immediately clear in the maps is the increasing prevalence of pickups as the target duration increases (10, 9, 14, and 19, respectively). Intuitively, an increased number of armor and health pickups makes sense as a design choice to make the game longer, as they extend the lifetime of a player. Perhaps the damage bonuses are increased as a countermeasure for the other pickups, in order to prevent the game from going on indefinitely. However, the total increase of pickups is not desirable at this rate. For example, with an almost doubled number of pickups compared to the initial design, the long map has become convoluted and gameplay will likely be chaotic. Moreover, several of these pickups will cancel each other out.

Another interesting aspect is the difference in the placement of powerups. In the short map, the pickups are placed on the diagonal between the two bases; thus providing easy access to resources and confining the gameplay to a limited space. In the medium map, the pickups are more spread out, with similar distances from both bases to the life preserving pickups. In the long map, there will always be a resource nearby, but the distribution of types is not balanced. Player one has easy access to damage bonuses, while the base of player two is surrounded by armor. Yet, this imbalance in resources does not negatively affect the balance of the kill ratio.

Considering all individual evolved maps in the experiment, regardless of which class matchup they were using, we observe that several structural metrics have strong and significant correlations with the actual match duration, based on the ground truth. The absolute highest correlations were the following: positive correlations with the A* distance from base of P2 to base of P1 ($r = 0.16$), with the number of damage boosts on the ground floor ($r = 0.21$), with the average Euclidean distance of all pickups to the map's center ($r = 0.25$) and with the average Euclidean distance of each pickup to its nearest neighboring pickup ($r = 0.26$). On the other hand, there were strong negative correlations between ground truth duration and the number of healthpacks on the ground floor ($r = -0.18$), with the number of damage boosts on the first floor ($r = -0.18$), with the shortest distance of the base of P1 to any damage boost ($r = -0.17$) and the total number of healthpacks ($r = -0.16$). It is surprising that healthpacks have a negative correlation with duration,

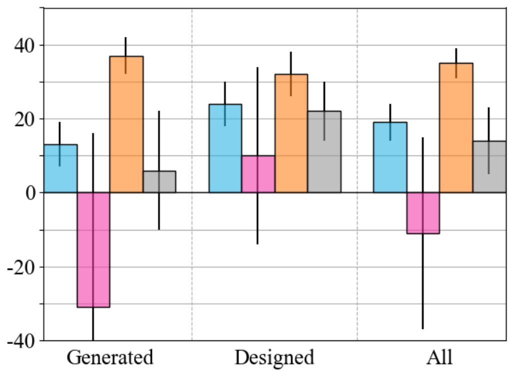


Figure 7.9: The improvements of the evolved maps with respect to the initial seeds for each of the three durations, short (blue), medium (fuchsia) and long (orange), and all durations together (gray). Results include the 95% confidence interval.

as it would be expected that more healthpacks would make the classes more survivable. A possible explanation could be that a higher number of health packs increases the odds that the stronger player recovers their health before the other player has respawned, thus starting the next fight with the same odds instead of a health disadvantage. Additionally, the discrepancy between damage boosts on the ground floor and on the first floor may point to an agent behavior that prefers to stay on the ground floor and thus damage boosts on the first floor are rarely chosen. A more thorough analysis of the agents' behaviors using heatmaps from play traces (and ideally gameplay videos) was considered to be outside the scope of this thesis, but could be interesting future work.

7.3.4 Performance with Different Initial Seeds

In order to test the generality of surrogate-based level generation, the same generation methods were applied using a broader range of levels as initial seeds. Unlike the in-depth assessment of the handcrafted level above, a high-level analysis is provided using O and E of Eq. (7.3) and Eq. (7.4) as performance metrics. Ten game levels are used, each of which acts as an initial seed for 75 evolutionary runs: one run for each of the 25 TF2 class matchups and each of three intended durations. The ten levels correspond to the generated levels G1-3, G6 and G7 and the designed levels D1-4 and D6 of Section 7.1.3. These levels were selected such that they cover a wide variation of paths between bases, ratios between level geometry and both presence and placement of pickups. These initial levels and a sample of the evolved levels for a specific class pair and match duration are shown in Fig. 7.10 and Fig. 7.11. The evolved map with the best overall improvement (O) is shown, but it is interesting to note that not all maps are changed much from their initial states. In many cases, there are minor architectural differences while most changes are focused on the number and type of powerups (e.g., Fig. 7.11j and 7.11f). In other cases the powerups largely remain the same, but entire pathways are blocked off (Fig. 7.10j and 7.11g) or balconies or galleries introduced (Fig. 7.10j).

The improvements, depicted in Fig. 7.9, show some similar trends as in Section 7.3.2 and some differences. On the one hand, due to the large number of initial maps being tested, improvements were highly varied. Each initial map had a different duration and balance per class matchup and thus its improvements could be minor or negative. With generated levels as initial seeds, in particular, many matches had durations very close to the intended medium duration (causing the negative O values). It is not surprising that generated levels generally had match durations close to 300 seconds, given the distribution in the original dataset. Despite a few extremely negative O values, in almost all cases the positive improvements were more than the negative ones per initial level and duration individually, and on average[2]. The only exception is for the 25 levels evolved for short duration based on G2 (9 positive versus 16 negative improvements).

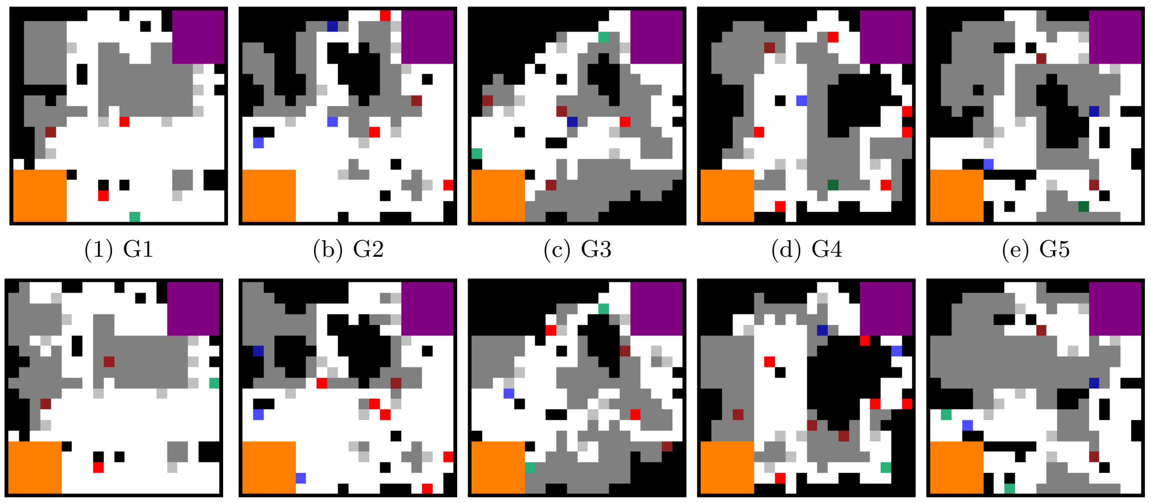
Averaged over all maps the prediction error lies around 0.15; with $E = 0.13$ and $E = 0.17$ when averaged over generated and designed maps respectively. It is not surprising that using a designed level as an initial seed is more challenging than using a generated one, since generated levels have patterns closer to those learned by the CNN model. For a similar reason, one might expect that the prediction error of the model is lower for medium durations; as it has seen a larger variety of those in the training set. However, there are no significant differences in E between the targeted durations.

7.3.5 Summary

Based on the improvements and the diversity of maps observed in this experiment, it is evident that the surrogate model can drive a map generation algorithm to adjust maps to specific class matchups with some degree of success. Though failures occur when the initial map was already very close to the intended gameplay. Especially in the experiments of Section 7.3.4, some initial seeds matched the intended balance and duration almost perfectly. Due to the random mutations evolution explored away from that, resulting in worse maps than the initial design. Stopping evolution when maps can not be improved further would enhance the results. The results are still fairly consistent: most evolved maps have an actual shorter duration when evolving for a short duration than when evolving for a medium or long duration and vice versa.

Considering the level structures, we have seen an increased number of pickups in the evolved maps (e.g., Fig. 7.8 and Fig. 7.11). Despite the high improvement in terms of metrics, some maps become convoluted and a designer would struggle with imagining the gameplay. This decreases the odds of a designer appreciating this as a design suggestion. It also hints at the evolutionary algorithm successfully exploiting the values that the surrogate model attributes to each pickup and moving to a map that falls outside the training distribution of the model (by having more than one pickup per cell). From both perspectives it would be good to constrain the number of pickups per map if this is to be implemented as a design tool.

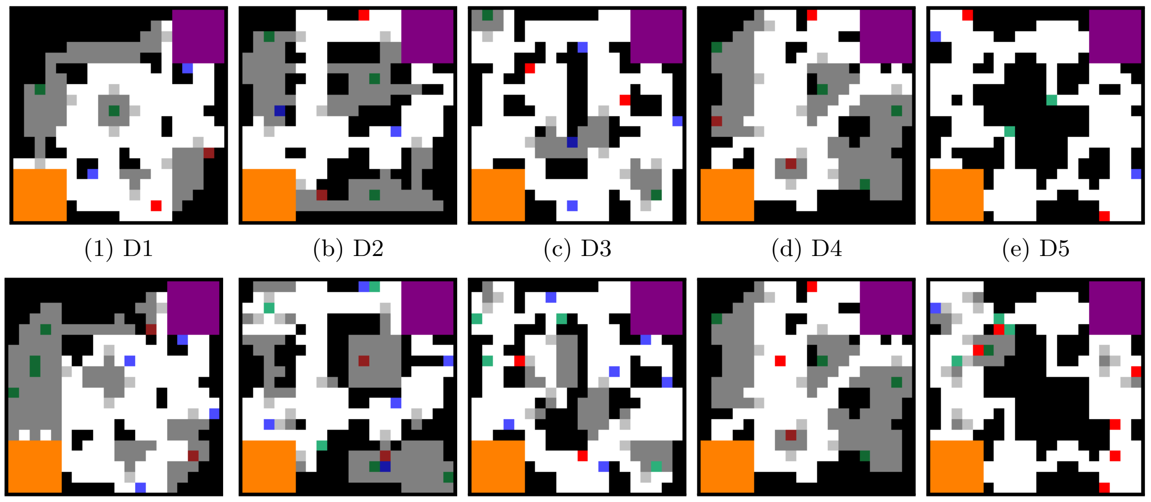
[2]There are 185 positive versus 63 negative in short matches, 173 versus 74 in medium matches, 222 versus 22 in long matches



(1) G1 (b) G2 (c) G3 (d) G4 (e) G5

(f) Scout vs Heavy (M), $O = 74\%$ (g) Heavy vs Heavy (M), $O = 89\%$ (h) Spy vs Spy (L), $O = 88\%$ (i) Spy vs Pyro (L), $O = 100\%$ (j) Heavy vs Scout (M), $O = 93\%$

Figure 7.10: Generated levels used as initial seeds (top) and best evolved maps in terms of improvement (O), based on each initial map (bottom). Intended durations are shown as M (medium) and L (long).



(1) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) Heavy vs Spy (L), $O = 96\%$ (g) Scout vs Pyro (L), $O = 93\%$ (h) Pyro vs Scout (L), $O = 100\%$ (i) Scout vs Scout (M), $O = 81\%$ (j) Spy vs Spy (M), $O = 95\%$

Figure 7.11: Designed levels used as initial seeds (top) and best evolved maps in terms of improvement (O), based on each initial map (bottom). Intended durations are shown as M (medium) and L (long).

7.4 Facet Orchestration: SO-EA vs MO-EA

The previous experiments consider a wide variety of pairings between classes to give a broad overview of the changes the algorithm can make in the design space of either the levels or the character classes. This section is instead aimed at comparing various content generation approaches applied to the same set of initial designs. More specifically, the approaches differ in the content that they can adjust (classes, maps, or both) and each of the experiments compares the use of a single, aggregated fitness function with multi-objective evolution as a means to find the desired gameplay. All generative algorithms in this section use the predictions of CNN3 of Table 6.1 as part of their fitness evaluations. The set of initial designs consists of the ten designed levels D1-D10 of Fig. 7.3 and one specific class pairing.

The choice for this pairing was based on a preliminary experiment. Simulations were run on the levels D1-D10 to derive the gameplay outcomes in pairwise matchups of the Scout, Heavy and Sniper classes. While the Sniper class dominated every opponent in every level (the kill ratio of the Sniper was over 0.90), the matchups between Scout for player 1 and Heavy for player 2 were imbalanced but diverse. Fig 7.1a shows the different kill ratios and match durations per level based on 30 simulated playthroughs. Fig 7.1a shows that while the Scout always has an advantage against the Heavy class, this differs from level to level. Moreover, the match durations were almost evenly distributed between medium (40%) and long durations (60%). This pairing was selected for further experiments since it was evident that changing the level itself could influence the player balance (unlike with the Sniper class), while the lack of short matches raised a challenge for the generator to tackle.

Initialization

The goal of the proposed system is to adapt an initial design. But an evolutionary algorithm requires a suitably diverse population in order to do global search, as a homogeneous population might disrupt the search process. It is therefore important that the individuals in the initial population are sufficiently different from the original design and each other, but not so much that we might as well initialize randomly.

As such, the initial populations in this section are created with a single mutation of the provided design. As indicative example of the resulting diversity, we compare ten populations that are based on a human design with ten random populations from our generated dataset. The map populations are based on D9 of Fig. 7.3, which is the map that is used as an example in Fig. 4.3 and Section 4.2.5. The class populations are based on the matchup between Scout and Heavy, which is used to explore orchestration in Section 7.4.3. The comparison is based on the above mentioned class distance C (4.5) and the total number of different tiles in a map. These metrics are reported with their 95% confidence intervals.

The distance of a random population of maps towards D9 is 207 ± 18 tiles, while the mutated populations differ 9 ± 2 tiles. The pairwise distances between the individuals in a random population is 204 ± 18, while that of the mutated population is 8 ± 3 tiles. The distance in a homogeneous population is 0 and the maximum possible distance between any two maps is approximately 323. Based on this, we can conclude that the mutated maps have some diversity, but are still close to the initial map, while random maps are too far away to be meaningful alternatives for a designer.

The distance of a random population of classes to the initial match is 2.65 ± 0.3, while the mutated populations have a distance of 0.3 ± 0.34. The pairwise distances within the random population average to 2.16 ± 0.3, and 0.51 ± 0.42 for the mutated individuals. Given that the maximum distance between any two of these vectors is 16, the two types of class populations seem closer together than the maps. Yet, the mutated population is clearly closer to the initial design than the random population.

The orchestrated populations have a 50% chance of mutating either the map or the classes. This is reflected in their lower distances to the initial designs, 2 ± 1 and 0.16 ± 0.27 for maps and classes respectively. Similarly, their pairwise distance to each other is lower than a single-faceted population, with a distance of 4 ± 0 and 0.29 ± 0.35 respectively.

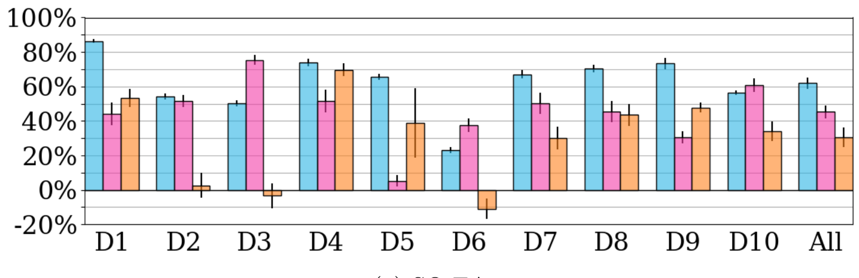
The mutated population does not include outliers which could drive evolution more efficiently (as a global search algorithm) and it is expected that a random initialization could lead to better results in terms of creating content with the desired gameplay targets. However, that broader search is also expected to ignore the designer's work. This would lean towards a fully-automated pipeline and does not satisfy our need for an adaptation of the human-authored content. All in all, the mutated population provides a better trade-off between being close to the initial design and maintaining a degree of diversity.

7.4.1 Class Generation

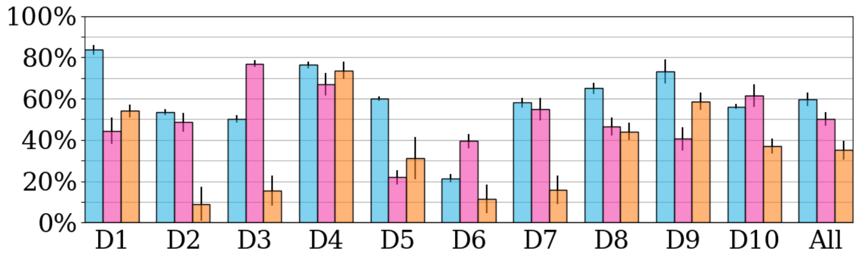
The first experiment concerns the fine-tuning of the classes for each of the ten selected maps. Player one's initial class is the Scout, while player two is seeded with the Heavy. The experiment compares the results of evolving via single-objective and multi-objective evolution. The desired gameplay outcomes are balanced matches (i.e., desired kill ratio $d_{KR} = 0.5$) of short, medium and long duration. The normalized values for the desired durations (d_t) are respectively 0.14, 0.36, and 0.72. An evolutionary run consists of a population of 100 individuals which are evolved for up to 100 generations. The run was stopped after 10 generations without improvements in terms of best fitness ($\epsilon = 10^{-5}$) or when the fitness was approximately zero ($\epsilon = 10^{-6}$), as the latter indicates the achievement of the design goal. The evolved classes are collected from ten evolutionary runs per map and target gameplay outcomes (for a total of 300 evolved classes per EA), and the actual gameplay outcomes of these classes for the map they were evolved in are calculated from 30 simulated playthroughs.

Figure 7.12 shows the percentage improvements over the initial classes on a per level basis and across all levels. It is evident that the evolved classes perform closer to the desired target in general, although the improvement is significantly higher in short or medium match targets than in long match targets for both MO-EA and SO-EA. On the one hand, for D4 the improvement is consistently high for all target durations (over 50% for SO-EA and over 60% for MO-EA). On the other hand, long matches for D2, D3 and D6 are rarely improved in both SO-EA and MO-EA. In fact, the evolved classes tend to move the gameplay away from the desired values on D3 and D6. For these maps the initial classes' match duration was already quite close to the desired long duration (see Fig. 7.1a) so the lack of improvement is to a degree expected. However, this is less obvious for map D5, for which the initial classes have very similar gameplay outcomes to D10 (see Fig. 7.1a) but class evolution in D10 manages to improve by at least 60% on medium matches.

To identify how the classes are adjusted towards different target durations, Fig. 7.13 shows the average difference from the original class parameters averaged across all ten levels. Trends seem to be consistent across SO-EA and MO-EA, while there are some interesting differences in how each class changes. For instance, in all cases both the Heavy and the Scout (which are both short-range classes) see an increase in range; which is, however,



(a) SO-EA



(b) MO-EA

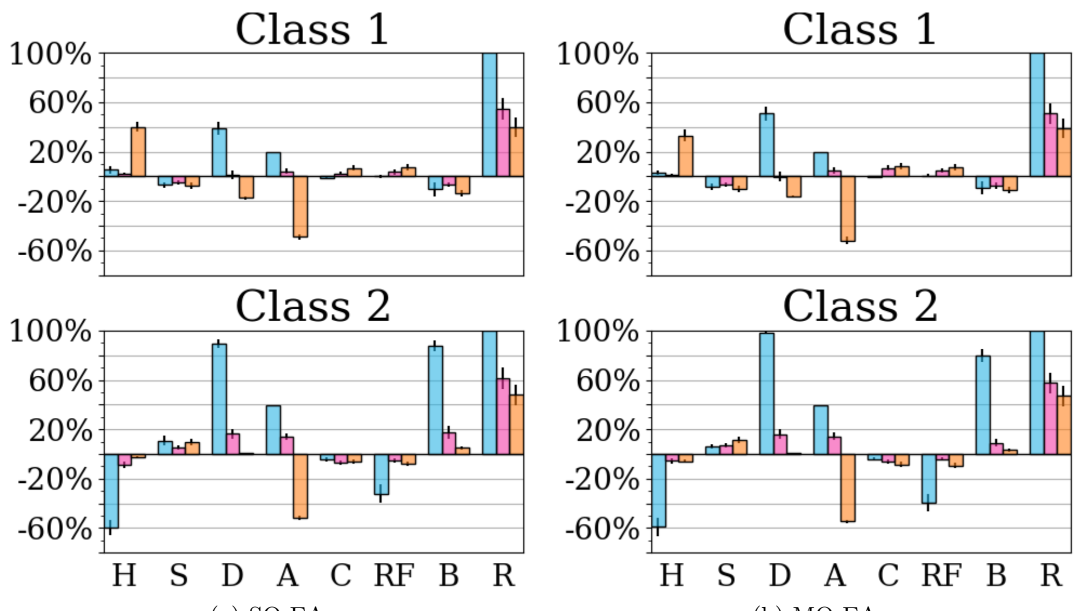
Figure 7.12: **Class Adaptation:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per map of Fig. 7.3. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.

less pronounced in long matches. Similarly for both classes the accuracy increases in short matches and drops in long matches. Some of the parameter changes make sense considering the initial values for the two classes: for instance, the Heavy class has the most HP and lowest damage per bullet of all TF2 classes and thus it is understandable that in short matches its HP is reduced substantially (to reduce survivability) while damage is increased (to increase the threat to the opponent). In that regard, these trends are consistent for the Scout which had few HP to begin with and these are increased in long matches to increase survivability. All changes are therefore intuitive, with long matches favoring classes with many hit points and low accuracy while short matches favoring high-damage, long-range classes.

This does raise the issue of diversity of the results. The different classes exist for various reasons, e.g., diverse gameplay. In both experiments, we have seen that the evolved classes tend to converge to a Sniper class when evolved for short durations; even with the MO-EA approach. In terms of game design, this convergence is not a desirable result. Changes to the algorithm are needed if this is to be implemented as a practical design tool, either by placing additional constraints making a different selection from the Pareto-front, or perhaps by applying a quality-diversity algorithm. On the other hand, one can imagine that there is only so much you can do when you can only change the classes. We will see in Section 7.4.3 that the effect is less pronounced when the maps and classes are orchestrated.

7.4.2 Map Generation

The second experiment uses the same setup as before, but now the maps are evolved while the two classes (i.e., the Scout and the Heavy) remain the same. Each of the designed maps of Fig. 7.3 is used as an initial seed and the desired gameplay target is again a balanced



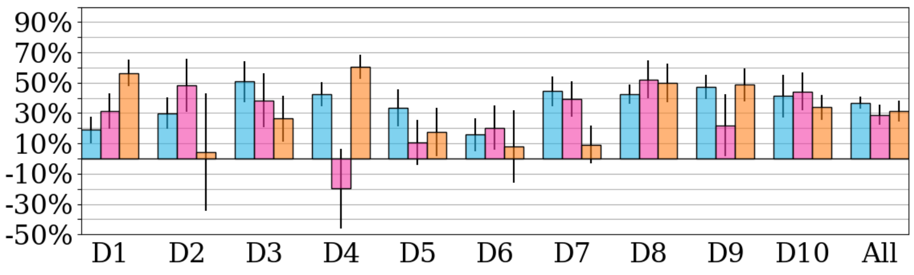
(a) SO-EA (b) MO-EA

Figure 7.13: **Class Adaptation:** Parameter changes of the two classes evolved for balanced matches of short, medium and long duration. Values show the difference of each parameter of the evolved class compared to the initial class (Scout for P1, Heavy for P2): hit points (H), speed (S), damage (D), accuracy (A), clip size (C), rate of fire (RF), bullets per shot (B) and range (R). Values are averaged from 100 evolutionary runs (10 runs per map of Fig. 7.3); error bars show the 95% confidence interval.

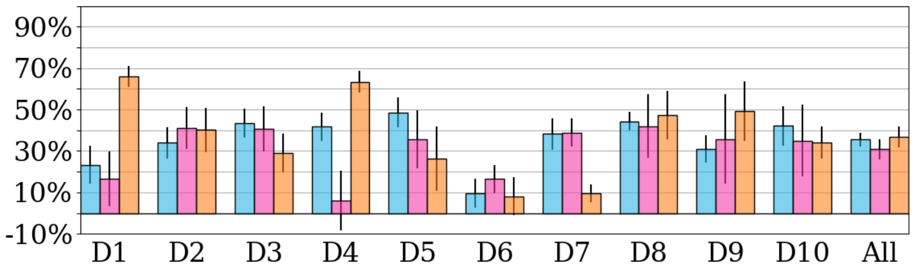
match of one of the three durations. An evolutionary run consists of a population of 100 individuals which are evolved for up to 100 generations. The run was stopped after 10 generations without improvements in terms of best fitness ($\epsilon = 10^{-5}$) or when the fitness was approximately zero ($\epsilon = 10^{-6}$). Ten independent evolutionary runs are performed for each of these settings. This results in 300 fittest maps for SO-EA and another 300 fittest maps for MO-EA; the actual gameplay outcomes are computed on these maps using the initial TF2 classes in 30 simulated playthroughs.

Figure 7.14 shows the percentage improvements over the initial levels. Overall the gameplay improvement is less than that obtained by the evolved classes. Unlike improvements in evolved classes in Fig. 7.12, the patterns are less consistent. Additionally, the evolved maps with negative improvements are more severely impacted than the evolved classes. Based on the overall improvements, evolving with MO-EA improves the levels more than SO-EA for all target durations, though the difference between the results is small. Some similarities in the results of SO-EA and MO-EA outcomes are that D6 is hardly improved for all durations, both algorithms struggle with improving D7 towards a long duration and D4 to a medium duration. However, MO-EA does tend to create more offspring with positive improvements for those maps. Notable differences between SO-EA and MO-EA can be found in D2 and D9 (with respectively a long and a medium duration target), where MO-EA obtains a significantly larger improvement. Similarly, MO-EA produces much better maps based on D5 than SO-EA for all targets.

To understand why the two EA approaches differ in terms of outcomes, Fig. 7.15 shows



(a) SO-EA

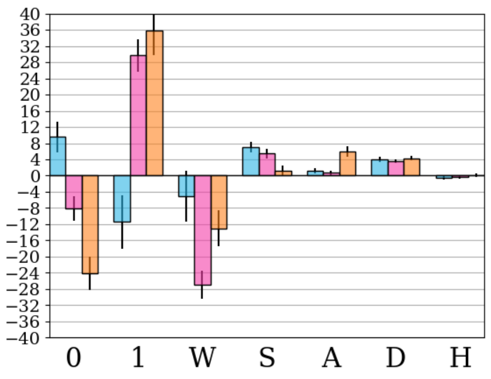


(b) MO-EA

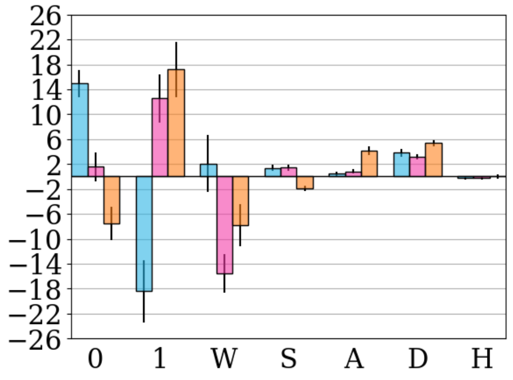
Figure 7.14: **Level Adaptation:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per level of Fig. 7.3. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.

how the number of different tiles (e.g., healthpacks) change from the initial level, averaged across all ten initial seeds. The changes by SO-EA and MO-EA display similar patterns, but the changes made by MO-EA are less profound. It is evident that in both SO-EA and MO-EA the number of ground-floor tiles increases when targeting short matches, to the expense of first-floor tiles. Medium and long matches see an increase in first-floor tiles. In medium matches this comes primarily at the expense of walls, while long matches also see a reduction of ground-floor tiles. This reduction of walls results in larger levels in terms of places that can be visited by the players. Simultaneously, first floor tiles can keep the chance of encounters lower than ground-floor tiles by blocking sight lines. Interestingly, for SO-EA the smaller number of first-floor tiles in short matches does not result in a decrease in the number of stairs but rather the opposite. The reason will be evident in the next paragraph. Finally, while the number of healthpacks and double damage powerups per level does not fluctuate on average from the initial values, the number of armor powerups increases substantially when evolving levels for longer matches. Similar to the class changes in Fig. 7.13, where classes tailored to longer matches had more hit points (HP), armor powerups artificially increase players' HP and are favored for long matches to increase players' survivability.

In order to better understand how levels can be adjusted towards balanced matchups of different durations, the most improved levels across target durations are shown in Fig. 7.16 and 7.17. We observe that for short matches (Fig. 7.16b and 7.17b) the top-left and bottom-right corners tend be devoid of powerups and either walled or empty, which makes them impossible or at least unattractive to visit. Scattered across the level are still several small 'chunks' of first-floor tiles (of one or two tiles) which need to be connected via one or more

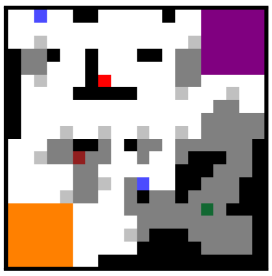


(a) SO-EA

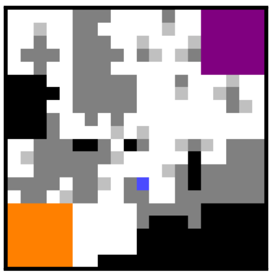


(b) MO-EA

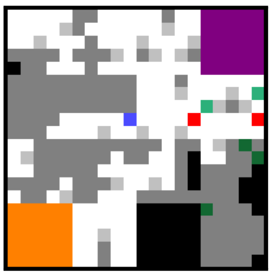
Figure 7.15: **Level Adaptation:** Changes in maps evolved for balanced matches of short, medium and long duration. Values show the difference in the number of tiles of the evolved levels compared to the initial level used as a seed: ground-floor tiles (0), first-floor tiles (1), walls (W), stairs (S), armor powerups (A), double damage powerups (D) and healthpacks (H). Values are averaged from 100 runs; error bars show the 95% confidence interval.



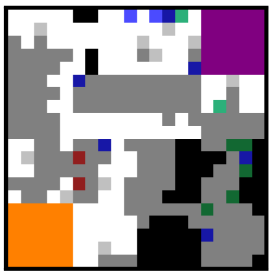
(a) Initial
$\boldsymbol{i} = (0.77, 0.53)$



(b) Short
$\boldsymbol{a} = (0.62, 0.31)$
$O = 56\%$



(c) Medium
$\boldsymbol{a} = (0.58, 0.35)$
$O = 75\%$



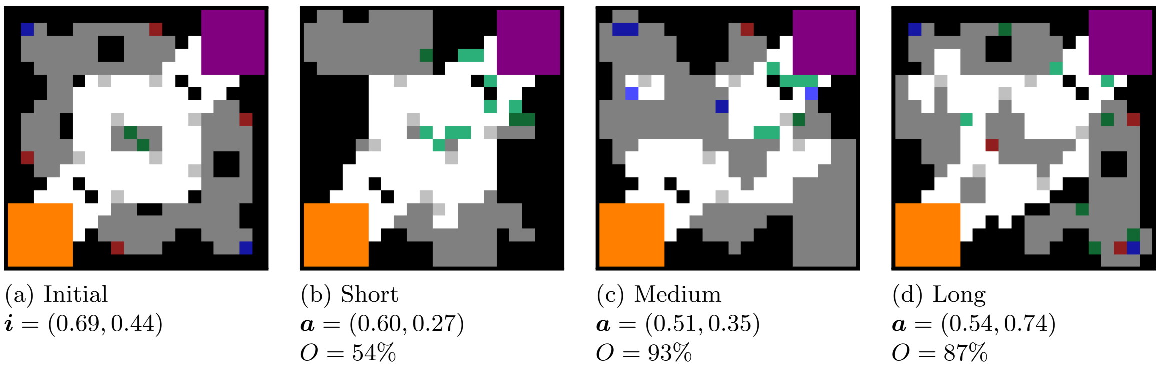
(d) Long
$\boldsymbol{a} = (0.59, 0.77)$
$O = 69\%$

Figure 7.16: **Level Adaptation:** Most improved levels for D8, via SO-EA.

stairs due to the constraints described in Section 4.2.5; this explains the prevalence of stairs even when first-floor tiles are few as observed in Fig. 7.15.

In terms of broader level structures, it is evident that levels adjusted towards short matches have a clear path on the ground floor between the two bases, while levels evolved towards long durations are denser, with more complex paths between the bases. Compare, for example, the first floor structure near the purple base in Fig. 7.16a with the evolved levels. It is removed in the short level, Fig. 7.16b, and instead extended in Fig. 7.16d. Note that the left path takes player 2 all the way around that structure, while the right path brings them to a first floor where the view to the orange base is blocked by walls. A similar removal and extension of a structure on the path between the two bases occurs in Fig. 7.17.

In terms of gameplay objects, we observe more double damage powerups in short matches (e.g., in Fig. 7.17b) and more armor powerups in long matches Fig. 7.16d). Level generation also attempts to correct for the imbalance between the classes: the Heavy class that spawns in the purple base has better access to powerups (e.g., double damage powerups in Fig. 7.17b,


(a) Initial
$\boldsymbol{i} = (0.69, 0.44)$

(b) Short
$\boldsymbol{a} = (0.60, 0.27)$
$O = 54\%$

(c) Medium
$\boldsymbol{a} = (0.51, 0.35)$
$O = 93\%$

(d) Long
$\boldsymbol{a} = (0.54, 0.74)$
$O = 87\%$

Figure 7.17: **Level Adaptation:** Most improved levels for D9, via MO-EA.

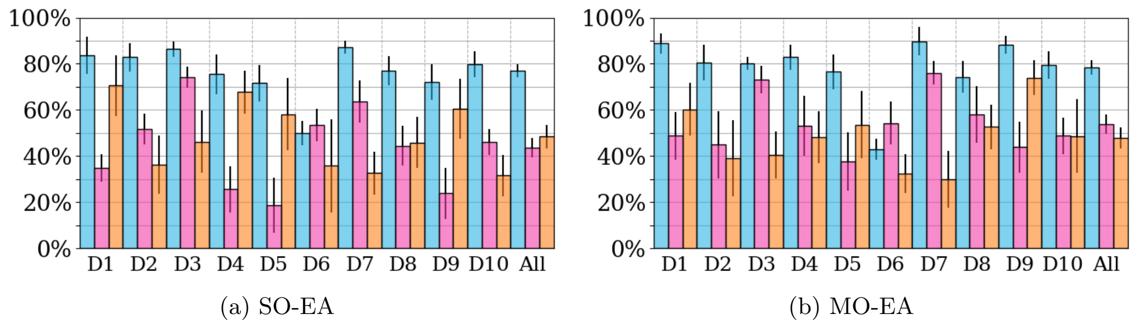


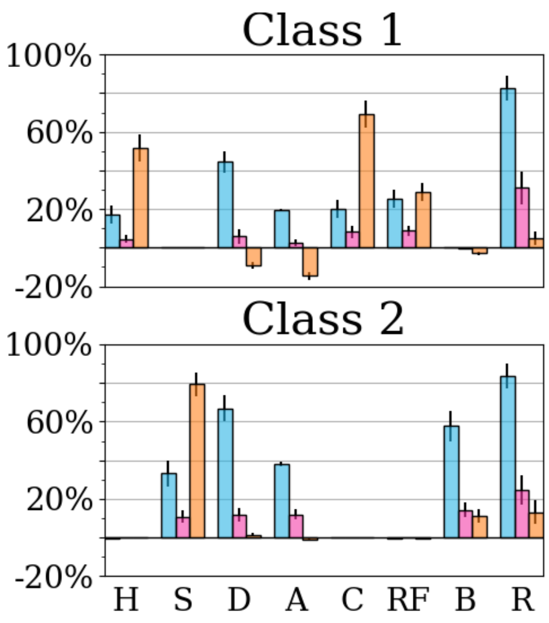
Figure 7.18: **Orchestration:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per map of Fig. 7.3. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.

7.17c, 7.16c and 7.16d) than the Scout in the orange base. The kill ratio of the Scout was much higher than the Heavy in the initial levels (e.g., $i_{KR} = 0.77$ in Fig. 7.16a), but this imbalanced distribution of powerups near one base improves the balance between classes.

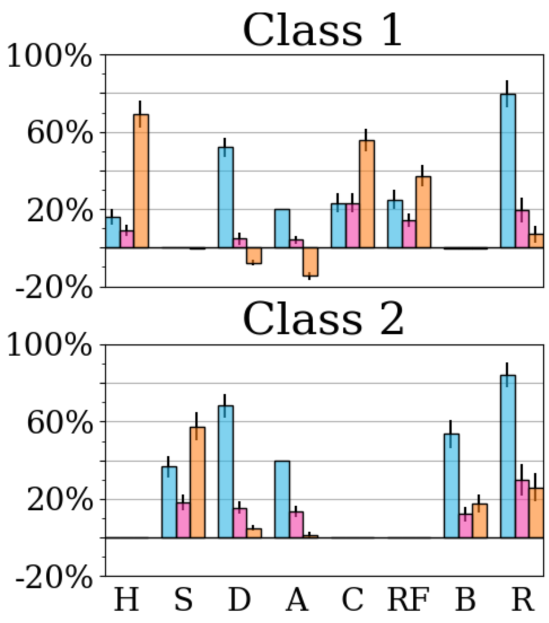
7.4.3 Orchestration

A core question posed in this thesis is how the generation of both facets (rules and levels) can be orchestrated via the surrogate model that maps them to gameplay outcomes. In this experiment the generative process adjusts both the initial level and the initial classes that compete in it. As in the previous experiments in this section, 10 evolutionary runs attempt to adjust an initial population of Scout and Heavy matchups in the same level, towards a desired gameplay outcome. One evolutionary run consists of 100 individuals which are evolved for up to 100 generations; early stopping is applied when lack of progress or goal achievement is detected. With 10 initial levels and three different target gameplay outcomes, a total of 300 evolved levels and classes are collected per EA.

Figure 7.18 shows the improvement over the initial setup. It is evident that on average the improvements are higher when targeting short matches, in part because the initial matchups had longer durations (see Fig. 7.1a). This behavior is consistent with class generation experiments summarized in Fig. 7.12, although the improvements themselves are significantly higher when both levels and classes are adjusted, on average.



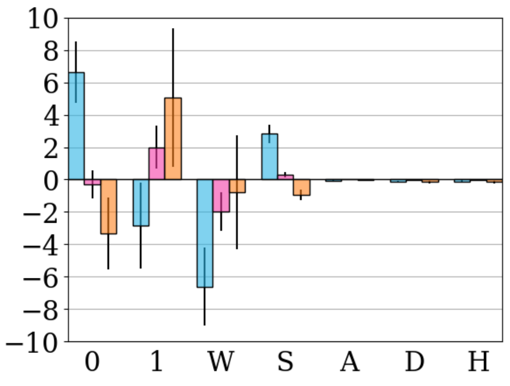
(a) Class changes (SO-EA)



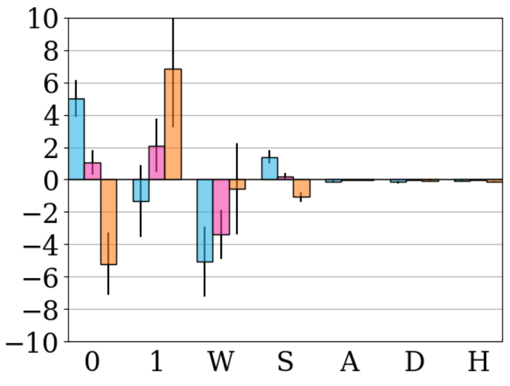
(b) Class changes (MO-EA)

Figure 7.19: **Orchestration:** Parameter changes of the two classes evolved for balanced matches of short, medium and long duration. Values show the difference of each parameter of the evolved class compared to the initial class (Scout for P1, Heavy for P2). Parameters' notations are the same as Fig. 7.13.

On the degree to which levels and classes are adjusted, Fig. 7.19 and Fig. 7.20 show the parameter changes from the initial seeds for class parameters and levels' tile counts. It is clear that there are few substantial differences between SO-EA and MO-EA in terms of parameter changes or level changes. It is also evident that changes in terms of levels are much more subdued than in level-only generation (i.e., Fig. 7.15), although similar patterns prevail (more ground-floor tiles and stairs for short matches, more first-floor tiles for long matches). However, the number of powerups does not change from its initial values in most cases: indeed, none of the three powerup counts changed in at least 91% of both the SO-EA and MO-EA experiments. The more subdued level adjustments are in part due to the crossover operators for the orchestrated approach which differ from level generation. Classes similarly do not change as much as when evolving on their own (i.e., Fig. 7.13). While there are no negative corrections (e.g., lowered HP for the Heavy class in Fig. 7.13), some familiar patterns remain, such as increased damage and range for short matches. Notably, four class parameters stay unchanged in at least 94% of SO-EA runs, while the same is true for five class parameters in at least 97% of MO-EA runs.



(a) Map changes (SO-EA)

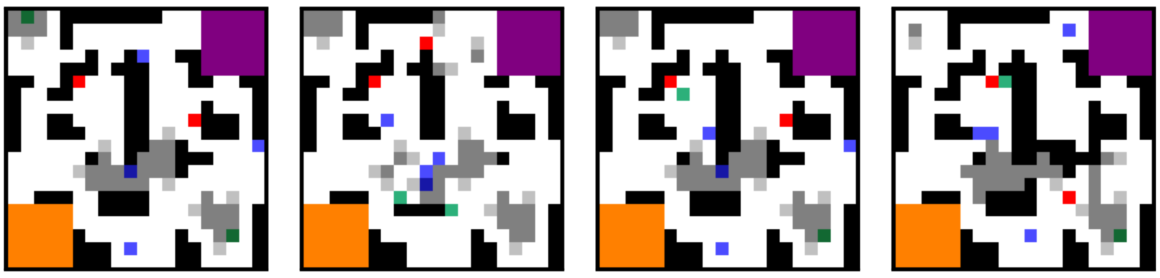


(b) Map changes (MO-EA)

Figure 7.20: **Orchestration:** Changes in maps evolved for balanced matches of short, medium and long duration. Values show the difference in the number of tiles of the evolved maps compared to the initial map used as a seed. Values are averaged from 100 evolutionary runs (10 runs per map of Fig. 7.3); error bars show the 95% confidence interval.

As indicative examples of how orchestrated evolution can improve matchups towards the intended gameplay outcomes, Fig. 7.21 shows the SO-EA results with the highest improvements for D3 and Fig. 7.22 shows the MO-EA results with the highest improvements for D4. D3 and D4 were chosen as they have the highest average improvements across all durations for SO-EA and MO-EA respectively. Familiar trends as in Section 7.4.2 are observed, e.g., first-floor chunks made up of one tile in short matches, and blocking paths near the diagonal between the two bases in long matches. However, it is also evident that changes in the level architecture are less pronounced and powerup placement does not particularly favor one player's base. Most of the tweaks therefore are performed on the classes rather than on the level. Indeed, SO-EA increases the accuracy of both classes, regardless of duration and increases the damage of both classes for short and medium durations. The Heavy class' parameters are boosted more than those of the Scout and it also sees an increase in the number of bullets per shot. On the other hand, MO-EA combines changing the damage of both classes mostly with changing the clip size of the Scout and the speed of the Heavy.

For short and long durations, the Heavy class is modified more heavily than the Scout class. Unlike the findings in Section 7.4.1, however, for short matches neither class evolves to match the Sniper archetype in TF2. In fact, the class of the first player stays closest to the Scout in five out of these six evolved matchups.



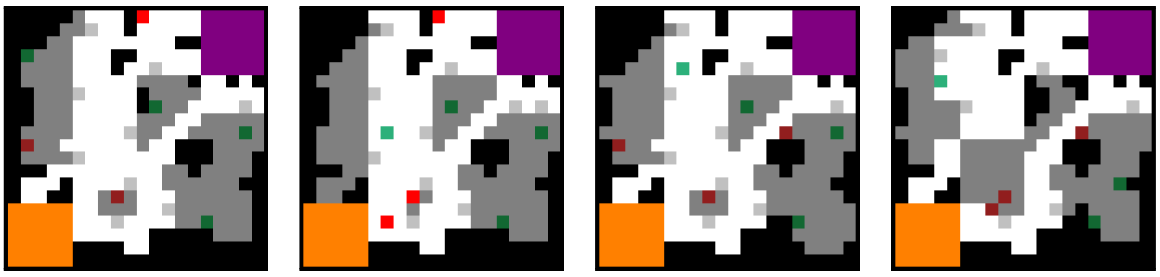
(a) Initial $i_t = 0.79$, $i_{KR} = 0.73$

(b) Short $a_t = 0.14$, $a_{KR} = 0.54$, $O = 94\%$

(c) Medium $a_t = 0.28$, $a_{KR} = 0.51$, $O = 84\%$

(d) Long $a_t = 0.65$, $a_{KR} = 0.51$, $O = 72\%$

Figure 7.21: **Orchestration:** Most improved levels for D3, evolved via SO-EA.



(a) Initial $i_t = 0.41$, $i_{KR} = 0.65$

(b) Short $a_t = 0.14$, $a_{KR} = 0.50$, $O = 98\%$

(c) Medium $a_t = 0.37$, $a_{KR} = 0.49$, $O = 92\%$

(d) Long $a_t = 0.81$, $a_{KR} = 0.52$, $O = 74\%$

Figure 7.22: **Orchestration:** Most improved levels for D4, evolved via MO-EA.

Table 7.4: Behavioral differences from the original and the predicted. Results are averaged from 300 evolutionary runs across the 10 maps and include the 95% confidence intervals.

Generator	$dist(\boldsymbol{p}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{d})$	$dist(\boldsymbol{a}, \boldsymbol{p})$
SO-EA class	0.029 ± 0.007	0.188 ± 0.011	0.160 ± 0.008
MO-EA class	0.028 ± 0.007	0.184 ± 0.011	0.158 ± 0.007
SO-EA map	0.052 ± 0.008	0.229 ± 0.016	0.182 ± 0.012
MO-EA map	0.081 ± 0.010	0.234 ± 0.014	0.169 ± 0.008
SO-EA both	0.015 ± 0.003	0.138 ± 0.009	0.140 ± 0.008
MO-EA both	0.028 ± 0.006	0.134 ± 0.009	0.127 ± 0.008

7.4.4 Analysis

Based on the results of the previous sections, there are some interesting differences in terms of the performance of evolution on a single game facet (class parameters, shooter levels) and when combined. Table 7.4 summarizes how the different experiments compare in terms of the distance between the desired ($\boldsymbol{d}$), the actual ($\boldsymbol{a}$) and the predicted ($\boldsymbol{p}$) gameplay outcomes. The prediction error, which is defined in Eq. (7.4) as the distance between predicted and actual (i.e., simulation-based) outcomes, is significantly lower when both level and classes are adjusted, while the highest errors occur when only levels are adjusted. The pattern is similar for the distance between desired and actual outcomes, which is closely related to the improvement metric of Eq. (7.3). Indeed, both average improvement and average $dist(\boldsymbol{a}, \boldsymbol{d})$ is significantly lower when both levels and classes are adjusted. On the one hand, this is hardly surprising since the generator has multiple degrees of freedom and can counteract imbalances between classes through map re-design but also through parameter tweaks on the classes themselves. On the other hand, it is also evident that the predictive model is able to recognize and match tweaks in both facets with a fairly accurate estimate of the actual gameplay outcomes. The latter is an important finding which shows the benefit of orchestration both as a generative medium but also as a surrogate for gameplay simulations.

Regarding the surrogate model itself, the distance between desired and predicted shows that the predictive model is understandably more "optimistic" than the actual simulations demonstrate, but still steers generation towards the right direction. An interesting observation is the low $dist(\boldsymbol{p}, \boldsymbol{d})$ for map adjustments which is not corroborated by the $dist(\boldsymbol{a}, \boldsymbol{d})$ metric: it seems that the large size of the maps and the complex nature of their representation is more likely to lead map evolution astray than other facets.

Table 7.5 compares the initial classes and map with the final best outcomes in the different generative approaches. Unlike tile counts in Sections 7.4.2 and 7.4.3, the tile difference metric is the pixel to pixel image difference[3] which is more granular. It is evident that orchestration makes fewer changes to either class or to the level, which is corroborated by findings in Fig. 7.20 and Fig. 7.19. It is also interesting that changes to the Heavy class are more prevalent than changes to the Scout class when the system can only adjust classes. This observed difference is not significant for orchestration. Meanwhile, map adjustments are smaller for MO-EA approaches (significantly so when evolving maps alone).

[3]Pixel to pixel difference using the representations of e.g., Fig. 7.3, directly reflects the tiles on that specific location, including whether a healthpack is on the ground or the first floor.

Table 7.5: Phenotypical distance from the initial classes (C1 for Scout and C2 for Heavy) and the initial map. Results are averaged from 300 evolutionary runs across the 10 maps, and the 95% confidence intervals is included.

Generator	C1 distance	C2 distance	tile difference
SO-EA class	1.507 ± 0.080	1.719 ± 0.098	—
MO-EA class	1.496 ± 0.081	1.697 ± 0.097	—
SO-EA map	—	—	110.5 ± 3.2
MO-EA map	—	—	74.1 ± 3.5
SO-EA both	1.251 ± 0.083	1.264 ± 0.095	34.2 ± 3.1
MO-EA both	1.236 ± 0.078	1.309 ± 0.097	30.1 ± 2.4

Generally, the differences between the single-objective approach and the multi-objective approach are surprisingly small: MO-EA is slightly ahead in terms of improvement, but the differences are not significant. The highest average improvement across all initial seeds and target durations is with MO-EA on both maps and classes (59.3% ± 2.8% improvement). The real benefit of MO-EA is that the improvements (however marginal) over SO-EA are made with fewer changes to the original maps and/or classes.

7.4.5 Summary

The case study in this section investigated how a specific matchup between archetypical character classes in popular shooter games (i.e., a fast, short-range Scout class and a slow and inaccurate Heavy class) can be tailored to a more balanced gameplay outcome of a certain match duration. The experiments demonstrated that tweaks on both the character class parameters and the level can lead to a closer outcome to the desired one, while attempting to tweak character classes alone was prone to major design shifts (e.g., making every class into a Sniper class for short matches). Moreover, it is evident that both the gameplay improvements as a result of evolution and the accuracy of the surrogate model vary by initial level and by target duration. This points to further enhancements that can be made on both fronts.

In terms of the presented evolved character classes and levels, there is room for improvement. On one hand, the evolved classes are often very different from the original ones (e.g., closer to the Sniper class than to the initial classes), especially when evolving classes alone. This could be avoided in future work by having a secondary objective to minimizing distance from the initial seeds (see Table 7.5 for sample similarity metrics for classes and levels), similar to the work of Gravina and Loiacono (2015). In terms of the evolved levels, it is clear that some architectural formations seem too chaotic, while the one-tile 'chunks' of first floor tiles serve little purpose and are visually jarring. Enhancements to the repair operators for evolved levels could improve their appearance (e.g., with cellular automata using a von Neumann neighborhood (Shaker et al., 2016a) to make corners and remove the smaller 'chunks') with the caveat of more a disparate mapping from genotype and phenotype.

7.5 Discussion

An obvious shortcoming of the current surrogate model is that it is not always accurate. While the MAE values in Section 6.3 are fairly low, experiments demonstrated that in many cases both score and duration predictions deviated from their simulation-based ground truth. Some match structures were more difficult to process and resulted in less accurate gameplay predictions than others, mainly when predicting match duration (as expected from the lower R_t^2 values during validation). The biased distribution of durations in the training corpus seems to be hampering the network's ability to accurately predict values outside the medium durations around 300 seconds. The model may often estimate values near the mean duration without suffering from a high error. To a certain degree, this even seems to be a problem in the balanced dataset, indicating that another balancing method might be required.

Another effect is that some maps seem to have a bias towards certain durations. Specifically in the designed maps we observed a potential bias towards long matches. We observed higher prediction errors but a lower distance towards the desired gameplay compared to the other duration targets. Results are still fairly consistent, e.g., most evolved maps have an actual shorter duration when evolving for a short duration than when evolving for a medium or long duration. Apart from these inaccuracies when predicting duration, the model was shown equally capable of handling handcrafted levels as input —compared to using generated levels, which are similar to the ones it was trained on. This points to promising applications as a game design tool.

It is difficult to ascertain whether the inaccuracies of the model have led evolution away from more promising matches than the final (tested) ones, i.e., whether the generation process converged to a false optimum provided by the surrogate model (Jin, 2005). This could be tested by comparing the outcomes of surrogate-based generation with generation based on simulations. However, the overwhelming computational overhead renders testing the surrogate model against a pure simulation-based model unrealistic. Indicatively, one evolutionary run as described in Section 7.3 lasts 3.5 minutes on a 12-core Intel i7 processing unit; a single death match simulation (optimized to run without graphics, and other speedups) lasts 50 seconds on the same machine. Assuming that we use the mean gameplay metrics from 10 simulations for each map (as used to calculate the ground truth in that section), one evolutionary run with 20 individuals evolving for 100 generations would last 12 days; even with a single simulation per individual one run lasts 1.2 days. A bigger difference is observed in the evolutionary runs of Section 7.2. In that section, one complete run with the surrogate model lasts 50 seconds. If we use an evaluation based on 10 simulations for each class pairing, one evolutionary run with 100 individuals evolving for 100 generations would last 58 days; even with a single simulation per individual it would last 5.8 days. Clearly, an important strength of the surrogate model is its speed when it replaces simulation-based evaluations for a search-based generator.

The system seems consistent in applying certain design rules when targeting different durations: increasing the health and lowering the accuracy and damage as the target duration increases. As a result, in short matches the evolved classes tend to be similar to the Sniper, while in long matches the classes are predominantly similar to the Heavy. From a design tool perspective, it will be necessary to place additional constraints on the evolved individuals, as a frequent suggestion to turn a class into a Heavy or a Sniper will quickly become obsolete. In addition, these experiments have focused on achieving gameplay in two numerical dimensions. Whether the players enjoy having a long match due to highly inaccurate weapons is considered another matter, but one very relevant to the tool's user. It would be interesting to explore which constraints are required for player satisfaction, which changes the model makes to compensate for this and to which degree these changes can include adjustments to the map. Aside from placing constraints, more diverse results could also be obtained by alternating between the CNN and the MLP as surrogate model. The MLP favors very different designs than the CNN, although this comes at the cost of reduced duration accuracy.

Based on the improvements and the diversity of maps observed in this experiment, it is evident that the surrogate model can drive a map generation algorithm to adjust maps to specific class matchups with some degree of success. Though failures occur when the initial map was already very close to the intended gameplay. In experiments of Section 7.3.4, the initial seed matched the intended balance and duration almost perfectly, and therefore evolution explored away from that (due to random mutations in 100 generations). A notable improvement would be prematurely ending evolution when fitness consistently decreases compared to the initial match or when the best fitness does not improve for a number of generations, such as in Section 7.4.

Considering the level structures, we have seen a highly increased number of pickups in the generated maps of Fig. 7.8. Despite the high improvement in terms of metrics (95%), the map has become convoluted and a designer would struggle with imagining the gameplay. This decreases the odds of a designer appreciating this as a design suggestion. It also hints at the evolutionary algorithm successfully exploiting the values that the surrogate model attributes to each pickup and moving to a map that falls outside the training distribution of the model. In order to avoid these undesirable properties it would be good to constrain the number of pickups per map.

7.6 Summary

This chapter has presented the results obtained from the experiments done with the proposed surrogate-based procedural content generation system. Experiments were performed on both generated and handcrafted designs. Though the model was only trained on procedurally generated content, the performance was equal on both types of input. The experiments characterized the artifacts created by character class generation, map generation and facet orchestration in terms of improvements made to the initial design as well as resulting gameplay. Section 7.2 and Section 7.3 captured a broad set of inputs and generated outputs. Whereas Section 7.4 focused on a very specific scenario in order to highlight the differences between the generative approaches, including the use of single and multi-objective evolution. The results provided insight into the accuracy of the surrogate model and the patterns that it has learned.

This chapter contains several summaries of the individual experiments. More general conclusions and a discussion of how the experiments answer the research questions are provided in the next chapter. Chapter 8 also addresses several limitations of this study and proposes directions for extensions.

Chapter 8

Discussion and Conclusions

This thesis has proposed a methodology for the orchestration of multi-faceted procedural content generation via a computational model of gameplay. In order to demonstrate its functionality we created *SuGAr*, a competitive two-player shooter game with procedural levels and mechanics, and built a corpus of gameplay via game simulations with artificial agents. This corpus was used to create a mapping from levels and mechanics to gameplay metrics via deep learning. The main research question throughout this thesis has been: *How can a computational model be used to orchestrate the procedural generation of multiple game facets?* This broad question was narrowed down via a number of subquestions, *RQ1* to *RQ4* (see Chapter 1). This thesis has aimed to provide general answers to these questions in various chapters, while using the domain of first-person shooters as a case study.

The experiments in Chapter 6 have demonstrated the feasibility of deep learning to predict elements of gameplay (*RQ1*). Several architectural considerations as well as options for the inputs and outputs of such a model have been described in Chapter 4 and Chapter 6. A convolutional neural network that fuses a vector of character class parameters with a one-hot encoding of the map was shown to be the best predictor of game duration and score balance; with an R^2 of respectively 0.61 and 0.91 it was significantly better than traditional baselines. Aside from evaluating the performance of the model, an effort has been made to explain the patterns learned by the model as well as having the model explain its decision by highlighting important areas of the input.

Chapter 5 has explored patterns in the data set, revealing correlations between the map and game duration on one hand, and character classes and score balance on the other. Indeed, models based on either input alone are not good at predicting the other target. Experiments in Chapter 6 have further demonstrated that fusing both input modalities does not only allow one model to predict both targets, but improves the accuracy on both targets with respect to single input models. This proves the benefit to evaluating levels and rules in conjunction (*RQ2*).

The obtained neural networks have been deployed in Chapter 7 as approximators of the objective function in three search-based PCG systems: a generator for shooter mechanics, a level generator and a system that generates both of these facets simultaneously. Despite the model's overestimation of the improvement, the generator was generally able to improve upon the initial designs. The question of how much improvement was achieved depends on the initial distance to the desired gameplay, which was not explicitly taken into account in the generation process. The results show that the predictive model can, in principle, replace other functions entirely (*RQ*3). Yet, it is advisable to keep some simulations for inspecting the true fitness, as the accuracy of the model on some targets and the resulting uncertainty would be a concern for commercial applications. This limitation is addressed from multiple angles in Sections 8.1.2 and 8.1.3. While benchmarking the different forms content evaluation and the creation of hybrid approaches are left for future work.

The experiments in Section 7.4 have tested a form of surrogate-assisted facet orchestration by considering an orchestrated representation of two game facets within a common genotype (*RQ4*). Single-objective and multi-objective optimization algorithms did not differ significantly in their success to find content with the desired gameplay. However, when searching for multiple objectives simultaneously, less changes are made to the original design than with a single, aggregated fitness function. Similarly, when multiple facets can be adjusted in conjunction, less changes are required overall in order to obtain the desired gameplay. Assuming that a designer prefers to achieve their goals with minimal changes, we can conclude that a multi-objective, multi-faceted approach is desirable for a mixed-initiative design tool.

8.1 Limitations

In order to create a functional proof of concept, decisions had to be made to restrict the system and the scope of the applied methods. Although precautions have been taken to implement the approach as general as possible, e.g., the representation of the model is not genre-specific, it remains an open question how well surrogate-based PCG extends to other game genres. This section describes the limitations of the proposed methodology and proposes solutions to overcome them.

8.1.1 Limitations of the Dataset

Generated game content will always be judged from the perspective of human players, especially when targeting specific gameplay. A generative algorithm that uses machine learning should ideally be based on player data if that data is available. The surrogate models in this thesis, however, have been trained on synthetic playtraces, which may not always match human decision-making. The agents that were used to generate this data set met the minimum requirements but still lack certain human-like behaviors. They do not reason about the tactical behavior of the opponent like humans do, nor do they actively seek out and utilize areas in the level that are advantageous for their class (e.g., Snipers camping on the top floor or Scouts rushing and circling the opponent). For example, when agents navigated the map of Fig. 7.8a a notable discrepancy was the fact that P2 rarely exploited the armor pickups of the right side of the map, even though this was designed to be safe hiding spot.

Convergence to specific gameplay targets might be easier to obtain when the exhibited behaviors are more strongly tied to patterns in the maps and the classes, as that would make the difference between generated artifacts more salient. Moreover, content evolved for certain gameplay between agents might well be evaluated differently when played by humans. While relying on human playthroughs for the vast data required to train deep learning models is not realistic for a study of this scale, a feasible improvement would be to use playtraces from human players on a small but diverse set of levels to fine-tune the current model. This will ensure that there is enough volume via simulated playtraces to perform the initial training, while allowing for some fine-tuning based on more realistic behaviors.

In the area of video games, one hundred thousand levels for a single game is a rare sight. With the exceptions of games as *Minecraft* (Mojang, 2011) and *Super Mario Maker* (Nintendo, 2015), which revolve around level creation, large numbers of game levels will typically involve a procedural system. Yet, as we have seen in Section 6.5, such numbers are required for training a model with a good precision (or simply: for machine learning). While care was taken to create a procedural level generator with a diverse enough output (Section 5.2), the resulting data is still constrained by the constructive rules of a single procedural system, e.g., the way paths are created or how pickups are distributed.

The surrogate model is based on a certain distribution of levels and will be most accurate when new inputs resemble the training data. The further the search algorithm will diverge from this distribution, the less accurate the model will be. While the experiments with an initial design outside of the training distribution (i.e., human designed maps) did not yield significantly different evaluations, this limitation remains a concern for a purely model-based approach. This can be mitigated in the development phase by alternating phases of training and generation, during which the generated levels are added to the dataset. Alternatively, simulations can be utilized during facet generation to update the model or to complement its evaluations.

8.1.2 Limitations of the Surrogate Model

Although many models have been tried in this thesis, the scope was limited to the area of neural networks and their linear counterpart. This was motivated by the recent advances of deep learning in computer vision and video game playing (LeCun et al., 2015; Mnih et al., 2015a). Obviously, many other machine learning algorithms, such as support vector machines and decision trees, could be applied as well.

There are two main caveats of the proposed surrogate model of gameplay. From a theoretical perspective, the metrics that the model can predict are a limited representation of gameplay. Only two gameplay outcomes have been quantified, learned and targeted through surrogate-based evolutionary search. While the concept of fun will always be hard to quantify, we acknowledge that there is more to it than score balance and duration. More nuanced gameplay metrics such as the entropy of players' movement or average combat duration could be added as alternative targets and could push evolution towards balanced matchups in more than one way. Section 8.2.1 describes the model's performance on several additional gameplay metrics and other extensions to the model.

From a practical perspective, the accuracy of the model regarding duration prediction is not good enough for production. Though the thesis has established that the general duration trends of the evolved content hold, e.g., that short matches are typically shorter than medium and long matches, there have been various exceptions. Increasing the accuracy of the model is not simply a matter of training on more data, as shown in Section 6.5. While searching for different network architectures is always an option in deep learning, a more feasible strategy might be to create an ensemble of networks. This ensemble might be homogeneous, but it could also be composed of models that are fine-tuned on different parts of the target space (e.g., specialized models for short, medium and long matches) or even have different architectures. As we have seen in Section 7.2.4, the MLP picks up on different aspects of the data than the CNN. An added benefit is that an ensemble could provide an uncertainty metric of the prediction, such as the 95% confidence interval. This uncertainty could either inform a designer or be used as a decision variable for simulation-based evaluation during evolution.

Since an ensemble would be trained on the same distribution (i.e., data from the same two generators), there is still a risk that all networks will drift in the same direction when confronted with new inputs during evolution, resulting in a false value of certainty. An alternative approach to providing a measure of uncertainty could be adopted from the field of curiosity-driven reinforcement learning. Specifically, the notion of *intrinsic reward*, a form of feedback that is supposed to drive exploration and does not depend on the goal the agent is trying to achieve. This is typically achieved by computing the novelty of each observed state. An algorithm that is particularly environment-agnostic is *Random Network Distillation* (Burda et al., 2019). Its core concept is that there is a fixed random network that produces an output for each state and a predictor network with the same architecture that is trained on the output of the random network via mean-squared error. Since it is trying to learn a random function, the predictor network is only successful when an input matches previously seen data. In our case, Random Network Distillation could be executed while training the surrogate model. Implicitly, the model is then treated as an agent in an environment with the training data (i.e., maps and classes) as states. The result is a very pure metric of what lies in the training distribution. In contrast to the ensemble method, there is no chance on a false agreement between networks during evolution, because the predictor network was not trained to model the target domain.

8.1.3 Limitations of the Generation Process

Although the improvement towards desired gameplay achieved by the orchestration process in Section 7.4.3 was satisfactory, a surprising result was that very few changes were made to the number of pickups in the map. In fact, this was an unintended side-effect of overriding the map crossover operator, which was responsible for increasing and decreasing the number of powerups in a map by swapping cells after a *Move Pickup* mutation had occurred. In light of having crossover for big steps and mutation for small steps in the search space, the orchestration process swapped level and classes as crossover operator; leaving the mutation operators of the individual facets for small steps. Yet, the map mutation operator can only decrease the number of powerups by converting walkable space into a wall. To increase the expressivity of the generative orchestration process, a solution would be to have a 50% chance of applying the crossover operator of either facet, as was implemented for mutation. Alternatively, the current operator could be kept as a third option.

Throughout this thesis, the accuracy of the computational model has been covered several times and suggestions for improvement have been given from a machine learning perspective. Yet, this limitation can also be addressed from the angle of evolutionary computing. A model, however perfect it seems, will by definition be an approximation. During evolution it is possible that mutations and crossover lead to genotypes outside of the data distribution that the model was trained on. When this occurs, it is likely that the prediction accuracy of the model decreases. *Evolution control* deals with this by keeping simulations as part of the evaluation procedure. There are various strategies for combining simulation-based evaluations with a surrogate model, such as individual-based, generation-based or adaptive evolution control (Jin, 2005; Jin et al., 2018).

Although the importance of evolution control is recognized, the approach in this thesis does not use the true fitness function during evolution; i.e., the fitness of the individuals was only based on evaluations by the model and not on any actual game simulations. One reason for this is that the game environment (Unity) and the deep learning framework (Tensorflow) were developed for different operating systems (Windows and Linux respectively).

Respective versions for the other operating system do exist, but these alternate versions were both highly unstable when the software for this thesis was developed. As such, we chose to keep the software for both frameworks on their own platform, using files to communicate between the systems. Secondly, with the response time of a design tool in mind, the game simulations (one minute per evaluation) were considered to be too expensive to run despite suggestions in the literature that this can lead to convergence to a false optimum.

A possible direction for future work could be to soften this constraint and try to increase the accuracy of the generator with respect to the gameplay targets by employing another strategy for evolution control. For example, it can perform a simulation-based evaluation when the predictive model deems that a match has sufficient quality that a more precise assessment of the gameplay outcomes (via simulations) is warranted, or by updating the model to new data during evolution.

8.2 Extensibility

As an exploratory study into the application of surrogate models for game facet orchestration, this thesis has focused on creating a functional pipeline. Now that this has been established, new areas of research can be explored. The majority of this section will focus on extensions to the surrogate model in terms of architecture, inputs, outputs or a combination thereof. This is followed by potential directions for research into the generative process, working towards a design tool and other domains that this approach could be applied to.

8.2.1 Extending the Output of the Surrogate Model

The gameplay metrics chosen for the experiments describe a summary of the gameplay during a match. Although game duration and fairness are typical goals for a game designer, they can be perceived as too broad. They might desire more fine-grained feedback on gameplay from a game design tool. For the purpose of showing the extensibility of this approach, more metrics were collected from the dataset of Chapter 5.

The datasets of the metrics described in this section are balanced via oversampling. The bins required to label the floating point values are based on the standard deviation of each dataset. The rules for binningare the result of preliminary experiments with different forms of binning. For the sake of brevity, this section only reports the results of CNN3, using the same training procedure as in Section 6.2.

Player Lifetime

Cardamone et al. (2011b) have argued that the interestingness of a map is directly linked to the time that a player spends in a fight. This is computed as the time between the first damage received and the death of the player (Combat Time or CT). Their intuition is that a good map offers prolonged fights because there are possibilities to escape, hide, or find pickups after a fight has started. An additional reason to maximize combat time is to limit the time players spend searching for each other; especially a map for two players should not be too maze-like. Inspired by these notions, we also collected the lifetime of each player, which is computed as the time between the spawn and death of the player (Lifetime or LT). The metrics collected from each player, $p1$ and $p2$, were normalized by the duration of the game and combined in two ways. As the mean of both players (m), similar to (Cardamone

Table 8.1: Average and standard deviation of Mean Absolute Error (MAE) and R^2 values obtained by CNN3 for combat time (CT) and lifetime (LT) prediction on the validation set (N=10). The metrics are either computed as the mean of both players (m) or as the ratio of player one over the total (r).

Metric	MAE	R^2
CT_m	0.038 ± 0.001	0.366 ± 0.012
LT_m	0.037 ± 0.001	0.370 ± 0.009
CT_r	0.109 ± 0.006	0.791 ± 0.016
LT_r	0.070 ± 0.002	0.895 ± 0.003

et al., 2011b), and as the ratio of the value of player one compared to the total of both players (r). For example, the combat time metrics are computed by:

$$CT_m = \frac{CT_{p1} + CT_{p2}}{2} \tag{8.1}$$

$$CT_r = \frac{CT_{p1}}{CT_{p1} + CT_{p2}} \tag{8.2}$$

where CT_{p1} and CT_{p2} are respectively the combat times collected from $p1$ and $p2$. The intuition behind the latter combination is that it describes the balance between the two players, which can serve as an alternative to score balance. A value of 0.5 shows an equal life or combat time for both players, while a value towards 0 or 1 shows that one player dominates the other in a fight. The metrics for LT_m and LT_r are computed in the same way, but with LT instead of CT.

The difference in performance between predicting CT_m and LT_m is small, as can be seen from Table 8.1. This is not surprising, given the small difference between the two metrics. In general the model's predictions on these metrics are not good at explaining the variance in the data despite a very small absolute error. This is reminiscent of the performance on the game duration target and it could be the result of a skewed data distribution. Another explanation could be that it is inherent to this target, as averaging the player lifetimes results in a lot of the same values. For example, an average CT of 0.5 is obtained both when one player dominates the other as when both players fight equally long. Mapping these situations to the same value seems counterintuitive. The model is more successful on the CT_r and LT_r targets. The predictions of the model explain the variance in the data very well. Indeed, the explained variance of LT_r is on par with the model's KR predictions. Given the difference in both meaning and error, it would be interesting to see how both of these variables change the generated matches.

Heatmap Entropy

In the field of game user research, a popular method for understanding player behavior is to make a heatmap, i.e., to make a map of events aggregated over the duration of the play session (Drachen and Canossa, 2009; Thompson, 2007). By mapping player positions or causes of death, heatmaps can indicate which parts of the maps are most used or not used at all and whether design patterns such as choke points or flanking routes are (ab)used by players. These rich descriptions of player behavior could also be an interesting component of PCG, e.g., to create level suggestions that effectuate a designer's intended heatmap or to add an audio facet to the orchestration by linking heatmaps to affective states and use these to create soundscapes such as in *Sonancia* (Lopes et al., 2016).

While heatmap prediction would open interesting lines of research for facet orchestration, it requires adjustments to the surrogate model that do not fall within the scope of this thesis. We can, however, turn the heatmap prediction into a regression task by computing the Shannon entropy of a heatmap. For heatmaps this metric can be interpreted as follows: an entropy of zero (i.e., a new event does not add any information), means that all fights take place in the same location, while an entropy of one means that fights are spread out over the entire map. As such, a low entropy might indicate an overly effective choke point, a lack of flanking routes or a lack of incentive to explore the map.

Based on the literature, this section explores two heatmaps: players' positions and players' deaths. The positions are obtained by logging the position of both players each second, while death positions are logged when a player died. The heatmap of the players' deaths consists of 16 cells on a 4×4 grid: each cell stores the number of deaths that occurred within the 5×5 tiles of that cell. Similarly, the heatmap of players' movement is 16 cells and each cell stores how many times each player was within that cell. With 16 values per heatmap, the entropy is calculated as:

$$H = -\frac{1}{\log_2 N} \sum_{c=1}^{N} \frac{p_c}{P} \log_2 \frac{p_c}{P} \tag{8.3}$$

where N is the number of cells in the heatmap, p_c is the number of entries in a specific cell, and P is the total number of entries in the heatmap ($P = \sum_{c=0}^{C} p_c$). These variables can be combined into several entropy metrics. The deaths and movements of both players can be joined before computing the entropy. Resulting in one entropy value for all death locations (H_d) and one for all player positions (H_p). The entropy can also be computed per player and merged afterwards, similar to Eq. (8.1) and Eq. (8.2). That is, by taking the mean of the two values (H_{dm} and H_{pm}) or computing the ratio of one player compared to the total (H_{dr} and H_{pr}).

Using the same training procedure as in Section 6.2, the validation results of CNN3 obtained from 10 runs are shown in Table 8.2. The model is most successful at predicting H_{dr} and H_p. These metrics translate to the comparison of death locations of the players and the overall entropy of player movement; as such, they are among the most interesting from a design perspective. The high R^2 value on H_{dr} combined with a high error indicates that there is a lot of spread in this data; as opposed to H_p and H_{pr}. The difference between the good performance on H_{dr} and the bad predictions of the other H_d metrics indicates that the interesting situations occur when the death locations are compared instead of aggregated. It is likely that the values of both players cancel each other out, similar to CT in the previous section.

8.2.2 Extending the Domain Knowledge of the Surrogate Model

Despite trends in field, preliminary experiments did not show an advantage of bootstrapping the model with a network that was pretrained on natural images, nor an advantage of overly large or deep models. Perhaps the input data or the modeling task as a whole are relatively simple. To deal with a higher task complexity, the model might benefit from residual connections that connect lower and higher level features (He et al., 2016)

Table 8.2: Average and standard deviation of Mean Absolute Error (MAE) and R^2 values obtained by CNN3 for death entropy (DE) and position entropy (PE) prediction on the validation set (N=10). The metrics are either computed without discerning between players, as the mean of both players (m) or as the ratio of player one over the total (r).

Metric	MAE	R^2
H_d	$0.079 \pm 4 \cdot 10^{-4}$	0.314 ± 0.006
H_{dm}	0.088 ± 0.002	0.474 ± 0.013
H_{dr}	0.152 ± 0.006	0.596 ± 0.031
H_p	$0.034 \pm 1 \cdot 10^{-4}$	0.553 ± 0.003
H_{pm}	$0.057 \pm 6 \cdot 10^{-4}$	0.543 ± 0.007
H_{pr}	$0.029 \pm 5 \cdot 10^{-4}$	0.503 ± 0.015

or embedding layers that convert parameter space into a semantic space (Mikolov et al., 2013). Both alterations change the input perception of the model and have successfully been applied to reinforcement learning agents for games in GVG-AI (Woof and Chen, 2018), DOTA2 (Matiisen, 2018) and Go (Silver et al., 2017).

A less comprehensive extension is the addition of domain knowledge via the input. In this section we discuss how the performance of the model changes when perceiving general heuristics of level design; in the form of 1D and 2D map metrics based on (Liapis et al., 2013c). Specifically, additional inputs include a vector of 90 level statistics (listed in Appendix B) [1] and several 2D visualizations of *safety* and *exploration*; depicted in Table 8.3. These features are used to predict the two targets that have been used throughout this thesis as well as two of the entropy metrics that were introduced in the previous section. Specifically, the entropy of the heatmaps of respectively the players' deaths (H_d) and the players' positions (H_p). These experiments are detailed in (Liapis et al., 2019a).

In order to explore how combinations of levels and weapons can better predict the different gameplay outcomes, the channels of Table 8.3 are used in full or in different subsets. Since the best model in this thesis is CNN3, which uses only the naive tile-based channels T1 to T7, that model will be used as a baseline. Different sets of channels are chosen based on intuition and range from few core channels to the full set of Table 8.3:

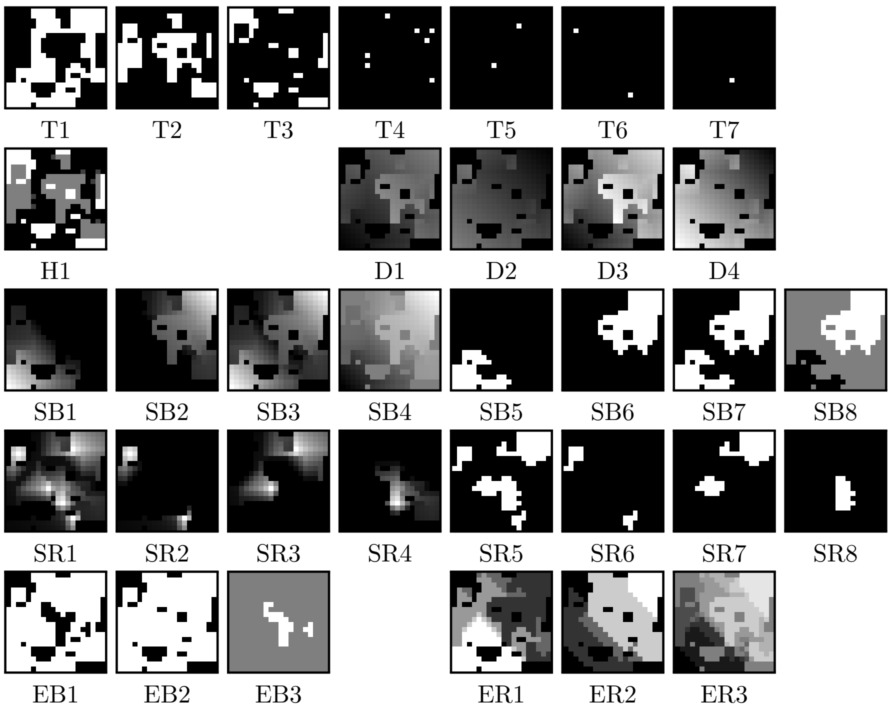
$S1$ T1-7 (7 channels)
$S2$ D1, D2, H1, SR1 (4 channels)
$S3$ T1-7, D1, D2, H1, SR1 (11 channels)
$S4$ all 34 channels of Table 8.3

Out of these channel sets, $S3$ and $S4$ include the baseline channels, while $S2$ only contains domain-specific metrics. The set $S3$ is the union of $S1$ and $S2$.

Table 8.4 shows the performance of the computational models for different sets of inputs and the four different game outcomes. It indicates that the naive binary channels used as a baseline ($S1$) are surprisingly powerful in predicting all gameplay outcomes. While the small set of channels chosen specifically for the rich information it contains ($S2$) underperforms rather substantially compared to the baseline, sets which combine $S1$ with other channels (i.e., $S3$ and $S4$) significantly outperform the baseline in several target outcomes. The best performance is with $S4$ which combines all 34 channels of Table 8.3. When combining

[1]Some of these have been used in Section 5.2.2 and Section 5.3.2 to analyze the dataset.

Table 8.3: Input Channels: binary channels per tile type (T1-7) and heightmap (H1), distance from bases (D1-4), safety metric before and after high-pass filter between bases (SB1-8) and between powerups (SR1-8), exploration from base to base (EB1-3) and from base to resources (ER1-3).



the level channels with quantifiable level statistics, the models become more accurate than their respective ones with the same channels. These level stats are fed to the model as an additional input stream, and fused in a way that is identical to the character classes. Using a model with $S4$ channels and level stats has the best overall accuracy, especially for match duration and H_d.

It is worthwhile to understand which streams of information (the level stats, the full level channels, or the character class parameters) contribute most to the models' predictions. To do this, we train the best architecture ($S4$) and the baseline $S1$ on a modified corpus without certain streams of information; by setting those input values to zero. Table 8.5 presents the MAE and R^2 metrics when different streams are omitted. While any combination of level channels and level stats seems unable to create useful predictive models for KR, the character classes are quite capable of predicting KR but are worse at predicting t and both entropy metrics. The winner of a match seems to be primarily determined by the classes of the competing players while the level can only do that much to affect power differences. Yet, both types of level information are valuable for predicting the other targets. This is similar to the effect we have seen in Section 6.4. The level stats alone are not as informative as the level channels, but in conjunction they are more informative than either input alone. When level stats and class parameters are used without level channels, both MAE and R^2 values are comparable to the baseline ($S1$).

These findings indicate that including domain-specific views of the level can lead to superior predictive models. The naive binary channels are surprisingly effective in this task as well; since it required many extra layers of information to significantly outperform the

Table 8.4: Validation results for different inputs and outputs. Values significantly better (lower for MAE, higher for R^2) than the baseline ($S1$) are in bold.

	KR		t		H_d		H_p	
	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2
Without level stats								
S1	0.069	0.910	0.082	0.605	0.079	0.312	0.0340	0.555
S2	0.078	0.900	0.086	0.570	0.086	0.198	0.040	0.410
S3	0.068	0.911	**0.080**	**0.624**	0.079	**0.320**	**0.033**	**0.570**
S4	0.067	**0.912**	**0.079**	**0.627**	0.079	**0.323**	**0.033**	**0.585**
With level stats								
S1	**0.066**	**0.914**	**0.079**	**0.627**	**0.078**	**0.327**	**0.033**	**0.578**
S4	**0.067**	**0.914**	**0.078**	**0.636**	**0.078**	**0.332**	**0.033**	**0.587**

Table 8.5: Validation results when streams of information are omitted or set to zero.

	KR		t		H_d		H_p	
	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2
Character classes only								
—	0.074	0.896	0.201	0.122	0.093	0.080	0.054	0.034
Level channels only								
$S1$	0.259	0.005	0.110	0.356	0.085	0.210	0.037	0.487
$S4$	0.259	0.004	0.110	0.354	0.085	0.209	0.037	0.487
Level stats only								
—	0.257	0.012	0.113	0.326	0.086	0.186	0.039	0.429
Level channels and level stats only								
$S1$	0.258	0.011	0.109	0.368	0.084	0.219	0.036	0.503
$S4$	0.258	0.012	0.109	0.367	0.084	0.217	0.036	0.502
Level stats and character classes only								
—	0.067	0.912	0.085	0.579	0.081	0.288	0.036	0.502

baseline. Moreover, models[2] that do not receive the original binary input perform worse than the baseline model. Combining 2D representations with summary level statistics and classes yields yet more powerful models. Removing any of the three streams of information lowers the overall accuracy, as each of the inputs contains valuable information for a subset of the gameplay outcomes. All in all, we can conclude that domain-specific knowledge can support, but not replace, the generic representation that we have used in this thesis.

8.2.3 Extending the Orchestration Process

The generative process proposed in this thesis orchestrates the generation of the two facets by putting them in one genotype. This rather limited form of orchestration can be extended by loosening the connection between the two facets, which gives the generators more freedom. One such approach would be to use *cooperative coevolution* as a search process. This equates to separate generators that inform each other about their best individuals via a blackboard. When only the best individual per facet is shared, such as in (Cook and Colton, 2011), the result is very similar to our approach. But when more individuals are shared between generators, this process can generate 'generally good' single facets and create the best pairing based on the model's predictions of their coherence.

With extensions to the output, the surrogate model could also be used as part of a *composer* or *director* in a top-down approach to game facet orchestration (Liapis et al., 2019b). Given a set of target values, the model could coordinate a more sequential process of generation while using predictions to evaluate the content. For example, the process could start with creating two balanced character classes, then find a map that complements these for a desired heatmap and/or duration. A heatmap of death locations could be used to warn players of an incoming battle or fake the history of a previous battle via 3D models or decals. Followed by the placement of audio cues based on a heatmap of positions (or deaths) to prime them for combat. All of these processes would be directed and evaluated by the model to ensure the desired gameplay experience.

This thesis has demonstrated the use of multi-objective evolution for gameplay targets, but this could be extended with diversity-based targets. Quality Diversity (QD) algorithms such as *Constrained Novelty Search* and *MAP-elites* have already been shown to be feasible for map generation (Liapis et al., 2015b; Alvarez et al., 2019) and a variety of other domains (Gravina et al., 2019). Section 7.4.3 has shown that few changes are typically required when pursuing multiple objectives. It would be interesting to see if this can be reduced further with a QD algorithm. Or perhaps it could produce a more diverse set of classes on the edges of the duration spectrum. Such a system would probably require a model management strategy to inject true evaluations, as diversity-based algorithms are much more likely to search areas of the design space that were not sampled by the model. Surrogate-assisted alternatives to MAP-elites are already being developed, e.g., (Gaier et al., 2017), so a first step could be to apply such an existing algorithm to procedural content generation.

Despite this thesis's focus on search-based PCG with a surrogate model, the general mapping between game facets and gameplay outcomes could be exploited in a generative adversarial network (GAN) setup. Following the PCGML paradigm (Summerville et al., 2017), a conditional GAN could be trained to output levels and classes based on a desired gameplay outcome provided as input. A network similar to the surrogate model in this paper could judge not only the resulting gameplay, but also whether generated output is good

[2]See (Liapis et al., 2019a) for a comparison with more models.

enough to pass as a real design. Such an architecture would likely resemble the Auxiliary Classifier GAN, which has been applied successfully to natural image synthesis (Odena et al., 2017).

8.2.4 Extending the System as a Tool

The experiments in this thesis have tested the premise of a surrogate-assisted design assistant that can evaluate content, highlight high impact areas and generate suggestions. While the system was designed with a tool in mind, it has not been implemented as such. A valuable extension would be a graphical user interface (GUI) that facilitates interaction with game designers. This can take inspiration from standalone mixed-initiative map design tools, such as *Sentient Sketchbook* (Liapis et al., 2013d) or *Evolutionary Dungeon Designer* (Alvarez et al., 2018). Alternatively, it could be implemented as a Unity plug-in to interact directly with the game content and give real-time feedback as the game is being created.

With the help of a GUI it would be easier to test the impact that this system can have on a game designer's workflow. For example, a user study could reveal which playability constraints on the classes are practical, and the ability to mark certain parts of the content as non-adjustable can teach us more about the desired division of labor and whether this can be supported. Like most research in PCG, this thesis has not actively included the iterative aspect of game design. Such an extension could validate the suggestion of Liapis et al. (2019b) to adjust facets in a sequential fashion and extend proposed single-faceted workflows (Beyer et al., 2016; Preuss et al., 2018) to multi-faceted ones.

8.2.5 Generalizing the System

The proposed facet orchestration approach was applied to a very specific task: tuning a 1 versus 1 match in a shooter game. One big question that remains is how well does the approach generalize to other tasks? As with most contemporary AI implementations, this particular system only works on this particular representation of this particular game. In fact, the predictive part of the system likely only creates desired gameplay for the artificial agents that were used to create the dataset. However, the entire pipeline was designed with generality in mind, and does in no way take advantage of any domain knowledge of the single use case that is was applied to. Therefore, the described methodology and the obtained results provide a handle on facet orchestration that can be applied to other domains as well.

Research into this could start with small changes. The levels could be made more complex, by including spawns at different positions, or tunnels and bridges as in (Liapis, 2018). This would make it more difficult to parse the map and to model the gameplay. Additionally, the game could be changed to play more like the *Unreal Tournament*-series (Epic Games, 1999-2007). Players could be given simple starting weapons, while the map contains various weapon pickups as in (Cachia et al., 2015). In that case, facet orchestration should not only balance the weapons' characteristics, but also their placement in the map. The gameplay could also be made more complex by adding players. Most online battle arenas have more than two players; typically ranging from 2 v 2 to 6 v 6, though some exceptions are 32 v 32. Given the decision earlier in this research to reduce the number of players, it would be interesting to find a way to increase this number again. An extension to 2v2 seems feasible, perhaps by imposing more constraints on the generation. The problem can be simplified by fixing 1 class on each team or by selecting from human designed classes, which resembles matchmaking. Predicting the gameplay of 32 v 32 seems impossible even for humans, so there must be a way to modularize this. Perhaps an algorithmic approach can be distilled from the way human designers handle this task.

Another opportunity would be to extend this system to other game genres. This would depend mostly on obtaining a usable dataset and creating different variation operators; a CNN should work for other genres as well. In particular the domain of real-time strategy games seems a suitable candidate, given their natural top-down view. In another context, CNNs have already been used to predict gameplay outcomes based on map views (Stanescu et al., 2016). Additionally, simulation-based PCG has been shown to be effective for tuning unit parameters towards balanced matches (Preuss et al., 2018). These successes indicate that a surrogate-based map and unit generator would not be out of reach. Creating a single system that can evaluate and create content for multiple game genres lies further on the horizon.

8.3 Summary

This chapter concludes the thesis by providing main insights from the research on multi-faceted content generation for a first-person shooter game. The system proved capable of creating content for gameplay goals set by a designer and demonstrated that less changes are needed to an initial design if more than one facet can be manipulated. In the process, a number of limitations were found. In particular, this thesis has focused on obtaining a large dataset of quantifiable gameplay, which resulted in the use of game playing agents. While the system has been demonstrated to facilitate certain gameplay for these agents, it remains an open question how this translates to affective gameplay experiences of human players. This chapter has addressed these limitations, for example by demonstrating that the proposed surrogate model can learn to accurately predict more than just the two gameplay outcomes that have been used in this thesis and outlining solutions for dealing with the uncertainty of the model's predictions. Directions have been provided for extensions towards different forms of orchestration as well as mixed-initiative design tools. Finally, the chapter considered related problem domains within procedural content generation that can benefit from the proposed methodology.

Appendix A

Machine Learning for Gameplay Prediction

This chapter describes the initial proof of concept of the proposed surrogate model and its evaluation. Specifically, it tests the validity of using a neural network to make gameplay predictions based on a map and the players' attributes. As described in Chapter 2, neural networks have been used for attributing values to a game state during the game. Yet there was no existing work on whether they could predict the value of an end state, e.g., score balance or game duration, based on an initial state. The results of this proof of concept study have been published in (Karavolos et al., 2017).

The experiment in this chapter focuses on the genre of first person shooters and one of its most common contests: team deathmatch. The goal of this game mode is to try to get more kills than the opposing team. This genre was chosen for its relatively straightforward gameplay (i.e., moving around and tagging opponents) and its common numerical summary of gameplay in the form of kill balance between opponents. Section 4.1.1 elaborates on the game that was made for this experiment, including the methodology for generating levels, weapons and gameplay. The representation of the game facets for the neural networks and architecture of the proposed model are described in Section 4.1.2 and Section A.3. The experiments in Section A.4 contain an analysis of the data, a comparison of the proposed model with several baselines (including models with only one input modality) and a breakdown of the test results along various dimensions. This is followed by a discussion of the results as well as a reflection on the major take-aways of this research in Section A.5.

A.1 Level Generation

To generate sufficient data for effective machine learning, a large number of diverse game levels must be created. For that purpose, 39 different generators were designed to create a broad range of levels. The generators primarily influence the spatial distribution of a set of large and small objects: due to concerns of objects colliding with each other, every generator places a total of 50 objects. Some generators place 50 large objects and no small objects (**L** row in Fig. A.1), some place 50 small objects and no large objects (**S** row in Fig. A.1), and some place 25 large and 25 small objects (**B** row in Fig. A.1). The spatial distribution of objects, as shown in Fig. A.1, includes placing objects (large or small) on one half of the level, in the center of the level, along a central row or column, or in the four corners of the level. As an addition, six generators feature a designer-defined level pattern

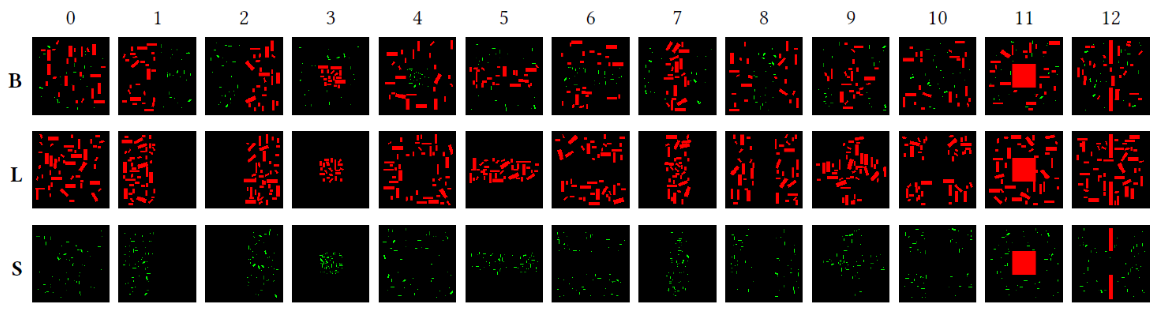


Figure A.1: Example maps of each level generator, with both types of objects (**B**), only large objects (**L**) and only small objects (**S**).

which is not randomized: either a large impassable block in the center of the level (column 11 in Fig. A.1), or a central choke point formed by two long walls (column 12 in Fig. A.1). While generators determine the object's coordinates, rotation, and type, the complete level is stored as a 100 by 100 pixel image where each object usually takes up more than one pixel.

A.2 Gameplay Data Collection

The neural networks are trained via supervised learning, which requires a set of labeled training data. Ideally, this would be data collected from actual players, but this is not feasible due to the volume of data needed for deep learning. Therefore, as an initial approach, we simulate player behavior with artificial agents.

The artificial agents are controlled by the *Shooter AI* plugin library (Squared55, 2015) and form two teams of three agents each. Each team starts at its respective 'base' (and agents re-spawn there if they die) and compete inside a generated arena as shown in Fig. 4.1. In terms of behavior, the agents wander around the map towards the enemy base. When they spot an opponent, they attempt to find cover and shoot the opponent from this cover position. If the opponent is not killed within a limited time frame (e.g., if the opponent is also in cover) the agent will try to move to another cover position. If no cover is found, agents switch to a chasing behavior, trying to kill the opponent from close range. Agents always aim for the head unless they carry a projectile weapon (which is aimed at the hips in order to take advantage of splash damage). For the sake of simplicity, all members of one team have the same weapon, which has unlimited ammo. However, the weapon's clip size affects how many shots can be fired before an agent needs to reload (reload time is one of the weapon parameters).

For experiments in this paper, five weapons were adapted from those of the *Shooter AI* library: the shotgun, the (assault) rifle, the sub-machine gun (SMG), the sniper rifle and the rocket launcher. These weapons have very different patterns of use (Hullet and Whitehead, 2010) and weapon characteristics, such as the high damage of the sniper rifle, the slow bullet speed of the rocket launcher, or the short range of the shotgun. In order to derive the desired mapping between a game level and a weapon pairing, each generated level was tested 25 times, once for each weapon combination (including matches where both teams had the same weapon). Simulations lasted until a total of 20 kills were scored, and balance between the teams was calculated based on the number of kills of the first team (the team to the left in Fig.4.1b) over 20 kills. A ratio of 50% indicates a perfectly balanced

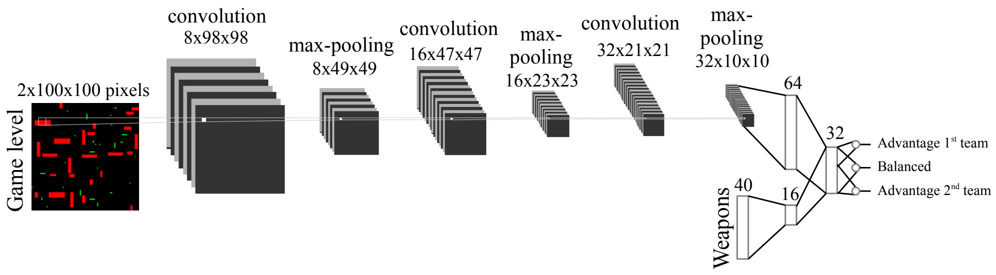


Figure A.2: CNN architecture for level and weapon input parameters.

(i.e., tied) match-up, a ratio over 60% indicates that the weapon pairing and the level favor the first team, while a ratio of 40% or lower indicates that the second team was favored. Since kill ratios between 40% and 60% amount to a couple of kills for one or the other team, these match-ups are still considered balanced. These three labels were stored for supervised learning.

A.3 Model

This experiment evaluates the use of a convolutional neural network (CNN) to predict game balance based on the weapons of each team and the layout of the level. The output of the network is a probability distribution over the three classes: balanced match, 1st team advantage (left in Fig. 4.1b), 2nd team advantage. The network has three output nodes, which are processed via a soft-max function to convert them into three real values between $[0, 1]$ that sum up to 1; the highest of the three values determines which class the match-up belongs to.

Based on extensive preliminary experimentation with different network architectures, learning rates and activation functions, the best performing CNN architecture used in Section A.4 is shown in Fig. A.2. The CNN uses three convolutional layers, each with several filters of 3 by 3 pixels applied on the previous layer (i.e., the original image for the first convolution). The first convolutional layer outputs 8 different feature maps of size 98×98 out of the original $2{\times}100{\times}100$ pixel image; each filter ideally detects different patterns of the input. Each of these feature maps is downsampled to half its dimensions through max-pooling, which outputs the maximum value of a 2 by 2 region of the convolution's output. Max-pooling is applied after each convolution, ultimately producing an output volume of $32{\times}10{\times}10$, which is then converted via a fully-connected layer into 64 values. The 40 weapon parameters are mapped via a fully-connected layer into 16 values. The 64 outputs from the level and the 16 outputs from the weapons are concatenated and passed to a fully-connected layer of 32 nodes which connect to three output nodes that predict the probability that the input belongs to one of the three classes. All nodes in the network use a rectified linear unit (ReLU) as their activation function (which applies element-wise non-linearity), except for the output layer which uses a soft-max function, as described above.

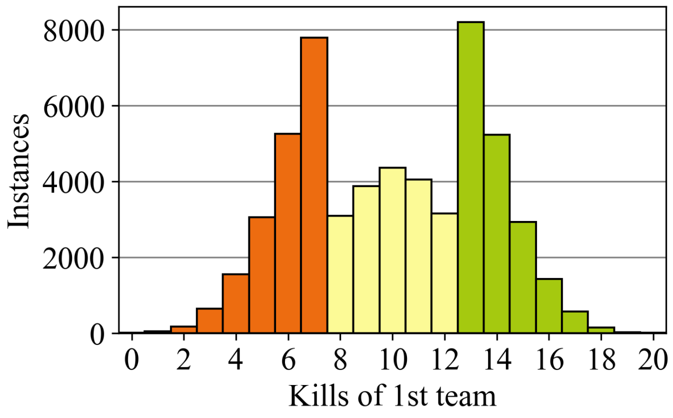


Figure A.3: Distribution of kills of the first team (left in Fig. 4.1b) in the pruned training set.

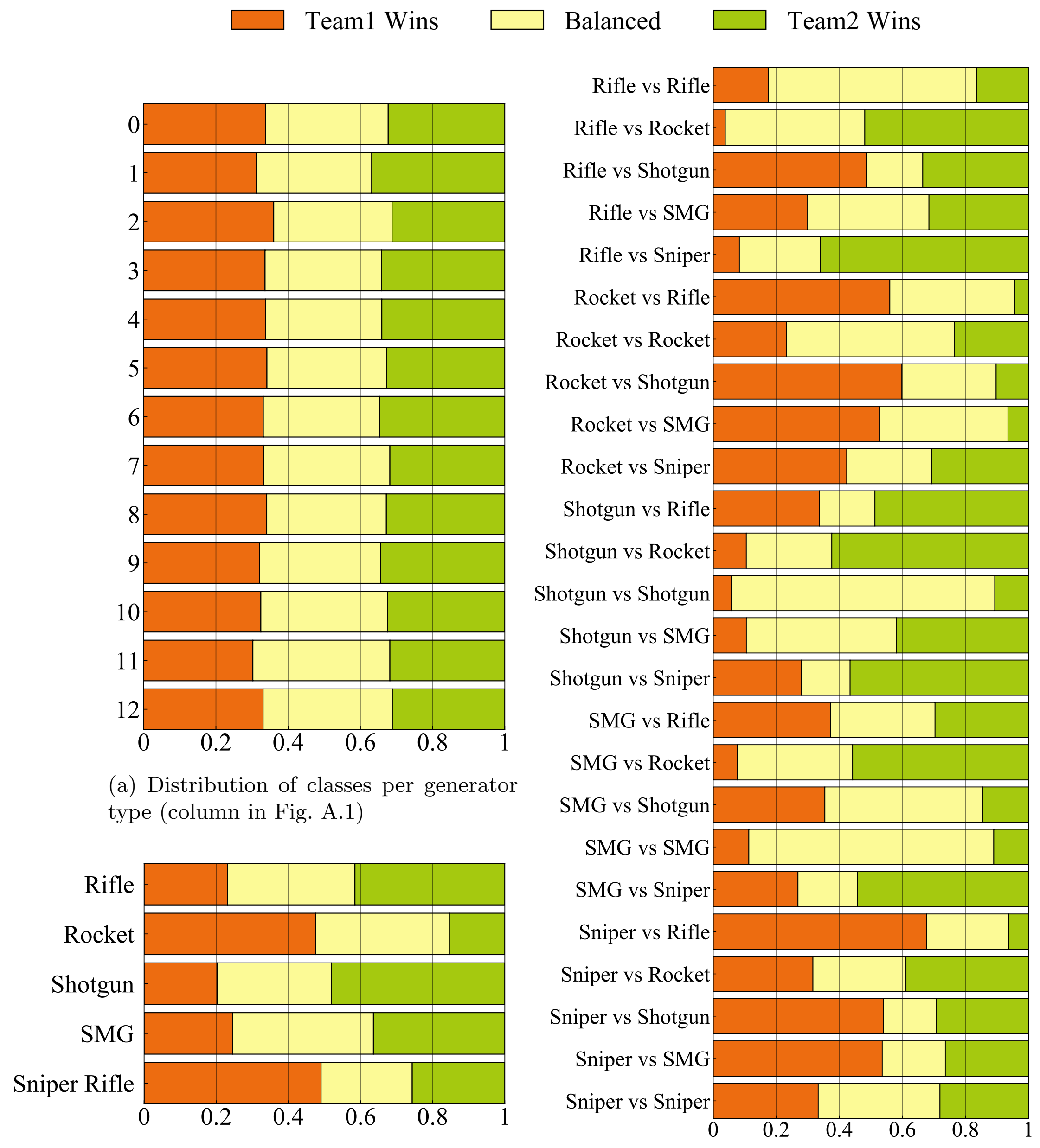
A.4 Experiments

This section discusses the training data collected from artificial gameplays, analyzes how different networks learn this data, and demonstrates how the computational model handles different weapons and level patterns.

A.4.1 Training Data

As noted in Section 4.1.1, training data was collected from generated levels played by teams of artificial agents which, for simplicity, use the same weapon within each team. The 39 generators presented in Fig. A.1 generated 100 levels each. These 3900 levels were playtested for all weapon pairings (i.e., 25 matches per level) resulting in $97.5{\times}10^3$ data points. Matches that did not finish before a timeout (less than 0.5% of the data) were removed. An inspection of the data showed that roughly two thirds of the matches classified as balanced. In order to prevent a machine-learned bias towards the most common class, undersampling was applied to make each class the same size as the least common class. At the same time, it was ensured that the data per level generator and per weapon pairing were roughly equally common in the data set. Each of the five weapons is used in 18% to 22% of all data points, and similarly each of the 39 generators was used in 2.2% to 3% of all data points. This resulted in roughly 17×10^3 data points per class. Taking a look at the raw data (before classification) in terms of the first team's kill ratio in the pruned dataset, we can see in Fig. A.3 that it is quite symmetric. Moreover, it is clear that before removing roughly half of the balanced matches, kill ratio followed a normal distribution.

In order to gain some insights into the patterns of the training data, we analyze them on a per generator, per weapon and per weapon pairing basis. For generators, we aggregate among all generators in the same column, as they constitute the folds that we train on: in Fig. A.4a we observe minor discrepancies, with all folds having an almost equal distribution of classes and only slight advantages to the first team (e.g., generators of column 2) or the second team (e.g. generators of column 1). There are more clear differences when looking at weapons used by the first team, in Fig. A.4b. Both the sniper rifle and the rocket launcher severely favor the team using them (with 49% and 48% of matches belonging to 1st team advantage class respectively); the rocket launcher also has far fewer instances of 2nd team advantage. The opposite is true for the other weapons, which tend towards advantage to the



(a) Distribution of classes per generator type (column in Fig. A.1)

(b) Distribution of classes per weapon, averaged over all opponents

(c) Distribution of classes per weapon pair

Figure A.4: Ratio of each class in the pruned training set.

team that does not use them or, at best, balanced matches. Looking at each weapon pairing individually in Fig. A.4c, differences become even more clear in terms of class imbalances. Indicatively, the rifle is severely handicapped against the rocket launcher and the sniper rifle, winning only 4% and 8% against each respectively; even so, the rifle versus rocket launcher match-up has more balanced instances than the rifle versus sniper rifle match-up (indicating that those pairings are inherently different). Even when both teams use the same weapon, the class distributions are quite uneven: for SMG versus SMG, 78% of match-ups are balanced while for sniper rifle versus sniper rifle only 39% of match-ups are balanced.

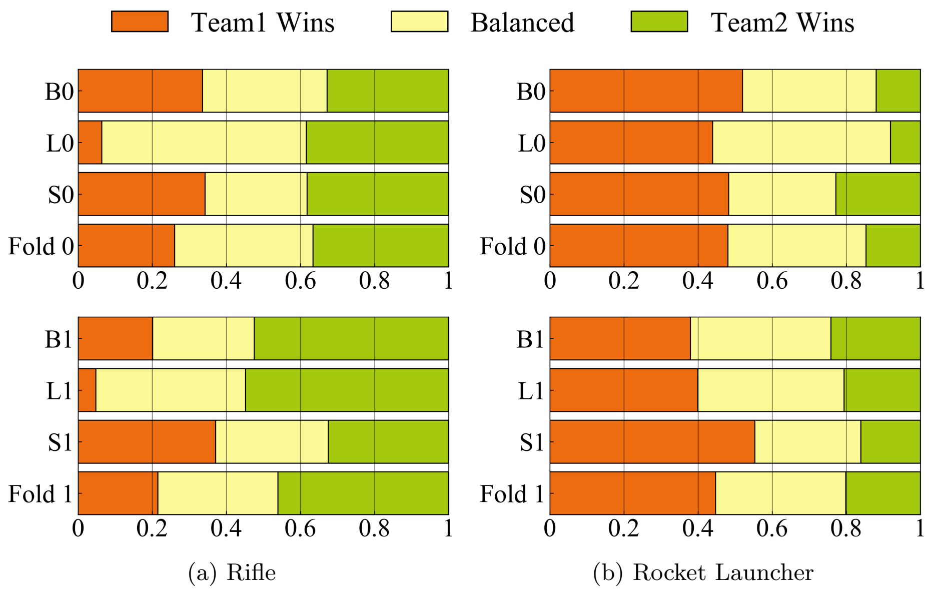


Figure A.5: Distribution of classes per weapon for one generator or one fold.

Based on the above analysis and the outlook of Fig. A.4a, it would seem that the patterns of the levels play a minor, if any, role in the matches' balance. However, this conclusion is mostly due to averaging factors; when looking at the impact of a generator to a weapon pairing, differences become far more evident. For the sake of brevity, only two example folds will be examined in detail (column 0 and column 1 in Fig. A.1), for two very different weapons: the rifle and the rocket launcher. Fig. A.5a shows how classes are distributed when the first team uses a rifle, per generator (B0, L0, S0 and their average as fold 0; B1, L1, S1 and their average as fold 1); Fig. A.5b shows the same information but for the rocket launcher. The first observation is that the ratio of small objects versus large objects plays a role regardless of their distribution: S0 and L0 are very different in terms of 1st team advantage instances for the rifle (although less so for the rocket launcher), and L1 and S1 are very different for both the rifle and the rocket launcher (S1 favoring both). The second observation is that each fold results in different class distributions. Fold 0 has a fairer distribution among the three classes for the rifle, while for fold 1 — which has many barriers on the side of the team using the rifle — the 2nd team has a clear advantage in 43% of instances. For the rocket launcher, differences when averaging across folds are less obvious; the rocket launcher is a consistently powerful weapon less sensitive to level differences.

A.4.2 Training Results

Several neural network architectures, topologies, and activation functions were considered and tested. For the purposes of brevity, this section focuses on the best performing CNN architecture, and compares it with the best performing fully-connected network (ANN) and the best performing single layer perceptron. All networks were trained according to a 13-fold cross-validation scheme, where the data in each fold coincides with the generators in the columns of Fig. A.1. More specifically, machine learning used the data and ground truth of 11 folds to train on, used one fold as a control for stopping training, and tested the

Table A.1: Mean accuracy and 95% confidence intervals of machine learning from 13-fold cross-validation.

Network	Epochs	Training	Validation
CNN	24	62%±0.5%	64%±0.2%
ANN	25	54%±11%	50%±2.2%
Perceptron	14	50%±2.2%	48%±1.1%
CNN levels only	12	36%±1.6%	36%±1.6%
CNN weapons only	36	53%±0.5%	55%±1.6%
ANN levels only	14	39%±0.5%	38%±1.1%
ANN weapons only	30	53%±0.6%	56%±1.1%

final model on the last fold to derive the validation errors reported here. The networks were trained on the cross-entropy loss, which measures the divergence of the predicted probability distribution with respect to the true class distribution and has several benefits compared to mean-squared error in classification problems (Kline and Berardi, 2005). Training was stopped after 5 epochs without improvement on the validation set.

The best CNN architecture (displayed in Fig. A.2) processes the level's image through three pairs of convolution and sub-sampling layers followed by a hidden layer that aggregates the different feature maps, and fuses it with the weapon parameters which are reduced via a hidden layer to 16 values, finally producing a probability distribution over the three classes of balance. Many different topologies (including one or two hidden layers) were also tested for ANNs; the best ANN discovered combines the level and weapon parameters (normalized to $[0, 1]$) into $20,040$ inputs ($2 \times 100 \times 100$ image pixels and 40 weapon parameters) and passes it through a single hidden layer of 256 nodes and again to three output nodes. Finally, the perceptron simply connects the $20,040$ inputs to the three output nodes.

Table A.1 shows the results of the learning process. A baseline of random guesswork would yield an accuracy of 33% among the evenly sampled classes. The perceptron improves on the random baseline by 15%, followed closely by the ANN which classifies 50% of the test data correctly. The CNN is clearly the best network with an average accuracy of 64% on the test data. Interestingly, the ANN tends towards quickly overfitting to the training set (54% accuracy) while the CNN does not do so.

Baselines

In order to determine the benefit of using both the level and the weapons as inputs for balance prediction, the same CNN and ANN topologies were trained using only one of those input modalities, setting the other inputs to zero. When level inputs are zero, the CNN essentially becomes a weapons-only fully-connected artificial neural network, although with a different topology than the tested ANN.

As can be seen from Table A.1, training on one input modality without the other makes the two networks (ANN and CNN) perform much more similarly. Especially for the levels-only case, this is surprising since one would expect that the benefits of applying convolutions, such as weight sharing, would still hold. Training only on the levels yields a much poorer test accuracy than training on both inputs, resulting in a classification rate of 38% and 36% for

Table A.2: Average confusion matrix of the 13 CNNs and 13 ANNs on their respective validation sets.

		Predicted class		
	Actual class	Team 2 wins	Balanced	Team 1 wins
CNN	Team 2 wins	0.74	0.17	0.09
	Balanced	0.26	0.46	0.29
	Team 1 wins	0.09	0.17	0.74
ANN	Team 2 wins	0.38	0.58	0.04
	Balanced	0.15	0.75	0.10
	Team 1 wins	0.07	0.60	0.33

the ANN and the CNN respectively, just slightly better than random. Observing Fig. A.4a, this should not come as a surprise, as levels on their own give very similar ratios in terms of classes, regardless of the features (types of pixels or their distributions) they contain. Only when paired with specific weapons (as shown in Fig. A.5) do level patterns show clear trends towards one class in the training set, and can be trained to give non-random predictions.

It seems that the weapons by themselves contain more useful information than the levels, as the drop in accuracy is much smaller. In fact, training solely on the weapons results in a better test accuracy for the ANN, equal or marginally higher than that of the 'CNN' architecture using weapons only (i.e., ignoring convolutions). The distribution of outcomes per weapon pair are much more diverse than those of the generators (see Fig. A.4c). Therefore, learning the distribution per weapon pair and predicting the most common class given can yield better results than doing this based on the generators. However, depending on the generator, the outcome for a specific weapon can actually be quite different (see Fig. A.5). If the network can identify the features that correspond to these different outcomes, it can become more accurate than based on the weapon pair alone. As can be expected, the CNN is better at identifying these weapon-level relationships than the ANN.

As another baseline, we trained networks based on the generator rather than the image of the level itself, and based on the weapon pairing rather than on the weapons' individual parameters. Using a variant of one-hot encoding to determine which generator created the level in the dataset, this reduces the size of the input vector and removes the computer vision task of processing a full level image. The generators were converted into two one-hot vectors based on the rows and columns of Fig.A.1, i.e., one vector that encodes which type of objects it spawns and one vector that encodes in which area the objects are spawned. The weapons are encoded as one of the 25 possible pairings. These three vectors are concatenated into one vector of size 41 (25 inputs for weapons, 13 inputs for columns and 3 inputs for rows of Fig. A.1); this input always contains three values of 1 (which weapon pairing, which column, which row) and the rest are 0. Given the limited input size, only fully-connected networks of one layer with a layer size smaller than the number of inputs were considered. The best ANN for this task has a hidden layer of size 16 and a test accuracy of 57%, while the perceptron has a test accuracy of 56%. The task becomes simpler with a smaller input size for the fully-connected networks, as both networks improve their accuracy from Table A.1 and do not overfit to the training data. However, neither network becomes more accurate than the CNN.

A.4.3 Test Results and Analysis

For these machine learning experiments, we have used 13-fold cross validation by splitting the data based on the levels' generators in columns of Fig. A.1. A test accuracy on the entire dataset can be calculated by aggregating all 13 test folds (collected during training and cross-validation). We can then partition the results on a per class, per generator type, and per weapon or per weapon pair basis to get different perspectives and insights.

Results per Class

Table A.2 has the confusion matrix of the ANN and CNN, which shows that the ANN has formed a general bias towards the balanced class. Approximately 60% of the time the ANN predicts that the match-up is balanced, regardless of what it actually is. The confusion matrix of the CNN shows that it does not have the same bias towards 'balanced' as the ANN; in fact it struggles to correctly classify the balanced class. Instead, it is very accurate in predicting either advantages. While the ANN correctly classifies 75% of the balanced class, the CNN correctly classifies 74% of both imbalanced classes. A positive note is that both networks have a low probability of misclassifying a match which is imbalanced in favor of one team as one that is imbalanced towards the opposite team.

Results per Generator

Figure A.6 shows the accuracy per generator type; each type includes three generators that place different types of objects at the same areas of the level (and coincides with the folds used for cross-validation). Firstly, it is obvious that the CNN has the highest accuracy for every generator type. Its lowest accuracies are with columns 1, 11 and 12 of Fig. A.1. It makes sense that levels of type 11 and 12 are hard to predict, as these levels differ the most from the rest of the corpus due to the designer-placed objects (central block, choke point). Levels of type 11 are also the most difficult for the ANN to predict accurately; however, type 1 levels are among the easiest to predict. While based on the confusion matrix one would expect that type 1 levels are predominantly balanced, based on the analysis in Fig. A.5 this is not the case. It is likely that the left-only placement of small and large objects in S1 and L1 is easier for the ANN to detect compared to the complex patterns of e.g., type 11.

The class distribution of the training data gives indistinct patterns when observed along generator types (as in Fig. A.4a). Therefore, assessing accuracy based on the type of obstacles in the level may be more insightful. Assessing accuracy on all levels of the same row (**B**, **L**, **S**), Table A.3 shows the distribution of classes in the training data and as predicted during cross-validation by the networks. The table reveals clear biases in the patterns found by the ANN, especially on generators with large objects (**B** and **L**), towards the balanced class. Astonishingly, while 32% of levels created by **B** generators had an advantage towards the 1st team, the ANN predicts that only 0.04% of total instances belong to this class. For **B**, the perceptron surprisingly has the closest distribution to the ground truth, although its predictions are not always accurate (accuracy of 47% compared to 60% of CNN). Based on Table A.3, it is clear that the different networks have a bias towards specific classes according to the type of pixels (red and green for **B**, red for **L**, green for **S**) in the image input. Looking into whether those biases lead to correct predictions, on the other hand, we find that the accuracy for all networks improves when levels contain only small objects (**S**). This is likely because in such levels weapons are less affected by blocked lines of sight which cause combat at shorter ranges; this would benefit traditionally weak weapons such as the

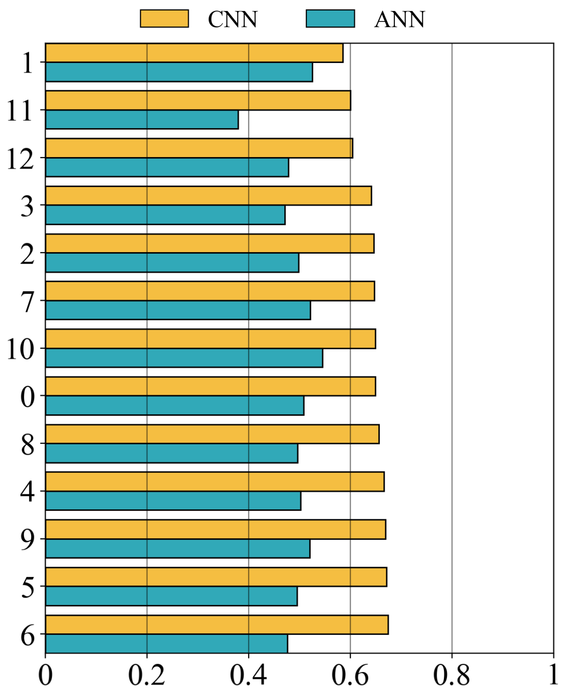


Figure A.6: Test accuracy of the networks categorized per generator type, sorted by the accuracy of the CNN.

shotgun. Interestingly, the ANN has a lower accuracy than the perceptron, and a clear bias towards the balanced class in levels of type **B**. We suspect that the ANN does not handle two types of objects (large and small) in the same level well, although the perceptron does not suffer from that. It is therefore more likely that the ANN has overfitted to patterns in other levels (**L** and **S**) which are easier to predict.

Results per Weapon

Grouping the data based on the weapon used by the first team, the CNN also has the highest accuracy in all weapons (significantly higher in 3 of 5), as shown in Figure A.7a. All three networks find the shotgun the easiest weapon to predict, which is not surprising as the weapon is consistently bad against all weapons in most levels (see Fig. A.4b). The second worst weapon, the SMG, is also easy for all networks to predict. Interestingly, for the most powerful weapons (rocket launcher and sniper rifle) the accuracy of the ANN suffers the most; this can be traced to its bias towards classifying most match-ups as balanced (see Table A.2) which is not often the case when those two weapons are used; especially for the sniper rifle, only 25% of the ground truth data belong to the balanced class.

When looking at the accuracies for each weapon pair, as in Fig. A.7b, clearer patterns can be observed. The CNN has higher accuracies in 24 of 25 weapon pairs, and significantly higher in 17. Both CNN and ANN have the highest accuracy when predicting Shotgun vs. Shotgun (86% and 73% for CNN and ANN respectively), followed by SMG vs. SMG (78% and 69% for CNN and ANN respectively). As shown in Fig. A.4c, both of these generally poor weapons when paired against each other lead to predominantly balanced matches (84% for Shotgun vs. Shotgun, 78% for SMG vs. SMG), the largest ratios for the balanced class among the weapon pairings. This explains the high accuracy of the ANN for those pairings,

Table A.3: Distribution of classes in the ground truth and predicted by the different networks, per **B**, **L**, **S** generator type.

Predictor	Advantage 1st	Balanced	Advantage 2nd	Accuracy
B generators				
Ground Truth	**0.32**	**0.34**	**0.34**	**100%**
CNN	0.38	0.23	0.40	60%±1.6%
ANN	0.00	0.90	0.10	36%±2.6%
Perceptron	0.34	0.29	0.37	47%±2.3%
L generators				
Ground Truth	**0.30**	**0.41**	**0.29**	**100%**
CNN	0.31	0.43	0.26	61%±3.7%
ANN	0.03	0.87	0.10	43%±4.4%
Perceptron	0.34	0.29	0.37	37%±4.3%
S generators				
Ground Truth	**0.37**	**0.27**	**0.36**	**100%**
CNN	0.42	0.17	0.41	70%±3.5%
ANN	0.40	0.22	0.38	66%±4.1%
Perceptron	0.39	0.13	0.48	57%±3.2%

as it tends to predict balanced classes overall. Other weapon pairings with high accuracy for the CNN have a low accuracy for the ANN, most notably Shotgun vs. Sniper Rifle, Sniper Rifle vs. Shotgun, Rifle vs. Shotgun and Shotgun vs. Rifle; again unsurprisingly, these four pairings have the lowest ratio of balanced match-ups (15%, 17%, 18% and 18%) which gives more predictive power to the CNN which leans towards predicting imbalanced classes rather than the ANN which leans towards the balanced class. As a final note, the only weapon pairing that the ANN is more accurate than the CNN is the Sniper Rifle vs. Sniper Rifle; this is an interesting case as it is not straightforward as to why. Sniper Rifle vs. Sniper Rifle match-ups do not particularly lean towards balanced (39% of instances), therefore the likelihood that ANN's bias towards balanced classes could be only partly an explanation. Based on qualitative evaluations of class distribution on a per generator basis, this weapon pairing is highly inconsistent (e.g., very rarely gives advantage for the first team for B1 and L1 and almost exclusively has balanced instances for L6 and S12; this erratic behavior seems to confuse the CNN more than the ANN, leading to its poor performance.

A.5 Discussion

Based on the quantifiable results of the machine learning task described in this paper, CNNs are particularly capable of discovering patterns between the level architecture and weapon parameters. While the ANN achieves an accuracy well beyond random, it hardly performs better than a perceptron and, in fact, is outperformed by a network that only receives the weapons of both teams. While using the level only as an input to either the CNN or the ANN does not give them a high predictive capability, when combined with weapon information it contributes to accurately predicting cases where the level architecture affects the relative balance of weapons. The CNN architecture processes the level and discovers higher-level patterns than merely those in the pixel image used by the ANN. The compact and information-rich outputs of the convolutions are better combined with the patterns found in weapons' parameters to classify with 64% accuracy on average between balanced matchups or matchups which favor one team or the other. Another positive result is that misclassifications when the matchup was imbalanced towards one team were not often "catastrophic", i.e., did not predict that the matchup was imbalanced towards the other team.

The ability of the model to fairly accurately predict whether a level and weapon combination will result in a balanced matchup without the need to playtest it can enhance generative processes immensely. Obviously, such a predictive model can replace simulations in a simulation-based evolutionary process such as (Cachia et al., 2015; Cardamone et al., 2011b), significantly lowering computation time needed to find new solutions. Based on the current model, however, classifying content into three classes may not be sufficient to provide the gradient towards better solutions that evolution can use as a fitness. Instead, the learned classes can be used in conjunction with simulation-based evaluations, as a first step. For instance, if evolving levels for a balanced matchup between a sniper rifle and an SMG, the predictive model can identify which of the levels are predicted to belong to the balanced class, and only those levels are then simulated to derive a more granular fitness for evolution to follow. The levels that are unbalanced in the above scenario can either be thrown away (e.g., regenerated or given the lowest fitness) or can be further evolved in the hopes of creating balanced ones using, e.g., a feasible-infeasible 2-population genetic algorithm (Kimbrough et al., 2008) as in (Liapis et al., 2013c).

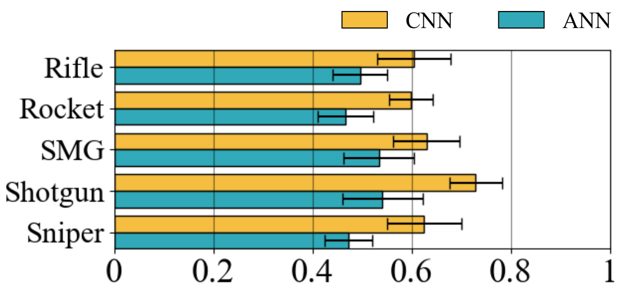
Since the model uses both the weapon pair and the level as input, it can also find the right combination of weapons per team for a provided level. This works in the same way as using the model for level generation (or critique) except now the level input is fixed and the weapon inputs are tested with the 25 combinations of existing weapons in the game. Beyond choosing a balanced set of weapons among those provided, however, the model can be used to generate new weapons or variations of existing weapons by fine-tuning their parameters (e.g. damage, clip size) to further improve the balance of the matches. As an example, the generally worse performance of the SMG can be improved by tweaking its different weapon parameters (via exhaustive or evolutionary search) and testing the modified weapon against all other weapons in a diverse set of game levels until a modified SMG receives a sufficiently increased ratio of predicted balanced instances compared to the original.

A.6 Summary

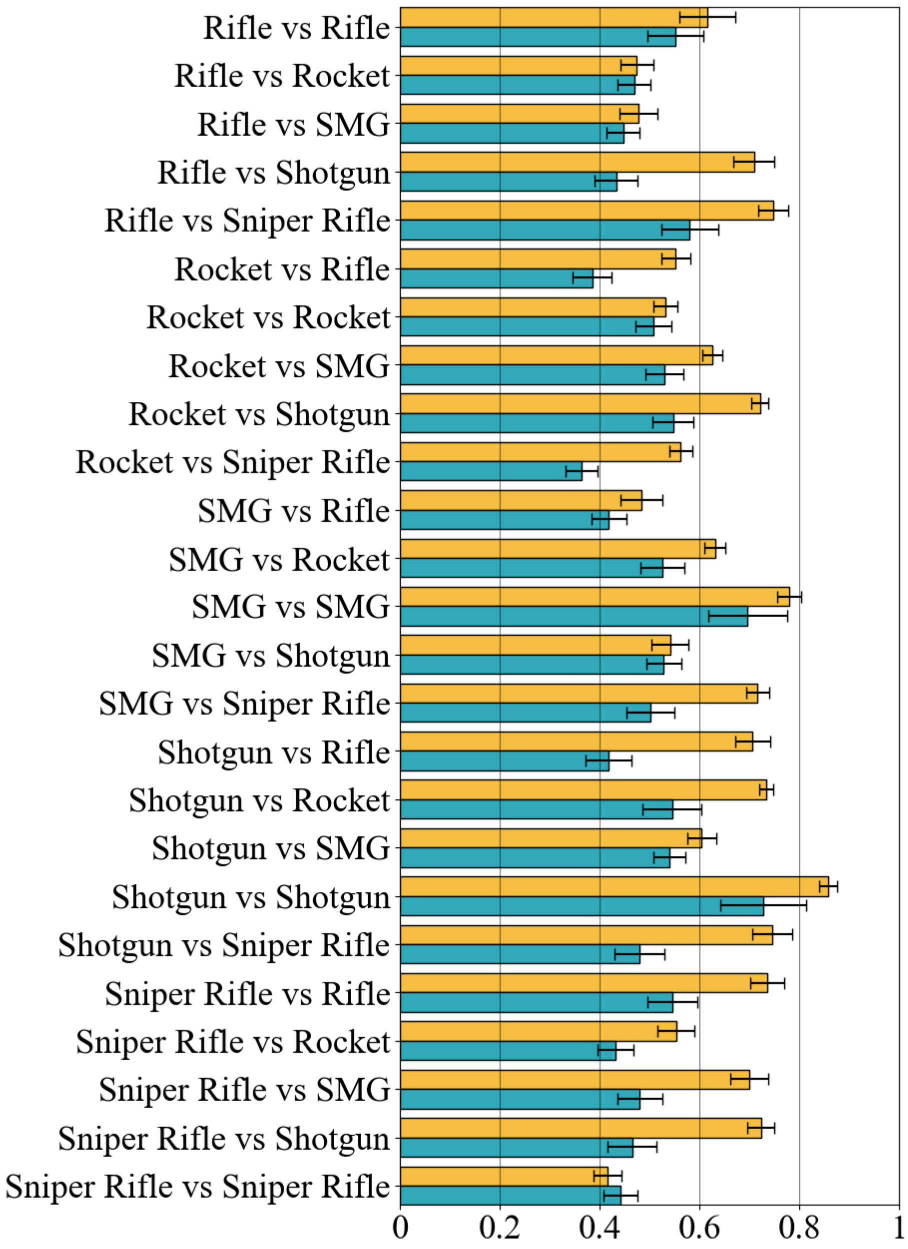
This appendix introduced a computational model which has learned to classify shooter game matches as balanced or as favoring one or the other team. In order to provide a large and diverse dataset for the model to learn from, over 50,000 simulated playthroughs by artificial agents were produced. In the shooter test bed introduced in this study, teams of three artificial agents, using one weapon type per team, attempted to score more kills than the other team. The arena in which combat took place had large objects to block line of sight and small objects to take cover in. In order for the model to learn a broad set of level patterns, 39 generators were used to create a diverse set of levels in which the matches were simulated.

Results showed that while weapons had quite clear imbalances in power level which were easy to learn, combining those learned patterns with levels allowed the computational model to increase its accuracy above all other tested approaches. By fusing levels stored simply as an image and weapons stored numerically, it seems likely that the learned model can be used to generate new balanced levels for a specific weapon pair, to fine-tune weapons so that the weaker weapons have a higher chance of being useful, and to provide real-time feedback to human designers without the need for computationally heavy simulations.

As a proof of concept for the computational model of gameplay, the described approach was successful. This appendix showed that the CNN can pick up on the relations between weapons and levels with some degree of success and provided directions for improvement.



(a) Accuracy averaged per weapon of 1st team.



(b) Accuracy per weapon pairing.

Figure A.7: Test accuracy of the CNN (yellow) and the ANN (blue) on matchups split by weapons. Error bars indicate the 95% confidence interval.

Appendix B

Game Level Metrics

This appendix describes the metrics that were used to compute correlations between map features and the targeted gameplay outcomes, i.e., the game duration (t) and the kill ratio of player 1 (KR). These metrics are based on the general game level metrics *Area Control*, *Exploration* and *Safety* as defined by (Liapis et al., 2013c). We compute the Pearson correlation and report the r values of significant correlations ($a = 0.0005$). The results of this analysis have been summarized in Section 5.2.2. These metrics have also been used as input channel in the multi-modal fusion extension that is described in Section 8.2.2 and (Liapis et al., 2019a).

All level metrics are in one way or another based on the occurrence of certain tile types. We define a level to be a set of tiles T, which contains subsets of tiles of a certain type: ground floor (F_0), first floor (F_1), wall (W), armor pickup (A), double damage pickup (D), health pickup (H), stairs (S). The number of tiles of a certain type can then be denoted as the size of this set, e.g., for the number of armor pickups we write $|A|$. For simplicity, we define two aggregate sets of respectively passable tiles, $P = F_0 \cup F_1$, and resource tiles $R = A \cup D \cup H$. Additionally, we define the bottom left and upper right corner tiles as the sets that indicate respectively the base of player 1 ($B_1 = \{b_1\}$) and player 2 ($B_2 = \{b_2\}$). As such, both of these sets always contain a single tile.

For the sake of readability this set notation ignores the structural properties of the level. Nevertheless, it should be noted that the structural relations are crucial for these metrics and computations were in fact done on a matrix representation of the level.

All levels in this thesis consist of 20×20 tiles, which means that each level has a total of 400 tiles ($|T| = 400$) and a perimeter of 80 tiles. Constructive level generation for the dataset utilizes an intermediate representation of 4×4 cells (each of which contains 5×5 tiles). The generator placed at most one pickup per cell, which means that a single map of the training data can contain a maximum of 16 resource tiles. The hard-coded numbers in the metrics are based on these properties.

Area Control

Area Control, as $f_a(S_N)$ in (Liapis et al., 2013c), evaluates the number of passable tiles much closer to one tile the set S_N than other tiles in the same set. This is computed both for the players' bases as well as the different types of pickups. See Table B.1 for the correlations with the gameplay outcomes.

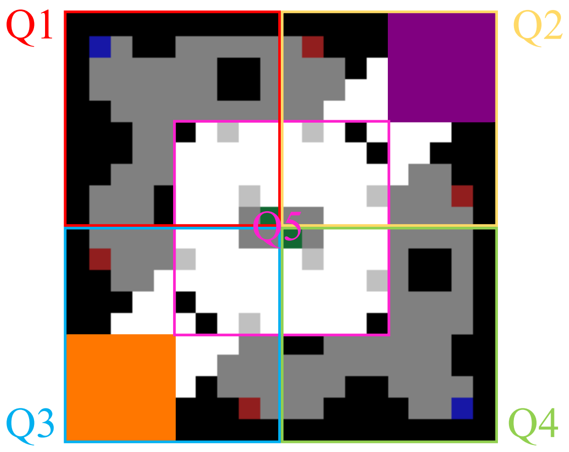


Figure B.1: The five quadrants that were used to compute the ratios in Table B.5.

Distances

These metrics evaluate the distance relations between tiles of different types. For this we define the function *dist(x,y)*, which computes the shortest walkable distance between two tiles. Based on this distance, we define two normalized distance metrics:

$$f_{d_1}(S_N, S_M) = \sum_{x \in S_N} \sum_{y \in S_M} min(\frac{dist(x,y)}{80}, 1) \tag{B.1}$$

$$f_{d_2}(S_N, S_M) = \sum_{x \in S_N} \sum_{y \in S_M} minmax(dist(x,y), S_M) \tag{B.2}$$

The former sums up the pairwise distances between two sets of tiles, which are manually normalized based on the perimeter of the map and then truncated to one if necessary. The latter computes a similar sum of pairwise distances, which are normalized via min-max normalization. Additionally, we compute a normalized average distance between all the tiles in S_N and all tiles in set S_M:

$$f_{d_3}(S_N, S_M) = \frac{1}{|S_N|} \frac{1}{|S_M|} f_{d_1}(S_N, S_M) \tag{B.3}$$

Table B.3 contains the correlations between these metrics and the gameplay outcomes. Note that these distance metrics are only applied to bases, which means that in this case S_N only contains one tile.

Exploration

Exploration, as $f_e(S_N, S_M)$ in (Liapis et al., 2013c), is based on a flood-fill algorithm that starts at a tile in the set S_N and counts the number of passable tiles that were filled until a tile of set S_M was encountered. In this research, the exploration distance as well as the shortest distance are computed between the players' bases and between the bases and the pickups. Aside from the exploration value between bases, we evaluate the average exploration value from a base to all resources ($f_e(S_N, R)$). Table B.2 contains the correlations between these metrics and the gameplay outcomes.

Safety

Safety (or strategic resource control), as $f_s(S_N, S_M)$ in (Liapis et al., 2013c), evaluates whether tiles of set S_N are significantly closer to tiles in set S_M than other tiles in the set S_N. Within the context of this game, it is used to compute whether pickups are much closer to the players' bases than other pickups and whether the base of one player is generally much closer to the pickups than that of the other. The latter is computed by the average distance between Table B.4 contains the correlations between these metrics and the gameplay outcomes.

Tile Ratios

Tile ratios compute the percentage of tiles in a certain area; either the complete level T or one of the quadrants $Q1 - Q5$ indicated in Fig. B.1. Table B.5 contains the correlations between the tile ratio metrics and the gameplay outcomes. While the tile ratios mostly correlate with game duration, there is a small effect of healthpacks on KR. Interestingly, this effect is stronger in the bottom left and top right quadrants, which contain the spawn areas of the players. Additionally, the correlation between the tile types and game duration is particularly strong in the center quadrant. Which makes sense because this area defines whether players need to take large detours to find each other and whether there are resources in the area that is expected to have the most battles.

Table B.1: Significant Pearson correlations ($\alpha = 0.0005$) of Area Control metrics.

Name	Formula	t	KR
AC1	$\frac{f_a(B_1)}{\lvert P\rvert}$	0.06	0.02
AC2	$\frac{f_a(B_2)}{\lvert P\rvert}$	0.06	-0.01
AC3	$\frac{f_a(B_1)+f_a(B_1)}{\lvert P\rvert}$	0.09	
AC4	$\frac{f_a(A)+f_a(D)+f_a(H)}{\lvert P\rvert}$	0.15	
AC5	$\frac{f_a(A)}{\lvert P\rvert}$	0.11	
AC6	$\frac{f_a(D}{\lvert P\rvert}$	0.02	0.01
AC7	$\frac{f_a(H)}{\lvert P\rvert}$	0.04	

Table B.2: Significant Pearson correlations ($\alpha = 0.0005$) of Exploration metrics.

Name	Formula	t	KR
E1	$\frac{f_e(B_2,B_1)}{\lvert P\rvert}$	-0.14	0.01
E2	$\frac{f_e(B_1,B_2)}{\lvert P\rvert}$	-0.12	-0.02
E3	$\frac{f_e(B_1,R)}{\lvert P\rvert}$	0.04	-0.04
E4	$\frac{f_e(B_2,R)}{\lvert P\rvert}$		0.04
E5	$\frac{E3}{TR7}$	-0.06	-0.02
E6	$\frac{E4}{TR7}$	-0.08	0.02

Table B.3: Significant Pearson correlations ($\alpha = 0.0005$) of Distance metrics.

Name	Formula	t	KR
D1	$\frac{f_{d_1}(B1,T)}{\lvert P\rvert}$	0.21	-0.01
D2	$\frac{f_{d_1}(B2,T)}{\lvert P\rvert}$	0.21	0.02
D3	$\frac{f_{d_2}(B1,T)}{\lvert P\rvert}$	0.07	
D4	$\frac{f_{d_2}(B2,T)}{\lvert P\rvert}$	0.09	
D5	$f_{d_1}(B1, B2)$	0.20	
D6	$f_{d_1}(B2, B1)$	0.19	0.01
D7	$f_{d_3}(B1, R)$	0.14	-0.05
D8	$f_{d_3}(B2, R)$	0.11	0.05
D9	$f_{d_3}(B1, A)$	0.12	
D10	$f_{d_3}(B2, A)$	0.09	
D11	$f_{d_3}(B1, D)$	0.06	
D12	$f_{d_3}(B2, D)$	0.08	
D13	$f_{d_3}(B1, H)$	0.05	-0.06
D14	$f_{d_3}(B2, H)$	0.02	0.04

Table B.4: Significant Pearson correlations ($\alpha = 0.0005$) of Safety metrics.

Name	Formula	t	KR
S1	$f_s(B_1, P)$	0.06	0.02
S2	$f_s(B_2, P)$	0.05	-0.01
S3	$S1 + S2$	0.1	
S4	$f_s(A, P)$	0.1	
S5	$f_s(D, P)$	0.03	
S6	$f_s(H, P)$		0.01
S7	$S4 + S5 + S6$	0.15	
S8	$f_s(B_1, R)$	-0.01	0.05
S9	$f_s(B_2, R)$	0.02	-0.05
S10	$f_s(B_1, A)$	0.02	
S11	$f_s(B_2, A)$	0.07	
S14	$f_s(B_1, D)$	-0.02	0.08
S15	$f_s(B_2, D)$	0.03	-0.09
S12	$f_s(B_1, H)$	0.01	
S13	$f_s(B_2, H)$		
S16	$\frac{S4}{TR4}$	0.05	
S17	$\frac{S6}{TR6}$	-0.01	
S18	$\frac{S5}{TR5}$		
S19	$\frac{S7}{TR7}$	-0.02	

Table B.5: Significant Pearson correlations ($\alpha = 0.0005$) of Tile Ratio metrics.

Name	Formula	t	KR
TR1	$\frac{\|F_0\|}{400}$	-0.13	
TR2	$\frac{\|F_1\|}{400}$	0.17	
TR3	$\frac{\|W\|}{400}$	-0.02	
TR4	$\frac{\|A\|}{400}$	0.13	
TR5	$\frac{\|D\|}{400}$		
TR6	$\frac{\|H\|}{400}$	0.05	-0.01
TR7	$\frac{\|R\|}{400}$	0.11	
TR8	$\frac{\|S\|}{400}$	-0.05	
TR9	$\frac{\|P\|}{400}$	0.02	
TR10	$\frac{\|Floor_0 \cap Q_1\|}{100}$	0.1	
TR11	$\frac{\|Floor_1 \cap Q_1\|}{100}$	-0.02	
TR12	$\frac{\|W \cap Q_1\|}{100}$	-0.09	
TR13	$\frac{\|A \cap Q_1\|}{4}$	0.08	
TR14	$\frac{\|D \cap Q_1\|}{4}$	0.09	
TR15	$\frac{\|H \cap Q_1\|}{4}$	0.05	
TR16	$\frac{\|S \cap Q_1\|}{100}$	0.03	
TR20	$\frac{\|Floor_0 \cap Q_2\|}{100}$	-0.3	
TR21	$\frac{\|Floor_1 \cap Q_2\|}{100}$	0.24	
TR22	$\frac{\|W \cap Q_2\|}{100}$	0.15	
TR23	$\frac{\|A \cap Q_2\|}{4}$		-0.01
TR24	$\frac{\|D \cap Q_2\|}{4}$	-0.05	
TR25	$\frac{\|H \cap Q_2\|}{4}$	-0.04	0.06
TR26	$\frac{\|S \cap Q_2\|}{100}$	-0.06	
TR30	$\frac{\|Floor_0 \cap Q_3\|}{100}$	-0.24	-0.01
TR31	$\frac{\|Floor_1 \cap Q_3\|}{100}$	0.18	
TR32	$\frac{\|W \cap Q_3\|}{100}$	0.14	0.01
TR33	$\frac{\|A \cap Q_3\|}{4}$	0.05	
TR34	$\frac{\|D \cap Q_3\|}{4}$	-0.06	
TR35	$\frac{\|H \cap Q_3\|}{4}$	0.01	-0.07
TR36	$\frac{\|S \cap Q_3\|}{100}$	-0.07	-0.01
TR40	$\frac{\|Floor_0 \cap Q_4\|}{100}$	0.02	
TR41	$\frac{\|Floor_1 \cap Q_4\|}{100}$	0.05	
TR42	$\frac{\|W \cap Q_4\|}{100}$	-0.06	
TR43	$\frac{\|A \cap Q_4\|}{4}$	0.12	
TR44	$\frac{\|D \cap Q_4\|}{4}$	0.03	
TR45	$\frac{\|H \cap Q_4\|}{4}$	0.08	-0.01
TR46	$\frac{\|S \cap Q_4\|}{100}$	0.02	
TR50	$\frac{\|Floor_0 \cap Q_5\|}{100}$	-0.48	-0.01
TR51	$\frac{\|Floor_1 \cap Q_5\|}{100}$	0.35	
TR52	$\frac{\|W \cap Q_5\|}{100}$	0.33	0.01
TR53	$\frac{\|A \cap Q_5\|}{4}$	-0.05	
TR54	$\frac{\|D \cap Q_5\|}{4}$	-0.14	
TR55	$\frac{\|H \cap Q_5\|}{4}$	-0.1	
TR56	$\frac{\|S \cap Q_5\|}{100}$	-0.26	

Appendix C

Class Parameter Activations

Detailed below are the results of applying Gradient-weighted Activation Mapping (Selvaraju et al., 2017) to the class parameters of CNN3. In Section 6.7.2 we described an example of how grad-CAM can be applied to highlight the important areas for the decisions made by the convolutional neural network. We examined the activation heatmaps for the generated level G2 of Fig. 7.2 and designed level D3 of Fig. 7.3 in two matchups with archetypal classes of *Team Fortress 2*: Sniper versus Scout, and Heavy versus Scout (see the parameters of each class in Table 4.1).

Instead of the convolutional features of the map, this appendix describes the results of applying this procedure to the class parameters. A summary of this was provided in Section 6.7.4. The grad-CAM algorithm can be applied to the class parameters by computing the gradient of the output with respect to the input layer. In which case the values are not averaged over all nodes in the layer, as this would result in a single contribution value for all class parameters. Table C.1 and Table C.2 respectively contain the activations with respect to predictions of duration and kill ratio.

Interestingly, when observing the most active class parameters (at 0.10 or above) the relative contributions share a pattern across maps. The accuracy is always the most important factor. For the Sniper vs Scout matchup it is the accuracy of the Sniper, with a value of -1 for duration and 1 for KR. For the Heavy vs Scout matchup it is the accuracy of the Scout (-1 for both duration and KR). These numbers indicate to the designer that in a match between Sniper and Scout, an increase in match duration would require a drop in the accuracy of the Sniper, while a higher kill ratio for the Sniper would require reducing the accuracy of the Scout (as increasing the accuracy of the Sniper is impossible). Similarly, in a match between Heavy and Scout, both increasing the match duration and increasing the kill ratio of the Heavy would require a drop in the accuracy of the Scout. For all Heavy vs Scout matchups, the relative effect of the HP of the Heavy is around 0.2. Indicating that the survivability of this class is an important factor in both its kill ratio as in extending the duration of the match. For all time predictions, the damage of the player with the highest damage is a factor of approximately -0.1. Interesting here is that even though the damage difference between the two classes is very different (0.79 between Sniper and Scout, 0.21 between Scout and Heavy), the effect on the duration is roughly the same.

Table C.1: The class parameter activations of CNN3 for duration prediction of four match-ups in the designed level D3 of Fig. 7.3 and the generated level G2 of Fig. 7.2. The parameters are: hit points (H), speed (S), damage (D), accuracy (A), clip size (C), rate of fire (RF), bullets per shot (B) and range (R). The most active class parameters (at 0.10 or above) are marked in bold.

		Sniper vs Scout		Scout vs Sniper		Heavy vs Scout		Scout vs Heavy	
		D3	G2	D3	G2	D3	G2	D3	G2
Player 1	H	0	0	0	0	**0.22**	**0.22**	0	0
	S	0	0	0	-0.01	0	0	-0.01	-0.01
	D	**-0.11**	**-0.10**	-0.01	0.01	0	0	**-0.11**	**-0.10**
	A.	**-1**	**-1**	**-0.15**	0.01	**-0.59**	**-0.25**	**-1**	**-1**
	RF	0	0	0	0	0.01	0.01	0	0
	C	0	0	0	0	0	0	0	0
	B	0	0	-0.03	-0.03	0	0	-0.09	**-0.14**
	R	-0.06	-0.06	0	0	0	0	0	0
Player 2	H	0	0	0	0	0	0	**0.23**	**0.18**
	S	0	0	0	0	0	0	0	0
	D	0.01	-0.03	-0.08	-0.05	**-0.10**	**-0.10**	0	0
	A	0.05	**-0.46**	**-1**	**-1**	**-1**	**-1**	**-0.28**	-1
	RF	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	-0.01
	B	-0.02	-0.06	0	0	-0.09	-0.07	0	0
	R	0	0	-0.05	-0.05	0	0	0	0

Table C.2: The class parameter activations of CNN3 for KR prediction of four match-ups in D3 of Fig. 7.3 and G2 of Fig. 7.2. The parameters are the same as in Table C.1. The most active class parameters (at 0.10 or above) are marked in bold.

		Sniper vs Scout		Scout vs Sniper		Heavy vs Scout		Scout vs Heavy	
		D3	G2	D3	G2	D3	G2	D3	G2
Player 1	H	0	0	0	0	**0.19**	**0.19**	0	0
	S	0	0	0	0	0	0	0	0
	D	0.03	0.03	0.05	0.05	0	0	0.09	**0.10**
	A	1	1	**0.74**	**0.78**	**0.44**	**0.45**	1	1
	R	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0
	B	0	0	-0.02	-0.02	0	0	0.04	0.04
	R	0.01	0.01	0	0	0	0	0	0
Player 2	H	0	0	0	0	0	0	**-0.22**	**-0.22**
	S	-0.01	-0.01	0	0	-0.01	-0.01	0	0
	D	-0.03	-0.03	-0.06	-0.07	-0.08	-0.08	0	0
	A	**-0.37**	**-0.36**	-1	-1	-1	-1	**-0.43**	**-0.42**
	R	0	0	0	0	0	0	-0.01	-0.01
	C	0	0	0	0	0	0	0.01	0.01
	B	0.02	0.02	0	0	-0.05	-0.05	0	0
	R	0	0	-0.01	-0.01	0	0	0	0

Bibliography

Alberto Alvarez, Steve Dahlskog, Jose Font, Johan Holmberg, Chelsi Nolasco, and Axel Österman. Fostering creativity in the mixed-initiative evolutionary dungeon designer. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, page 50. ACM, 2018. — Cited on pages 2, 19, and 118.

Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. Empowering quality diversity in dungeon design with interactive constrained map-elites. In *Proceedings of the IEEE Conference on Games*. IEEE, 2019. — Cited on page 117.

Nicolas A. Barriga, Marius Stanescu, and Michael Buro. Combining strategic learning with tactical search in real-time strategy games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017. — Cited on page 17.

Regina Bernhaupt. *Evaluating user experience in games: Concepts and methods.* Springer, 2010. — Cited on page 1.

Marlene Beyer, Aleksandr Agureikin, Alexander Anokhin, Christoph Laenger, Felix Nolte, Jonas Winterberg, Marcel Renk, Martin Rieger, Nicolas Pflanzl, Mike Preuss, and Vanessa Volz. An integrated process for game balancing. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016. — Cited on pages 12 and 118.

Anand Bhojan and Hong Wei Wong. Tital–asynchronous multiplayer shooter with procedurally generated maps. *Entertainment Computing*, 16:81–93, 2016. — Cited on page 15.

Rafael Bidarra, Antonios Liapis, Mark J. Nelson, Mike Preuss, and Georgios N. Yannakakis. Creativity facet orchestration: the whys and the hows. In *Dagstuhl Reports*, volume 5, pages 217–218. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. — Cited on page 3.

John A. Biles, Peter G. Anderson, and Laura W. Loggi. Neural network fitness functions for a musical iga. In *In Proceedings of the Soft Computing Conference (SOCO*, pages 39–44. ICSC Academic Press, 1996. — Cited on page 33.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738. — Cited on pages 24, 26, 29, and 62.

Staffan Björk and Jussi Holopainen. *Patterns in Game Design.* Charles River Media, 2004. — Cited on pages 3, 12, and 51.

Cameron Browne and Frédéric Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010. — Cited on pages 2, 3, 4, 14, and 21.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1lJJnR5Ym`. — Cited on page 110.

William Cachia, Antonios Liapis, and Georgios N. Yannakakis. Multi-level evolution of shooter levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2015. — Cited on pages 15, 118, and 132.

Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. *Interface*, pages 395–402, 2011a. — Cited on page 14.

Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi. Evolving interesting maps for a first person shooter. In *Proceedings of the Applications of evolutionary computation*, 2011b. — Cited on pages 15, 111, and 132.

Max Cherny. This violent videogame has made more money than any movie ever. `https://www.marketwatch.com/story/this-violent-videogame-has-made-more-money-than-any-movie-ever-2018-04-06`, 2018. Accessed: 2019-09-17. — Cited on page 1.

François Chollet et al. Keras. `https://keras.io`, 2015. — Cited on page 61.

Francois Chollet. How convolutional neural networks see the world, 2016. URL `https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html`. — Cited on pages 68 and 69.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *Proceedings of the International Conference on Learning Represenations*, pages 62:1–62:14, 2016. — Cited on pages 27 and 28.

Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, New York, 2nd edition, 2007. — Cited on page 32.

Simon Colton. Creativity versus the perception of creativity in computational systems. In *AAAI spring symposium: creative intelligent systems*, volume 8, 2008. — Cited on page 18.

Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *IEEE Conference on Computational Intelligence and Games*, pages 289–296. IEEE, 2011. — Cited on pages 20 and 117.

Michael Cook, Simon Colton, and Jeremy Gow. The angelina videogame design system—part i. *IEEE Transactions on Computational Intelligence and AI in Games*, 9 (2):192–203, 2016a. — Cited on page 20.

Michael Cook, Simon Colton, and Jeremy Gow. The angelina videogame design system—part ii. *IEEE Transactions on Computational Intelligence and AI in Games*, 9 (3):254–266, 2016b. — Cited on pages 4 and 20.

Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001. — Cited on page 32.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. — Cited on pages 32 and 42.

Olivier Delalleau, Emile Contal, Eric Thibodeau-Laufer, Raul Chandias Ferrari, Yoshua Bengio, and Frank Zhang. Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):167–177, 2012. — Cited on page 18.

Joris Dormans and Stefan Leijnen. Combinatorial and exploratory creativity in procedural content generation. In *Proceedings of the FDG Workshop on Procedural Content Generation in Games*, 2013. — Cited on page 18.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *Proceedings of the International Conference on Learning Representations*, pages 1–14, 2017. — Cited on page 17.

Anders Drachen and Alessandro Canossa. Analyzing spatial user behavior in computer games using geographic information systems. In *Proceedings of the 13th international MindTrek Conference*, pages 182–189. ACM, 2009. — Cited on page 112.

EA. Self-learning agents play battlefield 1. `https://www.ea.com/news/self-learning-agents-play-bf1`, 2018. Accessed: 2019-09-19. — Cited on page 1.

Edge. Forge ahead. Edge Magazine, 2018. URL `https://montreal.ubisoft.com/wp-content/uploads/2018/04/Forge-ahead.pdf`. Accessed: 2019-09-19. — Cited on page 1.

Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing.* Springer, 1st edition, 2003. ISBN 978-3-540-40184-1. Corrected 2nd printing, 2007. — Cited on pages 29 and 31.

Agoston E. Eiben and Selmar K. Smith. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011. — Cited on page 31.

Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game analytics.* Springer London Limited, 2016. — Cited on page 1.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies. — Cited on page 68.

Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons, New York, 1966. — Cited on page 29.

Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multi-objective optimization. *Evolutionary computation*, 3(1):1–16, 1995. — Cited on page 32.

Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games.* CRC Press, 4th edition, 2019. — Cited on page 12.

Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106. ACM, 2017. — Cited on page 117.

Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. Doom level generation using generative adversarial networks. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pages 316–323. IEEE, 2018. — Cited on page 16.

Robert Giusti, Kenneth Hullett, and Jim Whitehead. Weapon design patterns in shooter games. In *Proceedings of the FDG workshop on on Design Patterns in Games*, pages 3:1–3:7, 2012. — Cited on page 12.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`. — Cited on pages 26 and 29.

Jeremy Gow and Joseph Corneli. Towards generating novel games using conceptual blending. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015. — Cited on page 18.

Daniele Gravina and Daniele Loiacono. Procedural weapons generation for unreal tournament III. In *Proceedings of the IEEE Conference on Games, Entertainment & Media*, pages 173–180, 2015. — Cited on pages 2, 14, and 103.

Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. Constrained surprise search for content generation. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 166–173, 2016. — Cited on pages 14 and 21.

Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Procedural content generation through quality diversity. *arXiv preprint arXiv:1907.04053*, 2019. — Cited on pages 14 and 117.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. — Cited on pages 26 and 29.

Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen, Bartlomiej Kozakowski, Richard Meurling, and Lele Cao. Human-like playtesting with deep learning. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018. — Cited on page 1.

Christian Güttler and Troels Degn Johansson. Spatial principles of level-design in multi-player first-person shooters. In *Proceedings of the 2nd workshop on Network and system support for games*, pages 158–170. ACM, 2003. — Cited on page 12.

Matthew Guzdial and Mark Riedl. Learning to blend computer game levels. In *Proceedings of the Seventh International Conference on Computational Creativity*, 2016. — Cited on page 18.

Matthew Guzdial and Mark Riedl. Combinatorial creativity for procedural content generation via machine learning. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. — Cited on page 18.

Matthew Guzdial, Nathan Sturtevant, and Boyang Li. Deep static and dynamic level analysis: A study on infinite mario. In *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016. — Cited on page 16.

Matthew Guzdial, Nicholas Liao, and Mark Riedl. Co-creative level design via machine learning. *Proceedings of the AAAI Workshop on Experimental AI in Games*, 2018. — Cited on page 18.

Erin Jonathan Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009. — Cited on pages 14, 15, and 20.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016. — Cited on page 113.

John Henry Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, 1st edition, 1975. — Cited on page 29.

James Holloway. Deathmatch map design: The architecture of flow. `http://www.gamasutra.com/view/feature/195069/deathmatch_map_design_the_.php`, 2013. Accessed: 2019-09-04. — Cited on page 12.

Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Monte-carlo tree search for persona based player modeling. In *Proceedings of the Eleventh AIIDE Conference*, 2015. — Cited on pages 1 and 4.

Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 25–30, 2007. — Cited on page 12.

Amy K. Hoover, William Cachia, Antonios Liapis, and Georgios N. Yannakakis. Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay. In *Proceedings of the International Conference on Evolutionary and Biologically Inspired Music and Art (EvoMusArt)*, volume 9027, LNCS, pages 101–112. Springer, 2015. — Cited on page 20.

Kenneth Hullet and Jim Whitehead. Design patterns in fps levels. In *Proceedings of the Foundations of Digital Games Conference*, 2010. — Cited on pages 3, 12, 78, and 122.

Kenneth Hullett. *The science of level design: Design patterns and analysis of player behavior in first-person shooter levels.* PhD thesis, UC Santa Cruz, 2012. — Cited on page 12.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on International Conference on Machine Learning*, volume 37, pages 448–456, 2015. — Cited on pages 28 and 60.

Yves Jacquier, Olivier Delalleau, and Adrien Logut. Deep learning montréal @ autodesk – ubisoft laforge and some ai and machine learning projects, 2018. URL `https://youtu.be/PsBA907ytek`. — Cited on page 1.

Rishabh Jain, Aaron Isaksen, Christoffer Holmgard, and Julian Togelius. Autoencoders for level generation, repair, and recognition. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016. — Cited on page 16.

Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, 9(1):3–12, 2005. — Cited on pages 33, 104, and 110.

Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011. — Cited on pages 16 and 33.

Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. On evolutionary optimization with approximate fitness functions. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pages 786–793. Morgan Kaufmann Publishers Inc., 2000. — Cited on page 33.

Yaochu Jin, Handing Wang, Tinkle Chugh, Dan Guo, and Kaisa Miettinen. Data-driven evolutionary optimization: an overview and case studies. *IEEE Transactions on Evolutionary Computation*, 23(3):442–458, 2018. — Cited on pages 16 and 110.

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, February 2019. — Cited on page 17.

Giorgos Karafotias, Mark Hoogendoorn, and Agoston E. Eiben. Parameter control in evolutionary algorithms: trends and challenges. *IEEE Transactions on Evolutionary Computing*, 19(2):167–187, 2015. — Cited on page 31.

Daniel Karavolos, Anders Bouwer, and Rafael Bidarra. Mixed-initiative design of game levels: Integrating mission and space into level generation. In *FDG*, 2015. — Cited on page 19.

Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. Learning the patterns of balance in a multi-player shooter game. In *Proceedings of the FDG workshop on PCG in Games*, pages 70:1–70:10, 2017. — Cited on pages 47 and 121.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016. — Cited on page 17.

Ahmed Khalifa, Michael Cerny Green, Gabriella Barros, and Julian Togelius. Intentional computational level design. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019. — Cited on page 18.

Steven O. Kimbrough, Gary J. Koehler, Ming Lu, and David H. Wood. On a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2): 310–327, 2008. — Cited on page 132.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 5 2015. — Cited on pages 28 and 61.

Douglas M. Kline and Victor L. Berardi. Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Applications*, 14(4): 310–318, 2005. — Cited on page 127.

Pier Luca Lanzi, Daniele Loiacono, and Ricardo Stucchi. Evolving maps for match balancing in first person shooters. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 325–334, 2014. — Cited on pages 14, 15, and 21.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. — Cited on page 62.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 5 2015. — Cited on pages 1, 17, 26, and 109.

Scott Lee, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Predicting resource locations in game maps using deep convolutional neural networks. In *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016. — Cited on page 16.

Antonios Liapis. Piecemeal evolution of a first person shooter level. In *Applications of Evolutionary Computation*. Springer, 2018. — Cited on pages 12, 15, and 118.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012. — Cited on page 14.

Antonios Liapis, Héctor P. Martínez, Julian Togelius, and Georgios N. Yannakakis. Transforming exploratory creativity with DeLeNoX. In *Proceedings of the International Conference on Computational Creativity*, 2013a. — Cited on page 18.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Designer modeling for personalized game content creation tools. In *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013b. — Cited on page 14.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Towards a generic method of evaluating game levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013c. — Cited on pages 3, 12, 14, 15, 19, 47, 53, 58, 87, 114, 132, 135, 136, and 137.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of the Conference on the Foundations of Digital Games*, pages 213–220, 2013d. — Cited on pages 2, 19, and 118.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Designer modeling for sentient sketchbook. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014a. — Cited on page 19.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Computational game creativity. In *Proceedings of the International Conference on Computational Creativity*, 2014b. — Cited on pages 3, 18, and 35.

Antonios Liapis, Christoffer Holmgård, Georgios N. Yannakakis, and Julian Togelius. Procedural personas as critics for dungeon generation. In *Applications of Evolutionary Computation*, volume 9028. Springer, 2015a. — Cited on pages 1 and 13.

Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Constrained novelty search: A study on game content generation. *Evolutionary computation*, 23(1):101–129, 2015b. — Cited on page 117.

Antonios Liapis, Gillian Smith, and Noor Shaker. Mixed-initiative content creation. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, pages 195–214. Springer, 2016. — Cited on page 2.

Antonios Liapis, Daniel Karavolos, Konstantinos Makantasis, Konstantinos Sfikas, and Georgios N. Yannakakis. Fusing level and ruleset features for multimodal learning of gameplay outcomes. In *Proceedings of the IEEE Conference on Games*, 2019a. — Cited on pages 70, 114, 117, and 135.

Antonios Liapis, Georgios N. Yannakakis, Mark J. Nelson, Mike Preuss, and Rafael Bidarra. Orchestrating game generation. *IEEE Transactions on Games*, 11(1):48–68, 2019b. — Cited on pages 2, 3, 4, 20, 21, 39, 117, and 118.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Proceedings of the Conference on Neural Information Processing Systems*, 2018. — Cited on page 47.

Daniele Loiacono and Luca Arnaboldi. Fight or flight: Evolving maps for cube 2 to foster a fleeing behavior. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 199–206. IEEE, 2017. — Cited on page 15.

Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Framing tension for game generation. In *Proceedings of the International Conference on Computational Creativity*, 2016. — Cited on pages 20 and 113.

Andrew L Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning*, volume 28, 2013. — Cited on pages 27 and 28.

Tiago Machado, Daniel Gopstein, Andy Nealen, Oded Nov, and Julian Togelius. Ai-assisted game debugging with cicero. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018. — Cited on page 19.

Tiago Machado, Dan Gopstein, Andy Nealen, and Julian Togelius. Pitako–recommending game design elements in cicero. In *Proceedings of the IEEE Conference on Games*, pages 1–8. IEEE, 2019. — Cited on page 20.

Tambet Matiisen. The use of embeddings in openai five, 2018. URL `https://neuro.cs.ut.ee/the-use-of-embeddings-in-openai-five/`. Accessed:2019-07-22. — Cited on page 114.

Eric McDuffee and Alex Pantaleev. Team blockhead wars: Generating fps weapons in a multiplayer environment. In *Proceedings of the FDG workshop on Procedural Content Generation*, 2013. — Cited on page 14.

David Melhart, Ahmad Azadvar, Alessandro Canossa, Antonios Liapis, and Georgios N. Yannakakis. Your gameplay says it all: Modelling motivation in tom clancy's the division. In *Proceedings of the IEEE Conference on Games*, pages 1–8. IEEE, 2019. — Cited on page 1.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*, 2013. arXiv preprint, arXiv:1301.3781. — Cited on page 114.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015a. — Cited on page 109.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015b. — Cited on page 17.

Alexander Mordvintsev, Chris Olah, and Mike Tyka. Deepdream - a code example for visualizing neural networks, 2015a. https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html. — Cited on page 68.

Alexander Mordvintsev, Chris Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015b. https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. — Cited on page 68.

Mihail Morosan. *Automating Game-design and Game-agent Balancing through Computational Intelligence.* PhD thesis, University of Essex, 2019. — Cited on pages 14 and 16.

Mihail Morosan and Riccardo Poli. Online-trained fitness approximators for real-world game balancing. In *International Conference on the Applications of Evolutionary Computation*, pages 292–307. Springer, 2018. — Cited on pages 14 and 33.

Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, pages 807–814, 2010. — Cited on page 27.

NewZoo. 2019 global games market report. `https://newzoo.com/key-numbers/`, 2019. Accessed: 2019-09-17. — Cited on page 1.

Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3510–3520, 2017. — Cited on page 68.

David G. Novick and Stephen Sutton. What is mixed-initiative interaction? In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997. — Cited on page 2.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *Proceedings of the International Conference on Machine Learning*, 2017. — Cited on page 118.

Peter Thorup Ølsted, Benjamin Ma, and Sebastian Risi. Interactive evolution of levels for a competitive multiplayer fps. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1527–1534. IEEE, 2015. — Cited on pages 14 and 15.

Opsive. Ultimate first person shooter framework (ufps). `https://legacy.opsive.com/assets/UFPS/`, 2012. — Cited on page 40.

Opsive. Deathmatch ai kit documentation. `https://legacy.opsive.com/assets/DeathmatchAIKit/`, 2016. Accessed: 2018-02-02. — Cited on page 50.

Laura Parker. The science of playtesting. `https://www.gamespot.com/articles/the-science-of-playtesting/1100-6323661/`, 2012. Accessed: 2019-09-10. — Cited on page 12.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL `https://scikit-learn.org/`. — Cited on page 67.

Mike Preuss, Antonios Liapis, and Julian Togelius. Searching for good and diverse game levels. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014. — Cited on pages 2 and 14.

Mike Preuss, Thomas Pfeiffer, Vanessa Volz, and Nicolas Pflanzl. Integrated balancing of an rts game: Case study and toolbox refinement. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018. — Cited on pages 12, 13, 118, and 119.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. — Cited on page 28.

Ingo Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog, Stuttgart, 1973. — Cited on page 29.

Gabriel Rivera, Kenneth Hullett, and Jim Whitehead. Enemy npc design patterns in shooter games. In *Proceedings of the FDG Workshop on Design Patterns in Games*, page 6. ACM, 2012. — Cited on page 12.

Frank Rosenblatt. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms.* Spartan, 1962. — Cited on page 24.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. — Cited on pages 2 and 26.

Anurag Sarkar and Seth Cooper. Blending levels from different games using lstms. *Proceedings of the Experimental AI in Games Workshop*, 2018. — Cited on page 19.

Tom Schaul. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE, 2013. — Cited on page 19.

Jesse Schell. *The Art of Game Design: A Book of Lenses.* Morgan Kaufman Publishers, 1st edition, 2008. — Cited on pages 12 and 41.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015. — Cited on page 26.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017. — Cited on pages 68, 70, and 141.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *Proceedings of the International Conference on Learning Representations*, 2014. — Cited on page 68.

Sam Shahrani. Educational feature: A history and analysis of level design in 3d computer games - pt. 1. `https://www.gamasutra.com/view/feature/131083/educational_feature_a_history_and_.php`, 2013. Accessed: 2019-09-04. — Cited on page 11.

Noor Shaker, Mohammad Shaker, and Julian Togelius. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013. — Cited on page 19.

Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes, and Rafael Bidarra. Constructive generation methods for dungeons and levels. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, pages 31–55. Springer, 2016a. — Cited on pages 49, 50, and 103.

Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer, 2016b. — Cited on page 13.

Miguel Sicart. Digital games as ethical technologies. In *The philosophy of computer games*, pages 101–124. Springer, 2012. — Cited on page 4.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. — Cited on pages 2 and 17.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354–359, 2017. — Cited on page 114.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Proceedings of the International Conference on Learning Representations*, 2014. — Cited on page 68.

Ruben Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the FDG Workshop on Procedural Content Generation in Games*, page 2. ACM, 2010. — Cited on page 19.

Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):201–215, 2011. — Cited on pages 2 and 19.

Nathan Sorenson, Philippe Pasquier, and Steve DiPaola. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):229–244, 2011. — Cited on page 14.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *Journal of Machine Learning Research*, 15:1929–1958, 2014. — Cited on page 28.

Marius Stanescu, Nicolas A Barriga, Andy Hess, and Michael Buro. Evaluating real-time strategy game states using convolutional neural networks. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–7. IEEE, 2016. — Cited on pages 17 and 119.

Adam Summerville and Michael Mateas. Super mario as a string: Platformer level generation via lstms. In *Proceedings of DiGRA & FDG*, pages 127:1–127:16, 2016. — Cited on page 15.

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *arXiv preprint arXiv:1702.00539*, 2017. — Cited on pages 15 and 117.

Tyrnan Sylvester. *Designing Games: A Guide to Engineering Experiences.* O'Reilly Media, 1st edition, 2013. — Cited on page 12.

Clive Thompson. Halo 3: How microsoft labs invented a new science of play. `https://www.wired.com/2007/08/ff-halo-2/`, 2007. Accessed: 2017-10-20. — Cited on page 112.

Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *IEEE Symposium on Computational Intelligence and Games*, 2008. — Cited on pages 14 and 21.

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3), 2011. — Cited on pages 3, 13, 23, and 41.

Julian Togelius, Alex J. Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O. Stanley. Procedural Content Generation: Goals, Challenges and Actionable Steps. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 61–75. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013a. — Cited on page 18.

Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, Georgios N. Yannakakis, and Corrado Grappiolo. Controllable procedural map generation via multi-objective evolution. *Genetic Programming and Evolvable Machines*, 2013b. — Cited on page 3.

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. Is deep reinforcement learning really superhuman on atari? *arXiv preprint arXiv:1908.04683*, 2019. — Cited on page 17.

Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the FDG Workshop on PCG in Games*, pages 11–1, 2012. — Cited on page 20.

Alan Turing. Intelligent machinery. In B. Melzer and D. Michie, editors, *Machine Intelligence 5*, pages 3–23. Edinborough University Press, Edinborough, 1969. Originally, a National Physical Laboratory Report, 1948. — Cited on page 29.

Valve. Official team fortress 2 wiki. `https://wiki.teamfortress.com`, 2018. Accessed: 2018-02-02. — Cited on pages 43 and 44.

Rodrigo Vicencio-Moreira, Regan L. Mandryk, and Carl Gutwin. Now you can compete with anyone: Balancing players of different skill levels in a first-person shooter game. In *Proceedings of the Annual ACM Conference on Human Factors in Computing Systems*, pages 2255–2264. ACM, 2015. — Cited on page 13.

Vanessa Volz. *Uncertainty Handling in Surrogate Assisted Optimisation in Games*. PhD thesis, TU Dortmund, 2019. — Cited on pages 16, 17, and 33.

Vanessa Volz, Günter Rudolph, and Boris Naujoks. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 269–276. ACM, 2016. — Cited on pages 12, 14, and 16.

Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018. — Cited on page 16.

WePC. 2019 video game industry statistics, trends & data. `https://www.wepc.com/news/video-game-statistics/`, 2019. Accessed: 2019-09-17. — Cited on page 1.

Paul Werbos. *Beyond regression: New Tools for Prediction and Analysis in the Behavioural Sciences*. PhD Thesis, Harvard University, 1974. — Cited on page 25.

William Woof and Ke Chen. Learning to play general video-games via an object embedding network. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2018. — Cited on page 114.

Zuozhi Yang and Santiago Ontañón. Learning map-independent evaluation functions for real-time strategy games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–7. IEEE, 2018. — Cited on page 17.

Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018. — Cited on pages 13 and 18.

Georgios N. Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the Foundations of Digital Games Conference*, pages 37:1–37:8, 2014. — Cited on pages 5, 18, and 19.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833. Springer, 2014. — Cited on pages 68 and 69.

Jichen Zhu, Antonios Liapis, Sebastian Risi, Rafael Bidarra, and Michael Youngblood. Explainable ai for designers: A human-centered perspective on mixed-initiative co-creation. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 458–465, 2018. — Cited on page 68.

Alexander Zook and Mark Riedl. Learning how design choices impact gameplay behavior. *IEEE Transactions on Games*, 11(1):25–35, 2018. — Cited on pages 2, 12, and 13.