Personalized Procedural Map Generation in Games via Evolutionary Algorithms

A thesis submitted for the degree of
Doctor of Philosophy

William L. Raffe B.CompSc(Hons),
School of Computer Science and Information Technology,
College of Science, Engineering, and Health,
RMIT University,
Melbourne, Victoria, Australia.

5th August, 2014

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

William L. Raffe
School of Computer Science and Information Technology
RMIT University
5th August, 2014

This thesis and all of the work that has gone into it over the years is dedicated to my wonderful wife and the love of my life, Cynthia, who is always my biggest supporter and without whom I would never have reached this milestone.

Acknowledgments

This thesis was made possible by the considerable effort put forth by my supervisors, Dr. Fabio Zambetta and Dr. Xiaodong Li of the School of Computer Science and IT, RMIT University. The quality of the work herein has only been possible through their constant guidance and teachings through constant weekly meetings.

I would also like to thank all members of the Evolutionary Computation and Machine Learning (ECML) Group at RMIT. The insightful discussions on cutting edge research topics gave me additional education and exposed me to a wide range of research fields. Similarly, I would like to thank the research community at large for the effort put into anonymous reviews of research publications. The reviews that I received over the years gave direction to my work and strengthened the output.

This thesis was made possible by an RMIT University Ph.D. Scholarship and the availability of casual teaching positions at RMIT, without which it would have been financially impossible to continue my studies. Thank you to all academics and administrative staff at the School of Computer Science and IT for providing a creative and productive working environment.

Finally, I would like to thank my friends and family for the many years of support during the entirety of my university education. To my wife, Cynthia, for all of her love and care and for making every day an exciting adventure that we approach together. To my parents, Donna and John, for giving me every opportunity in life to succeed and for always guiding me towards a lifetime of happiness. To my brother and sister, Adam and Susannah, for being my best friends and role models throughout my entire life. To Cynthia's family, Timi, Louis, Daniel, and Katherine, for warmly accepting me into their family and for all the encouragement. And finally to the Cheeches for always making me laugh, no matter what.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- W. L. Raffe, F. Zambetta, and X. Li. Evolving patch-based terrains for use in video games. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, pages 363-370. ACM, 2011.
- W. L. Raffe, F. Zambetta, and X. Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In IEEE Congress on Evolutionary Computation (CEC), pages 1-8. IEEE, 2012.
- W. L. Raffe, F. Zambetta, and X. Li. Neuroevolution of content layout in the PCG: Angry Bots video game. In IEEE Congress on Evolutionary Computation (CEC), pages 673-680. IEEE, 2013.
- W. L. Raffe, F. Zambetta, X. Li, and K. O. Stanley. An Integrated Approach to Personalized Procedural Map Generation using Evolutionary Algorithms. In IEEE Transactions onComputation Intelligence and AI in Game (TCIAIG), 2014 (to appear).

The thesis was written in the LyX editor and typeset using the LaTeX 2ε document preparation system (*MiKTeX* distribution).
All trademarks are the property of their respective owners.

Contents

List of Figures

List of Tables

Glossary

AI	Artificial Intelligence
ANN	Artificial Neural Network
CF	Collaborative Filter
CPPN	Compositional Pattern-Producing Network
DDA	Dynamic Difficulty Adjustment
EA	Evolutionary Algorithm
EDPCG	Experience-driven PCG
ETT	Evolutionary Terrain Tool
FPS	First-Person Shooter (game)
IEC	Interactive Evolutionary Computation
NB	Naive Bayes
NEAT	NeuroEvolution of Augmenting Topologies
NPC	Non-Playable Character
PCG	Procedural Content Generation
PPM	Player Preference Modeling
RPG	Role-Playing Game
RS	Recommender Systems
RTS	Real-Time Strategy (game)
SBPCG	Search-based PCG

Abstract

In digital games, the *map* (sometimes referred to as the *level*) is the virtual environment in which a player navigates through and interacts with during gameplay. The map outlines the boundaries of play, aids in establishing rule systems, and supports the narrative. The map also directly influences the challenges that a player will experience and the pace of gameplay, a property that has previously been linked to a player's enjoyment of a game. Therefore, the entertainment value of the game is maximized by altering the challenge of a map to fit a player's expected ability.

In most industry leading games, creating maps is a lengthy manual process conducted by highly trained teams of designers. However, for many decades *procedural content generation* (PCG) techniques have been used to automate the map creation process to provide players with a larger range of experiences than would normally be possible. However, in recent years, PCG has been proposed as a means of tailoring game content to the preferences and skills of a specific player. These approaches fall into the recently established research field known as *Experience-driven PCG* (EDPCG), in which knowledge about the player is included in a generate-and-test PCG framework.

This thesis contributes to the growing EDPCG field with a focus on personalizing maps. Here, maps are represented via two distinct concepts; *geometry* and *content layout*. The geometry of a map defines the boundaries of play and the location of static virtual objects. Meanwhile, the content layout describes the location and quantity of interactive game assets, such as enemies and pick-ups. Both of these components affect a player's experience to varying degrees in different game genres. A third concept, *player preference modeling*, is added when adapting a map to a player's preferences and is conducted by collecting, interpreting, and utilizing knowledge about the player.

In this thesis, these three subcomponents are studied as interlocking mechanisms of the personalized procedural map generation process and solutions are proposed for each. Firstly,

geometry generation is conducted for both exterior and interior environments by connecting pre-made map segments to form larger, more complete maps. Player preferences are incorporated into this process via *interactive evolutionary computing*. The layout of content throughout the map is then determined by using features of the geometry as input to a *Compositional Pattern-Producing Network*, a population of which are evolved through *Neuroevolution of Augmenting Topologies*. Finally, a player's preferences for content layout are captured and utilized through the use of a player model based upon a *recommender system* (RS) framework.

The geometry generation process is firstly implemented into an *evolutionary terrain tool* that was created to aid novice game developers build the terrain of a map. All the solutions are then combined into the action-shooter game *PCG: Angry Bots* and evaluated through a large-scale public user experiment. The solutions are shown to perform well and can be generalized to other game genres. However, the experiment also gives rise to new questions and so this thesis concludes with a look at potential future work.

Chapter 1

Introduction

One of the most commonly stated motivating factors for video games research is that the size of the global games market warrants the investment of time towards improving it. For example, in the year 2011, video game retail sales were estimated to be generating US$21 billion in revenue [ESA, 2012]. For a comparison, worldwide box offices recorded US$32.6 billion in sales in the same year [McClintock, 2013] and the music industry's revenue was reported as US$16.6 billion [Smirke, 2012]. However, the manner in which these statistics are gathered are often contended [Kuchera, 2012] and some sources even report the entire video game industry, including both physical and digital sales, to be worth an astounding US$65 billion [Baker, 2011].

However, the exact sales figures for the industry are not so important. Regardless of which reports are chosen to be accurate, it is clear that the video games industry is big business. Far more important are the questions of why sales are growing and what impact this has on game developers. Video games are becoming more socially acceptable as an entertainment medium, are more accessible to novice players, and are incorporating more social interaction between players. What this means is that video game players now come from many backgrounds and statistics from the Entertainment Software Association [ESA, 2012] show that players can no longer be stereotyped into a single demographic. The study by Drachen et al. [2009] of player types in the game Tomb Raider: Underworld (Eidos Interactive, 2008) shows that, even within a single game, players' preferences can be varied and there are many ways of achieving the same goal.

The ESA also states that the average age of a game player is 30 years old, indicating an increase in the buying power of the average consumer. With this buying power comes

the desire for a larger variety in experiences and an expectation of longevity in a game, representing a better investment for the buyer. What this all means for game developers is that, given both the diversity and buying power of the market, it is becoming more difficult to design a game that will accommodate every player's past experiences and future expectations.

Incorporating different player preferences and offering increased replay value is not a trivial task though. The costs of developing a high profile game are skyrocketing. For example, Daly [2013] states that the game *Star Wars: The Old Republic* (BioWare, 2011) had an estimated development cost of US$200 million and only sold 2.56 million copies while the well selling game *Grand Theft Auto IV* (Rockstar Games, 2008) cost US$100 million. Game development has become a complex multidisciplinary task and with higher costs have come higher risks. Adding the additional complexity of accommodating many players' skill ranges only further increases costs.

A popular approach to appealing to a diverse audience is to simplify the game mechanics. This has given rise to term 'casual games' [Boyes, 2008] over the last few years. A casual game must be immediately enjoyable when picked-up by a player but must also be compelling enough to warrant continued play. Simplified game mechanics often lead to lower development costs, which is reflected by the quantity of small developers providing games on mobile touch devices, on which casual games have become most accessible. However, the downfall of this approach is that the simplified mechanics can result in a tedious and repetitive experience for seasoned game players, a lessening of the immersion of the game, or even require the removal of artistic elements within the game such as narrative, believable environments, and spoken audio. Also, if the game fails to be adopted in the currently saturated mobile app environment, it can cause a significant financial loss for small development companies.

An alternative approach, and one that has been adopted by developers for many decades in a variety of game genres, is to include a difficulty adjustment mechanism to allow players to manually set the level of challenge they will face [Pagulayan et al., 2003]. Challenge is commonly stated as a key element that influences a player's enjoyment of a game [Malone, 1981], with too much challenge for a specific player's skill leading to frustration and not enough challenge leading to boredom. However, these manual difficulty adjustment systems are usually reported to the player as a short list of nominal settings and do not perform well for those players whose abilities do not match any of the provided settings. An alternative solution is to automatically set the difficulty of the game for the player. For multiplayer games, this is done through matchmaking systems such as Microsoft's *Trueskill* system [Herbrich et al., 2007], which pairs players of similar experience with each other based upon win-loss

ratios and player performance during each game. In single-player games, this translates into dynamic difficulty adjustment systems [Hunicke, 2005] that typically automatically set one of the previously mentioned nominal difficulty options or make finer adjustments to the artificial intelligence (AI) of opponent behavior.

While these solutions can tailor the challenge of a game to suit a player, albeit not very accurately at times, they do not allow for changes in experience for a player. For example, once the player has experienced a map once, it is likely that consecutive attempts at the same map will give a similar experience. Similarly, mechanisms such as avatar customization will typically allow a player to decide how they will achieve goals within the game [Bostan and Ogut, 2009] but they do not allow for variances in the type of challenges that are presented to the player. In summary, there is a lack of customization of games to suit both the preferences and skill of an individual player.

Thus, we turn to *procedural content generation* (PCG) [Greuter et al., 2004; Togelius et al., 2011a] as a solution. The term *content* here refers to any game asset that is usually designed by a game developer and can include 3D models, 2D textures, audio assets, map design, rule systems, virtual object properties and functions, and narrative elements. Procedurally generating these assets involves using a programmatic approach to create them instead of requiring each asset to be handcrafted by a designer. These approaches typically utilize some form of randomness to help produce a large variety of content. Note that the artificial intelligence (AI) of virtual companions and enemies in a game is usually not considered to be content and the process of procedurally adjusting these non-player characters' (NPC) behaviors is usually labeled as *adaptive AI* [Ponsen et al., 2005].

As an example of the potential of PCG, the AAA game *Left 4 Dead* (Valve Corporation, 2008) changes the layout of enemies and pick-ups in a game map to balance the skills of two teams in a multiplayer game and to make each round of the game unpredictable [Booth, 2009]. While the approach is highly constrained with pre-designed possible locations of enemies and pick-ups and static environments, it represents a big step towards the game industry using PCG to balance gameplay. However, how to apply PCG to a game is not always clear, it can involve a long and costly process of researching appropriate technologies and it requires changes to the game design to accommodate random content. Despite all this effort, in the end the result is most often inferior in quality to handcrafted game content. Therefore, in order for PCG to be a viable approach to game development, investigation is needed to refine PCG techniques, increase their ease of integration, and reduce the risks involved with their use.

1.1 Motivation

PCG has been used for many years to provide players with a larger variety of game content than is usually feasible to manually design by game developers. For example, the roguelike game *Dwarf Fortress* (Bay 12 Games, 2006) randomly generates large facets of the game to provide the player with a unique experience every time the game is restarted. Alternatively, PCG can be used to reduce the memory cost of a game, such as in *Elite* (Acornsoft, 1984), which condensed 2048 worlds into less than 14 kilobytes of memory. It has also been used to aid in the production of games by taking care of mundane tasks. An example of this is the *SpeedTree* (Interactive Data Visualization Inc., 2013) application that inserts virtual vegetation into games and other 3D animations.

However, these past applications of PCG have typically utilized a one-off generation approach. That is, there is no validation process other than perhaps that which is done by a human game designer. Applications such as Dwarf Fortress and SpeedTree use a constrained random generation process that restricts how content is created but otherwise relies on chance to create a high quality piece of content. Meanwhile, the PCG in Elite utilizes a seed to reproduce a deterministic galaxy, which was roughly validated by the game developers before release. On top of this, the content created by these PCG approaches is rarely generated with a specific player's desires taken into consideration and it is even possible for a piece of content to not be appropriate for any player. For example, in the game Dwarf Fortress, it's typical for the random generation mechanism to produce an unforgiving game that is practically impossible for all but the most experienced players to win.

An alternative to the one-off generation process is to use a generate-and-test method. Here, content is tested after it is generated to make sure it meets a minimum criteria and if it fails, then it is rejected and the generation process is conducted again. A subcategory of the generate-and-test approaches, and the one that is focused on in this thesis, is known as *search-based procedural content generation* (SBPCG). SBPCG is defined by Togelius et al. [2010c] as being a special case of generate-and-test in which 'the test function does not simply accept or reject the candidate content, but grades it using one or a vector of real numbers. Additionally, there is a condition that future content is based upon previously generated content that has scored well, leading to a process of iterative improvement. The goal is no longer to generate content that is 'good enough' but instead to create content of a high quality.

Most PCG methods will have some constraints, even if they are very loose, and therefore all possible pieces of content that can result from this process can be thought of as a (typically high dimensional) solution landscape. Thus, the term 'search-based' relates to the process of searching this solution landscape for an optimal piece of content. While there are many types of search and optimization methods that can be used in SBPCG, the description above closely resembles that of *evolutionary algorithms* (EA) and so it is typical for EA to be used to achieve the desired effects. In this case, each generated piece of content is a candidate within a population, each content candidate is evaluated with a fitness function and given a fitness score, and the process of generating new content from previous candidates can be done via genetic mutation and recombination. In order for mutation and recombination to be used, a genetic representation is formulated as an abstraction of each content candidate. Thus, the two main challenges in this field are formulating an appropriate genetic representation and a fitness evaluation strategy or metric that produces the desired qualities in the content.

When the fitness function of an SBPCG takes into account some aspect of a player's behavior or preferences it is commonly referred to as *experience-driven procedural content generation* (EDPCG) [Yannakakis and Togelius, 2011]. Thus, the resulting content is expected to be of a high quality with regards to a specific player. This field has only begun to develop over the last few years but is gaining wide spread interest as a means of adapting games to suit the preferences of players.

However, a taxonomy constructed by Smith et al. [2011a] highlights that many EDPCG techniques utilize either a universal or class player model. That is, either a single, generalized model of behavior or preferences is deduced for all players of a single game or genre (a universal model) or a player is classified as one of a handful of nominal player types that each has its own generalized player model (a class model). These approaches fail to recognize that each player is a unique individual and that preferences and skill of a player are personal traits that may separate them from the norm or prevent them from fitting into a preordained player type. While providing varied experiences through PCG, these approaches have the same shortcomings that are present in manual difficulty selection mechanisms.

In this thesis, a focus on procedurally generating game maps is enforced. A *map* (sometimes also referred to as a *level*) is defined here as the virtual environment through which a player navigates during gameplay. The maps of a game have an important role in not only establishing an immersive background to the narrative of the game but can also greatly influence the challenge that a player experiences and, in combination with the game mechanics, aid in the formation of a player's strategies for achieving objectives. There are currently only

a handful of SBPCG solutions of generating maps for specific game genres and even fewer that utilize player preferences in an EDPCG approach. Of those that do utilize EDPCG, many are aimed at creating maps for 2D platform games such as *Super Mario Bros.* (Nintendo, 1985). However, 2D platform games, while experiencing a resurgence in popularity in the last few years due to casual gaming, only represent a small portion of the market [Reeves, 2012], with most other game genres utilizing 3D graphics and therefore 3D maps.

Finally, while the use of EA has become standard practice for SBPCG solutions there is not much other commonality. Many existing solutions for SBPCG and EDPCG are crafted to work with a specific game or a narrow genre and do not mention if (or how) the PCG method, content representation, content candidates, or player modeling approach could be generalized to other games. This makes it difficult for those new to the SBPCG field to begin formulating appropriate solutions. Similarly, no SBPCG or EDPCG technique has yet been demonstrated to be generalizable enough or produce content of a high enough quality to be adopted by industry developers. This is because the currently unrefined techniques are perceived to be at risk of producing content of poor quality, content that is not appropriate to specific player, or content that is outright game-breaking and paying customers are unlikely to tolerate these types of errors. While the work in this thesis does not fully achieve this standard either, it contributes towards this goal.

1.2 Aim, Research Questions, and Scope

With the above shortcomings in the existing literature in mind, the aim of this thesis is to investigate potential solutions for procedurally generating complete 3D game maps that are tailored to an individual player's preferences and skill through the use of evolutionary algorithms.

To do this, the problem is divided into three subcomponents: geometry, content layout, and player preference modeling (PPM). These components not only establish a line of investigation and make up the structure of this thesis but also constitute fundamental elements of any game map. Every game will place a different emphasis on the geometry of the map and the content within it. In some games, one aspect will influence a player's experience more than the other, while in other games they will both play an important role. While PCG of geometry and content layout allows for instances of maps to be created, the PPM component enables the maps to be tailored to an individual player's preferences and skill. Below is a

further description of each of the three subcomponents and the research questions associated with them. Following that is a discussion of what is not included in the scope of this project.

1.2.1 Geometry

The *geometry* of a map is defined here as the static environment that the player navigates through during gameplay. This may include the base terrain that a map is built upon, low level architectural elements such as the structure of floors and walls of a room, high level architectural elements such as the layout of rooms and corridors, and the arrangement of static clutter such as furniture. Some game genres may even have unique geometry elements such as the race track of a racing game or the platform arrangement in a 2D platform game.

The geometry offers no interaction to the player other than allowing the player to stand on it, bump into it, navigate around it, or hide behind it. In some game genres the geometry may simply act as a stage for the narrative, providing an immersive environment, but in other games it can have a drastic effect on the challenge the player experiences and the strategies they form. In simulation based racing games, the track acts as a primary challenge inducer, along with the competency of the other racers. If the terrain of a real-time strategy (RTS) game is not properly balanced, one player may have an easily defendable position while the other is left out in the open. Clutter such as furniture, vegetation, or rock outcroppings can be used for cover in a first-person shooter (FPS) game while increasing the distance between ledges in a platform game can substantially increase the difficulty of a map. Almost all modern games will have at least a basic geometry that dictates the boundaries of the virtual space. However in some games, such as some action games, the geometry may not affect the player experience other than providing aesthetically pleasing structure to the gameplay and narrative.

The research questions for this component are as follows:

1. How can the base terrain of a game map be represented in an SBPCG solution to provide a high level of control over the features in the generated maps?

2. How does this genetic representation compare to existing SBPCG techniques for generating terrain?

3. Can the ideas behind the genetic representation be abstracted and used to generate interior spaces in a SBPCG manner as well?

1.2.2 Content Layout

The term *content* here describes all game objects that the player can interact with in ways other than navigating around. This may include enemy NPCs that can be attacked, friendly NPCs that can be conversed with, item pick-ups, narrative and audio queues, or mission objectives. The similarity to the use of the word 'content' in the phrase PCG is purposeful; here it simply refers to a subset of game content that does not include the geometry of a map. Also note that we are investigating the layout of this content throughout the map, not the procedural generation of the aesthetics or function of individual pieces of content other than the geometry of the map.

Like geometry, content layout can also play an important role in the experience that a player has. In an action game, the balance of enemies and utility pick-up items from one area of the map to the next can control the pace and challenge of the game. The spread of objectives and powerful weapons and armor motivates exploration within an open world role-playing game (RPG) while the location of weapons and ammo pick-ups in a multiplayer FPS can dictate which areas of the map are likely to have higher player activity. However, in other game genres, such as driving or flight simulators, there may be very little content and instead the player's experience is affected only by the geometry.

The research questions for this component are as follows:

1. Given an example game, how does content affect a player's experience as compared to the geometry?

2. Is there a genetic representation for an evolutionary procedural map generation algorithm that can encompass the needs of content layout in the example game?

3. How can content layout be evaluated to determine whether it matches the player's preferences?

1.2.3 Player Preference Modeling

Player modeling in games is the process of collecting and utilizing data about a player (or a group of players), usually in an effort to improve the game or understand some physiological or psychological characteristics of a player. In this thesis, the focus is on *player preference modeling*, which can be seen as subcategory of player modeling with the goal of discovering what a player enjoys and using the knowledge to improve the game. For the sake of being concise, the term 'player preference modeling' is reduced to the more compressed parent term

'player modeling' throughout this thesis as many characteristic of preference modeling are inherited from the broader field of player modeling.

Player modeling is not restricted to being used in conjunction with PCG. On the contrary, the most common form of player modeling is conducted by leading game developers to monitor gameplay during testing sessions. This may be done in order to understand how the player is interacting with the game, what is contributing to the experience, what is inhibiting normal gameplay, and to make appropriate changes. In this case, the player modeling approach is being used as a reporting tool and is designed to give a descriptive analysis of player behavior.

In this thesis, however, we focus on using player models in a generative manner. That is, the understanding of some psychological or physiological trait of the player is combined with PCG techniques (forming an EDPCG application) to create or improve game content without the need of interpretation by a human designer. This approach has gathered popularity in recent years and various forms of player data have been investigated. These data sources have included physiological inputs, self-reporting, and in-game activity.

The research questions for this component are as follows:

1. Is there a player preference modeling technique that can be generalized to multiple game genres for the purpose of procedural map generation?

2. For this project, which source of player data is the most suitable and why?

3. When combined with an EDPCG application, does the player modeling technique provide an improved experience for players over random PCG?

1.2.4 Out of Scope

As mentioned earlier, the procedural generation of the structure or purpose of individual pieces of content other than the geometry of a map is not investigated. This is because these content pieces, while valuable components of the gameplay and present within the map, are not considered to be modifiable components of the map itself. If the scope of this project were expanded to include the creation of these content types, then it would quickly become a much larger problem.

Similarly, it may be argued that another important component of many game maps is how it fits into the narrative or what the objectives of the map are. The roles of these aspects are highlighted by the dichotomy presented by Dormans and Bakkes [2011], who separate a

game map into 'spaces' (similar to our definition of geometry) and 'missions'. We find that many game genres do not have the segregated mission structures analyzed by Dormans and Bakkes. The layout of objectives can even be considered as part of content layout process, though it will need more constraints to ensure a logical flow of objectives. In the games used in the experiments in this thesis, however, there is no narrative other than a background setting and the objective of every map is the same. Therefore, the need to address these issues is left for future work.

The closely related field of adaptive AI for NPCs is also not considered here. Currently, most existing literature uses either PCG or adaptive AI to enhance a player's experience but rarely use both. This is because both of the respective fields need further refinement before they can be combined into a single solution.

Finally, we acknowledge that there are many forms of search and optimization algorithms that can be used in SBPCG. However, the current norm within the field is to use EA as the base framework. Thus, to keep our work coherent with the existing literature, EA is the only optimization technique considered within this thesis.

1.3 Contributions

The central contribution of this thesis fits within the body of knowledge on procedural map generation; especially in relation to adapting game maps to meet player preferences. From this, more focused contributions arise:

1. The introduction and discussion of the concept of a game map in terms of its two constituent elements; *geometry* and *content*. An approach to applying PCG to both of these elements is provided. For personalized procedural map generation, an additional element of *player preference modeling* is added. How to capture and utilize a player's preferences in relation to both geometry and content within certain game genres is examined.

2. Two geometry generation techniques are provided; one for exterior geometries and the other for interior geometries. Both of these approaches use the idea of combining smaller pre-designed building blocks to form a larger map. In exterior maps, this is realized through *patch-based terrain* generation, while interior spaces are discussed in terms of *room templates*.

3. A content layout solution is provided that calculates content quantities within a separately generated geometry by using *Compositional Pattern-Producing Networks* (CPPN) [Stanley, 2007], which are evolved through *NeuroEvolution of Augmenting Topologies* (NEAT) [Stanley and Miikkulainen, 2002]. This solution creates patterns of content with regularities throughout a map and therefore is argued to generate patterns of challenge for the player.

4. A PPM solution is created through the use of a *recommender system* (RS) framework. We describe how the RS and EDPCG fields overlap and how a player model can be efficiently structured as an RS. A functional example is given of a traditional content-based RS approach being used to generate the maps of an action-shooter game to suit the preferences of individual players. This example shows that the use of an RS player model provides better experiences for players when compared to random PCG.

5. A performance metric is proposed to more clearly analyze the learning capabilities of an RS in the noisy player modeling environment. The metric has been named the *learning trend*, with mean and majority variants providing a complimentary view of the performance. This metric acts as an alternative to traditional confusion matrix based classifier accuracy measurements that can give distorted views of the performance of this type of RS.

The research presented within this thesis is primarily explorative and we make no claim to having solved the entire problem of personalized procedural map generation. The respective field is still in its infancy and best practices have yet to be established. Thus, much of the existing work in this field, as well as our own, examines potential solutions and frameworks that may promote future work and encourage personalized maps (and other content) to be adopted within the games industry. Towards this goal, the contributions of separating a map into geometry and content layout, constructing map geometries from smaller sample geometries, the use of a CPPN to create patterns of content throughout a map, and the use of an RS as a player model are all generalizable to many game genres. These ideas also aid in consolidating knowledge within the growing EDPCG field to enable developers and researchers new to the field to quickly create their own implementations of personalized procedural map generation.

1.4 Thesis Structure

There are nine further chapters within this thesis, with most of the core chapters addressing one of the three components of personalized procedural map generation as described in the research aim earlier. Chapter 2 first provides background knowledge in the PCG and player modeling fields as well as fundamental knowledge of evolutionary algorithms and recommender systems. Chapters 3 and 4 each present one solution to the problem of procedurally generating geometry. Chapter 3 deals with exterior terrain generation and the resulting solution can be seen as a development aid for novice game developers. Meanwhile, Chapter 4 shifts to a look at interior spaces and, while sharing similar mechanics to that of the terrain generation, is designed to be used by a player during gameplay. Additionally, the game *PCG: Angry Bots* is detailed, which acts as the test bed for the remainder of the thesis. With an interior geometry established, Chapter 5 demonstrates how to represent and optimize the layout of content within a map through neuroevolution. Chapter 6 discusses how an RS can be utilized as a player model and gives details of the implementation of the content-based RS used in *PCG: Angry Bots*. There are then two experiment chapters: Chapter 7 fist evaluates the tool described in Chapter 3 to evolve terrains, showing how parameter changes can affect the output of the tool. Then, Chapter 8 describes the *PCG: Angry Bots* experiment that was conducted to evaluate the systems described in Chapters 4, 5, and 6. The metrics that were used to evaluate the resulting player data are also described in this chapter. The results of the *PCG: Angry Bots* are then presented in Chapter 9. Finally, Chapter 10 concludes this thesis with a review of the work that has been done, recapping answers to the research questions, providing a discussion of limitations of the current work, and a look at future work.

Chapter 2

Background

Forms of artificial intelligence (AI) have been present in video games since their advent. They have ranged in complexity, with rudimentary examples including the basic scripted movements of non-player characters (NPCs) in many early console games, such as *Super Mario Bros.* (Nintendo, 1985), and the more advanced scripted enemies in modern games, such as the *Call of Duty* franchise (Activision, 2003-2013), that give the illusion of intelligent responses to a player's activity by using the $A^\star$ algorithm and finite state machines. Meanwhile, some games have adopted more advanced AI paradigms such as the use of agent-based planning to allow enemies in *F.E.A.R* (Monolith Productions, 2005) to make informed decisions and utilize complex strategies [Orkin, 2006] or the use of neural network machine learning techniques to teach virtual pets through reward and discipline in *Creatures* (Cyberlife, 1996).

Games have also risen as a primary means of evaluating AI research. AI techniques that are believed to be able to strategize as a human would or better are often tested on both chess-like board games and video games that require complex, multi-leveled planning such as *StarCraft* (Blizzard Entertainment, 1998) [Weber et al., 2010] and other real-time strategy (RTS) games [Buro and Furtak, 2004]. Alternatively, games are also being used in Turing tests in which judges must determine whether other virtual characters are being controlled by a human or an AI [Hingston, 2010].

However, despite this long history of opponent and NPC control algorithms, in this thesis we do not examine this aspect of AI in games. Instead, we focus on using AI techniques as means of improving the quality of a game's setting, either during the development of the game or during gameplay itself. To do this, we look towards procedural content generation (PCG) as a means of creating or altering elements of the virtual world around the player. A player's

experience can be improved by manipulating the appearance of a game, the structure of the virtual environment, the sounds in a game, the narrative, or even the rules of the game. Thus, just as advanced AI techniques have been used to improve the quality of NPC behavior, we investigate similar techniques in order to improve the quality of the presentation of the game. More specifically, in this thesis we use a combination of PCG, search-based optimization, and machine learning to create game maps that will provide a more enjoyable experience to the player.

In this chapter we first give an introduction and a brief history of PCG in Section 2.1, especially with regard to game maps. Included in this is a focus on procedural terrain generation, as it is a primary element of many games' maps and is examined in two of the chapters of this thesis. After that, Section 2.2 describes the emerging field of search-based PCG, which combines optimization techniques, such as evolutionary algorithms (EA), with a generate-and-test approach to content generation to improve the quality of output over traditional once-off PCG algorithms. Additionally, we provide a brief introduction to EAs and how the terminology of the field can be applied to PCG. Then, a motivation for and definition of player modeling is presented in Section 2.3, along with a discussion of the experience-driven PCG sub-field that combines PCG and player modeling to create content that is tailored to a player's (or a group of players') preferences. Finally, as recommender systems (RS) form the basis of the player model in this thesis, an introduction to the basic concepts and terminology of RS is provided in Section 2.4.

2.1 Procedural Content Generation

The traditional pipeline for producing a video game includes teams of artists hand crafting game content using specialized tools. This content is then imported into the game and used by the engine (the code of the game) to create immersive virtual worlds. *Content* can include 3D models, 2D textures, audio files, narrative structures, animations, and the properties of virtual objects, from physical properties, such as mass, to behavioral properties, such as the rate of fire of a weapon [Togelius et al., 2010c].

Procedural content generation (PCG) is a programmatic approach to creating this content instead. It can be seen as a means of automating content creation by using a small set of algorithms to create a large range of variations of a chosen content piece. Therefore, there is typically less effort required by the design teams of the project and more emphasis on the programming teams to implement algorithms that produce suitable content.

The motivations for using PCG in a game can be varied. It can be used as a tool to speed up the development process, it can provide suggestions to artists that they may not have considered, can reduce the memory and storage size requirements of the game, and it can be used while the game is being played to create more experiences for the player than would be possible through manually crafting the content. It's also worth noting that the AI controllers of virtual characters are usually not considered a form of content and thus the manipulation of the AI during gameplay is usually considered as its own research field, known as *adaptive AI* [Machado et al., 2011]. In this section we discuss common applications of PCG and discuss some of the content types that are explored in this paper; primarily the geometry of a game map.

2.1.1 Applications

Here, we briefly describe some of the common applications of PCG and give real-world industry examples of how PCG has been used. This is done to show that, while much of the cited work in this thesis comes from academic research, PCG can be used successfully in commercial endeavors. The PCG applications described here are broadly categorized as memory reducing techniques, development aids, and techniques used to create a large range of player experiences.

Memory Reduction

The first use for PCG is to reduce the size of the game, either in memory or on disk. In games, 3D models and 2D textures can use up a lot of these computational resources, especially as the fidelity of them increases. Thus, using PCG to generate game assets programmatically when they are needed causes the game content to utilize more of the processing power of a computer rather than storage resources.

With the ever expanding size and reducing price of storage media, this application of PCG is no longer necessary and is therefore studied less than the other two applications. However, it was a main driving factor of PCG research and development in early commercial video games. *Elite* (Acornsoft, 1984) is a role playing game (RPG) where the player can explore $2,048$ planets while traveling through space. Due to the limited storage capabilities of the computers of its time, the details of each planet are generated by feeding a single seed number to a deterministic PCG algorithm that outputs the location of the planet, the resources available at the planet, and even the planet's name.

More recently, the first person shooter (FPS) *.kkrieger (*Farbrausch, 2004) was created as a technical demonstration of the memory reducing capabilities of PCG. The game occupies only 98KB of disk space as all the 2D textures and 3D meshes are procedurally generated from basic building blocks when the game is first being loaded. Despite this, the graphical detail of the game is of a high quality and the developers estimate that the game would occupy between 200 and 300MB if the content were stored on disk.

Development Aids

PCG can also be used by game studios during the development stage to reduce the time and resources required to create game content. Tools such as *SpeedTree* (Interactive Data Visualization Inc., 2013), which populates a map with vegetation, or *Terragen* (Planetside Software, 2013), which creates photorealistic mountainous landscape, are popular in the game development community, even though most game players do not notice that they have been used. These tools reduce the complexity of their respective tasks, allowing developers to focus their attention on improving other aspects of the game and thus improving the quality of the entire game.

The use of these types of techniques is especially valuable for independent developers with limited resources. The highly stylized landscapes of *Darwinia* (Introversion Software, 2005) were procedurally generated using simple fractal methods (described in Section 2.1.2) on a low resolution heightmap. With the bulk of the map design being conducted through PCG, the small development team had more time to fine tune the landscapes and add other content to its surface.

These types of development aids can also be used by players if the game is designed with this process in mind. The God-game *Spore* (Maxis, 2008) allows players to create their own creature by adding limbs and other features to a body. A procedural animation process then ensures that every creature that a player creates behaves naturally and an evolution process develops the creature from a basic organism to the player designed creature. These two procedural processes give creative freedom to the player without requiring them to have animation knowledge and go through tedious manual validation processes.

Diverse Experiences

Possibly the most popular application of PCG in both commercial games and academic research is that of creating diverse experiences for the player. Depending on the design of

the generative algorithm and the content being generated, PCG can allow for a vast number of unique pieces of content, allow for the player to customize their experience, or, as is investigated in this thesis, personalize the game content to suit the preferences of the player.

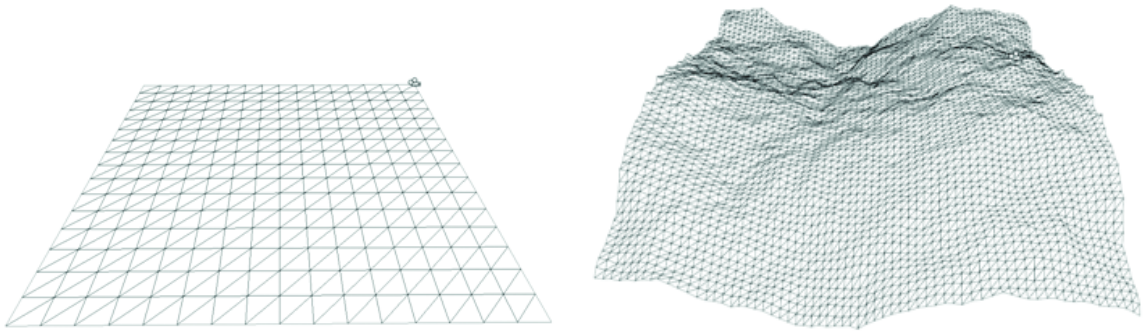
This goal sometimes overlaps with the previous two applications of PCG. For example, the reason for using a memory saving PCG algorithm to generate planets in *Elite* (Acornsoft, 1984) is because the developers wanted to give the player more experiences than was typically assumed to be supported by the computer hardware of the time. Meanwhile, the procedural generation of animation in *Spore* (Maxis, 2008) allows every player to create a unique alien race that meets their preferences for aesthetics, producing a more personalized gameplay experience.

Other examples from high quality commercial games include the use of PCG technique to generate over 18 million unique weapons in *Borderlands 2* (Gearbox Software, 2012), which is achieved by randomly combining weapon behaviors, features, and statistics. This means that throughout the hours long campaign in the game, the player is guaranteed to never see the same weapon twice and the player is motivated to explore the world and defeat enemies in the hope of finding better weapons. The turn-based strategy game *Worms* (Team17, 1995-2013) randomly generates arena style 2D maps to provide competitive experiences in unpredictable environments, meaning that no player can gain the upper-hand by learning the nuances of a map. The PCG algorithm is also loosely controlled by parameters, allowing players to dictate the style of the map without knowing the exact shape of it.

The most well-known genre of games that utilize this type of PCG is roguelikes. This genre was named after the game *Rogue* (Michael Toy, 1980), which first procedurally generated maps in ASCII graphics, a trend that continues in many roguelike games. At the height of the roguelike genre, *Dwarf Fortress* (Bay 12 Games, 2006) is a challenging game in which entire worlds, economies, cultural behaviors, temporal events, and in-game items are procedurally generated with little restriction. This leads to an experience that is unpredictable and unforgiving but from which the player can be a part of their own narrative.

2.1.2 Procedural Terrain Generation

The first investigation we conduct in this thesis is looking into how to procedurally generate the geometry of a wide range of maps. We deduce that, while every game has its own setting, terrain is a common element in many games on top of which the remainder of the



(a) (b)

Figure 2.1: (a) A flat heightmap and (b) a heightmap after fractal terrain generation.

map elements are placed. Therefore, we introduced a novel method for procedural terrain generation that aims at tackling some of the problems discussed in this subsection.

The procedural generation of virtual terrain has been successfully applied in many fields and has enjoyed ongoing research by both industry leaders and academics for almost three decades [Smelik et al., 2009]. Most existing research has been devoted to creating vast sprawling landscapes that are typically too large and too detailed to be created manually. These methods have been improved in recent years and powerful commercial tools, such as *Terragen* (Planetside Software, 2013), are capable of producing renderings of detailed terrain that resemble professional photographs of mountainous landscape. However, Terragen and other applications like it use fractal generation techniques and, as we will discuss in this section, these approaches lack applicability in games.

Terrain Representations

The most common data structure used for virtual terrain is a *heightmap* [Smelik et al., 2009]. A heightmap is usually represented as a three-dimensional array detailing the x,y and z coordinates of vertices. The vertices are evenly spaced across two of the coordinate planes (x, y) and the third coordinate's value (z) specifies the height of each vertex. The vertices are joined to their neighbors to form triangles and from this a solid surface can be rendered. Examples of heightmaps in their wire-frame form are shown in Figure 2.1.

Another popular approach to representing the terrain is to use *voxels* [Kaufman et al., 1993]. Voxels are three-dimensional pixels that can be thought of as cube shaped building blocks. At fine resolutions, voxels can theoretically be used to represent individual grains or

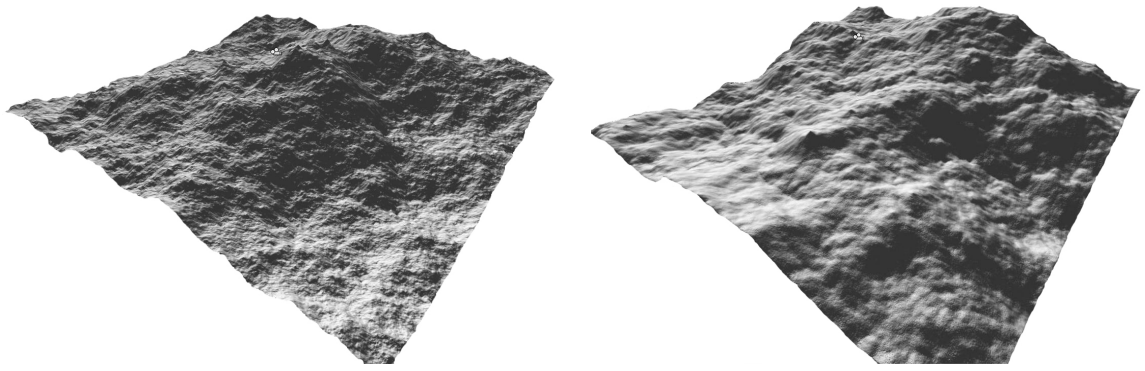
clusters of dirt. Peytavie et al. [2009] even associate materials (such as water, rock, sand, and air) with each voxel. Thus, these approaches lend themselves well to erosion simulation, as these algorithms can actually move small clusters of material around in a closed mass system.

The benefit of voxel structures is that they can create overhangs such as on cliff faces or inside caves; this is not possible for heightmaps as their cannot be two vertices (one for the floor of the cave and one for the roof) occupying the same (x, y) coordinates. Unfortunately, at high resolutions, voxels become much more computationally expensive than heightmaps, which are efficient to manipulate and render. One of the most well-known uses of voxels is in the stylized cube words of *Minecraft* (Mojang, 2011) in which the voxels are large so processing time is kept to a minimum yet allow players to remove individual voxels to change the terrain and explore underground caves. Alternatively, while Peytavie et al. [2009] store the voxel data and manipulate it at a low resolution, it is implicitly converted to a higher resolution before rendering, giving a smooth and highly detailed terrain. However, to generate higher resolution terrains with less computational effort and to remain coherent with much of the existing research in our related fields, we use heightmaps throughout the remainder of this thesis.

Terrain Generation Methods

Procedural terrain generation is the process of programmatically creating terrain for virtual worlds. In the case of heightmaps, this involves a system assigning the height value of each vertex. Fractal subdivision was one of the earliest procedural terrain generation techniques [Fournier et al., 1982] and it remains one of the most popular [Smelik et al., 2009] due to its ability to efficiently replicate natural terrain patterns. These algorithms typically recursively increase the resolution of the heightmap and stochastically adjust the height of each new vertex that is created. An example of a rendered fractal terrain is shown in Figure 2.2a.

Another popular technique is to use an erosion simulation algorithm [Musgrave et al., 1989] to refine an existing terrain, such as one generated by fractal subdivision. Erosion simulations iteratively move height values from high areas of a terrain to lower ones, much the same way that dirt would move in the physical world when being eroded by wind and water over time. While erosion simulation is usually computationally expensive, it can also add an increased sense of realism to the terrain. An example of an eroded terrain is shown in Figure 2.2b. However, the drawback of both fractal subdivision and erosion simulation is



(a) (b)

Figure 2.2: (a) A rendering of a heightmap terrain that has been generated through fractal subdivision. (b) The same terrain with erosion simulation conducted on it.

that they are highly stochastic and thus there is very limited control over the type of terrain that is generated.

Both of these popular techniques, as well as others such as random noise generators, provide little control over the types of terrain features (mountains, valleys, cliffs, etc.) and their location on the terrain. This is primarily due to their non-deterministic characteristics. In recent years there has been a push towards techniques with increased control. This has predominantly been achieved by increasing the number of parameters in the generative algorithm [Chiang et al., 2005; Kamal and Uddin, 2007; Doran and Parberry, 2010], by allowing the user to provide a basic sketch of feature layout [Zhou et al., 2007; Hnaidi et al., 2010], or by allowing users to quickly paint on height values [de Carpentier and Bidarra, 2009]. More recent techniques have used evolutionary algorithms (EA) to drive the terrain generation process, a concept that is explored more in Section 2.2.3 and that we study further in Chapter 3.

Terrain in Games

While virtual terrain can be used in films, images, and simulations, one of the most prominent applications for the procedural terrain generation is for use in video games [Doull, 2008]. To cater to players who enjoy a wide variety of experiences and exploring new environments, game developers can use procedural terrain generation techniques to provide players with a large sample of terrains to play on, as can be seen in games such as *Worms 2* (Team17, 1997), *Civilization V* (Firaxis Games, 2010), *and Minecraft* (Mojang, 2011). Alternatively,

a procedural terrain generator can also be used as a creative aid for designers or to simply streamline the map creation process by providing a base terrain for developers to work with, as was the case in *Darwinia* (Inversion Software, 2005).

The genre of the game that a piece of virtual terrain is to be used in will dictate the required terrain features. Olsen [2004] outlines and describes some requirements in order for a procedural terrain generation technique to be useful in the RTS genre. Primarily, Olsen states that the terrain must contain enough flat areas for game characters to move on realistically, as well as having a high connectivity such that players can traverse as much of the map as possible. However, the terrain also needs hills, cliffs, valleys and other features to promote strategic gameplay and to make maps visually appealing.

Though those criteria are true for many game genres, each genre will still have their own requirements for scale and feature arrangement. For example, an FPS game, such as *Call of Duty: Modern Warfare 3* (Infinity Ward, 2011), will typically have small scale maps, containing high densities of terrain features and virtual objects to impede line of sight. Meanwhile, an RPG, such as *The Elder Scrolls V: Skyrim* (Bathesda Game Studios, 2011), typically has large environments with dispersed townships, thus encouraging exploration of an expansive world. Furthermore, some game genres, such as flight simulators, don't require traversable terrain but rather require the terrain to appear realistic from an aerial perspective. While fractal and erosion techniques are suitable for this latter genre of games, the diverse and specific requirements for terrain in games makes stochastic procedural terrain generators impractical for a wide range of genres, thus motivating a need for further control over the generation process.

2.1.3 Cityscapes, Interior Spaces, and Object Placement

Once a terrain has been formed, it is usually required for other objects to be placed on top of it to make it a more immersive virtual world. As an example of an industry tool that does this through procedural generation, *SpeedTree* (Interactive Data Visualization Inc., 2013) gives developers powerful 3D modeling tools to manually create detailed vegetation models and then procedurally populates game maps with samples of those trees depending on developer specified zones on the terrain. The result is detailed and diverse forests that add a large amount of realism to a game with reduced effort from the developer when compared to placing individual trees themselves.

On the research side, there is a growing interest in procedurally generating cityscapes, complete with procedurally generated interior and exterior spaces and the placement of static objects such as clutter [Kelly and McCabe, 2006]. Smelik et al. [2008] propose a layered representation that adds details such as water, roads, vegetation, and urban structures to a base heightmap. Greuter et al. [2003] procedurally generate exterior building shapes and then procedurally generate a city vista in front of the players view. This process is deterministic so that as the player moves back and forth and turns around, the surrounding city will always appear to remain the same, even though it has been procedurally generated on request. Similarly, Müller et al. [2006] use grammar rules to create both the architecture of various styles of individual buildings and entire cities, saved as highly detailed 3D mesh models.

While not creating large cityscapes, Tutenel et al. [2011] create highly detailed and logical buildings using a combination of multiple algorithms. What separates their approach from others is that they not only generate the exterior of the building but also the interior floor plan, high quality 2D textures, and furniture locations. This produces a complete virtual environment for users to explore.

The placement of virtual clutter such as furniture is a little more complex than the placement of vegetation. This is because the furniture must be placed in logical positions, relative to the architecture and other furniture. For example, chairs should be placed next to a dining table, not facing a wall. Both Broughton and Howard [2006] and Taylor and Parberry [2010] propose methods for procedurally generating the layout of static objects to adhere to interior decoration principles.

There are further procedural generation techniques for creating virtual terrain [Smelik et al., 2009] and populating it with static objects [Kelly and McCabe, 2006]. While many of these techniques are capable of generating realistic virtual environments, they are often difficult to control and thus are either unable to produce a variety of results or cannot be used to optimize game maps to meet certain constraints on the fly; for example, optimizing a game map based upon an updating model of player preferences. Thus, in this thesis we use both search-based PCG (EDPCG) and experience-driven PCG (EDPCG) to gain more control over the procedural map generation process and personalize the maps for individual players. These two fields are discussed in the next two sections.

2.2 Search-based Procedural Content Generation

A trend in recent PCG research has been towards the use of search-based PCG (SBPCG) techniques, first termed by Togelius et al. [2010c]. The traditional PCG solutions described in the previous section use once-off generations, requiring every piece of content to be playable and of a certain quality. There is no attempt to find the best possible piece of content for the player but rather just utilizing the first valid piece of content to be generated.

Meanwhile, SBPCG solutions use a generate-and-test approach, evaluating candidate content against certain criteria after they have been generated in order to find the best solution. SBPCG has gained substantial popularity in recent years and has been applied to a variety of game content. It has been used to evolve the projectile path of weapons in a top-down space shooter game [Hastings et al., 2009], controls of in game camera orientation to keep specific objects in view [Burelli and Yannakakis, 2010], the structure of buildings to be placed on game maps [Martin et al., 2010], and even the rules of the game themselves [Togelius and Schmidhuber, 2008]. It has also been applied extensively to the generation of different types of maps in various game genres, which we explore further in Section 2.2.2 and to which we contribute through this thesis.

For further information, readers are referred to the extensive taxonomy and survey of SBPCG research, provided by [Togelius et al., 2011b]. From their work, one noticeable finding is that evolutionary algorithms (EA) have been a common choice in SBPCG as they offer a means of iteratively improving on previously found good content. Thus, in this section we give a brief overview of EA and how it applies to SBPCG, especially for the creation of game maps. Then, we discuss related work in the field of SBPCG, both in terms of general game content and of game maps specifically. Finally, a focus on the creation of virtual terrain is again provided as a follow-on to the traditional PCG techniques of terrain generation described in the previous section.

2.2.1 Evolutionary Algorithm Fundamentals

EAs in computing are inspired directly from Charles Darwin's *Theory of Evolution* [Darwin, 1859]. These algorithms are typically used in optimization and design problems where it can be said that one solution or output is better than another on either an objective or subjective scale. If solutions can be compared in this way, then survival of the fittest can take place from one generation to the next. Through breeding and mutation processes, good solutions are

encouraged to pass on their features and thus a refinement process can take place whereby each generation contains better solutions than the one before it.

Below are some simple definitions of the evolutionary computing terms that will be used in this thesis:

- *Candidate*: The counterpart of a candidate in nature would be an individual animal. Each animal encompasses its own genetics and, if it survives, can pass on its genetic traits to its offspring. In computing, a candidate is a potential solution to a problem. In generating game maps, this is a single map that may or may not be used in the game but is considered and evaluated nonetheless.

- *Genetic Representation*: An EA usually acts on the genetic representation (or the *genotype*) of a candidate. The genetic representation of most animals in the natural world is the double helix DNA structure. In an EA, the genetic representation may be a set of possible values for parameters, a modifiable mathematical program, the arrangement of design features, or any other representation that can be computed and modified efficiently.

- *Phenotype*: The phenotype is then the result of these genetics. While the genotype of an animal is its DNA, the phenotype is the animal's observable physical traits. When using an EA during procedural map generation, each map may be represented by only a few parameter values but after a genotype-to-phenotype conversion process a playable game map is rendered and ready for use.

- *Generation*: A generation (or *population*) is a set of candidate solutions. Typically, an EA will proceed in ordered generations, calculating multiple new candidate solutions for each generation, based upon the good solutions of the previous generation. The process of creating a new generation is typically referred to as *breeding*.

- *Parent* : A parent in the evolution process is a candidate that has been chosen as a good solution and can pass on part of its genetic makeup to the next generation, which will hopefully give its offspring the same attributes that marked it as being a good solution while also changing other attributes.

- *Child* or *Offspring* : A child is a new candidate that has been derived from one or more parents via mutation or crossover.

- *Crossover* and *Mutation* : When breeding occurs, an offspring can be generated in one of three ways; either by crossover, mutation, or a combination of both. Crossover (also known as *recombination*) involves combining the genetic structure of two parents to make one child that will exhibit features from both parents. It is hoped that the features that made both parents good solutions are kept while any negative features are removed. Mutation involves a stochastic change in a parent's genetic structure to create a child and is used to ensure diversity in the population and exploration of features.

- *Fitness Evaluation*: The fitness evaluation method is a means of determining how 'good' a candidate is and whether it will be allowed to be a parent. An automated approach to this is to use a *fitness function*. For example, a good map in a FPS game may be one with a lot of cover to hide behind, thus a fitness function would calculate the number of surfaces that can be used as cover in the phenotype. For a fitness function to be applicable, a definition of what is a 'good' or 'better' candidate must be created and a means of objectively calculating or measuring it must be formulated. Alternatively, in more subjective fields, fitness for each generation can be dictated by an outside source. This is referred to in this thesis as *interactive evolutionary computing* (IEC) [Takagi, 2001] and takes the form of a user providing feedback to the evolutionary process about which solutions they want to be the parents of the next generation.

This is only a simple overview of the concepts and key terms of EA and how they apply to the PCG of game maps. It is used as a framework for the solutions in the thesis but we do not make any claims to furthering knowledge in the EA research field. For more information, Eiben and Smith [2010] provide a comprehensive introduction to evolutionary computing.

2.2.2 Game Maps

The maps in a game play an important role in the experience that a player has [Byrne, 2005]. They can offer players a chance to use creative strategies by utilizing different paths and using obstacles to their advantage; they can provide opportunities for exploration; they can determine some aspects of the challenge that the player will face through skill based tasks such as jumping between platforms or the density of enemies in a specific area; or they can be used to increase the visual aesthetics of the game, establishing the theme of the virtual world and acting as a backdrop to the narrative. As with most game content, maps are traditionally produced by teams of experts to create an engaging and balanced experience

[Byrne, 2005]. However, for many decades there has also been an interest in procedurally generating game maps and this continues to be a popular topic of research as it investigates principles of game design as well as the implementation of human creativity into AI.

More recently, the use of SBPCG on game maps has risen in popularity, to the extent that much of the existing SBPCG literature focuses on some component of a game map [Togelius et al., 2010c]. A few examples include that of Cardamone et al. [2011b], who explore multiple genetic representations for maps of an FPS game, evaluating them based upon how long the player is expected to survive. Likewise, Ashlock et al. [2011] experiment on various genetic representations and fitness metrics for virtual mazes. Alternatively, Sorenson and Pasquier [2010] specify an abstracted genetic representation in combination with a feasible-infeasible dual population setup that can be adapted to different games given expert knowledge of the game design and used to generate maps that adhere to developer specified design constraints. Meanwhile, Togelius et al. [2010b] generate the landscape and the placement of objectives of RTS maps, in both 3D and 2D [Togelius et al., 2010a], to provide a fair game to all players. Finally, Frade et al. [2010b] use genetic programs to calculate the height values of each vertex in a heightmap and subsequently use the resulting terrains in a casual turn-based game. The use of SBPCG on game maps is diverse, with many solutions (such as those above) focusing on different game genres, having different genetic representations, and having different heuristics for evaluating the maps that are generated.

However, a common trait among many of these SBPCG solutions for generating maps, other than use of EA, is that they use a single generative process or evolutionary cycle. While not using a generate-and-test approach, Dormans and Bakkes [2011] have recently provided a motivation for describing the map generation process in terms of *missions* and *spaces*. In their work, they advocate using two independent grammars to generate these, even though the spaces are overlaid with the missions. However, this dichotomy is not necessarily useful in games that do not contain the kind of missions described by Dormans and Bakkes. For example, arena based multiplayer games, RTS games, arcade racing games, and some linear single-player games. It's not that these games do not have objectives, it is just that they are not multifaceted and are usually more constant: defeat all opponents, seize and defend control points, win a race, or get from one end of the map to the other.

For these kinds of games, in this thesis we instead suggest separating the map generation process into that of *geometry* and *content*. Geometry refers to the boundaries of the map and the structure of fixed game objects, such as architecture and furniture. The player cannot interact with these elements other than navigate around them but they still affect the

player's experience by providing obstacles to traverse or take cover behind, giving the player opportunities to explore, and establishing the visual aesthetics of the game. Meanwhile, the location of content, such as enemy characters and item pick-ups, within the map can provide varying levels of challenge depending on whether the content aids or harms the player, how it is strategically placed between multiple players, or how much extra exploration is needed to reach it. As mentioned earlier, content layout has typically been performed in conjunction with the geometry generation in past studies, however Horswill and Foged [2012] provide a discussion of the importance of content layout and give an example of procedurally populating game maps with enemies and items through the use of a constraint satisfaction algorithm.

In this thesis, we provide solutions for geometry generation in Chapters 3 and 4 and explore personalized content layout in Chapters 5 and 6. From Chapters 4 through 6, we describe multiple algorithms that make up a single personalized procedural map generation solution in an action-shooter game. Here, the geometry and content layout of an action-shooter game are given separate genetic representations, separate evolutionary cycles, and different approaches to fitness evaluation. While the geometry is represented by a fixed n-ary tree that was specifically designed for this solution, the content layout is indirectly represented by a Compositional Pattern-Producing Network (CPPN) [Stanley, 2007] and evolved through NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002]. CPPN-NEAT has previously been used to generate game content, including the trajectory of particle weapons [Hastings et al., 2009], 3D models of star-ship hulls [Liapis et al., 2012], and the shape and color of flower petals [Risi et al., 2012], but to the best of our knowledge has not been used in the procedural generation of maps, specifically the layout of content throughout the map.

2.2.3 Review of Evolutionary Terrains

In this section, descriptions of six approaches to applying EA to procedural terrain generation are provided, along with general advantages and disadvantages of each. To the best of our knowledge, these six approaches are the only solutions in this field, thus highlighting the infancy and the potential for growth. As in Section 2.1 previously, virtual terrains receive a separate analysis because they are the focus of our initial study into the personalization of game maps.

The first known proposal of using EA in terrain generation was put forth by Ong et al. [2005] and their work culminated in a terrain generation program called *Terrainosaurus*,

implemented by Saunders [2006]. Their algorithm proposed a two phase process with both phases using evolution to manipulate user provided examples. The first stage was to generate the silhouette of the terrain by mutating a basic user sketch of the terrain's borders into a more complex and rough outline. The second phase involves using a geospatial imagery sample heightmap as the initial seed and mutating the height of control points and their surrounding area. The authors propose using fitness functions that compare the resulting candidate to that of the seed heightmap, making sure it has a similar design but with unique placement of features. The advantage to the algorithm created by Ong et al. is that, in both stages of generation, the terrain can be seeded, thus introducing the user's preference at the start of evolutionary process. However, the disadvantage is that the user may not able to provide a geospatial imagery sample of the type of terrain they want to generate.

Ashlock et al. [2005] apply EA to L-systems [Lindenmayer, 1968] in order to generate fractal style terrains. They use an L-system that expands in two dimensions such that if each symbol is translated into a square, then an even grid of vertices for a heightmap is produced. A displacement value is assigned to each symbol in the L-system grammar so that when an expansion occurs, the new vertices are given height values that are modified by the displacement values of the symbols that they border. The authors use EA to find an appropriate set of replacement rules and displacement values. This is done by using a target example terrain and optimizing the replacement rules to produce similar terrains. The results provided by Ashlock et al. show that an L-System can be effective at generating fractal style terrains and in later papers [Ashlock et al., 2008] these results are improved further. However, the disadvantage is that it is not known whether this algorithm can produce terrains that differ from typical fractal terrains and that do not exhibit the symmetrical qualities that are shown in results provided by Ashlock et al. This L-System approach also suffers from the same problem as Ong et al. where if an appropriate target terrain cannot be provided then the system cannot perform well and even if a target terrain is available, all resulting candidates will be closely related and the solution space will not be explored.

A significant contribution to the field of using evolutionary algorithms on terrain generation is made by Frade et al. [2008] with their *genTP* algorithm. The basis of their algorithm is to use noise functions in combination with mathematical operator strings to calculate the height value of each vertex on the heightmap. Genetic programming [Banzhaf et al., 1997] was used to add, remove, or substitute operators in the function that calculates the height values. Initially, Frade et al. [2009b] used IEC to demonstrate the algorithm's explorative capabilities and the aesthetically appealing terrains that could be produced at variable levels

of detail [Frade et al., 2009a]. They have since investigated automated fitness functions that favor smoother terrains that are well connected [Frade et al., 2010a;b], which would allow the created terrain to be more appropriate for use in video games. They demonstrate the terrains' applicability to games by utilizing it in the game *Chapas* [Rodrigues et al., 2010]. However, these fitness functions cause the resulting terrains to be excessively flat, indicating the complexities in trying to control SBPCG of terrain programmatically rather than through IEC.

Walsh and Gade [2010] have used an EA to modify parameter values that describe the terrain, including feature height, feature roughness, water level, and atmospheric properties such as sun angle and cloud coverage. Walsh and Grade first used IEC to optimize these parameters and later investigated a fitness measure that judges the aesthetic qualities of the terrain [Walsh and Gade, 2011]. It should be noted that this algorithm does not create a new terrain, rather it alters the appearance of an existing one. A sample terrain must be supplied to the algorithm and the feature arrangement and orientation of this terrain is not changed, rather the features just increase or decrease in size and rigidity. Therefore, it only explores a small portion of the solution space of all possible terrain and also suffers from issues of requiring a sample terrain like some of the other solutions discussed here. However, this approach of optimizing terrain parameters could be applied to work by Kamal and Uddin [2007], who present a parameter controlled fractal generation method, or Doran and Parberry [2010], who use highly parameterized cellular automaton to generate island terrain.

Togelius et al. [2010b] have used multi-objective fitness functions [Deb, 2001] to generate terrain that is appropriate for use in real-time strategy (RTS) games. Unlike the approaches mentioned earlier that only address terrain generation, this algorithm focuses more on creating playable game levels. This includes the placement of player bases, collectable resources, and terrain obstacles on the map. Thus, terrain generation becomes a process of object placement rather than direct heightmap manipulation. Togelius et al. put more emphasis on automating the generation process through the use of multi-objective fitness functions that strive to provide fair and fun gameplay for all players. This technique has also been adapted to work on 2D RTS maps [Togelius et al., 2010a] for the game *StarCraft* (Blizzard Entertainment, 1998). The disadvantage to this change of focus, however, is that the terrain that is generated lacks detail and variety; providing fair gameplay at the cost of aesthetically pleasing terrain.

In Chapter 3 we present our own evolutionary terrain solution that combines smaller patches of terrain to form larger terrains and uses a unique two-leveled approach to IEC,

which gives the user greater control over the terrain features in the resulting candidate terrains. We also include a comparison of our solution against those described above. We then demonstrate this system in Chapter 7 by showing the results of manipulating various parameters of the algorithm. For a more detailed analysis of all the solutions presented here, please refer to our published paper on this topic [Raffe et al., 2012].

2.3 Player Modeling

There are two core topics in this thesis: one of them is PCG, which has been described in the previous section, and the other is personalizing player experiences through the use of player models. In this section we discuss the general concepts of player modeling, the motivation for it, and how it is applied in this thesis.

For decades, media such as novels, movies, and radio have provided hours of entertainment to millions of people around the world. The advent of video games in the 1970's enhanced these experiences by allowing consumers to be actors within the fantasy world rather than just observers. Games technology and game design has progressed substantially over the past few decades to provide players with ever diversifying experiences. However, despite these advancements, the player is still experiencing a game that is primarily designed to appeal to as many people as possible, rather than to them as a unique individual.

Thus, a growing trend in games research is towards exploring methods of programmatically adapting gameplay to suit the preferences, abilities, and past experiences of individual players while they are playing the game [Bakkes et al., 2012]. Lopes and Bidarra [2011] argue that continuously modifying a game can lead to less predictable game scenarios and increased replay value, while adapting a game to suite a player's individuality leads to a much more personal experience, rather than the player feeling as though they are playing a game designed for someone else. Adaptation may be through automatically setting difficulty levels to match the skill of the player [Hunicke, 2005], adapting NPC AI so that it responds uniquely to a player's actions [Machado et al., 2011], or allowing for a flexible narrative that gives the player feeling of agency and as though they affect the state of the virtual world around them [Thue et al., 2007b].

For all the above techniques to be possible and to effectively personalize a game experience, the game must first attempt to understand the player's preferences, skill level, or psychological state. For this, *player modeling* techniques are used to build computational models (typically through machine learning techniques) that are usually used to predict how

the player will react to future game events and adjust them accordingly. We focus on *player preference modeling* (PPM) and so we strive to model and understand the player's preferences for certain game scenarios and then make modifications to the game to maximize the player's enjoyment.

2.3.1 Challenge and Flow in Games

Challenge is commonly stated as a key element that influences a player's enjoyment of a game. Malone [1981] argues that it is primarily the presence of a challenge to overcome, a sense of fantasy, and curiosity to explore the virtual world that motivates players in a game. Meanwhile, Byrne [2005] describes how challenge should be considered when designing game maps to maximize the player's enjoyment.

Chen [2007] states that a game that matches the right amount of challenge with the abilities of a specific player will promote a 'flow' experience, as defined by Csikszentmihalyi and Csikszentmihalyi [1975]. Flow is a state of mind during which a person is fully immersed in a task such that all their attention is focused on the task and time passes quickly. This is usually seen as an enjoyable experience or at least a potentially rewarding one. This concept leads to the saying 'time flies when you're having fun'.

Chen's discussion of flow in games argues that, if the challenge is too great when compared to the player's skill, then the game will be frustrating. Alternatively, if the challenge is too low for the player's skill, then the game will be boring. Thus, designing a high quality game experience involves balancing the challenge to meet the average player's skill level and therefore promote a flow experience.

For these reasons, in this thesis we strive to use a player model to balance the challenge of a game map. The terms 'player preferences' and 'player skills' are also sometimes used interchangeably because we are attempting to model a player's preferences regarding the challenge of the game. Their preferences are subjective and usually known to them while their skill is more of a measurable quality of how well they can complete a certain task.

2.3.2 Dynamic Difficulty Adjustment

In most industry games, before the player begins to play they are asked to choose a difficulty level from a list of nominal settings such as 'Beginner', 'Intermediate', and 'Expert' [Pagulayan et al., 2003]. These settings are typical of single player experiences where these

settings adjust the amount of aid given to the player and the skills and strategies employed by AI controlled enemies.

The problem with these types of difficulty settings is that it can be difficult for a player to judge their own skill before playing the game. Thus, research in the field of dynamic difficulty adjustment (DDA) [Hunicke, 2005] aims at using player modeling techniques to automatically adjust these difficulty settings depending on the player's current performance.

However, even with using these types of DDA solutions, the use of a short list of nominal settings limits the total number of experiences and do not perform well for those players whose abilities do not match any of the provided settings [Johnson and Wiles, 2003]. Similarly, mechanisms such as avatar customization allow a player to decide how they will overcome challenges in the game [Bostan and Ogut, 2009] but they typically only present marginally different experiences to the player.

Thus, more advanced DDA solutions adjust individual parameters of the game on a much finer scale so that the game can be adjusted to suite any player. For example, Hunicke and Chapman [2004] change the health values of enemies and the strength of the player's weapon in small increments depending on the player's current in-game health and inventory. Additionally, Berkovsky et al. [2010] change the time limit of a round of gameplay down to the second based upon how well they performed in previous rounds of gameplay. In a multiplayer game, this may manifest as a match-making system such as Microsoft's *Trueskill* system [Herbrich et al., 2007] that estimates each player's skill from team based gameplay, predicts their skill level on a numeric scale, and finds player's with a similar skill level to place the player with in future games.

DDA is not without problems and it can sometimes lead to frustrating gameplay. An example of this can be found in many arcade racing games such as *Mario Kart* (Nintendo, 1992-2013) where DDA mechanisms reduce the speed of players in the lead and increase the speed and steering capabilities of players behind as well as giving them extra power-ups. This effect, known as rubber-banding [Pagulayan et al., 2003], is usually increased as the distance between players increases and is used to keep the majority of racers in a group to promote intense gameplay and make the game more fair for beginners. However, the disadvantage of this is that it becomes unfair for players who are working hard to be in first place, only to be overtaken by a rubber-banded opponent. Additionally, the effects of this can sometimes be painfully obvious and gameplay can become frustrating for all as it appears that their skill has no relevance in the game.

2.3.3 Experience-driven Procedural Content Generation

While DDA can affect various game parameters such as the speed of a vehicle, the strength of a weapon, or the prowess of enemies, the challenge of a game and the player's enjoyment of it can also be manipulated through PCG techniques. Experience-driven PCG (EDPCG), recently termed by Yannakakis and Togelius [2011], is the application of knowledge about a player's in-game experience to PCG. These techniques commonly use SBPCG as a base approach to iteratively improving content to cause a desired response in the user such as pleasure or frustration.

While the SBPCG solutions described in Section 2.2 established fitness evaluation metrics based upon hypothesized criterion of high quality content, EDPCG techniques capture a player's preferences, mood, or ability in order to evaluate the appropriateness of the content relative to that particular player, thus personalizing the content to that player. Examples of these are the use of multi-user IEC to give players control over the types of race tracks they play [Cardamone et al., 2011a] or more subtly monitoring which weapons the player interacts with to determine parents when evolving particle weapons [Hastings et al., 2009].

An alternative to allowing the player's input to directly affect the fitness evaluation is instead to build and use a player model. A player model can capture information about the player through subjective feedback, in-game activity, or physiological responses [Yannakakis and Togelius, 2011]. They can either be used to report statistics about the players of the game in order to better formulate a fitness evaluation metric or can be used directly to predict whether potential content candidates will be appropriate for the player [Smith et al., 2011a].

As an example, Shaker et al. [2010] train neural networks offline to model emotional states of players in a platform game. These models are then used during gameplay to optimize controllable parameters of future maps given the playing style of the player in previous maps. In this approach, a single model is created for each emotional state for all players. In contrast, Togelius et al. [2007] model individual players by constructing an agent to mimic a player's driving style and then using this agent to evaluate potential race tracks. The taxonomy provided by Smith et al. [2011a] highlights that the use of learned per-player models to generate game content (especially maps) is rare. Under their taxonomy, our approach uses an Individual Induced Generative Reaction model, for which only Polymorph [Jennings-Teats et al., 2010] is listed as a comparable solution, which combines a universal model of map segment difficulty with individual models of player skill.

We use a combination of methods to capture and use the preferences of individual players. We borrow from the literature by using IEC to give players direct control over the evolution of the map geometry in Chapters 3 and 4. Meanwhile, we use a trained per-player preference modeling technique to evaluate the content layout of a map, as described in Chapters 5 and 6. We believe that per-player models better capture the individuality of each player by preventing the preferences of other players from skewing the learning process.

2.4 Recommender Systems

In Chapter 6 we define our PPM as a *recommender system* (RS) and so in this section we give a brief overview of the RS field. As with the overview of EA previously given in Section 2.2.1, we do not make any claims to furthering the knowledge in the RS field. Rather, we argue the benefits of using an RS framework during player modeling, especially in relation to personalized procedural map generation. The only contribution that may be of some benefit to the RS field is the *learning trend* metrics that are proposed in Chapter 9, which may be useful when analyzing the performance of other RS applications where user data is irregular.

2.4.1 Goals of Recommender Systems

The goal of all RS applications is to monitor how a user interacts with items in a system and then recommend other items that the user may be interested in. RS was originally introduced in the field of information retrieval as a means of improving web searches by recording what websites or documents a user was viewing most often and suggesting websites with similar content [Belkin and Croft, 1992]. In this scenario, a single website or document represents a single item in the system.

However, RS has found its greatest use in recommending further purchases in internet commerce applications [Schafer et al., 1999]. In this case, each unique product that a business has to offer is a single item. By monitoring which items a user purchases or rates well, the business is able to recommend further products that the user may enjoy, thus stimulating future purchases. Some of the most well-known and well documented public application of RS are the recommendation of movies on *Netflix.com* [Bennett and Lanning, 2007], the suggestion of related products on *Amazon.com* [Linden et al., 2003], and the auto-generation of a user's music playlist based on their previous listening at *Pandora.com* [John, 2006].

Once it is decided what the items of the system are, then data about the user's preferences can either be recorded objectively or subjectively. In an objective system, the user's

interaction with the items of the system is directly monitored. For example, this may be done by recording which products a user views and purchases [Linden et al., 2003] or how long they spend on a given website [Pazzani and Billsus, 1997]. In subjective systems, the user is typically asked to rate items they have previously interacted with, directly specifying their preferences. This may be in the form of specifying a rating out of five-stars or simply indicating thumbs up or thumbs down to indicate whether they like or dislike the item [John, 2006]. While objective systems are popular in research related to information retrieval, Adomavicius and Tuzhilin [2005] state that the primary focus of the RS field has been on subjective user data and they even define RS as a problem of estimating user ratings for items. Thus, a subjective system is assumed for the remainder of the discussion in this section and is the primary focus of the RS applications in this thesis.

The final step of designing an RS is to formulate how to find items that are similar to those that the user has previously enjoyed. There are two primary ways to achieve this: either through content-based recommendations [Pazzani and Billsus, 2007] or via collaborative filtering [Su and Khoshgoftaar, 2009]. Additionally, solutions have been proposed that use a hybrid between these two approaches to gain the benefits of both [Basu et al., 1998]. However, these approaches are typically more complex and as our work is an initial study into the use of RS as player models, we focus on the two independent approaches. Both of these techniques can be further distinguished as either memory-based or model-based approaches [Adomavicius and Tuzhilin, 2005]. The next three subsections describe these concepts in more detail.

2.4.2 Content-based Recommendations

In this thesis we use a *content-based RS* [Pazzani and Billsus, 2007] to form a PPM for each player. A content-based approach examines each user in isolation from all other users. The goal is to compare items that have previously been rated by the user with other items in the system. Therefore, there must be an item-to-item similarity calculation to find items similar to those that were previously enjoyed.

Figure 2.3 shows examples of the *item-content* matrix that forms the core of a content-based RS. In this system, there are only two users, each with their own matrix. Note that these two matrices never interact because each user is considered in isolation. Each matrix also only includes items that those users have previously rated, even if there are many other items in the system.

User 1:

	x_1	x_2	...	x_K	c
i_1	0.77	0.21	...	0.64	0
i_2	0.18	0.35	...	0.82	0
i_3	0.72	0.46	...	0.27	1

User 2:

	x_1	x_2	...	x_K	c
i_1	0.54	0.17	...	0.89	1
i_2	0.33	0.28	...	0.06	0
...	...	...	...	...	...
i_m	0.57	0.92	...	0.41	1

Figure 2.3: Examples of the item-content matrix. Each user has a separate matrix, indicating the items they have previously rated. User 1 has rated 3 *items while User 2 has rated* m *items. Each item has* $(x_1, .., x_K)$ *features that describe the content of that item. Each item also has a class* c*, which represents the rating given to the item. Here,* 0 *may represent the user disliked the item while* 1 *represents a positive rating.*

This approach is known as 'content-based' because it is the content of each item that is compared to one another. This is represented as the features $(x_1, .., x_K)$ of the items in Figure 2.3. Every item will have the same features but not the same feature values. For example, if a user has previously enjoyed movies with a genre feature of comedy, then it may be good to recommend other movies in the comedy genre. One of the major limitations of this approach though is overspecialization: always recommending comedy movies will limit the user's exposure to other genres, which they may also enjoy. They also suffer from a cold-start user problem: when a new user joins the system there is very little existing rating data to compare with the items of the system so recommendations are inaccurate.

2.4.3 Collaborative Filtering

An alternative to content-based RS is to instead use a *collaborative filter* (CF) [Su and Khoshgoftaar, 2009]. While a content-based RS considers each user in isolation, a CF bases its recommendations for a single user on what similar users have enjoyed. To do this, user-to-user similarity calculations are conducted instead of item-to-item ones, thus finding users that are similar to the current user.

User-to-user similarities can be determined through profile data or demographic data that is supplied by the user before they use the system [Rich, 1979]. In this way, stereotypes are assumed whereby it is expected that people with similar cultural backgrounds will enjoy the same items. In some commercial applications, recommendations are made purely from purchase history, such as how the *iTunes* (Apple Inc., 2013) music store states "Users who bought this album also bought these following albums".

However, the most common approach to RS is to use a user-item matrix [Adomavicius and Tuzhilin, 2005], such as the one shown in Figure 2.4. Here, there is only a single matrix

	i_1	i_2	...	i_M
u_1	2	3	...	
u_2	4		...	1
...	...	...	...	...
u_N	1	5	...	4

Figure 2.4: An example of the user-item matrix. There are a total of N users in the system and a total of M items, recorded in this single matrix. If user u_n has rated item i_m then there is a rating value present (a $1-5$ rating scale is used in this example). If a user has not rated an item, then the corresponding matrix entry is empty.

that contains every item and every user in the system. If a user has rated an item, then there is a rating value (between 1 and 5 in this example) and if they have not rated an item then it has been left blank. Notice that there are no longer any item features. Instead, CF relies on items to be co-rated by multiple users. That is, multiple users have provided a rating on an item. In this way, a user is said to be similar to another user if they have provided similar ratings on co-rated items. CF can be more powerful than content-based approaches because it does not rely on interpreted features that could fail to capture the true reason of why a user likes an item. However, the reliance on co-rated items means that they perform poorly when there are only a few co-rated items between the users of the system. CF also suffers the cold-start user problem that degrades the performance of the content-based RS (a new user has no items co-rated with any other user) but they additionally suffer the cold-start item problem whereby a new item is never recommended to any user because no user has provided a rating for it yet [Park et al., 2006].

2.4.4 Memory-based vs. Model-based

The final distinction that can be made of an RS is whether it is memory-based or model-based. In a memory-based content-based RS, all the ratings for a single user are stored in memory. Then, when trying to find similar items, all existing items in the item-content matrix are compared to all the other items in the system. A popular approach to this is to use the cosine similarity measure [Adomavicius and Tuzhilin, 2005] that calculates similarity as the dot product of two items' feature vectors, divided by the product of the magnitude of those two vectors. A similar measure can also be used in a CF for calculating the similarity of two users' ratings on co-rated item. The downside to this approach is that it can become computationally expensive in systems with many items or when a user has rated many items.

The alternative is to use a model-based approach. Here, machine learning classifiers are trained using the features and classes (ratings) of the items in the user's item-content matrix. Then, given the features of un-rated items as input, the classifiers are used to predict the class (rating) of each item. These models are usually simple probabilistic models, artificial neural networks, or decision trees. As the user rates more items, more training data becomes available and the accuracy of the classifiers improves.

The benefit of a model-based approach in a content-based RS is that many classifiers are efficient to train with the relatively small amount of data expected by each user in a typical RS. Additionally, as each item is only compared with one model rather than all other items in an item-content matrix, the evaluation of un-rated items is more efficient than that of memory-based approaches. For this reason, we use a model-based approach when formulating the per-player preference models in Chapter 6.

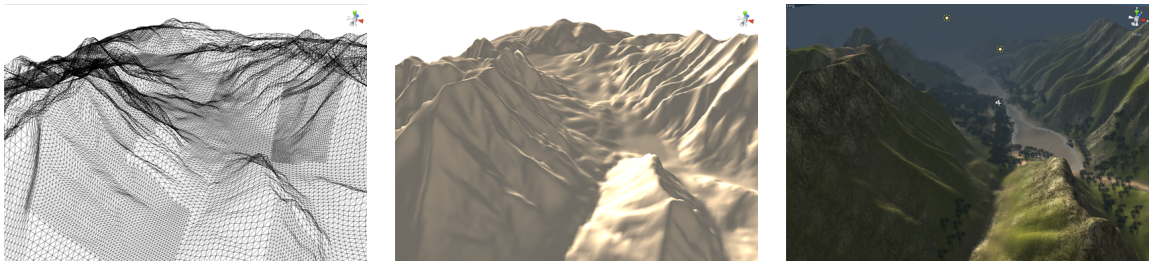
Chapter 3

Patch-based Terrain Generation

The first component of a complete procedural map generator that we chose to focus on is *geometry*. The geometry defines the boundaries of the map, the location of static objects within the map, and visual properties of the environment. In a sandbox game the player is free to explore the geometry as they please; in a multiplayer game the geometry acts as the arena for a competition; in a traditional challenge-based game the geometry may be a linear obstacle course to be traversed from start to finish. This terminology was chosen because it aligns with the definition and use of 'geometry' by Tutenel et al. [2011] in procedurally generating virtual environments.

The boundaries of the geometry may be in the form of a terrain that the rest of the map is built upon, the floors and walls of an interior space, the track of a racing game, the platforms of a 2D platform game, or even the cells of chess-like game. Meanwhile, the static objects of the map geometry may include rock outcrops, vegetation, furniture, or even entire buildings. The player can navigate around or through all these virtual objects or use them as cover but as they are static the player cannot interact with them any further. The virtual objects that the player can interact with are referred to in this thesis as the *content* of a map and are discussed in the background knowledge of Chapter 5. Finally, visual properties such as the color theme of the map, lighting effects, and other atmospheric effects are also considered as part of the geometry as they set the mood of the map but typically remain static during gameplay and cannot be interacted with.

We chose to investigate the geometry of a map first because typically all games have some form of geometry, aside from games set in infinite, non-detailed spaces such as Asteroids (Atari, 1979) or Flow (Thatgamecompany, 2006). Additionally, the content of the map is



(a) (b) (c)

Figure 3.1: The same heightmap terrain shown as a (a) wire-frame, (b) basic rendering, and (c) textured rendering. We seek a PCG solution to generating the heightmap as shown in (a) and (b) and do not investigate procedural methods of texturing and adding static objects such as in (c).

typically laid out across the geometry. Therefore, it is important to know the structure of the geometry before the location of the content is decided on to ensure the content is not placed outside the boundaries of play. It is typical for game genres such as simulation racing and flying games to have map geometry but no content, though only in the rare cases mentioned earlier does a game have content but no geometry. Therefore, we find the geometry subcomponent to represent the minimum requirement for a playable game map in a typical game.

Every game has its own unique geometry shape and style which is appropriate to the gameplay. Therefore, it is difficult to create a general procedural content generation (PCG) solution for creating geometry that will be applicable to all, or even the majority, of games. However, in this chapter we focus on generating 3D virtual terrains because many 3D games, and indeed real-world spaces, have some form of underlying terrain. This terrain may act as the sole form of geometry for the map or may contain additional forms of geometry on top of it. The player may be more cognizant of the structures on top of the terrain, such as rocks, flora, buildings, roads, and rivers and these geometry components may be important to the gameplay; however they are all additional geometry features that are placed on top of the terrain. Therefore, terrain is the lowest common factor of many game maps.

All terrains in this chapter are created from heightmaps, as defined in Chapter 2. Figure 3.1 shows the same terrain in three different states and shows what the focus of this chapter is. Figure 3.1a shows the wire-frame of the heightmap; a graph view of all vertices and edges that make up the heightmap surface. This is the underlying structure of all heightmap terrains. Meanwhile, Figure 3.1b shows how the heightmap terrain may appear in game, as a

single rendered mesh with a uniform color and a single light source shinning upon it. Many of the figures in this chapter are shown in this form. Finally, Figure 3.1c shows how the terrain may look when it is expertly textured, has water effects, atmospheric effects, multiple virtual light sources, and static objects placed upon it, all of which may be considered as part of the geometry of the map as well. However, here we only focus on the heightmap alone as shown in Figures 3.1a and 3.1b and leave detailing tasks of Figure 3.1c for future work as these aspects are more likely to depend on the genre of game and the theme of its maps and therefore the PCG process for these cannot be generalized as well as the terrain itself.

The remainder of this chapter is as follows: first we describe the process we developed for building a terrain in Section 3.1, which involves combining existing patches of heightmap data into a larger terrain. Next, in Section 3.2 we discuss how an evolutionary algorithm (EA) can be used to increase control over the types of terrain features that are present in the resulting terrains, as oppose to terrains that are randomly generated. More specifically, the patch-based terrain representation acts as the genetic representation for an interactive evolutionary computation (IEC) system. A novel subcomponent of this interactive EA is the use of a two-level selection scheme that allows every patch of each parent to be separately marked for crossover and mutation. The resulting solution is an *evolutionary terrain tool* (ETT) that can be used by a novice game designer to quickly produce detailed terrains. Finally, Section 3.3 provides a high-level comparison of this implementation with the other prominent evolutionary terrain solutions described in the literature analysis of Chapter 2. This is done to identify an appropriate direction to move forward with in order to investigate a complete preference-based procedural map generator and thus motivates the remaining chapters of the thesis. A lower-level look at how the effects of changing the various parameters of the ETT and how well it performs at creating terrains with specific features is provided in the demonstrations of Chapter 7.

3.1 Patch-based Terrains

The approach we take to generating 3D terrain in this thesis is to combine smaller *patches* of terrain to form larger terrains. This idea is not novel and is sometimes referred to as terrain synthesis, which borrows from the fields of 2D texture [Wei et al., 2009] and 3D model [Bhat et al., 2004] synthesis. While not using a heightmap representation, Aliaga et al. [2008] create top-down views of urban environments by extracting fragments from existing satellite imagery of urban layouts and recombining them to make new ones. Similarly, Merrell [2007]

does not use a heightmap representation but does produce complex 3D environments that could potentially be used in a game. Merrell achieves this by combining architectural building blocks together to form a larger, far more complex, floating cityscape.

Perhaps most similar to the patch-based technique presented in this thesis is that of Zhou et al. [2007]. Their work extracts patches of heightmap terrain from real-world digital elevation models (DEM). The system then takes a user provided sketch of terrain features, such as the ridge line of a mountain range, and overlays the DEM patches to produce a 3D terrain. Extracting patches from a DEM input produces realistic terrain features, while using the user provided sketch allows for control over the shape of the features in the procedurally generated terrain. The downside though is that because the user sketch is quite basic, the user is unable to specify terrain features other than mountain ridges or canyons, are unable to specify the location of both of these features in a single sketch, and cannot dictate different height values for different areas of the dominant terrain features. Instead, the style of the terrain features is controlled by the DEM terrains provided to the system, which prevents fine tuning of specific areas of the resulting heightmap.

Rusnell et al. [2009] address many of these problems by allowing the user to provide feature profiles in the form of additional side-on sketches of terrain feature heights and shapes, though this comes at the cost of increased user effort and knowledge of terrain aesthetics. Lagae et al. [2005] also generate virtual terrain by extracting features from a sample terrain but achieve it through a voxel grid representation rather than a heightmap. However, the results provided by Lagae et al. are not as seamless or as realistic as those by Zhou et al.

While the concept of patch-based terrains is not novel, the details of our implementation are. Additionally, the use of EA with a patch-based terrain representation is indeed unique and thus describing the PCG technique is a prerequisite to the discussion of the search-based procedural content generation (SBPCG) solution presented later in this chapter.

3.1.1 Core Principles

There are two types of terrain in this system, which we refer to as *sample terrains* and *candidate terrains*; the former being the input of the system and the latter being the output. Sample terrains are provided to the system as input and are decomposed into smaller patches. All patches are square and of a uniform size, which can be controlled by the user of the system. The patches are extracted from the sample terrain by dividing the sample terrain up into $n \times n$ patches. These patches are then stored in a *patch database* for later use. There is no

limit on the number of sample terrains provided to the system and the more sample terrains there are, the more variety there is in the patch database. However, the user should be cautious of the kinds of sample terrains provide to the system, as the features of the patches will become portions of larger features in the candidate terrain; thus, realistic sample terrains will more likely produce realistic candidate terrains.

Candidate terrains are the output of the system and are the result of selecting patches from the patch database and placing them in an $n \times n$ grid. This $n \times n$ grid is henceforth referred to as the *patch-matrix*. Each element of the patch-matrix is simply an identifier that links to a single patch in the patch database. Thus, the patch-matrix is the intermediary representation between raw heightmap vertex values and a complete rendered terrain. The only way to change a candidate terrain in this system is by changing the identifier in an element of the underlying patch-matrix. Once each element of the patch-matrix has been filled, either randomly or through the interactive EA approach discussed later in this chapter, the system renders the candidate terrain by taking the heightmap data of each patch from the patch database and joining adjacent patches together.

How adjacent patches of heightmap data are joined together seamlessly is an important topic as it can affect the realism and coherency of the candidate terrains. We refer to the process of connecting adjacent patches as *stitching*, which is conducted by first *overlapping* two patches of terrain and then conducting *seam removal* to smooth the transition from one patch to the next. This terminology is similar to that used by Efros and Freeman [2001] when describing the act of using a small sample texture to create a larger one for memory efficiency. The terminology is an analogy of quilting; the process of stitching together small patches of material to make a larger piece of material that exhibits either various visual styles or a repeating pattern as a result of the chosen patches and their position. A well-constructed quilt will be seamless such that it is not obvious where one patch ends and another one begins. Thus, the quilt will appear to be constructed from a single large piece of material.

These goals also apply to the process of stitching heightmap patches together. For example, Figure 3.2 shows two terrain patches with quite different average height values. When connected together without any overlapping or seam removal, there is an obvious boundary of where one patch ends and the next one begins. Once overlapping and seam removal is conducted though, it appears to be a single piece of terrain with a variety of high and low areas, joined by a slope. The next two subsections detail the processes we used for overlapping and seam removal.

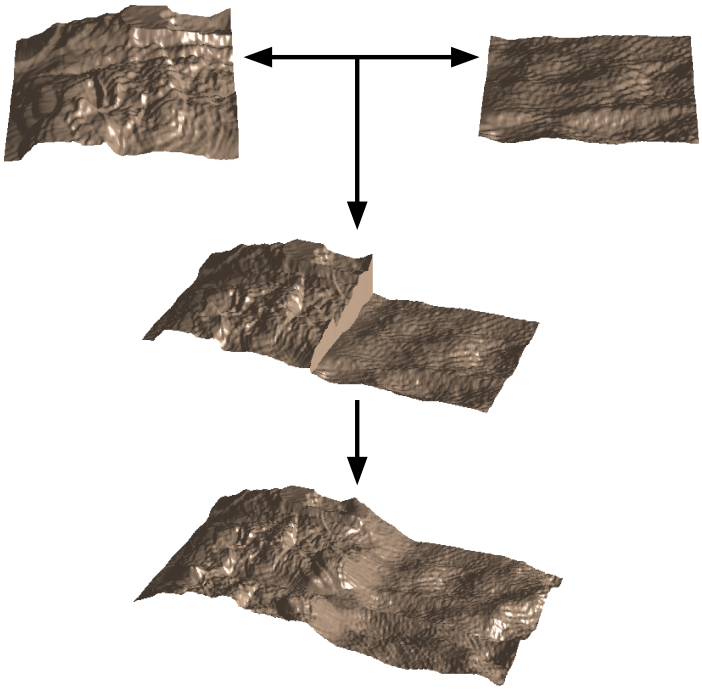


Figure 3.2: Two patches first joined with no overlapping and then with overlapping and cubic spline interpolation.

3.1.2 Overlapping: Roof-Tiling

Patches are overlapped with each other so that the area between two patches contains features of both patches. That is, the vertices in the overlap region get their height values as a weighted average of the height values of the corresponding vertices of each patch. This ensures that the area between two patches exhibits features of both patches, blended together. A smaller overlap region will result in a more immediate transition from one patch to the next and therefore will create terrain features such as steep inclines. Meanwhile, a larger overlap region will allow for gradual transitions and result in smoother terrains. Another point to note is that a patch must overlap with all adjacent patches. Thus, a patch in the middle of the patch-matrix will need to overlap directly with four other patches.

In order to accomplish the overlapping of the entire patch-matrix in a single process, we employ a *roof-tiling* overlapping algorithm, so called due to its likeness to laying tiles on the roof of a house. This process is outlined in Algorithm 3.1. It involves working from left to right across the patch-matrix, stitching each patch with the one before it (to the left of it), including both overlapping and seam removal. Once an entire row is stitched together, it is

Algorithm 3.1 The 'roof-tiling' algorithm.

```
1: vertPos = 0
2: while(vertPos + patchLength < terrainLength) do
3:   horiPos = 0
4:   while(horiPos + patchWidth < terrainWidth) do
5:     select a new patch from database
6:     place new patch at (horiPos − overlapSize, vertPos)
7:     stitch patch with any existing patch to the left
8:     horiPos = horiPos + patchWidth
9:   end while
10:  shift row from vertPos to vertPos − overlapSize)
11:  stitch row with any existing row above it
12:  vertPos = vertPosition + patchLength
13: end while
```

likewise overlapped with the row before it (above it) and seam removal is conducted on the entirety of the two rows. By constructing an entire row before overlapping it with one above it, the overlap region of two patches is also overlapped with overlap region of the row above it. That is, a patch will overlap with all eight side and corner neighbors rather than only its four side neighbors.

Overlapping patches does however introduce a small error. As mentioned earlier, patches are extracted by dividing the sample terrains up into an $n \times n$ grid. As the candidate terrains are forced to be the same size as the sample terrains, the patch-matrix of a candidate terrain also has $n \times n$ elements. However, if each patch is overlapped with the patch to the left and right of it by a ratio x, then the number of patches needed in each dimension of the patch-matrix to make the candidate terrain the same size as the sample terrain is $(n + 1)x$. Additionally, if $x > 0.5$ then at least 50% of each patch will be overlapped with the patch to the left of it and then again with the patch to the right, meaning that the entire patch is overlapped and its individual terrain features may be lost.

To solve both of these problems, extra vertices are extracted to the left of and below each patch when it is being extracted from the sample terrains. The quantity of extra vertices on each side is equivalent to the number of vertices in a single overlap region between two patches. That is, each extracted patch contains additional heightmap data from adjacent patches in the sample terrain. For the patches at the end of a row or column in the sample terrain where there are no additional vertices, the patch is unfolded by mirroring the existing vertices in the patch until an entire patch worth of data has been generated. Similarly, when

creating a candidate terrain, the extra overlap vertices for patches at the end of a row or column are discarded.

3.1.3 Seam Removal: Linear vs. Cubic Spline Interpolation

Once two patches have been overlapped, the vertices in the overlap region need to be assigned height values. It is not acceptable to simply take the average height value of the two overlapping vertices from the patches because this may still lead to a sudden change in height from a vertex outside the overlap region to a neighboring vertex within the overlap region.

To join adjacent patches, Zhou et al. [2007] also overlap patches of heightmap data and use an algorithm they call Poisson seam removal, which is a variation on the popular Poisson image editing technique [Pérez et al., 2003] used in blending colored images. However, both Zhou et al. and Pérez et al. state that the Poisson algorithm works best if the intensities of the pixel are within similar ranges for both images. In the case of heightmaps, this means that the area around the seam must have similar height values. To solve this, Zhou et al. include a process of finding a patch with similar boundary features to an existing patch of the candidate terrain to minimize height differences. Additionally, once such a patch is found, the two patches are analyzed simultaneously with a graph-cut algorithm [Kwatra et al., 2003] to find the optimal single dividing seam between them.

These limitations mean that the Poisson seam removal technique is not suitable for our application. Through random generation and evolutionary operators, patches are selected from the patch database stochastically and as the patches are placed on a fixed grid, all patches must overlap by the same amount. This means that it is highly likely that two adjacent patches will contain very different average height values and there will be no optimal seam line between them. For example, our seam removal algorithm must accommodate for a patch with a mountain range and a patch with a valley to be placed next to each other. Additionally, in each generation the ETT, up to seven new candidate terrains must be generated, in which the entire patch-matrix undergoes overlapping and seam removal to render the final heightmap. In order for the ETT to respond in real-time to the user's input, these processes must be efficient and by eliminating the need of finding suitable adjacent patches and optimal seam lines we can increase performance.

Thus, we interpolate the height values in the overlap region by weighting two overlapping vertices based upon their position in the overlap region. That is, assuming a left patch and a right patch that are being stitched together, then the height values closer to the left side in

the overlap region will be largely influenced by height values of the left patch and not much by the patch on the right side. This applies vice versa and height values in the center of the overlap region should be influenced equally by both patches.

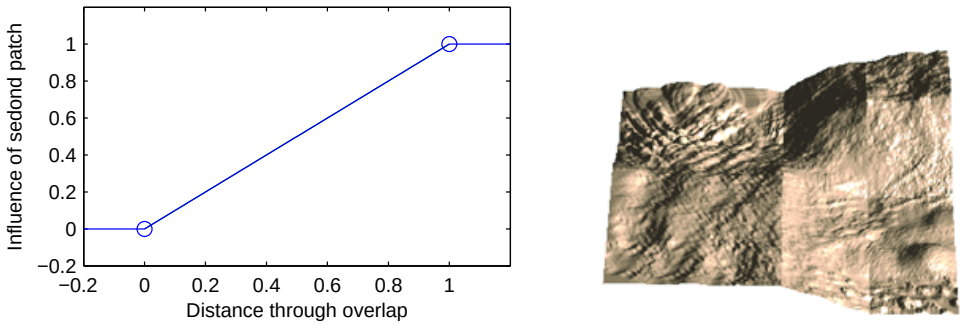
If linear interpolation is used, the height value h_o of any vertex (x, y) in the overlap region can be calculated as:

$$h_o(x,y) = h_l(x,y) + \left[\left(\frac{x}{x_{max}}\right)(h_r(x,y) - h_l(x,y))\right] \tag{3.1}$$

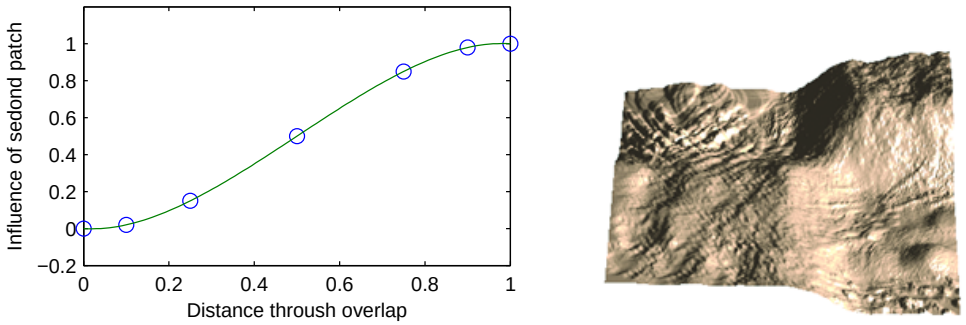
This is assuming that two patches are being stitched horizontally in a row of the patch-matrix but the process is similar for stitching entire rows together. $h_l(x,y)$ returns the height value of the vertex in the left patch with coordinates (x, y) and $h_r(x,y)$ returns the height value of the corresponding vertex that it is overlapping with from the right patch. x_{max} is the total number of vertices in the x-axis of the overlap region. Thus, the height of the vertex in the left patch is adjusted by the weighted difference between the height values of the right and left patches, where the weight is linearly correlated with the distance of the current vertex through the overlap region. This simple linear relationship is shown in the graph of Figure 3.3a.

Linear interpolation ensures that the height of vertices in the overlap region transition appropriately from one patch to the next without losing details and features in the overlap region. However, due to the sudden change in line direction (weight value) that is shown in the graph of Figure 3.3a at $x = 0.0$ and $x = 1.0$, the boundaries of the overlap region are made apparent. For example, the terrain example in Figure 3.3a is constructed from a 2×2 patch-matrix. While there is no single obvious seam between patches, there are now at least two subtle seams per overlap region.

To solve this, the linear interpolation is replaced with cubic spline interpolation. This is calculated similarly to linear interpolation except that $\left(\frac{x}{x_{max}}\right)$ is instead fed to a cubic spline function bounded by $(0.0, 0.0)$ and $(1.0, 1.0)$ in order to calculate the weight to be used. The graph in Figure 3.3b shows the cubic spline curve used in the ETT and the remainder of the terrain figures in this thesis. The curve is defined by the indicated points and allows for a gradual change in weight at the start and end of an overlap region and a steep change of weight in the center. The terrain example in Figure 3.3b uses the same patch-matrix as in Figure 3.3a and while the beginning and end of each patch can still be found with a keen eye due to the height differences of adjacent patches, there is now a more natural transition from one patch to the next due to the smoother boundaries of the overlap region. In order



(a) Linear interpolation



(b) Cubic spline interpolation

Figure 3.3: Linear interpolation and cubic spline interpolation.

for linear interpolation to create the same effect, at least of 90% every patch would need to be used in each stitching operation.

The benefit of using the above interpolation techniques for the vertices in the overlap region is that new terrain features can be constructed by the combination of features of two adjacent patches, how much they overlap, and how seam removal is conducted. In the work by Zhou et al. [2007], the features must be present in the sample terrains otherwise that feature is unlikely to appear in a candidate terrain. Thus, our approach grants the user more control to construct terrains that closely match their requirements. However, this is at the cost of a small degree of realism when compared to the terrains produced by Zhou et al.

3.1.4 Summary of Terrain Parameters

From the discussion of the patch-based terrain generation process described above, a few parameters have arisen. They are briefly described below to provide an overview of the system. Two of these parameters, patch-matrix dimension and overlap ratio, are demonstrated in

Chapter 7 to show how they affect the terrain generation process and how the modification of them provides the user with an increased level of control over the types of terrains that are generated.

The following are parameters that can be specified by the user: the terrain resolution determines the number of vertices in sample terrains and candidate terrains (the same value used for both). Typical terrain resolutions are 128×128, 256×256, or 512×512. A higher resolution produces more realistic terrains but, because the resolution of the terrain affects the resolution of the patches extracted from it, this will also increase the number of vertices that need to be considered in each patch stitching procedure. The user can also specify the dimension of the patch-matrix and an overlap ratio parameter. The patch-matrix dimensions state the size of the patch-matrices that make up every sample and candidate terrain. Patch-matrices are square ($n \times n$) and typical values are between 2×2 and 10×10. Meanwhile the overlap ratio determines how much of each patch will be used in stitching procedures and takes a value between 0.0 and 1.0.

From the above user specified parameters, the following dependent parameters can be calculated: the base patch resolution is calculated by dividing the terrain resolution by the patch-matrix dimensions and determines the number of vertices in the core section of a patch. The overlap size is calculated by multiplying the base patch resolution by the overlap ratio. This specifies the number of vertices used in each stitching operation. Note that both of the above calculations are rounded down to an integer value. Finally, the full patch resolution is calculated by adding the overlap size to each dimension of the base patch resolution. This is the size of each patch as it is extracted from the sample terrains, allowing for extra heightmap data to be used in stitching operations without losing the entirety of a patch when the overlap ratio is greater than 0.5.

Figure 3.4 shows all these parameters from a top-down view of a blank terrain. As an example of the calculations, values are given to each of the user specified parameters. There are 512 vertices in each row and column of heightmap of the terrain, which is edged by the outer border in the figure. Each inner square is a single patch and indicates the size of the base patch. Each patch has 170 vertices in each row and column of its heightmap. The shaded square shows an example of the size of a full patch, including additional data for the overlap operations. The size of the overlap area is an additional 85 vertices in each row and column, which makes a total of 255 vertices in each dimension of the heightmap of the full patch.

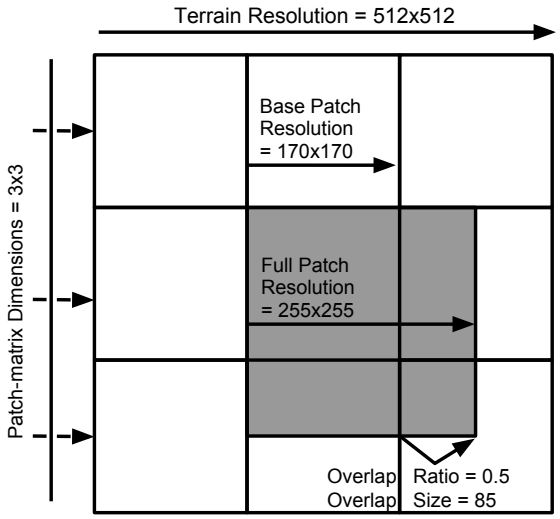


Figure 3.4: All the terrain parameters in the patch-based terrain generation system. User specified: terrain resolution, patch-matrix dimension, and overlap ratio. Dependent: base patch size, overlap size, and full patch size.

3.2 Evolving Patch Placement

To allow a user to control the types of terrains that are produced by the procedural terrain generation system, an interactive EA is utilized. Combining EA and patch-based terrains produces an evolutionary terrain tool (ETT) that can be used by novice map designers to create detailed terrains. The IEC interface can be used with very little explanation and is similar to those used in genetic art applications [Xu et al., 2007; Secretan et al., 2008]. The user does not need the same level of skill that is required to handcraft terrains using techniques such as terrain brushes [de Carpentier and Bidarra, 2009] nor does the user even need to search the patch database and choose patches themselves. However, if the user wishes to invest further learning time, more complex terrains can be generated through the manipulation of terrain parameters, evolutionary parameters, and the use of the *two-level interactive evolution* setup described below. In this section, we describe the EA used in the ETT; briefly describing the basic setup of the EA, the evolutionary operators used, and the fitness evaluation method.

3.2.1 Genetic Representation

The patch-based approach for generating terrains has the added benefit that it is easily utilized in an EA. Here, the patch-matrix becomes the genetic representation of each solution

(each candidate terrain). That is, the patch-matrix is the genotype and each slot of the patch-matrix is a single gene that can take an integer value between zero and the number of patches in the patch database (an identifier). In this case, the completely stitched heightmap is the phenotype, which is further processed to add color and lighting in order to make it visually appealing to the user.

The size of the patch matrix cannot be changed by the processes of the EA. Therefore, this genetic representation allows evolutionary operators, such as crossover and mutation, to be efficient as they only need to substitute existing patch identifiers with other identifiers in the patch database. Unlike many other heightmap-based procedural terrain generation algorithms and SBPCG solutions, the height values of each patch are never changed directly and only experience adjustments during the stitching process, which is only conducted in the genotype-to-phenotype transformation. Thus, instead, a variety of terrain features are created by rearranging the placement of patches rather than manipulating individual vertices, an act that can cause artifacts in the heightmap, as is the case with the work by Frade et al. [2009b].

3.2.2 Crossover and Mutation

The initial population in the ETT is typically randomly generated by selecting identifiers at random for each slot of each candidate's patch-matrix. Alternatively, though, the initial population can be seeded by assigning one or more of the user provided sample terrains as a candidate terrain. This is useful for when the user wishes to discontinue using the tool and resume at a later stage.

After the initial population, all candidate terrains in subsequent generations are created via crossover, mutation, a combination of both, or random generation, depending how many parents are chosen by the user in the previous generation. Parent selection is part of the interactive approach to fitness evaluation and is discussed in the next section. If no parents are selected in the current generation, then the candidates of the next generation (the offspring) are completely randomly generated in the same manner as the initial population.

If only one parent is selected, then mutation alone is conducted to generate the offspring. This is performed by first copying the single parent's patch-matrix to all offspring. Each offspring then tests to see which of its patches will mutate by randomly assigning each slot of the patch-matrix a random value between $[0.0, 1.0]$. If this value is equal to or below a user specified *mutation rate* parameter, then the respective patch mutates. Mutation then simply

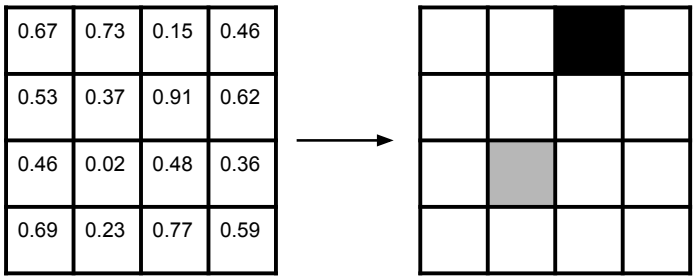


Figure 3.5: An example mutation operation for patch-based terrains. The parent (left side) is represented by all white patches. The offspring (right side) has mostly white patches as well but also contain two randomly mutated patches (gray and black). A mutation rate of 0.2 *was used.*

involves swapping the indicated patch with a new patch, chosen at random from the patch database. If a patch is not marked to mutate, then it simply remains the same. Figure 3.5 shows an example of this process. The patch-matrix on the left is that of the offspring before it mutates and is thus the same as its parent's patch-matrix. The values in each element are the randomly assigned mutation values of each patch. With a mutation rate of 0.2 being used in this example, only two patches mutate, which are marked in black and gray in the resulting patch-matrix on the right.

If two or more parents are selected by the user, then the offspring are generated by first conducting crossover, followed by mutation. A uniform crossover technique [Sywerda, 1989] is used with similar logic to the mutation process. Crossover is always conducted between two parents, which we will refer to as Parent A and Parent B. The first parent that the user selects during interactive evolution will always be Parent A. If the user only selects two parents, then the second selected parent will naturally be Parent B. However, if the user selects more than two parents, Parent B is randomly selected from the set of all other parents other than Parent A for each offspring. This means that while all offspring will always be a result of the first user selected parent, the other parent involved in the mating process may be different for each offspring.

To conduct crossover, the patch-matrix of Parent A is copied to all offspring, similar to the process of mutation. Each element of the patch-matrix of each offspring is then given a random value in the range of $[0.0, 1.0]$ and is compared to the user specified *crossover rate* parameter. If the value is greater than the crossover rate, then the patch from Parent A is kept. That is, crossover is not conducted, just as mutation is not conducted if the corresponding value is above the mutation rate. However, if the crossover value is below or

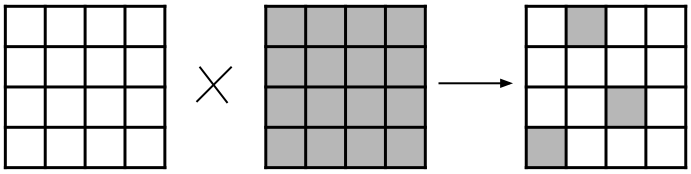


Figure 3.6: An example crossover operation. The white parent is the primary parent whose genetic structure is first copied to the offspring. A crossover rate of 0.2 *was used.*

equal to the crossover rate then that element of the patch-matrix is filled by the patch in the same position in the patch-matrix of Parent B. Figure 3.6 shows an example of crossover with a white parent (Parent A) and a gray parent (Parent B). A crossover rate of 0.2 is used here and so the probability of a patch being selected from the gray parent is relatively low. Thus, the resulting offspring is made up of mostly patches from the white parent, with only three patches being taken from the gray parent. After the crossover operation has completed, each offspring then undergoes mutation in the same way as described earlier.

3.2.3 Two-Level Interactive Evolution

In IEC, the fitness of a candidate is supplied by the user. Typically, this is conducted by allowing the user to examine a population of candidates and requiring them to either choose one or more candidates to become the parents of the next generation. The ETT deviates from this norm slightly through the use of a hierarchical mechanism that we refer to as *two-level interactive evolution*. *Parent selection* is the first level of interaction and functions much the same as many other IEC applications [Xu et al., 2007]. The user is presented with fully rendered images of all the candidate terrains and they are allowed to choose zero or more parents, the number of which affects the evolutionary operators as described earlier. Figure 3.7 shows an example of this screen. In the ETT, a population size of 8 is used and so each terrain in this screen is a candidate of the current generation. The candidate terrains in the top-left of the screen are the parents that were selected in the previous generation (the first two terrains in this example) and the rest are the offspring of those terrains. The user is required to tick the box under each terrain that appeals to them and that they would like to become parents of the next generation. After that, at the bottom of the screen the user is given the option of either evolving those parents immediately or moving onto the next level of interaction.

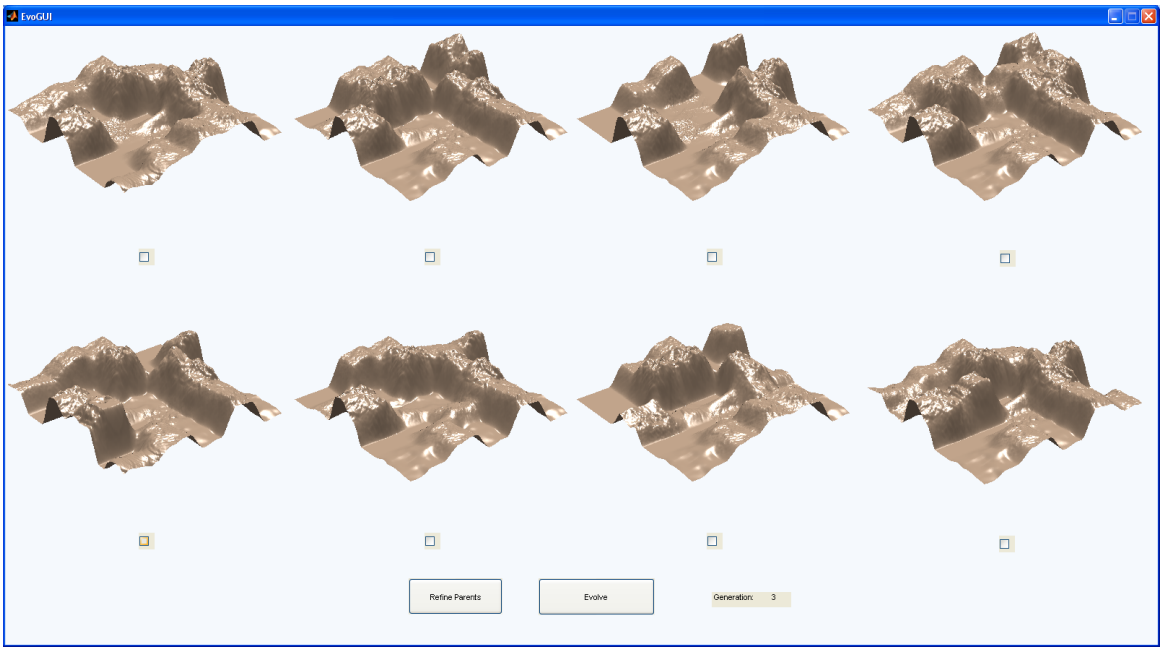


Figure 3.7: The user interface for parent selection. The candidates in the top-left of the screen are the parents that were selected from the previous generation and all other candidates are their offspring. Once parents are selected, the user can choose to either evolve them immediately or conduct gene selection.

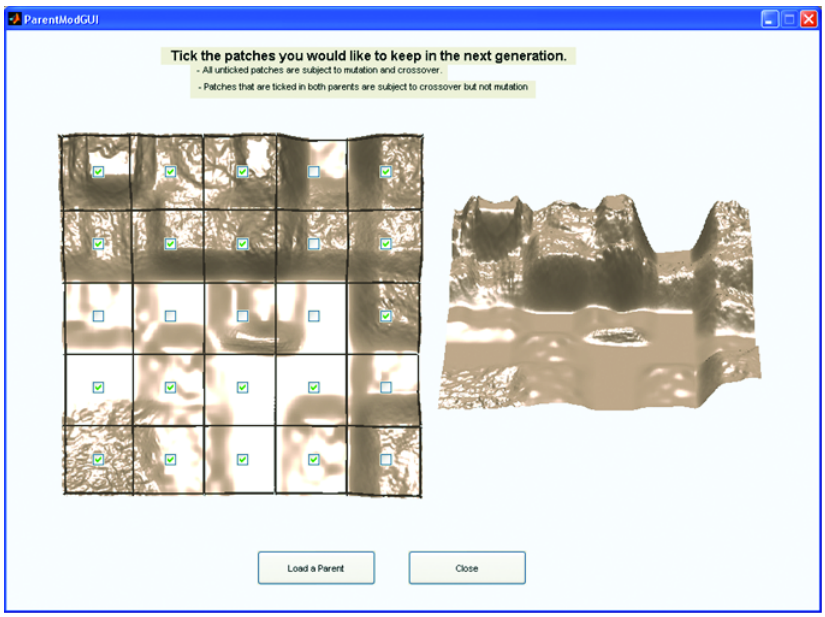


Figure 3.8: The gene selection *interface. The left image is a top-down view of the terrain on the right with the borders of each patch marked on it. By ticking a patch (a gene) the user is preventing mutation and crossover from affecting that patch.*

The second, more unique, level of the two-level IEC approach is called *gene selection*. At this stage, the user is shown the patch-matrix of each selected parent, one at a time, and asked to indicate which patches (genes) should be exempt from crossover and mutation operations. Figure 3.8 shows the interface used to allow the user to visually select which patches of a single parent are desirable and which are not. If no patches are selected then it is assumed that all patches are subject to crossover and mutation.

For mutation, if a patch is ticked in this screen, then that patch will be unable to mutate and therefore will be present in all offspring, assuming only a single parent was selected. For crossover, if a patch is selected in one of the mating parents (Parent A) but not in the other (Parent B), then, because that patch-matrix element is also prevented from mutating, the patch from Parent A will be in every offspring that is a result of those two parents mating. If a patch is ticked or unticked in both parents, then crossover is conducted as normal.

The gene selection mechanism was introduced after we perceived a flaw in our early experiments that only utilized the parent selection system. Figure 3.9 shows a terrain with an undesirable patch. In this scenario the user wants a mountain ridge on one side of the terrain and flat plains on the other side. Thus, the ridge on the bottom right in this figure is preventing the terrain from matching the user's desires. With a mutation rate of only

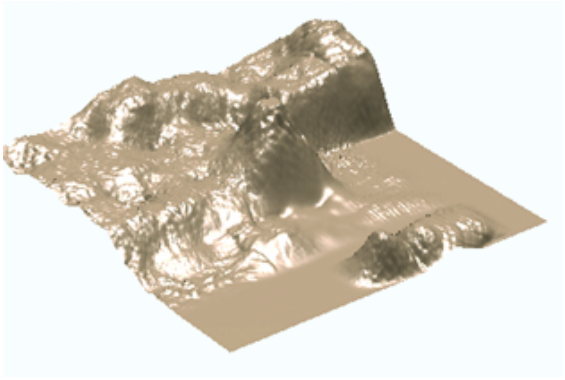


Figure 3.9: The ridge in the bottom right is undesirable. Removing this patch through parent selection alone typically takes many generations of waiting for a suitable mutation to appear in an offspring. With gene selection, this patch can be removed in a single generation.

0.1 or 0.2 there is only a small chance that this one patch would change and when it does another patch may change as well and become undesirable. This meant that it often took dozens of generations to substitute a single undesirable patch, increasing both user fatigue and frustration, a common issue of parent selection only IEC systems [Cotta and Fernández-Leiva, 2011]. Instead, with gene selection, the user can mark all other patches to be immune to mutation and so only that patch will change in the offspring.

Using parent selection alone or combining it with gene selection gives the user different levels of control at different stages of using the ETT. By using just parent selection the user can quickly explore the solution space early on in using the ETT in order to find the basic terrain features they desire. Once the general shape of the terrain has been discovered, the user can begin to use gene selection to improve smaller areas of the terrain; that is, exploitation of the surrounding solution space. We have not witnessed such a gene selection mechanism in any of the IEC literature that has been investigated but suggest that the increased level of control that it provides to the user may be useful in other applications.

3.2.4 Inverse Mutation Rate

For the terrain in Figure 3.9, using gene selection will ensure that no new undesirable mutations will occur while attempting to mutate the single undesirable patch. However, if the mutation rate is 0.2, a suitable value for when parent selection alone is used, it will still take many generations for that single patch to mutate. This is because the probability of that patch mutating is independent of the probability of all other patches in the patch-matrix.

This can be even more frustrating for the user as they will need to conduct gene selection in multiple generations but still not get the result they desire.

Thus, an inverse mutation rate is used such that the mutation rate increases as the number of mutation prone patches decreases. This is calculated as:

$$r^{inv} = 1 - \left[\frac{k}{n^2} \cdot (1 - r) \right] \tag{3.2}$$

where r is the user specified mutation rate, r^{inv} is the inverse mutation rate, k is the number of patches that can be mutated, and n^2 is the total number of patches in the $n \times n$ patch-matrix. If $k = n^2$ then all patches are subject to mutation and the user specified mutation rate is used. As the number of patches that can be affected by mutation decreases, the mutation rate increases towards 1.0. This ensures that, in the example in Figure 3.9, the single undesirable patch is highly likely to change in all offspring.

While the tested system conducted gene selection by requiring the user to mark patches that would not mutate, a further improvement to reduce user fatigue would be to instead have the user mark patches that should be mutated. Alternatively, the user's gene selections could be carried over from one generation to the next so that they would not need to re-mark candidate terrains in every generation. In many scenarios, such as when gene selection is used to refine small portions of a terrain candidate, these approaches will reduce the number of patches the user would have to mark. Even with this change though, the benefits of the inverse mutation rate are still apparent.

3.2.5 Summary of Evolution Parameters

Similar to the patch-based terrain algorithm, there are a few parameters involved in the interactive EA described above, most of which have been described in detail already. The *population size* in the ETT is fixed to 8 candidates per generation. A few candidates of each generation are the parents that were selected in the previous generation. Therefore, to maximize the number of offspring that are generated, it is generally recommended that the user does not select more than three parents.

In comparison to many other evolutionary algorithms, a population size of 8 is quite small. This can cause the population to converge quickly such that many of the offspring are very similar to the parents after just a few generations. However, this behavior is desired in this setting to reduce the number of generations the user needs to conduct during the exploration phase of their search. Additionally, this small number prevents the user from

being overwhelmed with information in the IEC interface in every generation and reduces the screen space needed to properly visualize small details of the terrain candidates. If the user does wish to see more offspring, they can simply select the same parents of the previous generation again.

The *crossover rate* determines from which parent a patch is inherited from for each element of the patch-matrix of an offspring. The *mutation rate* likewise controls how likely it is for an element in the patch-matrix to mutate and have the corresponding patch substituted with a randomly selected patch from the patch database. Both of these parameters take a value between 0.0 and 1.0. Both of these parameters will be evaluated in more detail in the experiments of Chapter 7. Finally, the *inverse mutation rate* is calculated such that as the number of patches that are allowed to mutate decreases, the probability of mutation on the remaining patches increases.

3.3 Comparison of Evolutionary Terrain Solutions

In this section, the ETT is compared with five other evolutionary terrain solutions, all of which were described in the literature review of Chapter 2. This is a high level comparison of the capabilities of each solution rather than an experimental evaluation. Experiments involving the ETT are reported in Chapter 7. This comparison is made here in order to discuss the benefits and limitations of the patch-based approach and the ETT in general. From this, insights are drawn that aid in the motivation and formulation of solutions throughout the rest of this thesis.

3.3.1 Benefits of Patch-based Terrains

There are two primary strengths of the ETT described in this chapter: the variety of terrain features that are enabled by the use of the patch-based terrain representation and the level of control afforded to the user through the two-level interactive evolution. For a comparison of these two aspects, we regard six other evolutionary terrain solutions that utilize a heightmap terrain representation. Figures of these terrains can be found in their respective papers and five out of the six have been republished in Raffe et al. [2012] with the permission of the primary author of each paper. They are not reproduced here due to copyright considerations. Note that each author has used a different rendering program and so the difference in texturing and lighting effects should be ignored. Instead, the focus here is on the shape and distribution of terrain features. As a base of comparison, Figure 3.10 shows an example of a

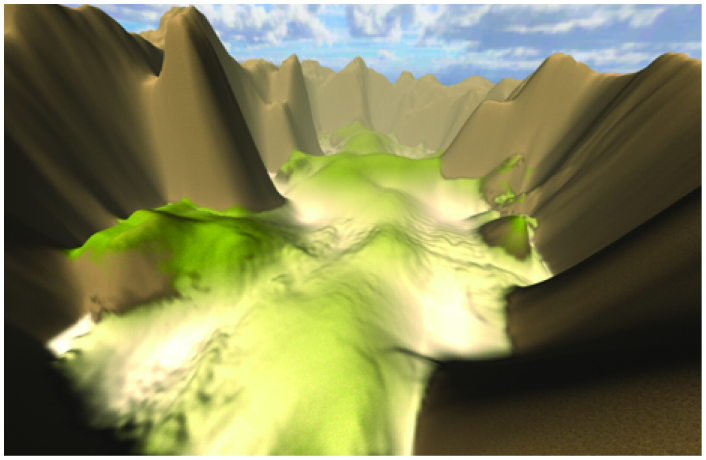


Figure 3.10: An example of the type of terrain that can be created by the ETT tool and that is compared with the outputs of six other evolutionary terrain solutions.

terrain created with the ETT; how this terrain was produced is described in the experiments of Chapter 7.

The terrains by Ong et al. [2005] and Saunders [2006] are smoothed over due to the use of low resolution satellite imagery as the seed to the evolutionary process as well as genetic operators that raise and lower heightmap vertices in a circular radius around selected control points. Similarly, the solution created by Togelius et al. [2010b] appears only to be capable of forming rounded hills. Though these hills can be of different heights, lengths and widths, the candidate terrains shown by Togelius et al. all appear smooth due to the lack of asymmetric terrain features, small noise on the surface of the terrain, raised plateaus, and sharp cliff faces.

Meanwhile, the solutions presented by Frade et al. [2009b; 2010b] make a trade-off between interesting terrain features and playable surfaces. A playable surface is defined as one that is relatively flat, such that a virtual human character can walk across it without appearing to break the laws of physics. It is rare to find a candidate terrain in the solutions provided by Frade et al. that exhibit both of these qualities. The terrains produced by Ashlock et al. [2005] always contain a dominant single feature in the middle of the heightmap. This is problematic if a user desires to create a single candidate terrain with a variety of terrain features.

In contrast to all these, the patch-based approach allows for a variety of terrain features, including steep cliffs, rolling hills, plateaus, craters, and other minor details. On top of this, each candidate terrain can have multiple of these terrain features and they can be asym-

metrically shaped. The ETT presented in this chapter is capable of producing both distinct terrain features and playable surfaces as the user desires, making the resulting terrains ideal for use as maps in games. However, it should be noted that the one feature that is difficult to produce with patch-based representation is a sharp angled cliff face, peninsula, or any triangular terrain feature. This is because these features require extremely small patch resolutions, which increases the difficulty of using the gene selection interface.

Not only does the patch-based approach allow for all the above terrain features to be created but the two-level interactive evolution setup gives the user the control to place those features where they wish. The terrains created by Walsh and Gade [2010], are realistic in comparison to the output of the other evolutionary terrain solutions. However, they are simply variants of an input terrain, with the height and noise of terrain features, along with atmospheric affects, being adjusted to provide various aesthetics rather than different terrain features. The system presented by Ong et al. [2005] and Saunders [2006] similarly uses only a single sample terrain. Thus, in both of these systems, if the sample terrain does not exhibit a similar feature layout to what the user desires, then it is unlikely that a suitable candidate terrain will be found. The ETT instead allows the user to supply multiple sample terrains to extract terrain features from and the recombination of patches can even produce new, unseen terrain features. Additionally, the use of both parent selection and gene selection gives control over the location of these terrain features.

Finally, unlike the two previously mentioned examples, in the ETT the user is able to change their desired goal throughout the evolutionary process rather than being required to know their ideal terrain beforehand and finding a similar sample terrain. This is even more apparent when compared to the system created by Ashlock et al. [2005], which requires a target terrain to be supplied to the system. The ETT here can be used as a design aid for novice game developers wishing to efficiently create game terrains. Requiring the user to supply a suitable sample or target terrain defeats this purpose.

3.3.2 Limitations and Future Considerations

Despite the numerous benefits stated above there is one major limitation of this approach regarding the objectives of this thesis; this system can neither be used by player's or used as a background process to improve a player's experience without their knowing. The ETT presented here works well as a tool for creating new terrains that can be used as game maps. While we believe it makes the terrain creation process easier and more time efficient for novice

developers over other forms of terrain creation, the ETT still requires some skill and time to use effectively. For example, creating a new terrain can take anywhere from ten minutes to half an hour. We cannot expect players to conduct this process for each new map that they play; more time may end up being spent creating maps rather than playing them.

In short, while the ETT can capture a user's preferences, it is too intrusive to be used by players during gameplay and rather should be used by designers during the game development stage. This does support some of the objectives of the thesis; increasing a game developer's productivity means that they can provide many more maps to their players, increasing the number of experiences for the player and the longevity of the game. However, while using SBPCG to create developer aids is an important and active research field [Liapis et al., 2013b], requiring developers to manually creating new maps to match the preferences of each player of a game is impractical.

There are two possibilities for rectifying this problem: either simplify the interactive evolution process such that it poses barely any inconvenience to the player or replace it entirely with an automated fitness function. Using a fitness function to drive the EA and produce map geometries that are suitable for a player's preferences is the more ideal situation of the two. However, it is not a trivial problem for terrain generation; what makes one candidate terrain better than another?

Both Frade et al. [2010b] and Togelius et al. [2010b] attempt to answer this question and both utilize automated fitness functions. Frade et al. [2010b] use a measure that encourages playable surfaces divided by obstacles. The goal of this measure is to generate terrains that would be appropriate in many game genres. Unfortunately, the results of this fitness function are terrains that are almost entirely flat. Togelius et al. [2010b] use a multiobjective EA with numerous fitness functions to generate maps for the real-time strategy (RTS) game genre. However, many of these fitness functions consider the layout of content, such as player bases and collectible resources, rather than the geometry of the map. Despite this, the terrains produced by Togelius et al. are both more interesting and more playable than those of Frade et al.

What this indicates to us is that how a map's geometry is evaluated is dependent on the game and the effect that the geometry has on a player's experience. By choosing the RTS genre, Togelius et al. were able to formulate the fitness functions to measure aspects of the map that played an important role in the player's experience. They improve this even further in later work [Togelius et al., 2010a] by focusing on creating maps for the game *StarCraft* (Blizzard Entertainment, 1998). While the geometry is no longer a heightmap

terrain, they use similar techniques to their previous work but with the unique gameplay elements of *StarCraft* in mind, which produces a stronger solution.

Similarly, successful EDPCG solutions for game maps such as the creation of race tracks [Togelius et al., 2007] and 2D platform maps [Shaker et al., 2010] have focused either on a genre of games or a specific game. Either way, the solutions were built into a test bed game for the purposes of demonstrating an example and experimentation. In both of these examples, the geometry of the maps also greatly influences the player's experience. In contrast, the ETT described in this chapter was designed to be able to create terrains for a wide variety of game genres. This makes automating the terrain evaluation process difficult without experiencing the same shortcomings as Frade et al. [2010b]. Thus, it is better to create a solution for a test bed game and generalize it rather than to attempt to formulate general solution from the beginning.

These final observations motivate the remainder of this thesis. In the next chapter we introduce a test bed game. We then go on to describe the geometry generation technique. Though some of the ideas from the ETT are utilized, the geometry is no longer in the form of a heightmap terrain but rather interior rooms and corridors. As the objective of this thesis is to investigate all three components of personalized map generation, we go on to describe a content layout algorithm and a player preference model in subsequent chapters. With all three of these components implemented, the test bed game contains a complete EDPCG solution that is experimented upon in Chapter 9.

3.4 Summary

- Patch-based terrains combine smaller patches of terrain to create a larger one. This is achieved by first extracting patches from user provided sample terrains and then inserting them into a patch-matrix to form a new candidate terrain.
- Once a patch-matrix has been created and the terrain is to be rendered, the heightmap of the candidate terrain is formed by stitching the heightmaps of the specified patches together.
- Stitching involves overlapping and seam removal. Overlapping proceeds in a roof-tiling manner. Seam removal is done by interpolating the height values of each vertex in the overlap region using cubic spline interpolation.

- There are three user specified parameters associated with the patch-based representation and processes: terrain resolution, patch-matrix dimensions, and overlap ratio.
- An evolutionary terrain tool (ETT) was created by using the patch-based terrains in a SBPCG solution. In the ETT, the patch-matrix becomes the genetic representation of each candidate terrain in an interactive EA.
- Crossover between two parents is done by first copying one parent to all offspring. Then, each element of the patch-matrix of each offspring is given a random value. If this value is less than or equal to the crossover rate parameter, then the patch is substituted with the patch in the same position in the other parent. Otherwise it remains unchanged.
- Mutation is carried out when one parent is selected or after crossover has finished. For each element of the each offspring's patch-matrix, if a randomly generated value is less than or equal to the mutation rate parameter, then it is substituted with a randomly selected patch. Otherwise, it remains unchanged.
- A novel two-level IEC system is used for fitness evaluation. Parent selection involves the user marking which candidate terrains are appealing to them. Gene selection asks the user to mark any patches on each parent that they wish to make immune to crossover and mutation operations.
- The two-level IEC allows users to explore the broader solution space early on by only using parent selection. Later, when a roughly suitable candidate terrain has been found, gene selection can be used to make small refinements.
- In comparison to other evolutionary terrain solutions, the patch-based genetic representation can produce a wide variety of terrain features and gives users control over their shape and location.
- The biggest short-coming of the ETT is that it is better suited as a developer tool rather than a means of personalizing map geometry to individual players. In order for geometry to be optimized in this manner, either the IEC process needs to be simplified or it should be replaced with an automated fitness evaluator.
- An EDPCG solution benefits from having a specific game genre in mind. The ETT, however, is designed to create terrains for a variety of game genres and so the simplification or removal of the IEC process is non-trivial. Thus, in the following three chapters,

we introduce a test bed game in which geometry generation, content layout, and player preference modeling systems are all implemented to form a complete EDPCG solution.

Chapter 4

Interior Generation via Room Templates

The aim of this thesis is to investigate all three components of personalized map generation; the geometry, the content layout, and the player preference modeling. However, the time restriction of this candidature would not permit an individual experiment on each of these components in isolation. Thus, we utilized a single testbed game in which all the algorithms in the remainder of this thesis were developed into and experimented upon as a singular experience driven procedural content generation (EDPCG) solution.

The genre, mechanics, and visual setting of the game have guided certain design decisions within the solutions of each component. However, where possible, we try to show how these algorithms and genetic processes can be generalized to a broader category of games. The decision to adopt a fixed testbed game was made as a result of the analysis of the patch-based terrain generation algorithm, which is provided more at length in the previous chapter. That investigation attempted to generate terrain that would be applicable to a variety of game genres. However, each genre, and indeed even each game, has its own requirements for the geometry and content layout of its maps. Therefore, attempting to define a solution that could be used directly by the player was difficult without knowing what potential preferences the player may have.

Thus, instead of trying to develop general solutions from the start, we instead chose to develop solutions for a specific game and show how they can be generalized. This also gives us a chance to explore the role of geometry and content in a specific game, which may act as an example to follow when designing similar solutions in other games. This also allows us to

develop a complete playable game that can be experienced by real game players. Therefore, we can evaluate the personalized procedural map generation solutions in a real-world context.

This chapter begins by first briefly describing the game that was used as the testbed. Then, from Section 4.2 onwards, the geometry generation technique is described. The role of geometry in the context of the chosen game is first explained, followed by a brief description of the room template approach to geometry generation used. This approach continues the trend of smaller geometry samples being used to construct a more complete geometry, as was done with the patch representation for terrains in the previous chapter. The scope of the geometry generation is also discussed here. Design considerations for the representation of the geometry are then discussed in Section 4.4. This is followed in Section 4.5 with a description of the *fixed n-ary tree* genetic representation of the geometry. This is discussed within the paradigm of an evolutionary algorithm (EA) and so includes the fitness evaluation and mutation operators. Finally, this chapter concludes with a discussion of how this geometry representation could be generalized to work with other games.

4.1 The PCG: Angry Bots Game

The game that serves as a testbed for the remainder of this thesis is titled *Procedural Content Generation: Angry Bots* (*PCG: Angry Bots*). It is built upon the *Angry Bots* technical demonstration provided free with the Unity game engine[1]. The game is a single-player action-shooter, played from a top-down perspective, and has mechanics similar to those of the popular *Alien Swarm* (Valve Corporation, 2010) multiplayer game. A screenshot of the game being played is shown in Figure 4.1.

The objective of the game is simply to get from one end of a map to the other. The players must navigate from room to room (the geometry) while killing enemies and collecting useful items (the content). A map can be skipped with no penalty. A leader board was implemented in the game prototype that reported the player with the most enemy kills, items collected, and maps successfully completed. This was put in place to give participants of the *PCG: Angry Bots* experiment an incentive to continue playing.

4.1.1 Game Cycle

Figure 4.2 shows a simplification of the game cycle in *PCG: Angry Bots*, with all system processes represented as rectangles and all player actions represented as ovals. From a player's

[1]Unity Game Engine: http://unity3d.com

Figure 4.1: Screenshot of gameplay in PCG: Angry Bots.

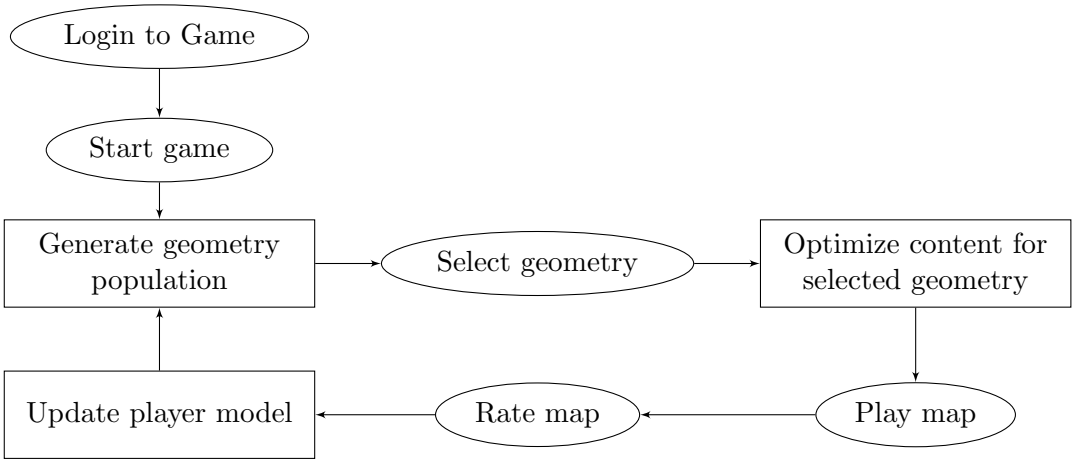


Figure 4.2: Simplified game cycle in PCG: Angry Bot. All blocks are system processes and all clouds are player actions.

perspective, they first log into the game and choose a menu option to begin playing. They are then shown possible map choices and are required to select a map that appeals to them. Next, the player will see a brief loading screen and then be able to play through the map. After they complete the map they are required to rate it on a six point scale. The process then repeats. However, what the player does not see are mechanisms to handle all three subcomponents of personalized procedural map generation. In Figure 4.2, those processes involve generating the geometry, optimizing the content layout for that geometry in accordance with that player's preference model, and then updating the player model with the rating that the player gives to the map.

Algorithm 4.1 gives a more in-depth view of the above game cycle. The added detail here from the player's perspective is that after each map, they are first asked whether the next map should be randomly generated or optimized using their player profile. Providing the player with this option gives two benefits: firstly, we anticipated that the recommender system player model or the content layout evolutionary cycle (see below for both) could get stuck in local optimums and therefore provide the player with repetitive experiences; an issue that does eventuate in the experimental results in Chapter 9. While the player is unlikely to understand this concept, it was expected that they would choose the randomized option if they felt they were continuously playing similar maps, which would not only provide an immediately different experience but would also add more explorative data to help the models break away from the local optimum. Secondly, by gathering ratings for both optimized and randomized maps, we are later able to compare our solutions with a plausible random baseline that would not be out of place in an independently developed roguelike game.

If the 'randomize' option is chosen, then the player is immediately provided with a new map and play commences. If the 'optimized' option is chosen, the player is then shown eight potential maps and is required to select one that they would like to play. The eight potential maps that are shown to the user only portray the geometry (the boundaries and static objects) of the map, which is optimized through an interactive evolutionary computation (IEC) mechanism. After the geometry is selected, the layout of the content (the location of enemies and pick-ups) in the map is calculated by a Compositional Pattern-Producing Network (CPPN) [Stanley, 2007], which is optimized via Neuroevolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] and a recommender system (RS) player model. Once the player has experienced the map, the rating that they provide is combined with features of the map to update the RS player model via supervised learning.

In the remainder of this chapter we describe the geometry representation of the maps in *PCG: Angry Bots* and how they are optimized via a search-based procedural content generation (SBPCG) solution similar to that used to evolve patch-based terrains in the previous chapter. The CPPN-NEAT approach to content layout is then discussed in the next chapter (Chapter 5) and the RS player model in the chapter after that (Chapter 6). All these systems are experimented upon through a public user experiment involving the *PCG: Angry Bots* game, the results of which are presented in Chapter 9.

4.2 Role of Geometry in PCG: Angry Bots

Throughout this thesis it has been mentioned that the geometry and content of a map can have a variety of effects on the player's experience depending on the game genre. In creating an EDPCG solution, it is important to understand how changes to each of these will impact the player. From that, an appropriate procedural generation algorithm can be developed to make meaningful and logical changes to the map and a means of capturing the player's preferences regarding those changes can be investigated.

In *PCG: Angry Bots*, the geometry acts as a backdrop to the gameplay, providing aesthetics and acting as a stage for the gameplay rather than affecting the challenge that the player will experience. Indeed, this may even be the case for many other action-shooter and first person shooter (FPS) games. Hullett and Whitehead [2010] describe design patterns of FPS maps that primarily influence a player's experience through the geometry. However, these design patterns are a result of the game mechanics of a typical AAA multiplayer FPS game. How the player utilizes the geometry of the map in a shooter game becomes important when competition with other intelligent human player's requires complex strategies, such as making extensive use of cover, gaining higher ground, and having proper spatial positioning and awareness depending on the current weapon that the player is using.

Through play testing of *PCG: Angry Bots*, it is apparent that most of these complex strategies are never needed. The simplistic routines of the artificial intelligence (AI) of the enemy non-player characters (NPCs), combined with their quick movement speed, makes strategic maneuvers such as flanking redundant, though utilizing cover in a basic manner is still useful. The player can only shoot on a horizontal plane due to the top-down view and the controls of the gameplay so there is no chance to use higher-ground tactics. Finally, all weapons have the characteristics of a typical automatic rifle and so nothing can be gained from a player's ability to adapt their spatial awareness.

Algorithm 4.1 An overview of the main game cycle. Player actions are marked with asterisks.

```
1: player := PlayerLogin()
2: if (player is newPlayer)
3:  generationMethod := randomize
4:  CPPN := new CPPNObject(randomPopulationFlag)
5:  RS := new RSObject()
6: else
7:  generationMethod := optimize
8:  CPPN := new CPPNObject(LoadLastPopulation())
9:  RS := new RSObject(LoadPlayerData())
10: while (playing)
11:   if (generationMethod is randomize)
12:    geometry := RandomGeometry()
13:    content := RandomContent(geometry)
14:   else if (generationMethod is optimize)
15:    geometryCandidates := Evolve(geometry)
16:    Display(geometryCandidates)
17:    geometry := GetPlayerInput(geometrySelection)
18:    while (evaluations < 10000)
19:      cppnCandidate := CPPN.NeatEvolution()
20:      content := CPPN.Process(geometry, cppnCandidate)
21:      mapCandidate := Combine(geometry, content)
22:      features := ExtractFeatures(mapCandidate)
23:      fitness := RS.ProbabilityOfLike(features)
24:      CPPN.UpdatePopulation(cppnCandidate, fitness)
25:      if (fitness is 1.0)
26:       break
27:      evaluations++
28:    end while
29:    content := CPPN.Process(geometry, CPPN.GetElite())
30:   end else if
31:   map := Combine(geometry, content)
32:   RenderMap(map)
33:   PlayMap()
34:   rating := GetPlayerInput(ratingSelection)
35:   features := ExtractFeatures(map)
36:   RS.UpdatePlayerModel(features,rating)
37:   generationMethod := GetPlayerInput(methodSelection)
38: end while
```

The maps in this game are also not typically complicated enough to be classified as a maze. Thus, navigating through a map can be made more time consuming with an increase in the size of the geometry but rarely requires enough mental focus to be considered as part of the challenge of the game. An added benefit of the closed interior environments that have a limited number of doorways is that the issues of human wayfinding in procedurally generated environments described by Biggs et al. [2008] are not as prevalent, further simplifying the generative process.

It may then be argued that the geometry should simply remain static and only change the layout of content in each round of the game, as was done in the commercially successful game *Left 4 Dead* (Valve Corporation, 2008). However, while the challenge that a player faces and the strategies they use are not overwhelmingly affected by the geometry, other aspects of the player's experience may still be affected. As mentioned, the size of a geometry will dictate the amount of time it takes to complete a map and how much content can fit into it. Changing the structure of the geometry gives the player a chance to explore. The player may appreciate the variety in appearance of the rooms that they pass through. The player may even have a preference over the shape and size of the rooms they pass through. Thus, the procedural generation of the map geometry is warranted here, though the design choices made for the solution are influenced by the above analysis.

4.3 Elements of Interior Spaces

The setting of *PCG: Angry Bots* is that of an interior space that can be described as the rooms and corridors of a science fiction space station. Thus, the geometry of a map includes the layout of rooms and corridors in relation to each other, the structure of floors and walls of each room, the location of doorways in each room, and the arrangement of static clutter such as furniture and computer consoles.

4.3.1 Scope Reduction

Despite all the elements of the geometry in *PCG: Angry Bots*, the procedural generation of geometry here only deals with the layout of rooms and corridors. Meanwhile, the remaining facets of the geometry are manually designed. Additionally, the background of all maps is empty as if the rooms were floating in space and therefore there is no need for terrain generation or the placement of rooms on top of a terrain.

This design choice was made to reduce the complexity of the overall EDPCG solution being developed, as procedurally generating the structure of rooms and buildings [Müller et al., 2006], arranging clutter logically [Taylor and Parberry, 2010], and placing architecture on terrain [Kelly and McCabe, 2006] are all extensive 3D graphics research fields in themselves. By manually designing these elements, we can ensure visually appealing, complex, logical, and playable rooms. This limited procedural geometry generation is also acceptable in *PCG: Angry Bots* because, due to the limited effect that the geometry has on player's experience in this game genre, changing the finer details of room structures adds no extra benefit to the player and therefore adds no benefit to the investigations of this thesis.

4.3.2 Room Templates

In the *PCG: Angry Bots* game, the geometry of a map is procedurally generated by connecting pre-designed *room templates* together via corridors. A room template defines the floor space, walls, door location, furniture, and general graphical appearance of a room. The use of room templates is an extension of the patch-based concept used for terrains in the previous chapter; each room is a 'patch' of geometry which is joined with other patches to form a larger, more interesting geometry. However, while the patch-based terrains were created by assembling patches of terrain in a uniform grid and stitching them together to blend their terrain features, the rooms here are connected to each other at doorways and are not regulated in shape or size. There is always a corridor between two rooms and those corridors can be placed at any door of the room template.

Each room template is a self-contained unit and so there is no reliance on any specific order of rooms to ensure that a candidate map is playable or appears logical. Additionally, by making each room self-contained, there is a higher chance of appropriate spatial and challenge segmentation [Zagal et al., 2008] occurring within the procedurally generated maps once the content is added.

Figure 4.3 shows an example of a room template and corridor template, which are joined together with other room templates to form a full map geometry. In the implementation in this thesis, 10 room templates and 5 corridor templates were manually crafted and used. These templates are shown in Appendix B. There is one additional room template that always acts as the starting room that the player begins the map in as well as the exit room that they must reach to successfully complete the map. These are marked with an 'S' and an 'E' in Figure 4.3.

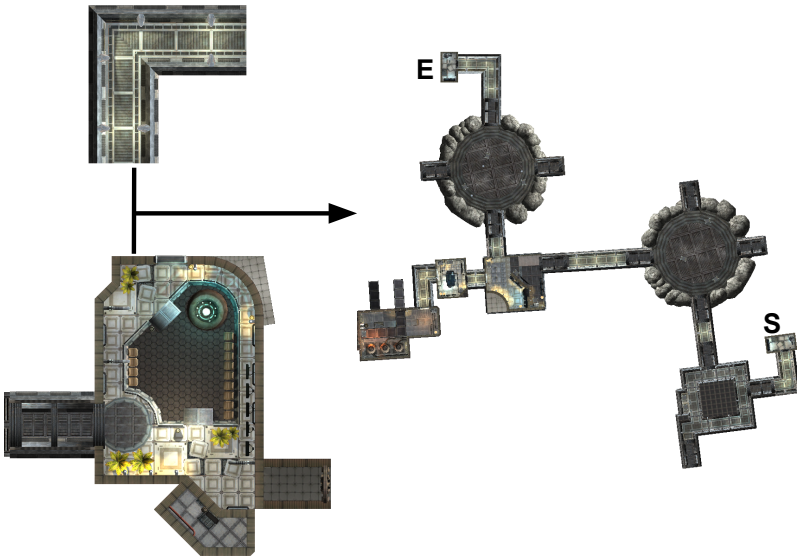


Figure 4.3: Top-down views of an example room template (bottom left) and corridor template (top left). When multiple templates are combined, they produce the geometry of a complete map candidate (right), with rooms connected together via corridors. Rooms 'S' and an 'E' are the 'start' and 'exit' rooms respectively.

4.4 Linear, Tree, or Graph Geometry Representation

With the considerations of the effect of geometry on the player's experience and the use of room templates in mind, this section presents some high level design choice for procedurally generating a geometry in *PCG: Angry Bots*. In preliminary implementations, the geometry in *PCG: Angry Bots* was linearly structured. A linked list data structure was used as the underlying representation and therefore each room of the map had exactly one room before it and exactly one room after it. However, this representation further reduced the player's experience of the geometry by eliminating most of the exploration of the map. The player only needed to remember which door they had entered through and find the door that lead to the next room.

Thus, a tree structure was instead investigated and used in the final build of the *PCG: Angry Bots* game. The details of this structure are provided in the next subsection. The tree structure still has a linear path from the starting room that the player begins in and the exit room that they attempt to reach. However, there are also paths that branch away from this direct path, increasing the complexity of the geometry and providing a slight increase in challenge as the player attempts to find their way through a simple maze. It also offers

those players that enjoy exploration of virtual worlds a chance to experience the content in many rooms in a single map.

The ultimate extension of the exploration idea is to use a graph representation. The primary difference here is that a graph representation would allow for circular paths in geometry. This further increases the challenge involved in navigating the map because if the player does not have good spatial memory then they may become disoriented and lost. This also mimics how most real world interior spaces are arranged, providing multiple paths to access the same physical space.

Unfortunately, there is also an added layer of complexity involved in using a graph representation. The room and corridor templates have been manually designed so that they can contain a high level of detail. This means that the shape, size, and entry/exit points of the room and corridor templates are fixed, which makes aligning the doorways of multiple rooms in a circular path becomes a non-trivial problem. Even in the simplest example where two rooms are cyclically connected by two corridors (such as if the first two rooms after the starting room in Figure 4.3 had two connected doors), establishing a corridor path between the second set of doorways using fixed corridor templates is a complex problem. Joining separate branches of the graph also increases the likelihood of the graph being nonplanar and therefore having overlapping geometry assets that would block the player's navigation path.

Rogue (Michael Toy and Glenn Wichman, 1980), an adventure role playing game (RPG) that spawned an entire genre of similar games known as roguelikes, resolved this issue by making the corridors adaptable in size. However, *Rogue* and many other roguelikes either use ASCII graphics or other simple 2D graphics and so the procedural generation of dynamically sized corridors was simplified. However, procedurally generating the size and shape of highly detailed, 3D corridors in games like *PCG: Angry Bots* without introducing graphical artifacts is also a non-trivial problem to solve. Thus, it was decided to stay with a tree representation for the geometry in order to spend more time focused on the research questions of this thesis and not on aside development problems.

4.5 Fixed n-ary Tree Evolution

The underlying genetic representation of the geometry that was used in the *PCG: Angry Bots* experiment is that of a *fixed n-ary tree*, an example of which can be seen in Figure 4.4. Each node in the tree is a room and each edge represents a corridor. In this figure, each

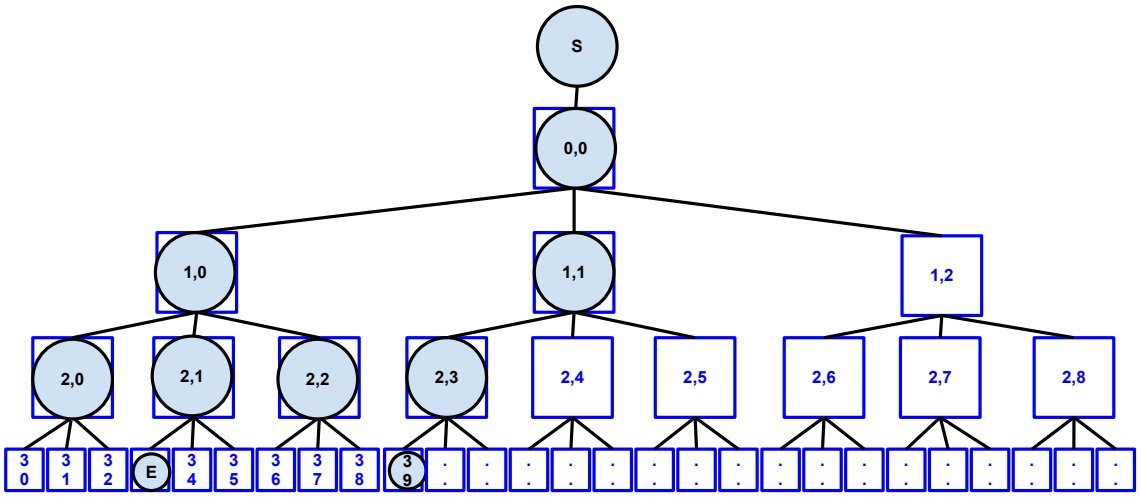


Figure 4.4: An example of a map's geometry represented as a fixed n-ary tree. 'S' is the starting room and 'E' is the exit room.

square slot represents a possible position of a node and each circle is an instantiated node that will be rendered as a room in the actual game. Each instantiated node and edge of the tree holds a reference to the room or corridor template to use when the map is rendered for play during the genotype-to-phenotype conversion process.

The node labeled with the coordinates $(0, 0)$ is the first room with content in it and is considered the root node. The node above the root node is the *starting room*, in which the player starts the map, and is only connected to one other room (the root node of the tree). Exactly one leaf node is the *exit room*, to which the player must try to reach to successfully complete the map. Like the starting room, the exit room only has one door and therefore only one parent node and no child nodes. As a tree structure is used instead of a graph structure, there exists only one path between the start room and the exit room, which is hence forth referred to as the *direct path*. All edges and nodes not on the direct path are referred to as *branching paths*.

Each node may have a maximum of n children. For this experiment, $n = 3$ simply because there are no room templates with more than four doors; one door leads to the room before the current room (the parent node) and up to three doors lead to rooms after the current room (the child nodes). Not all geometries have 4 doors but this does not interfere with the use of the 3-ary tree. Each door of each room template has an identifier so that the node also keeps track of which door leads to the parent node and which door leads to which child

node. If a door does not lead to another room then it is locked during gameplay to prevent the player from falling out of the map. If the current node is on the direct path, then at least one child node must also be on the direct path.

4.5.1 Random Generation and Recursive Branching

In the initial population, all geometry candidates are generated randomly. The direct path is first generated to be a random length between $[2, 7]$ rooms, resulting in small to medium sized maps for the player to experience first. As each node is created, a room template is randomly chosen as well as the corridor template that connects it to its parent node. When randomly generating a tree, the door of the current room template that each corridor connects to is chosen at random. This is so that a room template can be rotated so that the player does not always enter a specific room template through the same door, further increasing the variety of experiences.

After the direct path has been created, all nodes are recursively branched. Two variables control the number of branching paths in a tree: the *branching rate* and the *branching decay*, both of which are values between $[0.1, 0.9]$. For every possible branching path at every existing node in the tree, a random value between $[0.0, 1.0]$ is generated and if it is less than or equal to the branching rate then a child node is created. For each depth along a branching path, the branching rate is reduced by multiplying it by the branching decay value. This ensures that a node on the direct path always has a higher probability of having branching paths and that branching paths are restricted in length, preventing excessively large branching paths.

4.5.2 Node Labeling

Each node of the tree is given a label of $(depth, sibling)$, which can be seen inside of each possible node position Figure 4.4. The sibling number takes into account all the possible nodes at a specified depth, assuming a complete n-ary tree was generated. This is why the term 'fixed' is used to describe the tree. Assuming a complete tree at specified depth allows for the metaphor of placing each node in a bucket. This is important because even if a node is added or removed, the bucket remains and therefore sibling identifiers of all other nodes at that depth will not change. For example, in Figure 4.4, if node $(2, 0)$ is removed, nodes $(2, 1)$, $(2, 2)$, and $(2, 3)$ do not get re-assigned new coordinates. If node $(1, 0)$ is removed, nodes $(1, 1)$, $(2, 3)$, and $(3, 9)$ are not affected. Finally, if a new node is inserted at position $(3, 7)$, then node $(3, 9)$ is not re-labeled.

This fixed representation is chosen because the algorithm for determining the layout of content throughout the map is reliant on the coordinates of the nodes. This gives a relationship between the geometry and the content but prevents a mutation in the geometry from causing a severe change to the content layout. Thus, while the geometry and content genetic processes are interconnected, if so desired, one evolutionary cycle could be halted while the other was further refined. For example, the content evolution can be stopped and the current evolutionary champion used for consecutive maps. Meanwhile, the geometry can continue to evolve and it is expected that the fitness of the champion from the content evolution will remain similar. This is further discussed in Chapter 5 where we explain how the content layout is determined with regards to the node coordinates.

4.5.3 Genotype-to-Phenotype Conversion

When a valid candidate has been generated it is usually rendered, either to provide a preview during the fitness evaluation process described in the next section or to be experienced by the player. The fixed n-ary tree is the genotype (the genetic representation) and the rendered geometry is the phenotype (the observable result of the genes).

In order to convert the genotype to the phenotype, the starting room is first placed at the origin of the virtual world's coordinate system and its rotation is zeroed. This results in the player always starting a map by leaving south out of the first room. After this, the corridor between the starting room and the root node of the tree is instantiated. They are connected by aligning handcrafted reference points of the door of the room to the entry way of the corridor, creating a near seamless transition between the two. Next, the room template referenced by the root node is instantiated and the indicated door is similarly joined to the other end of the corridor. This process then repeats for all nodes and edges of the genotype, moving through the tree from parent node to child node. Unlike the patch-based terrain technique described in the previous chapter, there is no stitching process other than aligning reference points at each connection, thus making the genotype-to-phenotype conversion highly efficient.

4.5.4 Fitness Evaluation of Interior Geometry

Fitness evaluation of the geometry is conducted through IEC. One of the observations that we made from the patch-based terrains investigation of the previous chapter was that in order for an EDPCG solution to utilize the preferences of an individual player, either a minimally

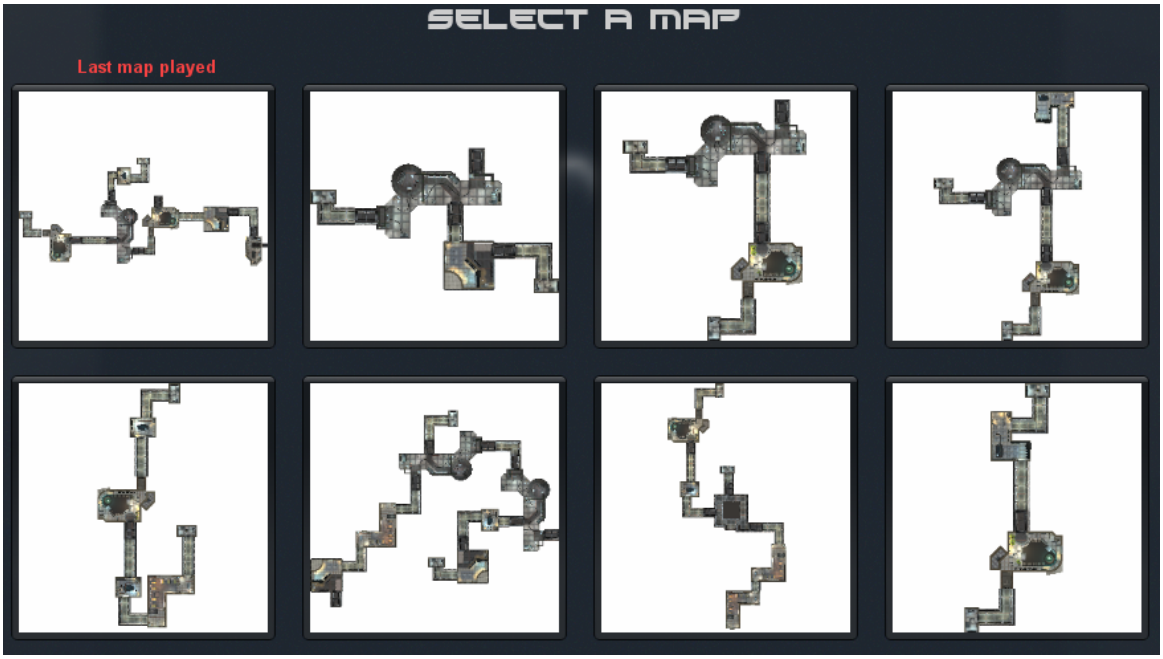


Figure 4.5: The geometry selection menu in PCG: Angry Bots. *The menu facilitates the use of interactive evolutionary computation (IEC).*

invasive IEC approach was needed or the fitness evaluation would need to be automated. For *PCG: Angry Bots*, we choose to use a simplified interactive EA to optimize the geometry and an automated fitness function to optimize the content layout.

For every map that a player experiences, they are shown a menu screen such as the one depicted in Figure 4.5. This menu shows the geometry of the parent of the generation in the top left (if this is not the first generation) as well as all the offspring generated from that parent. The player is then required to choose the geometry of the map that they wish to play next, which then becomes the parent for the next generation of geometries. Though some of the offspring may appear to be substantially different from the parent when the maps are rendered, there are typically strong similarities in their underlying fixed n-ary tree representation. This is due to the fixed labeling structure of the tree and the mutation operators described in the next section.

IEC was judged to be a sufficient method of fitness evaluation here because a player can quickly decide for themselves which geometry is most appealing to them. The geometry is displayed clearly through the previews in the geometry selection menu and can capture a player's preferences regarding the size of the map, the number of branching paths, and the type of rooms they will be navigating through. These are the primary means by which the

geometry will affect the player's experience and therefore the most accurate way of capturing the player's preferences regarding these aspects is through IEC.

Experienced players can also use this opportunity to memorize the paths through the map to help them navigate during gameplay. Using IEC to optimize the geometry of a map is suitable when knowing the map shape prior to play is beneficial (e.g. racing games) and geometry candidates can be clearly previewed but is not appropriate when the exploration of unknown areas is a key entertainment factor of the game (e.g. many role playing games). Additionally, in our deployed system, it was more likely for small maps to be shown in the IEC interface and we later observe that this may have influenced player preferences. When using IEC to generate game content, the interface should present a diverse range of candidates. This allows the player to naturally express their preferences without pressure from the system. In PCG: Angry Bots, diversity could be enforced by ensuring that no two map candidates in a single generation have the same number of rooms. However, as this observation was made after the user experiment, this constraint isn't enforced here.

4.5.5 Mutation Operators

Reproduction of the geometry trees is done purely through mutation. Recombination is not used because the fixed tree structure is not conducive to its use. This is because mixing the depth and sibling identifiers of two parents could result in offspring that is substantially different from either parent and using identifiers predominantly from one parent or the other could introduce favoritism to one parent that the player did not intend for. Instead, we restrict the player to selecting one geometry to be the parent of the next generation. This also aids in the player modeling process of Chapter 6 because the player will experience and rate every parent that they select.

In terms of the mutation, each offspring is first made to be an exact copy of the chosen parent. In addition, the branching rate of each offspring is randomly modified to be within the range of $[-0.5, +0.5]$ of that of the parent and the branching decay is similarly adjusted by $[-0.2, +0.2]$, both of which result in offspring of varying sizes.

Each node in an offspring is given a random probability to mutate. An inverse mutation rate similar to that used for patch-based terrains in Chapter 3 is first calculated for the tree using:

$$r^{inv} = 1 - \left[\frac{k}{k^{max}} \cdot (1 - r^{min})\right] \tag{4.1}$$

where r^{min} is the mutation rate parameter and is also the lowest mutation rate that can be used for the tree, k is the number of active nodes in the tree, and k^{max} is a parameter that specifies number of nodes at which r^{min} is used. The result is a mutation rate that is higher when there are fewer nodes and lower when there are more nodes, encouraging a better chance of mutation in small trees but protecting from rampant mutation in larger trees. This ensures that small map parents have a variety of offspring geometries while larger parents still produce offspring that have at least some hereditary commonalities. In this experiment, r^{min} was set to 0.5 and k^{max} was set to 10.

Once the inverse mutation rate is determined, the tree is stepped through from top to bottom, with each node being given a random chance to mutate. If a node is to mutate, one of three following genetic operators are used, each with an equal chance of occurring:

- *Addition*: A new randomly generated node is inserted between the current node and its parent node. Branching is then conducted on the new node.

- *Subtraction*: If the current node is on the direct path, then it and all of its branching child nodes are removed and the next node in the direct path is moved up one depth and connected to the current node's parent. If the current node is not on the direct path, then it and all of its child nodes are removed.

- *Permutation*: The current node's reference to a pre-designed room template is changed to a randomly selected new pre-designed room. Note that this may cause the number of doors available in the room to change, which affects the number of child nodes that the current node can have. In this case, randomly selected child nodes that are not on the direct path are removed until the maximum number of child nodes is achieved.

When the tree mutates, any existing door references are kept; any new child nodes that are added are randomly assigned a remaining door in their parent node and any subtracted nodes cause a door to free up, unless it was on the direct path.

The combination of the above operators and the fact that they have an equal chance to occur results in no noticeable signs of bloat or shrinkage in small and moderate sized maps. Without validation, most offspring populations have a sufficient number of maps that are smaller, larger, or the same size as the parent. However, with the validation process described in the next section, evolution favors smaller geometries as they are more likely to be valid.

4.5.6 Candidate Validation

After each offspring is generated, the map is validated to make sure that no rooms or corridors will overlap once the map is rendered for play. This is done by doing a simplified rendering process that only utilizes a handful of bounding boxes for each room and corridor. If an intersection is found among any of the bounding boxes then the validation has failed. The mutation and validation process is quick enough that if validation fails, then the offspring is discarded and mutation is attempted again. If the mutation process fails more than 100 times, then the offspring resorts to random generation. This rarely occurs but when it does, it's typically because the map parent is quite large and therefore the chance that an offspring will have intersection is higher. Note that for the same reason, a large map parent can result in having an offspring population of smaller maps because the subtraction genetic operator is less likely to introduce new intersection and therefore that operator is inadvertently favored.

Some of the room templates have slopped areas, which can lead to a map geometry with a variety of depths. This gives a visually appealing affect when the player is in a higher location of the map and can see other sections of the map below them. The slopped areas of the room templates also offer unique challenges, as the player cannot aim up or down and so must draw the AI controlled enemies out into flat ground to defeat them. However, one issue that has been observed is that when a higher area of the geometry overlaps a lower area, it can block the player's line of sight to the character when they are in the lower area due to the fixed distance camera that follows the character. This issue is not detected by the validation process. This issue could be resolved by implementing a more intelligent camera control mechanism, as detecting it during validation would create a higher calculation cost and restrict the size of the feasible solution space of geometries further.

4.6 Generalizing the Geometry Representation

The highest level abstraction of this geometry representation, as in the previous chapter, is the process of combining smaller geometry samples into a larger, more complete geometry. This idea can be used in many game genres and map styles such as the exterior and interior examples shown here, segments of road being connected together to form a track, buildings or city blocks being combined to create an entire cityscape [Greuter et al., 2003], or groups of platforms being combined to make a linear geometry in a platform game [Smith et al., 2011b].

The fixed n-ary tree itself though is a little more restricted in use. For example, it cannot be used to represent a track in a racing game. However, it can be used in most structured linear game maps. That is, games that have maps where there is one entry point and one exit point and where each node of the tree is self-contained, unlike the patch-based terrains of the previous chapter where patches freely flow together and the only boundary of the geometry is the outer limits of the heightmap.

Replacing the room templates with terrain patches that have high walls on some borders would allow terrains to be created in the tree structure. In this way, the edges of the tree could even be removed and nodes (patches of terrain) could instead overlap with each other. Alternatively, a large tree structure with many branching paths could be used as a maze, using either terrain or interior geometry samples.

Many top selling games, such as the *Half Life* series (Valve Corporation, 1998-2007) and the *Call of Duty* series (Activision, 2003-2012), have linear single-player experiences. These types of maps can be generated with the fixed n-ary tree representation by simply restricting the creation of branches, thus only having a direct path between the start and end of the map. However, the creation of graph-like geometries with circular paths in them would require substantial further investigation. For these types of maps, Dormans and Bakkes [2011] have proposed a more free-form graph representation that, rather than using pre-designed geometry samples, generates the structure of each room and corridor through generative grammars. The downside to this approach, however, is the limited visual quality of the geometry. Alternatively, Tutenel et al. [2011] use a combination of techniques to construct the interiors of individual buildings and their surrounding exteriors that exhibit natural floor plans and appropriate aesthetics.

4.7 Summary

- After the implementation of the evolutionary terrain tool (ETT) (see Chapter 3) and the experimentation on it (see Chapter 7), it was decided to conduct the remainder of the investigation through a single test bed game. This would unify the solutions of all three components of personalized procedural map generation and represent a complete, deployable solution.

- The test bed game is *PCG: Angry Bots*, a third person shooting game where the player must navigate from one side of a map to the other while defeating various enemies and collecting utility items.

- The geometry in *PCG: Angry Bots* provides an aesthetic backdrop to the gameplay. It also gives players opportunities to explore and to use basic strategies, such as taking cover behind static objects.

- To generate the geometry in *PCG: Angry Bots*, the concept behind the patch-based terrains was used, building interior map geometries by combining smaller sample geometries. In this case, room and corridor templates were manually crafted to ensure playability and visual quality. They were then used as building blocks, connecting them to one another to form a complete map.

- The room and corridor templates were combined in a *fixed n-ary* tree structure. A tree structure was decided on as linear geometries did not offer enough chances for exploration while a graph based structure introduced too many additional generative tasks to ensure connectivity and validity.

- The tree is described as 'fixed' because once the nodes of the tree have been labeled after random generation, the labels are only minimally altered by the mutation operators. This is important as the labels of the geometry are later used in generating the content layout of the map (see Chapter 5).

- Fitness evaluation is conducted via IEC. This IEC process is a more simplified one than that of the ETT, ensuring that all geometry candidates are playable and of a certain quality. Thus, the system can be used to capture a player's preferences over the size and shape of a map during gameplay with minimal intrusion to the player's experience.

- Mutation operators for this representation include adding a node and allowing branches to be created from it, removing a node and all its sub-trees that do not lead to the exit of the map, and permutating the room template reference that the node holds.

- After a geometry has been generated, it is validated by performing a basic rendering of the geometry and testing whether any segments of the geometry overlap with each other. As the generation and validation process are efficient, if a geometry is invalid, it is simply discarded and mutation or random generation is repeated.

- While the fixed n-ary tree representation has limited use in other games (depending on the structure of their maps), using smaller geometry samples to construct larger geometries is an idea that can be easily abstracted for use in many game genres.

Chapter 5

Neuroevolution of Content Layout

In the previous chapter we established an approach to procedurally generate the geometry of maps in the game *PCG: Angry Bots* and used interactive evolutionary computing (IEC) to utilize the player's preferences during the generation process. Now that the player has selected a geometry in the game cycle, that geometry must be populated with content to provide a more interactive experience for the player.

Content is defined here as the interactive elements of the map. Content can include:

- Artificial intelligent (AI) controlled friendly non-playable characters (NPCs) that the player can communicate with.
- AI controlled enemy NPCs that will attack the player and that can be destroyed by the player.
- Personal resources such as health and ammo in a first-person shooter (FPS) game.
- Constructive resources such as wood in a real-time strategy (RTS) game.
- Collectable items that add to the player's score or social status in a multiplayer game.
- Player bases or spawn points in a multiplayer game.
- The finish line in a racing game.
- Self-contained puzzles and mini-games.

These are just some examples of content within a game map but are by no means exhaustive; some games have unique forms of content that the player can interact with that do not appear

in other games. Some of these pieces of content can be procedurally generated themselves, while others need only be placed in a logical location within the geometry.

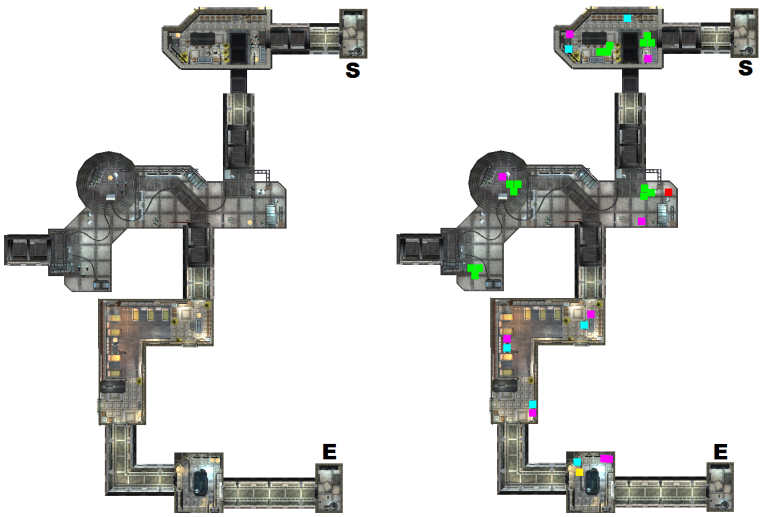
Not included in this definition of content are static objects such as architecture, furniture, doors, or clutter. Even if the player can interact with these objects (a player may be able to open a door, sit in a chair, or enter a building), they typically do not influence the player's experience in the same way as the other forms of content and instead usually add to the shape and visual appeal of a map. Therefore, static objects are considered as part of the geometry of the map, predominantly there for the player to navigate around, and thus would be included in the geometry representation in a procedural map generator.

The remainder of this chapter first discusses the effect that content has on the player's experience in *PCG: Angry Bots*. Then, in Section 5.2, details are given on the types of content in the game and how they are distributed throughout the geometry. Along with this is a discussion in Section 5.3 of why the geometry and content layout are not procedurally generated at the same time and are instead separated into individual search-based procedural content generation (SBPCG) processes. Our approach to calculating the layout of content throughout a map is then described in Section 5.4, along with the neuroevolution techniques used to generate-and-test content layout candidates. Finally, as with the interior geometry, this chapter concludes with a brief discussion regarding how this approach could be generalized to other games.

5.1 Role of Content in PCG: Angry Bots

As with geometry, it is important to consider the role of content in a game when designing a personalized procedural map generation solution. In *PCG: Angry Bots*, there are two categories of content; enemies and utility pick-ups. Enemies attack the player and must either be destroyed or escaped from in order to succeed. Pick-ups on the other hand are resources that help the player achieve their goal of getting from one side of the map to the other. As we will discuss more in Section 5.2, this content does not undergo PCG itself, rather the process of procedurally generating personalized maps here involves finding an optimum layout of these content types throughout the geometry. Figure 5.1 shows a map with just the geometry and then the same map with both the geometry and content layout, where each colored square represents a different type of content.

With these characteristics in mind, the layout of content within a map in *PCG: Angry Bots* is the primary determinant of the difficulty (or *challenge*) that the player will experience.



(a) Geometry only (b) Geometry and content

Figure 5.1: A map with four rooms between the start room (marked with an 'S') and the exit room (marked with an 'E'). The map is first shown with (a) just the geometry and then (b) with the content added to the geometry. Each colored square represents a single piece of content.

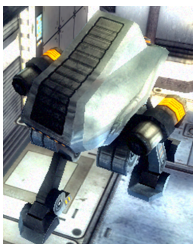
Increasing the number of enemies will increase the challenge of the map, while increasing the number of pick-ups should decrease the challenge. As discussed in the background knowledge in Chapter 2, providing the right level of challenge is a key element in determining the player's enjoyment in a game [Malone, 1981]. Thus, personalizing a map requires finding the level of challenge that is appropriate for an individual player.

However, while finding an appropriate level of challenge for the player is important, the use of a consistent challenge setting will also likely be uninteresting to a player. That is, if every room in a *PCG: Angry Bots* map had the same quantities of content, it may result in a repetitive experience for the player. The *pace* of a game is determined by the change of challenge over time [Adams, 2010], with patterns of high intensity and low intensity gameplay giving the player a variety of experiences and preventing them from becoming bored or frustrated [Zagal et al., 2008].

Therefore, we believe that when the content of a map plays a significant role in the challenge that a player experiences, such as in *PCG: Angry Bots*, then there is a direct correlation between the patterns of content throughout the map and the patterns of challenge. So, in summary, the goal of the content layout process of procedural map generation in *PCG: Angry Bots* is to not only create a map that has an appropriate level of challenge for the

(a) Buzz Bot

(b) Spider Bot



(c) Mech

(d) Pick-ups

Figure 5.2: (a), (b), and (c): The three enemy types in PCG: Angry Bots. *(d) The three pick-up types in* PCG: Angry Bots. *New weapons are in boxes (left of image), health is in blue barrels, and ammo is in yellow barrels.*

player's skills and preferences but also to create interesting and coherent patterns of content that will maintain the player's engagement throughout the map.

5.2 Elements of Content Layout

In this section we describe the types of content in *PCG: Angry Bots* and how they are procedurally laid out across the geometry of the map. Included in this is a description of the limitations in scope regarding the content of the map that were introduced to make the research goals of this thesis achievable within the available time.

5.2.1 Content Types

There are six types of content in the *PCG: Angry Bots*; three types of enemies and three types of pick-ups, all of which are shown in Figure 5.2. They are:

- Enemies: Buzz Bots, Spider Bots, and Mechs.
- Pick-ups: Ammo, Health, and New Weapons.

The three types of enemies each have different movement behaviors, damage output, and health values. Buzz Bots do little damage and are easy to kill but move quickly and attack in swarms. Spider Bots attack by walking close to the player and self-destructing to do moderate damage. Finally, the Mechs are the most dangerous enemy; while they are slow moving, they are difficult to kill and they shoot from a distance, doing a large amount of damage per shot.

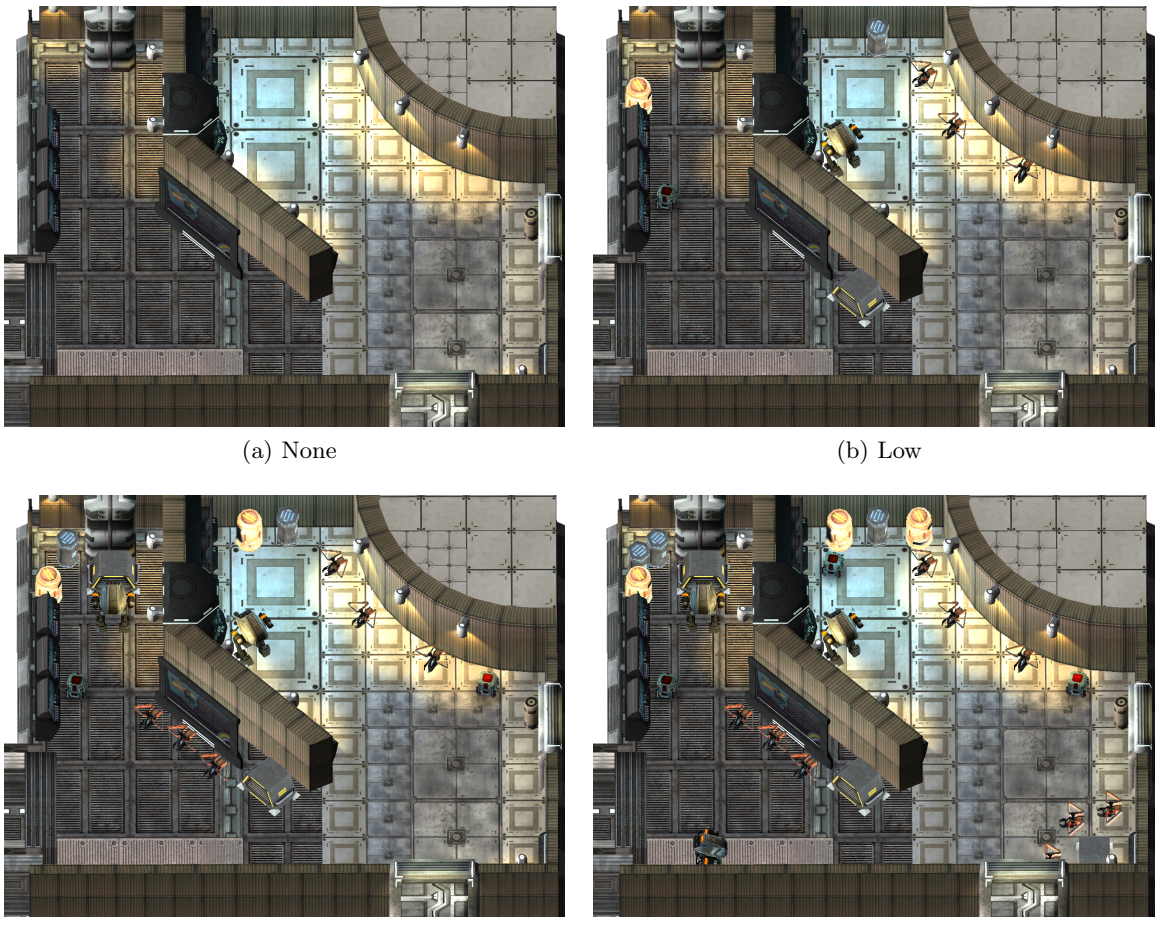
Pick-ups can be interacted with by standing next to them and pressing a designated keyboard key. Ammo and Health pick-ups give fixed amounts of their respective resource per barrel. Meanwhile, New Weapon pick-ups are randomly generated variants of the default weapon with altered damage per shot and rate of fire parameters. These weapons provide incentives for the player to explore the map as a randomly generated weapon could be substantially more powerful than the one they start the map with and therefore increase their chances of completing the map.

5.2.2 Content Settings

Procedurally generating the layout of content in a map in *PCG: Angry Bots* is conducted by calculating the amount of each content type in each room of the player-selected geometry. The manually constructed room templates utilized for the geometry construction also define the maximum quantity of each content type and the possible locations that each piece of content can be placed in.

Each content type in each room can have a discretized setting of *None*, *Low*, *Medium*, or *High*. Figure 5.3 shows an example of a single room template with each of the four settings assigned to all content types. If every room in the map has a setting of None for a content type, then there will be no instances of that content in the map. Conversely, if every room has a setting of High, then the maximum allowable quantity of that content type will be present. Note that content is only placed in rooms and not in the corridors that connect the rooms.

From Figure 5.3 it can also be seen that these settings also differ for each content type. For example, a High setting for the Mech enemy type will result in three Mechs in that room, while for the Buzz Bot enemy it will result in nine (three groups of three) being present. This is due to the behaviors of each content type. Nine Mech enemies will be too difficult even for the most experienced players while three Buzz Bots is too easy for a maximum setting. It should also be noted that if all enemy types and the Ammo pick-ups are set to High in every



(a) None (b) Low

(c) Medium (d) High

Figure 5.3: The same room template shown with each of the four content settings applied to all the content types.

room, then the player will not have enough ammo to combat all enemies. This was done on purpose in order to see whether the system could find an appropriate balance between ammo and enemies for an individual player, rather than simply maximizing all values.

5.2.3 Scope Reductions

As mentioned in the introduction to this thesis, the structure or the behavior of the content types themselves is never modified, other than the randomly generated rate of fire and damage per bullet stats of the New Weapons. This is because this task would fall under other PCG research, rather than that of procedural map generation. While the content is placed on the map and the layout of the content plays an important role in the quality of a map, modifying the behavior of the content itself would fundamentally alter the mechanics of the game, which would require its own process of optimization. Likewise, modifying the visual structure of the content may confuse players as to the purpose of each piece of content and thus unnecessarily increases the difficulty of the game. For examples of this type of research, Liapis et al. [2012] evolve the structure of space ships in games, giving the player more choice in how they represent themselves in the virtual world, while Stanley et al. [2006] make a game out of evolving different character behaviors by requiring the player to train an army of AI controlled agents.

The position of the content is also fixed as it is defined by the manually designed room templates. While this reduces the diversity of the possible maps, it simplifies the content layout problem to simply calculating how much content should be in each area of the map. It reduces the need for any checks for logical content layouts. For example, pick-ups of similar types should be spread out around a room to give player multiple opportunities to collect the pick-up before engaging enemies and to encourage the player to move around the entire room. It also means that there is no validation process for the content layout evolutionary cycle; every possible combination of content settings is valid. Otherwise, an algorithm for deciding the location of the content in each room on a fine grained coordinate scale would need to ensure that not only did content not overlap with each other or illegally intersect the geometry but also, for example, ensure that all enemies in a room are not crowded around a door, which would block the player's path and most likely make the map impossible to complete.

5.3 Combining vs. Separating Geometry and Content

As there is an existing link between the geometry and the content layout due to the position of the content being determined by the room templates, it would be possible to simply have each node of the geometry tree hold a setting value for each content type at that room. Then, the values could simply be mutated at the same time as the geometry as part of the geometry's permutation operator, as well as being affected by tree nodes being added and removed. In fact, this approach was implemented in an earlier version of *PCG: Angry Bots*, when a linear geometry structure was also being tested. As there are 10 geometry room templates, 6 content types, and each content type can have one of 4 possible settings, there are 40960 unique room templates. Thus, in this early implementation, each node simply contained an integer between $[0, 40960]$, which would be decoded during the genotype-to-phenotype conversion to determine the rooms geometry and content settings. With a combined linear genotype, a more straightforward (μ, λ) (multi-parent, multi-child) evolution strategy [Beyer, 1994] with both crossover and mutation was devised.

However, this approach has numerous flaws. Firstly, it ignores the different effects that the geometry and content layout have on the player's experience. The geometry provides the player with opportunities for exploration while the content layout primarily determines the challenge of the map. Thus, there are two separate objectives that are being optimized for with each only marginally affecting the other.

Of primary consideration is the use of a single fitness evaluation mechanism for both the geometry and the content layout. Not only are two separate objectives being evaluated but the means in which they can be appropriately evaluated do not overlap. As we discuss later in Section 5.4.5, using IEC for content layout would require increased effort on the player's behalf due to a lack of a suitable visualization. Meanwhile, the per-player preference model used for evaluating the content layout (described in Chapter 6) is built upon a classifier that would suffer in performance if features were added to evaluate the geometry.

Additionally, the mutation operators that would act upon the content layout in this setup would lead to a volatile representation. Mutating the single reference value by just a small amount could result in an entirely different room geometry and content layout. Also, if the content layout of a room has been optimized well, it could be ruined if the node is removed from the tree. Finally, there is no coherency between the rooms of the map. That is, there is no correlation between the content of one room and the content of the room that follows it,

and therefore, offers little towards optimizing the pace of the map and producing a pattern of experiences over random generation approaches.

Thus, we separate the representations, evolutionary cycles, and fitness evaluation of the geometry and content layout of map. We also recommend that this approach be considered when procedurally generating the maps of any games, as these two components of a map affect the player's experience differently in many games. Separating the two solves the issues described above and comes with additional benefits. It allows for either of the two to be optimized with only minimal effects on the other. For example, once a good content layout candidate has been found it can be applied to multiple geometries to give similar experiences. Conversely, if a superior geometry has been found, then that evolutionary cycle can be halted and the content continuously optimized for that geometry while the player model is being trained with more player data. This could be useful in existing games such as *Left 4 Dead* (Valve Corporation, 2008) in which there are only a dozen or so fixed geometries but the placement of enemies and pick-ups varies.

5.4 Neuroevolution

With the above framework and considerations in mind, the goal of procedurally generating the layout of content in a map is to calculate the content settings for each content type in each room of the geometry. We take this opportunity to explore *neuroevolution* as a potential solution. Neuroevolution is a subfield of the broader evolutionary algorithms (EA) research field which involves applying EA techniques to artificial neural networks (ANN) [Floreano et al., 2008]. This is in contrast to optimizing an ANN through other machine learning techniques such as backpropagation [Hecht-Nielsen, 1989]. In this section, we first describe the structure of the ANN used to indirectly represent the content layout. We then discuss the method used for evolving a population of content layout candidates, as well as the means by which the fitness of each candidate is evaluated.

5.4.1 Compositional Pattern-Producing Networks

The exact representation used to calculate the content settings of a map is a *Compositional Pattern-Producing Network* (CPPN) [Stanley, 2007]. A CPPN is a type of ANN that produces enhanced patterns with regularities in the output [Secretan et al., 2008]. The primary difference between a traditional ANN and a CPPN is that while an ANN typically assigns the same activation function to all nodes in the network, a CPPN can assign a separate function

to each node. While using a single activation function can produce a specific pattern in the output, the combination of multiple different activation functions in a single network can lead to more complex patterns that exhibit properties of all the used activation functions.

In this application, a library consisting of the sine, Gaussian, bipolar sigmoid, and linear activation functions was used. Each activation function has an equally likely chance of being selected for use at each node in the hidden layer of the CPPN. Thus, any favoritism towards a specific activation function is a result of evolution promoting the patterns that result from that functions use.

The versatility of the CPPN representation is visually presented in the *Picbreeder* application, created by Secretan et al. [2008]. Here, the complex patterns in the images, some of which result in abstract representations of real-world objects, are the result of CPPNs being evolved through IEC in which users can build upon the solutions found by past users.

The pattern-producing properties of CPPN have also previously been used in games to produce projectile paths for particle weapons in the game *Galactic Arms Race* [Hastings et al., 2009], aesthetic 3D models of spaceships [Liapis et al., 2012], and visually appealing plant petals in the social media game *Petalz* [Risi et al., 2012]. In *PCG: Angry Bots*, it is hoped that these properties produce a diverse range of patterns of content throughout a map.

5.4.2 Calculating Content from Geometry

Figure 5.4 shows the input and output structure of the CPPN network. The input to the network utilizes the depth and sibling coordinates of each node in the geometry that was selected by the player during IEC. Additionally, a bias value of 1.0 is used to further diversify the output of the activation functions.

Both the depth and sibling coordinates are normalized to the range of $[0.0, 1.0]$, as values greater than 1.0 typically produce little change in the output due to the asymptotic nature of the activation functions after this value. Depth is normalized by dividing by the maximum depth of the tree. The room after the start room will always have a depth of 0.0 and, in a linear map with no branches, the room before the exit will always have a depth of 1.0. The sibling coordinates are normalized by the total number of nodes at the given depth assuming a full 3-ary tree. Note that this means that increasing the height of the tree will also affect the normalized sibling coordinate of some nodes.

There are six outputs, one for each of the content types, that each provides a value between $[-1.0, 1.0]$. Each output value is then discretized into one of the earlier mentioned

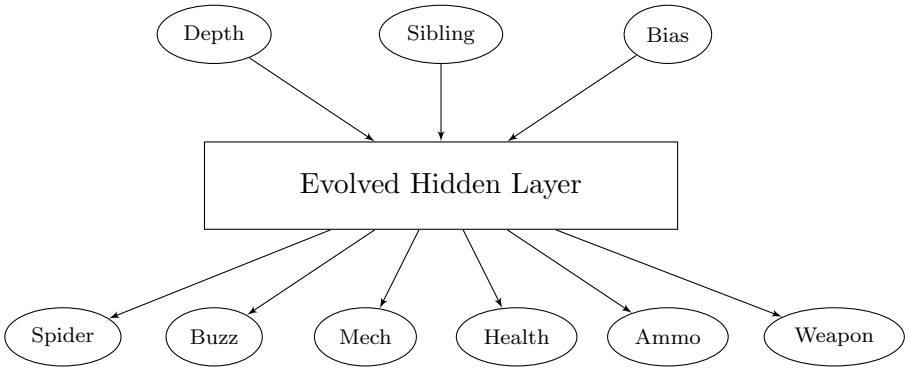


Figure 5.4: The input and output structure of the CPPN genetic representation used for calculating the quantity of each content type in each room of a map. From top to bottom are the input layer, hidden layer, and the output layer. The depth and sibling values are the coordinates of the current room (node) in the fixed n-ary tree geometry representation. The bias is always set to a value of 1.0*.*

settings. Thus, an output of -1.0 to -0.5 is a None setting, -0.49 to -0.0 is a 'Low' setting, 0.01 to 0.5 is a Medium setting, and 0.51 to 1.0 is a High setting. If there is no connection to an output node, then the output value will default to the Medium setting. Once this calculation has been performed for each geometry tree node (room) in the map, the content layout of the map has been determined and the map is ready to be rendered and played.

5.4.3 Neuroevolution of Augmenting Topologies

The hidden layer of the CPPN is optimized through Neuroevolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002]. The NEAT algorithm evolves ANN and CPPN populations by adding or removing nodes and connections, permutating the weight of a connection, or changing the activation function at a node. Recombination of two parent networks to produce an offspring is made possible by assigning a historical marker to each new node and connection and then using these markings to align crossover operators. Additionally, speciation, the process of grouping similar evolutionary candidates (both in nature and in EAs), is used in NEAT to aid in grouping candidates with similar historical markings to ensure compatibility between two parents and maximize the performance of recombination operators.

The NEAT algorithm was chosen to optimize the CPPN because it has previously been shown to be efficient in other real-time applications [Hastings et al., 2009], a property that is not present in backpropagation [Hecht-Nielsen, 1989]. Additionally, backpropagation would

require training examples of desired output with a given input. As the geometry is always changing and the node coordinates are normalized, it is difficult to provide enough training examples of different geometry structures. Moreover, as we are optimizing for each individual player, training examples generated from one player are not appropriate for other players. Taking this approach would be similar to the universal player models created by Shaker et al. [2010]. Instead, NEAT allows us to search the solution space of content layout candidates and evaluate them through a fitness function, which can be a player model of any form we desire. In this approach, the input to the generative model (the CPPN) is different to the feature inputs to the evaluative model (the player model).

An alternative to NEAT is to use conventional neuroevolution [Floreano et al., 2008] in which the structure of the CPPN does not evolve, rather it is only the weights of each connection within a fixed hidden layer topology that is altered. However, deciding upon a fixed topology can introduce limitations into the search process, while a free form topology allows for more possibilities. The process of network complexification in the NEAT algorithm begins with populations of small, simple CPPN candidates and allows them to grow if needed; that is, if evolution finds that the combination of more activation functions is better than only a few hidden layer nodes. We hope that this will allow for more complex patterns of output and therefore a more intricate patterns of content throughout a map.

5.4.4 CPPN-NEAT Setup and Parameters

Both the CPPN and NEAT implementations used in *PCG: Angry Bots* are part of the $C^\sharp$ distribution of NEAT, named *SharpNEAT*, created by Green [2004]. As mentioned earlier, once a player has selected a geometry through IEC, the content layout for that geometry is calculated and optimized through CPPN-NEAT. For each geometry, $10,000$ fitness evaluations are allowed to occur to attempt to find an optimal content layout. This fixed number provides an extensive number of candidates to be evaluated but ensures that the evolution is bounded so that the game continues to operate within real-time expectations of the player.

There are 50 candidates per generation, thus there are roughly 200 generations of evolution for each geometry. This is an estimate though because candidates are only evaluated once; if a candidate CPPN caries over from generation to the next due to elitism or simply having a better fitness than the offspring, then it is not re-evaluated.

Other CPPN-NEAT parameters include the use of the sine, Gaussian, bipolar sigmoid, and linear activation functions, each with a 0.25 probability of being chosen at each node. The

population is divided into 5 species, which was chosen due to its use in other example NEAT applications provided with the SharpNEAT framework. An absolute complexity regulation strategy was used with a complexity threshold of 50. This means that there can be at most 50 nodes and connections in the hidden layer of a CPPN candidate. This approach was chosen as it is less restrictive on complexification than a relative regulation strategy. Finally, the CPPNs are acyclic networks, meaning that all values are passed forward through the network and that connections are only ever from a node in one layer to a node in a later layer, never to the same layer or previous layer.

5.4.5 Fitness Evaluation of Content Layout

One of the primary reasons for separating the generative systems of geometry and content layout is due to the fitness evaluation of both aspects. With regards to the game type discussed here, it is expected that the only preferences that the player may have regarding geometry are the size of the map, the number of branching paths, and the types of room templates used in the map. Thus, IEC is capable of efficiently previewing the geometry of the map and capturing the player's preferences.

This is not the case with content, as there is no simple means of visualizing the content for IEC such that there is little effort required by the player. This is primarily due to the complexity of the interactions between various content types in this game. The player would need to carefully study each content layout preview in order to make an informed decision. Moreover, the player may not fully know their own preferences regarding the content layout. Requiring the player to analyze the content layout of similar maps may even require such effort that it would simply be better to require the player to manually decide the location of content themselves, which would not be in line with our research objectives.

Thus, in order for the procedural map generator to be as unimposing to the player as possible, we build per-player preference models, formulated as content-based recommender systems (RS). While the CPPN-NEAT approach to evolving content layout is capable of producing various pattern of content, the player model is a means of determining whether a pattern of content is appropriate for an individual player. These player models are trained with features that are extracted from the maps that are played along with subjective feedback from the player, the details of which are provided in Chapter 6. Then, during NEAT, the predicted probability that a candidate map will be liked by the player acts as the fitness for the CPPN that was used to generate it.

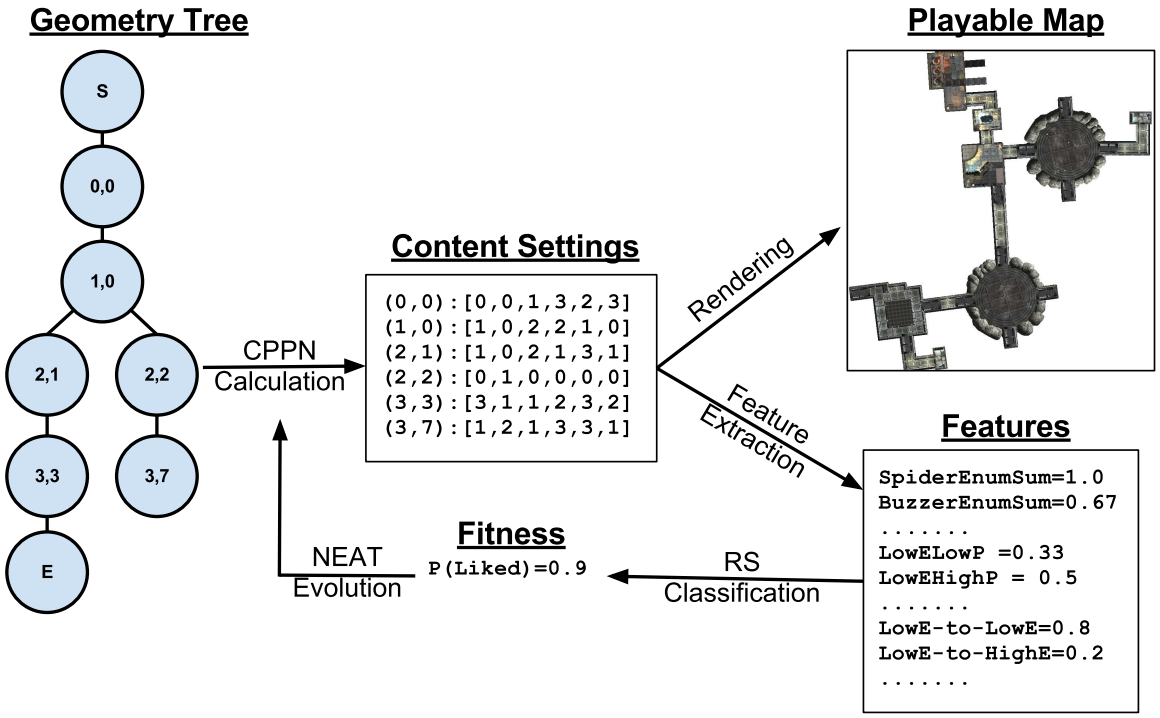


Figure 5.5: The process of calculating content layout with a CPPN and the NEAT evolutionary cycle.

The complete CPPN-NEAT evolutionary cycle is depicted in Figure 5.5. The coordinates of each node in the tree of the player-selected geometry are passed through a CPPN candidate to determine the content settings in each room. The map can then either be rendered for play or, if NEAT evolution is being conducted, undergo feature extraction. In the latter case, features are then used by a content-based RS to predict the probability that the map will be liked by that player. This probability value is used as the fitness of that CPPN candidate and the NEAT evolutionary algorithm continues.

The features that are extracted from played maps and candidate maps primarily describe the layout of content in the map. However, due to the features being extracted from a map that is a combination of the geometry and content, some features do implicitly describe some aspects of the geometry. Additionally, the calculation of the content is calculated from the coordinates of the geometry tree. Thus, the reason for utilizing a fixed n-ary tree for the geometry representation is to minimize the influence of changes in the geometry on the prediction of a player's preference on the layout of the content, while still maintaining a union between geometry and content that will allow for a logical flow of content throughout the map. Without the fixed n-ary tree geometry representation, every time a node (room) is added to or removed from the tree, the labeling of all other nodes at that depth and below are likely to change, especially if the node is on the far left of the tree.

5.5 Generalizing the Content Layout Representation

The CPPN-NEAT algorithm is already highly generalizable given carefully chosen input features and the translation of output values and previous research has applied CPPN-NEAT to other forms of game content [Hastings et al., 2009; Risi et al., 2012; Liapis et al., 2012]. However, in this section we specifically look at how the CPPN-NEAT approach to content layout described in this chapter can be generalized to work in other procedural map generation solutions.

The most general form of the CPPN representation above is to use the location on a geometry as the input to the neural network and to translate the output as some form of quantity for the amount of content at that location. In this case, the input was the location of nodes in the geometry tree representation. However, this could also be 3D or 2D coordinates of the virtual worlds Cartesian coordinate space or some other form of sequence tags placed throughout the geometry. It may also be possible to first determine the content layout and then procedurally generate the surrounding geometry, in a similar way to how Dormans

and Bakkes [2011] build spaces around mission objectives using grammars. It is also worth noting that all the CPPN inputs should be normalized between the range of $[0.0, 1.0]$ as the activation functions within the CPPN will give their full range of output with these values.

For the output of the CPPN, we recommend that each piece of content be given a separate output node of the network, as we have done here. Alternatively, one node could specify the type of content, while another specifies the quantity or difficulty setting of that type of content at the given input location. The output values will typically be between $[-1.0, 1.0]$, so they will need to be translated into something more useful. Here they were converted into discretized ordinal settings but they could also be converted to integers to specify an exact quantity.

Once the input and output representations have been decided on, the main design choice is then how to evaluate fitness. In the next chapter we go into the details of the RS style player preference models that are used to determine the fitness of CPPN candidates by extracting content layout features from a generated map and predicting whether the player will enjoy it or not. Other forms of player models can also be used to determine fitness, as well as metrics that are formulated based upon what is believed to be important to the player. Togelius et al. [2011b], Smith et al. [2011a], and Yannakakis and Togelius [2011] all provide discussions on the various approaches to the fitness evaluation of procedurally generated game content through both player models and researcher designed metrics.

5.6 Summary

- The content of a map encompasses most interactive elements of the map. In *PCG: Angry Bots*, there are six types of content; three kinds of enemies that attack the player and can be destroyed and three kinds of pick-ups that aid the player in getting from one side of the map to the other.

- In each room of a map, each content type can have a discretized setting of None, Low, Medium, or High. The location and quantity of content for each of these settings is defined by the room templates that are used for the geometry generation and thus every map is guaranteed to be valid, though it may not be playable to some players due to excessive difficulty.

- In *PCG: Angry Bots*, the layout of content throughout a map determines the challenge that the player will face and the pace of the map. Increasing the number of enemies

will increase the difficulty of the map, while increasing the number of pick-ups should reduce the difficulty. Increasing either of these in successive rooms will either increase the intensity (pace) of the game or reduce it. This is the same with many action games.

- The goal of procedural content layout is to determine the quantity and location of each content type throughout the map such that it provides an appropriate level of challenge for each individual player.

- The geometry and content layout evolutionary processes are separated because each of these elements has a different effect on the player's experience. Separating them allows for each to have a genetic representation that encapsulates their respective properties of an enjoyable map and for candidates of each to be evaluated on whether or not they provide an appropriate experience for the player.

- The content layout in *PCG: Angry Bots* is calculated through a Compositional Pattern-Producing Network (CPPN) [Stanley, 2007]. The node coordinates of each room in the geometry tree are passed as input to the CPPN, with six output nodes specifying the quantity setting of each of the six content types in that room.

- Neuroevolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] is used to evolve a population of CPPN candidates. Fitness evaluation is conducted by extracting content layout features from a map that is the result of combining a user-selected geometry and a candidate CPPN and using them to predict the probability that the map will be enjoyed by the player. This per-player preference model is described in the next chapter.

- The CPPN-NEAT approach to content layout described in this chapter can be used in the procedural generation of maps for other games by ensuring that the input represents some location within the map while the output determines the type of content as well as either its quantity or difficulty setting.

Chapter 6

Player Preference Modeling via Recommender Systems

In this chapter, we address the final component of a complete personalized procedural map generator; *player preference modeling* (PPM). Interactive evolutionary computing (IEC) was decided be powerful enough to capture a player's preferences regarding the geometry of maps in *PCG: Angry Bots* with little inconvenience to the player. However, the same is not true for content layout and so a more indirect approach to player modeling is utilized.

There are a wide variety of player modeling techniques that have been investigated for EDPCG with little commonality between them and thus there is no established best practice. On top of this, many of these techniques are substantially coupled with the domain (the game genre or the optimization algorithm). Unfortunately, this means that there is no existing solution that is appropriate to adopt and further investigate or even a generally accepted paradigm to work within, despite the taxonomies compiled by Yannakakis and Togelius [2011] and Smith et al. [2011a]. However, crafting a tightly tuned solution to suit the needs of only *PCG: Angry Bots* and the use of CPPN-NEAT is not ideal because while this would provide a solution to the current problem, the contribution would be lessened as the solution may never again be adopted by other researchers.

The generalizable framework that we are looking for actually lies within the well established research domain of recommender systems (RS). In fact, PPM can be thought of as an RS; the goal of an RS is to suggest items to a user that they may like and the goal of EDPCG is to provide players with game content that they will enjoy. The only difference in these definitions is that in a traditional RS items are only suggested and the final decision is left

to the user, while in many EDPCG solutions the content is provided to the player without any final input from them.

Moreover, many existing EDPCG solutions are actually complex, non-standard, domain dependent recommender systems. Though, the authors of these solutions either fail to notice this property or at least fail to report it. This is a significant gap in the literature and by bridging it the field of PPM can become more approachable. For example, at many educational institutions, RS is one of the first applications of machine learning that undergraduate students learn, as they are likely to interact with some form of an RS on a daily basis. By framing EDPCG as a process of recommending (rather than generating) appropriate content to a player, it makes the technique immediately accessible to novice researchers and developers and allows for easier integration of knowledge from the abundant RS field. Thus, in this chapter, a traditional and clearly defined content-based RS implementation is used as a PPM. The experimental results in Chapter 9 show that even techniques that lack the power of recent advancements in the RS field can produce good results in an EDPCG setting. As such, we predict that more advanced RS techniques would provide even better results.

The structure of the rest this chapter is as follows: Firstly, a more thorough description of the overlap between RS and PPM is provided along with comments on how to use RS in an EDPCG setting and the limitations of doing so. Secondly, Section 6.2 gives a review of existing literature that explicitly use RS as some form of a player model, whether it be in the fields of EDPCG or adaptive AI. This section is sparse as it was difficult to find existing research on this line of thought. Sections 6.3, 6.4, 6.5 then describe the implementation of the content-based RS that was used as a PPM in *PCG: Angry Bots* and explains why certain design decisions were made. Again, this chapter concludes in Section 6.6 with a discussion of how the ideas contained within this chapter can be generalized to other games.

6.1 Recommender Systems as Player Preference Models

In this section, the similarities and differences between RS and components of EDPCG are discussed. This is done by comparing the two fields in terms of their problem statements, the data structures used in them, and evaluation techniques used in them.

The goal of the EDPCG technique in this thesis is to maximize the player's enjoyment of a game through PCG; that is, a PPM is used to discover and utilize the player's preferences. This short summary and the topic title 'PPM' itself allude to the similarities between EDPCG and RS; the goal of an RS is also to maximize a user's enjoyment.

To be more precise, Adomavicius and Tuzhilin [2005] define the RS problem as such:

> Let U be the set of all users and let I be the set of all possible items that can be recommended, such as books, movies, or restaurants... Let f be a utility function that measures the usefulness of item i to user u... Then, for each user $u \in U$, we want to choose such item $i' \in I$ that maximizes the user's utility.

They also more formally communicate this as:

$$\forall u \in U, \quad i'_u = \arg \max_{i \in I} f(u, i) \tag{6.1}$$

The terms user u and item i in the above definition can be substituted with the terms player p and map m. With these substitutions in mind, the definition closely resembles the goal of this thesis: to find maps that maximizes a player's enjoyment. Here, the utility function is defined as the amount of enjoyment that map m provides to player p. Even more abstractly, the term 'map' can be generalized to 'content', and now the same statement above applies to the entire EDPCG field, with substitutions in the type of game content that is being optimized and the type of utility function. The utility function can be used to maximize either positive or negative experiences depending on the objectives of the researchers or game developers. For example, it may be desired to find a race track that maximizes challenge for the player, find a story that maximizes the player's curiosity, or even to find a set of rules that maximizes the player's frustration. We are using a PPM paradigm, so the aim of the utility function in this thesis is to maximize enjoyment.

6.1.1 Solution Space as an Item Set

There is, however, one issue that confounds the above definition overlap: an RS is traditionally used to recommend an item from a finite set of existing items, while in the SBPCG domain, the items (the game content) are 'generated'. However, it should be remembered that the EAs commonly used in SBPCG solutions are in fact search algorithms for sorting through all possible solutions and finding the optimal ones. The only 'generative' processes of SBPCG are those of assigning genotype values and converting the genotype to the phenotype in order for it to be utilized in game. Thus, it can instead be stated that all possible solutions already exist as a result of the chosen genotype and phenotype structures. Therefore, if each point in the solution space is instead thought of as an item in a finite item set, then RS

techniques can be directly applied and thus the terms 'solution space' and 'item set' can be used interchangeably.

6.1.2 Size of the Item Set

The primary difference between RS and EDPCG now becomes the size of the item set. In many traditional commercial RS applications, there are typically a tractable number of items. For example, if the items are movies, books, or attractions to recommend to the user, then the system will typically contain a database entry of every item with information about each, even if there are millions of items. The RS algorithm can be conducted on all items or, if need be, a subset of items decided through other heuristics. This is in contrast to many SBPCG solution spaces that can have a immense or even continuous number of solutions. However, this is similar to some RS applications, typically in the information retrieval field, such as those that attempt to recommend websites or documents from a rapidly expanding item set. It is too computationally expensive to evaluate all possible items with the RS and so only a relatively small subset of items must be chosen to be evaluated.

As a full example, let us calculate an approximate size of the item set of maps in *PCG: Angry Bots*. This calculation will be done on the phenotype of the maps. That is, total number of unique maps from the player's perspective. The size of the item set may be further constrained by the features of the RS, discussed in Section 6.5.2 of this chapter, but as these features are all on a continuous scale it is simpler to approximate the size of the item set from the discrete and observable phenotype.

Firstly, the total number of unique rooms (R) that can be generated from the combination of the 6 content types, each with 4 possible settings, can be calculated as:

$$\begin{aligned} R &= settings^{types} \\ &= 4^6 \\ &= 4096 \end{aligned} \tag{6.2}$$

This does not take into account the 10 room geometry templates that can be used because the RS is primarily concerned with optimizing the content layout, while the fixed n-ary tree genotype of the geometry simply acts as input in the CPPN calculations. How the geometry does affect this calculation though is in determining the number of rooms in a map. The

total number of unique sub-trees (T) that can be generated assuming a maximum geometry tree depth of 5 is:

$$\begin{aligned} T &= \sum_{d=2}^{D} \sum_{s=1}^{S_{d-1}} ns \\ &= \sum_{d=2}^{5} \sum_{s=1}^{S_{d-1}} 3s \\ &= 1290 \end{aligned} \tag{6.3}$$

where D is the maximum depth that is being measured to (with the root node at $d = 1$), S_d is the maximum number of siblings at depth d, and n is the term of the fixed n-ary tree used for the geometry representation. This calculation ignores the strict indexing of the fixed n-ary tree and sub-trees with the same relative position of nodes are counted as the one sub-tree. Note also that this equation starts at $d = 2$ because a map with just one room between the start and exit (the root node) is not valid. Here, $D = 5$ because it was rare for player's to experience maps with more than 5 rooms on the direct path between the start and the exit room in the experiment of Chapter 9. Finally, the size of the item set I (number of unique maps) is equal to:

$$\begin{aligned} I &= \sum_{t=1}^{T} R^t \\ &= \sum_{t=1}^{1290} 4096^t \end{aligned} \tag{6.4}$$

and by taking the greatest factor this is approximately:

$$\begin{aligned} I &\approx 4096^{1290} \\ &\approx 8.79 \times 10^{4659} \end{aligned} \tag{6.5}$$

However, this calculation assumes that a map with 1290 rooms has been generated, which is not only impractical for a player to experience but also essentially impossible as the curving nature of the rooms and corridors leads to invalid geometries with an increasing probability

proportional to the number of rooms in the map. The above equation for T would need to assume that the geometry was rendered to appear as its tree genotype, in which case there would be no invalid maps. However, in this case, there is no sense in setting a maximum depth and therefore the item set is still discrete but it is now infinite. Instead, in the *PCG: Angry Bots* experiment discussed in Chapter 9, the maximum number of rooms in a map that was played was 10. Therefore, a much more realistic number of rooms in a map for the average player are 4. However, even in this case the size of the item set is $4096^4 = 2.81 \times 10^{14}$, which is still larger than most RS applications.

This is where the common practices of the EDPCG field can benefit the RS field. In EDPCG, an SBPCG algorithm, usually based upon an EA, is used to generate content which is then evaluated against some knowledge of the player's psychology (typically via a player model). Using an RS as a player model now becomes a logical and sensible choice. The EA is, essentially, acting as a filter to the RS; searching the item set intelligently in order to reduce the number of items that are required to be evaluated by the RS. Conversely, and more directly, the RS is acting as the fitness evaluator to the EA. Note that while the earlier comparison between an EA solution space and an RS item set only holds true for finite discrete solution spaces, the use of an EA to minimize the number of items to be evaluated by an RS can still be applied in theoretically continuous and infinite solution spaces.

A final point to note is that in both an adaptive game and a typical RS application, the number of users (or players) can vary wildly depending on the popularity of the system (or game). However, as discussed further in Section 6.4, because we use a content-based recommender system and each user is considered in isolation, the size of the user set is irrelevant. However, for a collaborative filter, it would be important to consider how the size of the user set would impact the performance of RS, an issue that is quite prominent in the collaborative filter research field [Su and Khoshgoftaar, 2009].

6.2 Recommender Systems and Adaptive Games

The overlap between the techniques used in the RS field and those used in player modeling are often overlooked. However, Medler [2011] makes a qualitative comparison between RS and adaptive gameplay to highlight these similarities. Medler gives examples of research involving player modeling that can be likened to content-based RS techniques, as well as commercially active matchmaking and dynamic difficulty adjustment (DDA) systems built upon principles that are similar to those of collaborative filtering (CF).

Medler states that the primary differences between RS and adaptive games is that adaptive games are used to challenge the player, rely on implicitly gathered player data, and have real-time processing constraints and large item sets. The only one of these differences that we believe to be valid is the last, regarding time constraints and data sizes, which was discussed in the previous section. While challenge is often stated as a primary factor that influences a player's satisfaction of a game [Malone, 1981; Byrne, 2005], it's only one element of a player's experience. We argue that the goal of any game, adaptive or not, is to maximize the player's enjoyment, which can be achieved through varied approaches; challenge balancing being one of them. As for the second point, implicit gameplay data is not the only form of player data that a player model can be built upon, which is discussed further in the next section.

Medler also identifies potential future work towards applying RS to adaptive games. While the author's discussion of memory-based versus model-based RS appears to be a little confused, the categorization of potential solutions into ranking content, profile matching, and action tracking provides strong guidelines for implementation of RS in adaptive gameplay. These categories also align closely with the player data types discussed in the next section. Under this categorization, the system used in *PCG: Angry Bots* is similar to a content ranking approach, though the decision for this was made independently of Medler's analysis, as the publication was discovered after the implementation of the player modeling approach in *PCG: Angry Bots* had begun.

Similar to Medler's analysis, Riedl [2010] draws upon the analogy between RS and personalized experiences in games. Riedl makes a distinction between recommending an experience (which is typical in RS) and tailoring an experience (which is typical of EDPCG). This builds upon the notion that EDPCG and other forms of adaptive gameplay change an experience to create a new one that is more suitable to the current player. However, as was shown in the previous section, the set of all possible changes to an experience can be viewed as the item set in a recommendation process, with each experience being its own item. Therefore, we argue that there is in fact no difference between Riedl's definitions of recommending experiences and tailoring experiences.

6.2.1 Examples of Recommender Systems in Games

Despite the similarities of RS and EDPCG, the relationship between the two fields is typically implicit. To the best of our knowledge, the following references are the only examples in which

an RS was explicitly stated as being used to model a player. This shows the novelty of using an RS framework in an EDPCG application.

Berkovsky et al. [2010] use a CF to compute the time limit of a game map for individual players. This is achieved by taking the average completion time of other players that have had a similar performance in previous maps. That is, user-to-user similarity is computed by comparing the completion times of the current player so far to those of all other players for the same maps and finding a subset of similar players. The predicted completion time for the next map is then the average of the completion times of the similar players on that map. Note that this is not an EDPCG application and so all players experience the same sequence of maps.

Zook and Riedl [2012] use a tensor factorization technique, which has recently gained popularity in the CF [Rendle et al., 2009] research field, to model a player's change in skill over time. This model is used to detect how the player performs in skill based challenges and, when combined with a time factor, predict how the player will perform on future challenges. From this, the individual challenge events can be tailored to meet the player's predicted skill. While tensor factorization is a popular CF tool, this implementation is actually a content-based RS because the skill of other player's is not taken into account.

Yu and Riedl [2012] use a prefix based CF to select the sequence of plot-points in a game's narrative. This system requires players to rate the story so far and then correlates their ratings with those of other players to decide on future plot-points. In this system there are a discrete number of handcrafted plot-points that can be linked to one another in specific orders, all of which is represented by an authoring data structure known as prefix graph. RS is especially effective in this environment because if each plot-point is thought of as an item in the RS, then at the current plot-point there are only a dozen or so possible items that the RS needs to make predictions on. Additionally, this setup will most likely lead to a dense user-item matrix, which is favorable to using CF.

While not explicitly stating the use of RS, Medler [2011] points out that the EDPCG application created by Togelius et al. [2007] can be easily considered as using an RS solution. Togelius et al. optimize tracks in a racing game through the use of an EA. Here, an AI agent is trained to mimic a player's driving style, as observed over a few sample tracks that are experienced by the player. This agent is then used to evaluate tracks in an EA, with fitness values derived from the agent's performance on each candidate track.

This can be likened to a content-based RS in which the player's enjoyment of a track is being predicted by the agent. However, the agent is not making direct predictions on the

player's rating. Instead, objective data is collected from the agent's performance (as a proxy to the player's actual performance) and evaluated by three fitness functions. Unfortunately, the three fitness functions are designed by the authors and so impose what they believe to be a good track upon the player, rather than allowing the player's own unique reasons for enjoyment of a track to influence the recommendations. This is something that our implementation seeks to overcome, as discussed in the Sections 6.3.2 and 6.5.2.

Similarly, the optimization of map parameters in *Super Mario Bros.* (Nintendo, 1985) by Shaker et al. [2010] can also be thought of as an RS. Here, artificial neural networks (ANN) are used to model different emotional responses given certain features of a map and the player's performance on recent maps. The feature values are altered such that when combined with the player's performance on the most recent map as input to the ANN, the output of the ANN is maximized, indicating a maximal effect on that emotional state.

This again is similar to a content-based RS. Notice though that it is the map features (or parameters) that are being recommended rather than the actual maps themselves. This is because of the chosen features, which allow many maps to appear different to the player but to have the same feature values. The features define the item set and are constrained enough such that an exhaustive search of the item set can be conducted. That is, if this were an RS application, every item could be evaluated by the RS and the most appropriate item recommended to the user. These types of analogies between RS and EDPCG applications can continue to be drawn for any EDPCG solution that involves a player model assigning a rating or fitness to content in the player's stead.

6.3 Player Data

A player model is a representation of what is known about the player. More specifically, here it represents the player's preferences for map. As human beings, we learn from examples in our everyday life; that is we learn from surrounding data. Most machine learning algorithms used for player modeling function in a similar manner in that there must be data about the player to learn from and to make predictions on. The question answered in this section is what forms of data are available during the player modeling process and which one is most applicable to the current context.

6.3.1 Types of Player Data

Yannakakis and Togelius [2011] define three approaches to player modeling within the ED-PCG paradigm, each with their own form of data: subjective, objective, and gameplay. *Subjective* player modeling approaches rely on the player to self-report their experiences. They may be asked to rate their experience, compare multiple experiences, or even verbally discuss their experience. The benefit of this approach is that there is minimal ambiguity or misinterpretation of the player's experience. However, the downside is that this method is usually somewhat intrusive of normal gameplay; requiring the player to stop what they are doing and respond to a direct question. There may also be issues of inconsistent reports due to a change in the player's mental state or the ambiguity of their own memories.

Objective player modeling involves monitoring a player's physical condition in order to detect a change in emotion or mental activity. This has been conducted in the past via monitoring heart rate [Drachen et al., 2010], brain activity [Chanel et al., 2008], skin conductivity [Rani et al., 2005], and respiration [Tognetti et al., 2010]. This removes the potential for distorted subjective player responses. However, if the physiological responses of the player are misinterpreted by the experiment designer, it can provide misleading results. For example, a fast heart rate could indicate excitement or rage. Additionally, these types of systems can be financially prohibitive as they often require additional peripheral devices to monitor the player's body.

Finally, *gameplay* player modeling involves collecting data about a player's actions in the game. Like objective player modeling, it involves monitoring the player and removing any form of subjectivity on the player's behalf. The data collected may include how far the player has traveled, what items they have picked up [Hastings et al., 2009], how many shots have they fired, how long the player has been playing for [Cardamone et al., 2011b], or even the player's exact path through a map. Any player action can be recorded in exact detail and used as a data source for a player model. The additional benefit of this is that the player themselves can be replaced by an AI agent and the same data can still be collected [Togelius et al., 2007; Cardamone et al., 2011b]. However, gameplay modeling suffers the same shortcomings as objective player modeling in that it is up to the researcher or game developer to interpret how the collected data reflects the player's preferences. This makes it more likely that the designer's own preferences and thoughts of what is important to a player will influence the modeling process.

6.3.2 Player Data in PCG: Angry Bots

While it may be expected that the choice of player data is limited by the use of an RS, the truth is actually the opposite. This is, in fact, one area that RS and EDPCG overlap significantly. While objective data sources are rare in RS applications, subjective and gameplay (or more generally *user-activity*) data sources are the main forms of data used in RS applications. Many commercial RS applications require users to rate items that they have experience with on a five star rating systems [Bennett and Lanning, 2007] or provide information about their own preferences and demographic upon joining the system [Rich, 1979], which are both forms of subjective data sources. Meanwhile, researchers in the information retrieval field often use user-activity data sources, such as statistics on which documents the users view, in RS applications [Billsus and Pazzani, 2000] to provide a more personalized online experiences without any additional requirements from the user.

Though a case can be made for all the data sources mentioned, we chose to use a subjective data source for the implementation in this thesis. Objective data was never an option here as the aim of this solution is to make PPM more approachable to game developers. Few amongst the game player market would be willing to purchase a new peripheral for a single game and so objective player modeling would represent a financial loss to the developer. Gameplay data sources, though, are a strong choice and have been utilized in prior EDPCG research with good results [Togelius et al., 2007; Hastings et al., 2009].

However, in the end, a subjective rating system was chosen for use. After the player experiences each map, they are asked to provide a single rating of the maps. This rating is on an ordinal scale and is in response to the question 'How would you rate the map that you just played?' The details of the ordinal rating scale can be found in Section 6.5.3 below. This type of subjective approach was primarily chosen because of the use of similar rating systems in many commercially active RS applications [Linden et al., 2003; John, 2006; Bennett and Lanning, 2007]. In fact, Adomavicius and Tuzhilin [2005] define the RS problem as one of estimating user ratings to items in a system, given previous ratings. By keeping the player model representation coherent with a significant portion of the RS literature, we strive to demonstrate the applicability of existing RS techniques to EDPCG and to make the transition to more advanced RS techniques in future work more direct.

Shaker et al. [2010] use a similar subjective approach in predicting a player's emotional state during gameplay, though it is not described as an RS application. However, while Shaker et al. ask the player to respond to multiple questions about various emotions, we

only ask one, rather ambiguous question about enjoyment. The reason for this is that, while current game theory states that challenge, fantasy, and curiosity are important elements of a player's experience [Malone, 1981], the reality is that there may be many more factors that influence a player's experience, at much finer granularities, and to varying degrees for different players. This makes it hard to generalize what exactly contributes to a positive experience for all possible players. The player themselves may also not recognize what influences their enjoyment, let alone a third party researcher or game developer attempting to interpret the player's actions or responses. The ambiguous question of enjoyment instead only requires the player to respond on an instinctual level. This allows each player to have their own internal process for decision making, whether the player is conscious of it or not, which does not require any interpretation in order for it to contribute to an EDPCG system.

As an aside, this setup also leads to indirect IEC. In Chapters 3 and 4, IEC was conducted by requiring the user to directly specify which map geometry they preferred and the EA used the map as the parent for the next generation. Using an RS paradigm with a subjective data source allows for IEC to be conducted in a less intrusive manner. Explicit feedback is still required from the player but it is in a manner that requires less effort than traditional IEC, especially when the potential candidates cannot be clearly visualized to the player, which is the case with content layout in *PCG: Angry Bots*. The RS uses the feedback from the player to evaluate the content in the player's stead, similar to how Togelius et al. [2007] use trained agents to experience and evaluate race tracks in the player's stead.

An additional difference between our work and that of Shaker et al. [2010] is that while both use post-play self-reporting techniques, we ask the player to rate each map independently after they have experienced it, while Shaker et al. require players to rank two maps against each other in order to establish preferences. Yannakakis and Hallam [2011] provide a comparative analysis of post-play reporting techniques for player modeling and found that a pairwise preference approach introduces less error, especially regarding the order that the player experiences the content. The results presented later in Chapter 9 show that there is indeed an issue associated with the order that maps are experienced, especially as a player's preferences and skill change rapidly in the first few maps that are played. However, it was found that while the classifier may take a sudden penalty from contradicting training data, this is typically quickly negated by further training data that aligns with the new preferences. This also addresses the requirement put forth by Medler [2011] of the need of adaptive games to accommodate an improvement in a player's skill. Additionally, we believe that this approach is less intrusive to gameplay, more compatible with a user's previous experiences with

commercial RS, allows for the classifier to be updated after every map, and only requires the player to recall their experience on the immediately prior map.

Linn and Gronlund [2000] also mention that there may be errors in user surveys related to the misinterpretation of an ordinal rating scale. This is not a significant issue here as it is expected that each player will have their own interpretation. However, this is one reason labeled ordinal ratings were used rather than a 5-star system. As the ratings are symmetrical around an unavailable 'Neutral' rating, it is possible to clearly identify positive and negative experiences. This may introduce problems in a CF though, where the understanding and internal rating guidelines of one player may affect another. However, this issue has not prevented the CF approach from becoming popular in commercial systems with large quantities of users [Linden et al., 2003].

6.4 No Co-ratings

With the source of player data decided, the actual RS implementation can begin to be discussed. In Chapter 2, definitions for the two main RS approaches were given; collaborative filters (CFs) [Su and Khoshgoftaar, 2009] and content-based RS [Adomavicius and Tuzhilin, 2005]. Before discussing the implementation used in *PCG: Angry Bots*, this section describes which of the above approaches was ruled out for now and why. Descriptions of both of these techniques can be found in the RS background knowledge of Chapter 2.

In the first section of this chapter it was stated that the item set can be much larger in an SBPCG application than in a typical RS application. However, the number of users of the system is likely to be roughly equivalent or even much less, depending on the popularity of the game that this technique is used in. Therefore, the logical choice would at first seem to be to use a CF. A content-based RS would need to evaluate a potentially vast number of items in a typical SBPCG application, finding items from the item set that are similar to ones that the active user has enjoyed in the past. Here, the *active user* is the user that the RS is currently evaluating and discovering recommendations for. A comprehensive content-based RS solution would evaluate every item in the item set for every user and therefore recommend the absolute best item in the system (based on the current training data). However, this process is clearly computationally infeasible here. Meanwhile, a CF would compare the known preferences of a finite and computationally feasible number of users in order to find those that have similar interests to the active user.

	i_1	i_2	...	i_M
u_1	2	3	...	
u_2	4		...	1
...	...	...	...	...
u_N	1	5	...	4

(a)

	i_1	i_2	...	i_M	$i_{M+?}$
u_1	2		...		$\rightarrow$
u_2			...		$\rightarrow$
...	...	...	...	...	$\rightarrow$
u_N			...	4	$\rightarrow$

(b)

Figure 6.1: User-item matrices used in recommender systems. (a) A tractable user-item matrix with N users and M items and many co-ratings. (b) A sparse user-item matrix with N users and theoretically continuous item set and few co-rated items.

However, it must be remembered that CFs operate on *co-rated* items. A co-rated item is an item that has been experienced by multiple users. In a movie RS for example, it is expected that many users have seen and rated each movie. Thus, in finding similar users, the CF looks at which movies the user has liked in the past and finds users that have also liked the same movies. The user with the most co-rated items with similar ratings is expected to have the most similar preferences to the active user. Thus, in a traditional CF, a movie from that user's previously well rated movies that has not yet been rated by the active user is recommended.

The problem with this is that as the number of items increases, the probability of two or more users rating the same item decreases. Figure 6.1 visually demonstrates the difference between a typical RS user-item matrix and a sparse one. In both user-item matrices, a value in the matrix indicates that user u_n has rated item i_m with the given value. A lack of a value indicates that the user has not experienced or rated the item. The first user-item matrix shows a known total of N users and M items. This may, for example, be a snap shot of a movie RS where at any given time there is a known number of movies in the database. The second user-item matrix still has N users but has an unknown quantity of items greater than M. That is, the item set is excessively large, difficult to calculate, or completely continuous and therefore has no practical upper limit.

As the number of users has not changed and the number of items that each user is expected to experience has not changed in the second user-item matrix of Figure 6.1 from the first, there are very few items that have been rated by more than one user. That is, co-rated items are exceptionally rare. Therefore, in an SBPCG application such as the one in this thesis, the chance of two player's experiencing the same map out of the large number

of possible maps in *PCG: Angry Bots* (as discussed earlier in Section 6.1.2) is very low and so there are no co-ratings for a CF to operate on.

6.5 Content-based Recommender System for PCG: Angry Bots

With the considerations above in mind, we test the belief that player preference modeling is analogous to recommending appropriate content to the player by constructing player models using a traditional RS technique. If this can be shown to work reasonably well, then it is expected that more advanced techniques from the RS field could be adapted to be functional in an EDPCG setting. In the *PCG: Angry Bots* test bed, a *content-based recommender system* [Adomavicius and Tuzhilin, 2005] approach is used as a per-player preference model. Instead of drawing similarities between users as CF does, this approach calculates similarities between the items that a user has already rated and those that they have not yet experienced. That is, every user is considered in isolation; the recommendations that are made to them are based on their own past transactions and not influenced in any way by the actions of other users.

More specifically, in *PCG: Angry Bots* a model-based approach is used in which items (maps) that have previously been rated by a user (player) are used as training data for a classifier. This classifier is then used to predict the likelihood that the player will like a map candidate, which is then used as the candidate's fitness score in the CPPN-NEAT content layout algorithm described in the previous chapter.

A memory-based approach, while just as effective as a model-based approach, would have had a performance decrease in this setting as every map resulting from a CPPN candidate would need to be compared to every previously rated map of that player, rather than conducting a single calculation of a classifier. In contrast, the retraining of the classifier after a player rates a new map is a quick process because the number of maps likely to be played by each player has an inconsequential effect on training time. For a further discussion of the difference between a model-based and memory-based approach, please see Section 2.4.4 of the Background chapter.

6.5.1 Naive Bayes Classifier

The classifier that was chosen for use was a Naive Bayes (NB) classifier [Duda et al., 1973]. This was due to its reported ability to learn with very few training examples, its simplicity,

and its success in other RS applications [Pazzani and Billsus, 1997]. The probability that a map will belong to a rating class is:

$$P\left(C_i \mid f_{1,m}\&...\&f_{n,m}\right) \tag{6.6}$$

That is, the probability that map m belongs to class C_i given the features of the map $(f_1, f_2, ..., f_n)$. This can be reformulated using the Bayes' theorem as :

$$\begin{aligned} posterior &= \frac{prior \times likelihood}{evidence} \\ P\left(C_i \mid f_{1,m}\&...\&f_{n,m}\right) &= \frac{P(C_i)\,P(f_{1,m}\&...\&f_{n,m} \mid C_i)}{P(f_{1,m}\&...\&f_{n,m})} \end{aligned} \tag{6.7}$$

where the posterior is the predicted membership of the map in class C_i. More simply, the prior and likelihood can be calculated as:

$$\begin{aligned} P(C_i) &= \frac{\#\,C_i\,instances}{\#\,instances} \\ P(f_{1,m}\&...\&f_{n,m} \mid C_i) &= \frac{\#\,C_i\,instances\,with\,features\,(f_{1,m}\&...\&f_{n,m})}{\#\,C_i\,instances} \end{aligned} \tag{6.8}$$

An instance of C_i is a training sample (a previously rated map) that belongs to the rating class C_i. The denominator of Equation 6.7, the evidence, can be ignored as it is not dependent on C_i. Additionally, if it is assumed that the features are independent of each other than the probability of each feature given the class can be evaluated separately. Thus, the entire process can be rewritten compactly as:

$$P\left(C_i\right) \prod_{x=1}^{n} P\left(f_{x,m} \mid C_i\right) \tag{6.9}$$

As the likelihood each feature $f_{x,m}$ is now calculated independently, the probability of having training samples with at least one feature that matches the current map that is being classified increases. This logically reduces the number of training samples needed to produce sensible predictions, which is what gives the NB classifier its characteristically strong performance with smaller training sets. Also note that the above equations work directly for discrete

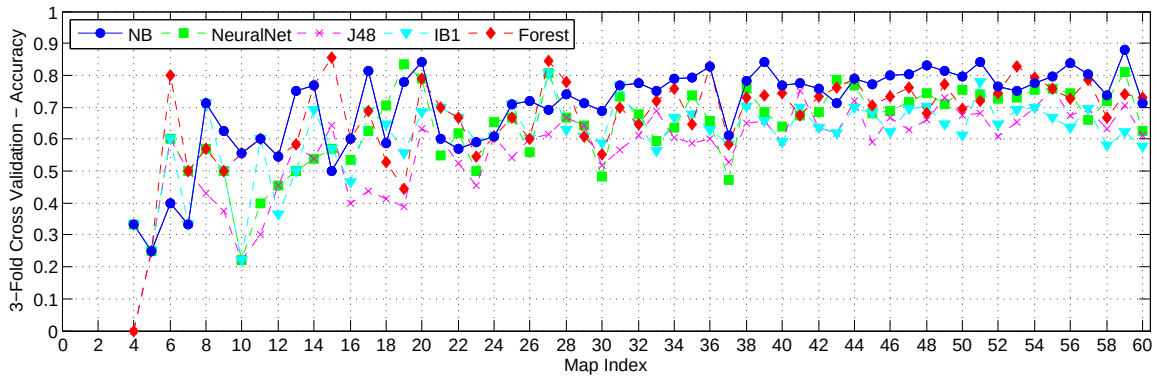


Figure 6.2: Classifier performance on an example data set of 60 *map ratings.*

features. However, the map features described in the next subsection are all theoretically continuous. Thus, a kernel estimation function is used in calculating the likelihood [John and Langley, 1995], with a separate precision for each feature that is calculated from the average difference of that feature value between adjacent training instances.

In choosing NB as the classifier to be used in the experiments in this thesis, initial tests were conducted on five sample classifiers from the WEKA machine learning suite [Hall et al., 2009]. The classifiers were evaluated with 3-fold cross validation on an example training data set of 60 randomly generated maps that were rated by our research team. The details of the four alternative classifiers and the 3-fold cross validation test can be found in Chapter 8. The features and classes used by the classifiers are described in the next subsection. Figure 6.2 shows the results of this initial test, which plots the 3-fold cross validation prediction accuracy against an increasing data set size. For most data set sizes NB is performing better than most of the other classifiers. While it has been reported that many other classifiers will outperform the NB classifier with larger volumes of training data, it is unlikely that a player of *PCG: Angry Bots* will ever play long enough to generate that level of data. The NB classifier also has the additional benefit of being one of the quickest models to both train and evaluate with.

6.5.2 Map Features

In a content-based RS, it is the features of the items that are compared with one another. Thus, once a geometry tree and CPPN candidate have been used to calculate the content layout of a map candidate, the following 18 features are extracted:

- Enumerated Sums - The settings {None, Low, Medium, High} are enumerated as $\{0, 1, 2, 3\}$ and a sum of each of the 6 content types across the entire map is calculated.

- Room Composition Counts - The six content types are condensed to two categories, Enemies (E) and Pick-ups (P), and the four settings are condensed into Low and High. This creates 4 possible room types: LowE-LowP, LowE-HighP, HighE-LowP, and HighE-HighE. The quantity of each room composition is counted throughout the map.

- Room Transition Counts - As with the Room Compositions, content types and settings are reduced. However, these features determine the transitions from one room to another. There are 4 enemy transition types: LowE-to-LowE, LowE-to-HighE, HighE-to-LowE, and HighE-to-HighE. There are also 4 similar transition types for pick-ups. Each edge of the geometry tree will belong to one enemy transition type and one pick-up transition type.

All the above feature values are normalized by dividing by the total number of rooms in the map. For the enumerated sum features, this gives an average value for each content type in the map. For the room composition and transition counts, it gives the proportion of rooms in the map that belong to each composition or transition type.

The independent feature assumption that defines the NB classifier holds true for some of the features used. For example, the value of one enumerated sum feature does not affect the value of the others. While this does not necessarily hold true for features such as room composition counts that sum to unity, assuming that these features are independent does not appear to negatively impact the performance of the classifier as shown in Figure 6.2. This behavior has also been witnessed and examined in depth by Domingos and Pazzani [1996].

These features were determined through expert knowledge and without the rigorous feature evaluation conducted in previous procedural map generation studies [Pedersen et al., 2009]. This is because we wanted to evaluate the system's performance in less-than-ideal circumstances, which would allow the system to be quickly adapted for other games given expert knowledge in that game genre. Additionally, as a practical consideration of the research project, rigorous feature selection that is supported by statistical evidence would only be possible after carrying out a user study and we did not have enough resources to conduct two major user studies. However, if the system can be shown to perform well under these

conditions then it is likely that the performance would increase with a more thorough feature selection process.

Also note that these features do not necessarily represent what aspects of the game that designer believes to be influencing the player's experience, which would counter the argument put forth for the use of a subjective player data source. The only assumption being made is that it is the content layout, more so than the geometry, that is influencing enjoyment of a map. The features are instead constructed as a means of simplifying the description of the dynamic elements of the map (the content layout) to as small of a representation as possible without losing information about how one map differs from another.

6.5.3 Weighted-Binary Classes

In a content-based RS, it is often required that the users of the system rate items that they have interacted with to indicate whether they liked them or not. In a model-based approach, these ratings then become the classes for the classifier. For example, a rating system that allows a user to select 'Like' or 'Dislike' for each item will then result in a binary class classifier. Meanwhile, other systems may allow the user to rate items between one and five stars, which would result in multinominal class setup.

In *PCG: Angry Bots*, a non-dichotomous ordinal rating system is used, formatted as a Likert item [Likert, 1932]. After a player completes a map or chooses to skip it and move on to the next map, they are asked to rate their experience on the six point scale of {Very Bad, Bad, Poor, Fair, Good, Very Good}. No 'Neutral' option is provided as we wanted to clearly define positive and negative experiences. This rating system logically leads to the use of a multinominal class setup for the NB classifier. However, it was found in early testing that using multiple classes (greater than three) caused significant performance loss for all the comparative classifiers in Figure 6.2. This is because, assuming a fixed number of training samples (maps rated), the samples must now be split across more classes, which means each class has less evidence to support it. This compounds with the large number of features used in the classifier, as described in the previous subsection, which also require an increase in training samples to properly find the class boundaries in the high dimensional data. All of these requirements for increased training samples conflict with the expected number of maps to be played by the average player of *PCG: Angry Bots*.

After the poor performance of these initial tests we considered using a simpler {Dislike, Like} rating system and adopting a binary class setup to increase classification accuracy.

Table 6.1: Conversion table for nominal player rating to weighted binary classes.

Nominal Rating	Very Bad	Bad	Poor	Fair	Good	Very Good
Binary Class	Dislike	Dislike	Dislike	Like	Like	Like
Weight	2.0	1.0	0.5	0.5	1.0	2.0

However, in this type of application we argue that a binary rating scale does not properly capture the finer details of a player's preferences. For example, the player may have enjoyed the previous map but not as much as other maps before it.

Therefore, a trade-off between the two approaches was formulated in which the player's ordinal rating is converted into weighted-binary class value. The player ratings {Very Bad, Bad, Poor, Fair, Good, Very Good} are evenly divided into two classes {Dislike, Like} and the associated training sample is given a weight of {2, 1, 0.5, 0.5, 1, 2} respectively. A conversion chart for this is shown in Table 6.1. This allows for the performance benefits of a binary rating system to be gained but also allows for the finer details of the players preferences to be included. For a NB classifier, the weights are taken into account during the prior probability and likelihood calculations in Equation 6.8 by counting an instance doubly for a strong rating and as half for a weak rating. Thus, a rating of 'Very Bad' or 'Very Good' will have a strong influence over the classifier while ratings of 'Poor' and 'Fair' will only make weak contributions to the classes.

6.6 Generalizing the Player Preference Model

The RS based PPM described in this chapter is the most generalizable element of this thesis. We believe it can be used during personalized PCG of game maps in most game genres that use distinctive maps (i.e. the player knows when the maps starts and finishes). Moreover, it can be used almost entirely as it is presented here, with the only required change being made to the map features being used by the classifier. These features should be appropriate to the game genre and the style of maps and can be designed with expert knowledge.

This PPM can be used in single player games, as shown here, or potentially in multiplayer games, where the best candidate map is the one that has the highest average probability of being liked by all players. It may also be used for other types of game content, DDA algorithms, or other evaluative processes that require a player model. However, this PPM is not suitable as a descriptive model that is meant to report statistics about the player that may be used by developers during the game design process.

The key to adapting this PPM to other applications is to always think of it within the RS framework. The players will usually be the users of the RS but what are the items that are being recommended to them? In this case, game maps were the items of the RS. The source of player data should also be considered at this time. While subjective player feedback was used to train the PPM in this example, the player's actions during gameplay could also be monitored. However, for future work, we plan to continue to use subjective data as it gives a clearer indication of the player's preferences.

The items should be able to be evaluated by the users subjectively at appropriate times throughout the game. The maps in *PCG: Angry Bots* are individual and self-contained, so it is easy for the player to know when one starts and ends. If the maps were part of continuous linear gameplay, it may have been difficult for the player to judge exactly what they were rating. For similar reasons, if the PPM is being used with other game content, the user should not be interrupted very often to gather feedback; it should be requested during a natural break in gameplay.

From here, the remainder of the solution involves designing the RS. There are many possible RS solutions that can be investigated for both content-based RS and CF [Adomavicius and Tuzhilin, 2005]. If a traditional content-based RS is used as was done with *PCG: Angry Bots*, then the designer needs to choose a classifier, the features of the content, and the classes of the classifier. In the results of the *PCG: Angry Bots* experiment in Chapter 9, we evaluate the prediction accuracy of numerous classifiers with real-world player data to demonstrate alternatives to the NB classifier used in the actual game.

The features of the maps in *PCG: Angry Bots* that were used by the NB classifier were designed here with expert knowledge and minimal testing. However, a more rigorous feature design stage can be conducted to maximize the prediction accuracy of the classifier. As a general rule, the game content should be described as completely as possible with as few features as possible. Finally, any type of class setup and user feedback can be used. However, as a binary reporting system does not offer enough granularity for the user preferences and a multiclass setup leads to poor classifier performance, we recommend adoption and further investigation of converting ordinal player feedback to a weighted-binary class setup.

6.7 Summary

- There is overlap between the knowledge and algorithms of the recommender system (RS) field and the experience-driven procedural content generation (EDPCG) field.

Both fields attempt to understand a user's preferences and utilize this knowledge to provide personalized experiences in the future.

- The solution space of an EDPCG is analogous with the item set of an RS. Each piece of content that could be potentially generated by a PCG algorithm can be thought of as a fully realized item for the purposes of RS.
- The main difference between EDPCG and RS is that the quantity of all possible content is much greater in a typical EDPCG application than that of a typical RS application.
- When combining EDPCG and RS, search methods that are used in many EDPCG applications can act as filters to an RS. For example, when EA is used, the RS becomes the fitness evaluator. This can also be thought of as an EA searching the solution space intelligently in order reduce the number of items that need to be evaluated by the RS.
- There are three types of player data that a player model can be built upon: subjective, objective, and gameplay player data. In our research we use a subjective data source in the form of player ratings of maps. This subjective data source was chosen due to similar rating systems being used in commercially active RS applications.
- Collaborative filtering (CF) cannot be used in this domain without investigating alternative methods of calculating user-to-user similarity. The user-item (player-map) matrix is intractable due to its immense size and so it is rare for two or more players to experience the same map, which prevents co-ratings from occurring.
- Instead, a content-based RS is used, in which the rating of unseen items (maps) is predicted based upon the ratings that the user (player) has given to other items. This works on item-to-item similarity calculations.
- A model-based approach was used. Each player has a Naive Bayes (NB) classifier that is trained with features and ratings of maps that were previously experienced by the player.
- For the NB classification, a total of 18 features are extracted from each map and used by the classifier. These features summarize the layout of content in a map, both in terms of the quantity of each content type and the transition of content quantities between adjacent rooms of the map.

- A weighted binary class system is used. Players rate each map that they play on an ordinal scale of {Very Bad, Bad, Poor, Fair, Good, Very Good}. The ordinal ratings are evenly divided into the binary classes {Dislike, Like} and given a weight of {2, 1, 0.5, 0.5, 1, 2} respectively. This insures that strong opinions have a greater influence over the learning of the classifier.
- The player model described here is highly generalizable, with most design decisions focusing on how the model is to be used and what type of RS to use. The system can be used 'as is' for personalized map generation in other games, given appropriate map features that have been designed through expert knowledge of the game.

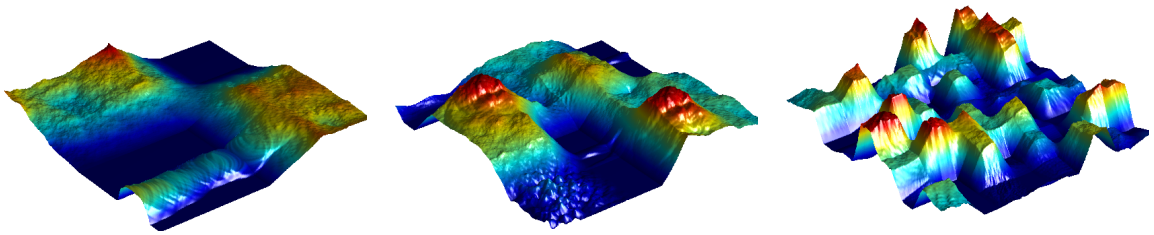
Chapter 7

Evolutionary Terrain Tool Demonstration

In this chapter we present a qualitative demonstration of the evolutionary terrain tool (ETT) that was described in Chapter 3. Through various tests, we show the level of control that this tool provides to the user to aid them in achieving their goals regarding virtual terrain creation. Firstly, an analysis of some of the key parameters of the ETT is provided with examples of how each parameter can affect the terrains that are generated. Then, in Section 7.2, sample runs are shown in which the user is attempting to produce a terrain with a single simple feature. After this in Section 7.3, the user's goals expand to generating the multitude of terrain features present in game maps from industry leading games. Finally, Section 7.4 concludes this chapter with a discussion on how the ETT can be used by novice game designers but how it has limited uses from the perspective of an individual player's preferences. It is worth noting that throughout all these demonstrations, the sample terrains shown in Appendix A were used during the patch extraction process within the ETT.

7.1 Observations on Parameter Settings

In Chapter 3, the parameters of the ETT were described in detail. This section provides examples of how some of these parameters can be changed to create a variety of terrain effects. Each of the subsections below addresses a specified parameter from either the set of terrain parameters or the set of evolutionary algorithm (EA) parameters. A brief description of the parameter will be given as a reminder of its purpose followed by figures to show its effect on the terrain creation process.



(a) *patch-matrix* $= 2 \times 2$ (b) *patch-matrix* $= 4 \times 4$ (c) *patch-matrix* $= 8 \times 8$

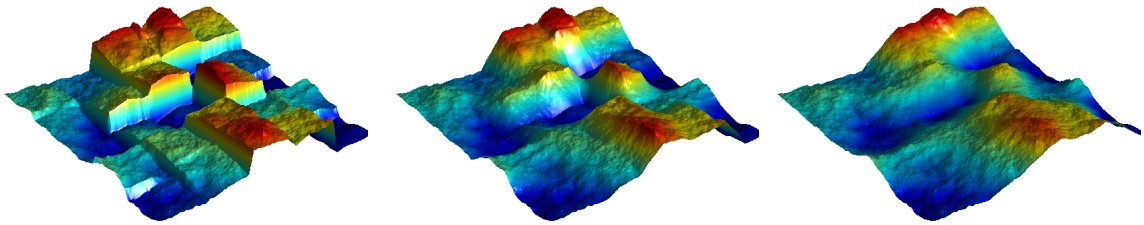
Figure 7.1: The effect of changing the dimensions of the patch-matrix. The patch-matrices of all three terrains were randomly generated.

In all these figures, the terrain resolution parameter is set to 512×512. This is the number of vertices in a single candidate terrain. Additionally, all the terrains in this section are rendered using a height dependent color-map. In this tool, each vertex of the heightmap can take a height value between $[0.0, 1.0]$. Each vertex is given a color based upon its height value; vertices at 0.0 are dark blue, vertices at 1.0 are dark red, and vertices and height values in between pass through the color range of cyan, green, yellow, and orange. The mesh surface is colored by interpolating the color value of adjacent vertices. This representation aids in distinguishing the relative height of neighboring terrain patches.

7.1.1 Patch-matrix Dimensions

The dimensions of the patch-matrix determine how many patches are extracted from the sample terrains and how many patches make up the candidate terrains. All the terrains in this thesis use square $(n \times n)$ patch-matrices. Figure 7.1 shows the result of randomly generating patch-matrices with different dimensions. An overlap ratio of 0.5 was used for all three terrains. As the patches in these examples are randomly selected, the probability of two adjacent patches having vastly different height values increases as the number of total patches increases. This can be seen by comparing the average height difference between adjacent patches in Figures 7.1a and 7.1c.

This characteristic results in the numerous high and low regions when an 8×8 patch-matrix is used. This is made more visible by the steepness of the transition between adjacent patches. As the terrain resolution parameter does not change in these demonstrations, increasing the number of patches causes a reduction in the size of each patch. As the overlap ratio parameter does not change either and the size of the overlap region is dependent on the resolution of a patch, this means that the size of the overlap region also reduces, resulting in



(a) *overlap* = 0.1 (b) *overlap* = 0.5 (c) *overlap* = 0.9

Figure 7.2: The effect of changing the overlap ratio parameter. The same patch-matrix was used for all three terrains.

a steeper transition between adjacent patches. Additionally, because each patch is smaller, less terrain features from the sample terrains are captured in each patch. This means that in using the ETT, with higher dimensions of the patch-matrix the user will need to craft more terrain features themselves, rather than relying on them to be present in the sample terrains.

However, there are disadvantages to using smaller patch-matrices as well. As the details of each patch are dependent on the sample terrains they are extracted from, a 2×2 patch-matrix relies on the sample terrains being of high quality. If the sample terrains do not contain a specific feature (such as a canyon or mountain ridge) that the user is seeking, then it will not to be possible to create a candidate terrain with that feature. Therefore, we have found that using a 4×4 or a 5×5 patch-matrix usually gives a good balance between the advantages and disadvantages of smaller and larger dimensions.

It is also worth mentioning that while an 8×8 patch-matrix results in an unrealistic and likely unusable terrain when the patch-matrix is randomly generated, these kinds of high dimension patch-matrices can be useful during EA. Due to the use of the two-level interactive evolutionary computation (IEC), having each patch containing a smaller portion of the entire terrain allows users to alter minute details through the gene selection process, giving the user further control over the resulting candidate terrains. However, the downside to this is that the more patches there are, the more time the user will spend on the gene selection screen, selecting which patches should mutate and which should not. This increases the time requirements of the user and can lead to user fatigue.

7.1.2 Overlap Ratio

The overlap ratio parameter determines how much of a patch overlaps with adjacent patches. For example, an overlap ratio of 0.3 means that 30% of the vertices of a patch are overlapped

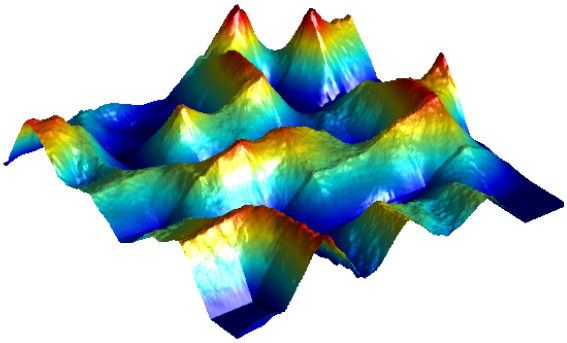


Figure 7.3: A terrain generated with a large number of patches and a proportionally large overlap area between each patch. Here, patch-matrix $= 8 \times 8$ *and overlap* $= 0.9$.

with adjacent patches. Figure 7.2 shows the same patch-matrix rendered with three different overlap ratios. The size of the patch-matrix is 5×5. With a small overlap ratio of 0.1, the border of each patch can clearly be seen. This is because there are fewer vertices in the overlap region to interpolate the height difference of adjacent patches. In this case, it has resulted in a blocky terrain. Meanwhile, when an overlap ratio of 0.9 is used, almost the entirety of a patch with four adjacent patches is used in stitching calculations. This leads to a much smoother terrain with rolling hills. While this appears more realistic, the downside to a high overlap ratio is that it can cause interesting features of each patch to be lost as the patch is essentially smoothed over. Additionally, if the dimensions of the patch-matrix don't change, then it is impossible to generate terrain features such as cliff faces.

A good balance between these two examples is to use an overlap ratio of about 0.5. This causes adjacent patches with similar average height values to form into smooth terrain. Meanwhile, adjacent patches with extremely different average height values can form steep slopes that resemble angled cliff faces. Thus, with a single parameter setting, multiple types of terrain features can be achieved.

Figure 7.3 shows how the patch-matrix dimensions and the overlap ratio can be combined. As with Figure 7.1c there are many small patches that make up this candidate terrain. However, now that a larger overlap ratio is being used, the boundaries of each patch are less evident. This also highlights how much of each patch is being used during the overlap and stitching processes. While Figure 7.1c contains areas of plateaus due to individual patches, in Figure 7.3 the majority of those features are now being overlapped with adjacent patches, resulting instead in terrain features reminiscent of peaks and ridges. This shows that by

adjusting these two parameters at the same time, an even more diverse set of terrain features can be produced.

7.1.3 Crossover Rate

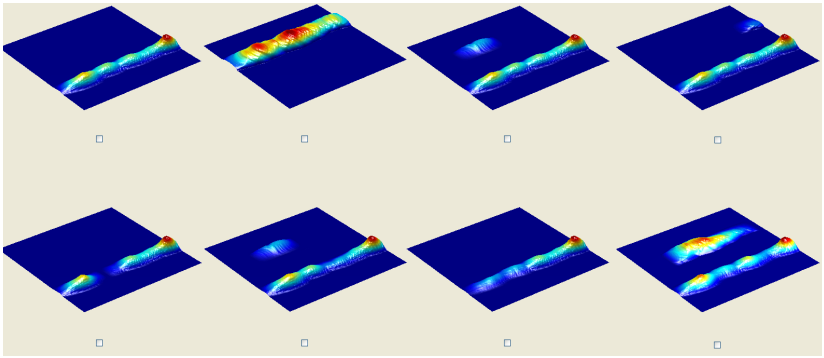
The crossover rate dictates which parent a patch is inherited from for each slot in the patch-matrix of an offspring. Figure 7.4 shows how changing the crossover rate affects the offspring produced by the ETT. The first two candidate terrains in the top-left of each image are the two parents of the rest of the population. It can be seen that with a crossover rate of 0.1 or 0.9, the features of the offspring terrains are predominantly from one parent or the other. Meanwhile, a crossover rate of 0.5 results in each patch being selected from either parent completely at random and thus produces variety of terrains with features from both parents.

While the user can select multiple parents in the ETT, the first parent they select will always be the first parent of the crossover operation, with the other parent being chosen at random from the remainder of the selected candidates. Therefore, if the user is mindful of the order in which they select the parents for crossover, a crossover rate below 0.2 or above 0.8 can be used to make small changes to a candidate terrain by attempting to include a handful of features from other candidate terrains.

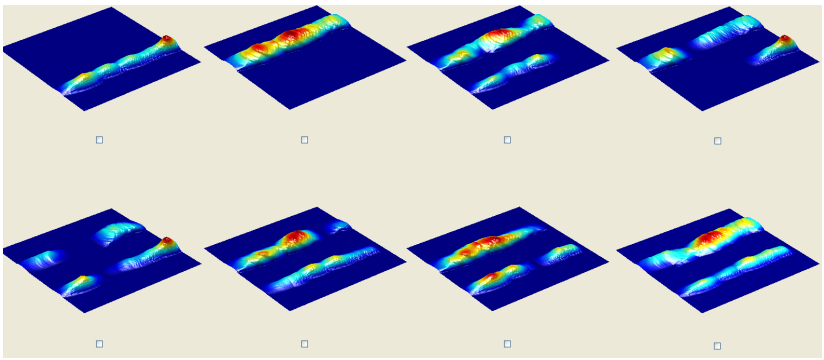
However, this can be achieved more effectively through the use of the gene selection mechanism in the two-level IEC process. If two or more parents are selected by the user, then the user is shown all the parents during gene selection, one at a time. If the user exempts a patch from crossover and mutation in one parent but not in the other, then the patch from that parent will be kept for that position in the patch-matrix. If both parents have the patch marked or unmarked during gene selection then crossover is conducted as normal. Therefore, it is recommended that the crossover rate be kept at 0.5. This provides increased exploration of the solution space early on in a run of the tool by producing a large variety of candidate terrains. Gene selection can then be used later on when a good foundation has been found and the user wants to make more controlled changes.

7.1.4 Mutation Rate

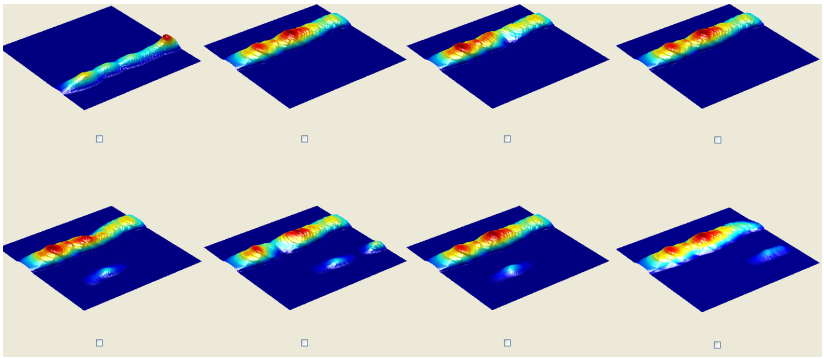
The mutation rate determines, probabilistically, how many patches in an offspring will be substituted with a randomly selected patch from the patch database. Figure 7.5 shows how increasing the mutation rate causes an offspring to differ more from the selected parent. Only a single parent is selected, shown in the top-left of each figure. Thus, the crossover operation



(a) *crossover* = 0.1



(b) *crossover* = 0.5



(c) *crossover* = 0.9

Figure 7.4: The effect of changing the crossover rate parameter within the ETT. The first two terrains in the top-left of each screenshot are the parents of the crossover process. The same parents were used in all three screenshots.

is ignored and all offspring are a result of mutation alone. When the mutation rate is set to 0.1, many of the parent's terrain features still exist in each of the offspring. Only minor changes are being made so the overall feel of the terrain largely remains the same. However, the downside to this is that some maps are so similar to the parent that they may be of no real interest to the user. One offspring in this figure is even an exact replica with no mutations.

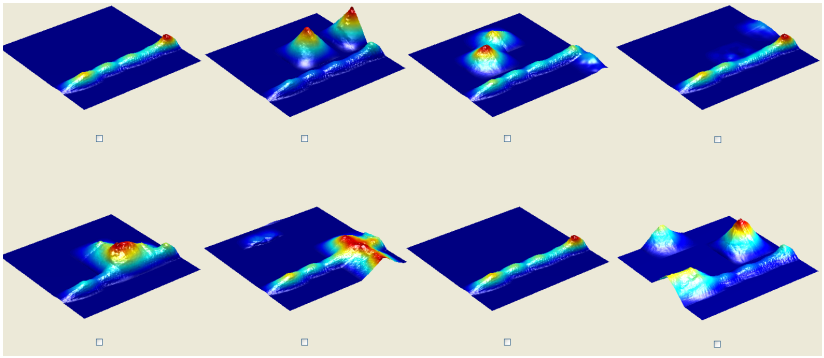
Increasing the mutation rate to 0.5 causes more variety in the offspring population. Some offspring still contain traces of the terrain features in the parent while others bear little resemblance. This may be a desired property, as it offers diversity in the population, or it may be counterproductive if the user is aiming to simply refine the features of the parent. Once the mutation rate reaches 0.9, almost all the recognizable features of the parent have been removed. At such a high mutation rate, the offspring population is essentially randomly generated as most patches in all the candidates will experience mutation.

Therefore, we suggest that a good choice for mutation rate is around 0.2 or 0.3. This gives enough variety to aid in exploration of the solution space while still producing offspring that have noticeable similarities to the parent. However, with such a low mutation rate, the probability of a specific undesirable patch mutating while leaving most others unchanged is low and can cause frustration for the user as generation after generation is presented to them with the desired mutation missing in all them. This is where the two-level IEC and inverse mutation rate improve the user's experience. By allowing the user to precisely select which patches are to mutate and increasing the mutation rate for them, problem areas of an otherwise desirable candidate terrain can be efficiently swapped out.

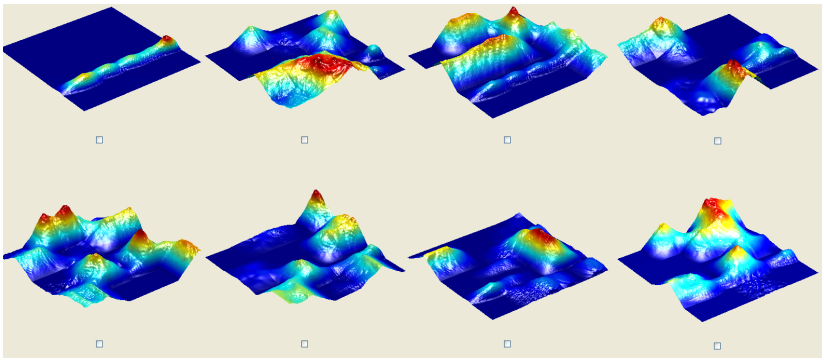
7.2 Sample Runs

The tests in this section were conducted to show that the user can easily and repeatedly produce terrains with simple basic features. All the final user selected terrains in the figures shown here were all generated within 15 evolutionary generations, representing a small time commitment from the user.

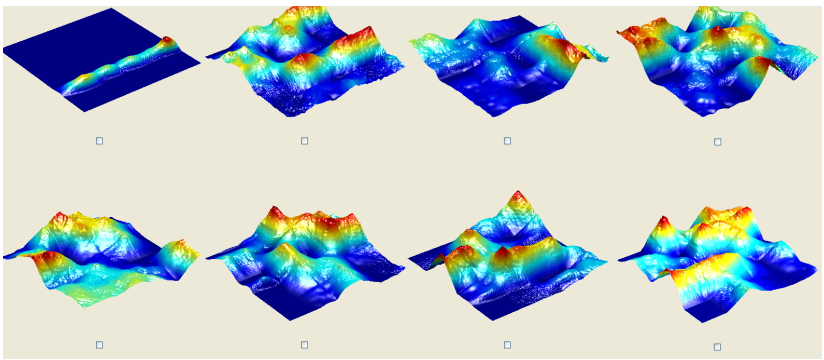
Additionally, the candidate terrains in the following figures are not rendered with a height dependent color map as before. Instead, the entire terrain is uniformly colored with a more neutral color, with the only contrast being produced by a single virtual light source. This is how the final version of the ETT prototype renders terrains and was chosen as such because



(a) $mutation = 0.1$



(b) $mutation = 0.5$



(c) $mutation = 0.9$

Figure 7.5: The effect of changing the mutation rate parameter within the ETT. The first terrain in the top-left of each screenshot is the single parent of all other candidate terrains. As there is only one parent, no crossover operation was conducted prior to mutation. The same parent was used in all three screenshots.

it looks more like a dirt or sand texture with the sun shining upon it. This gives the user a rough guide of what the final terrain may look like once properly textured.

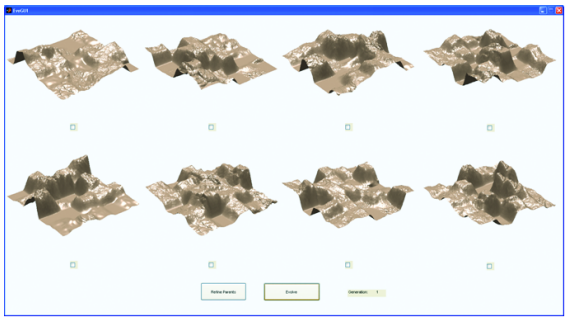
Figure 7.6 shows a single run of the ETT where the user desired a terrain with a canyon running between two ridges, from the bottom corner to the top corner. This type of specific feature requirement can be difficult to achieve in procedural terrain generation applications other than those that are built upon a sketch or other existing example [Zhou et al., 2007; Hnaidi et al., 2010]. In this run, the parameter settings were as follows: the patch-matrix dimensions were 5×5, the overlap ratio was 0.6, crossover rate equaled 0.5, and the mutation rate was 0.2.

Screenshots are shown for evolutionary generations one, four, and six, as well as a resulting terrain from the ninth generation. Up until the fifth generation, only parent selection was used, allowing the user to quickly get candidate terrains that exhibited some resemblance to the desired terrain. This can be seen in the fourth generation in Figure 7.6 as all the candidate terrains have similar features. After this, gene selection was used to refine the terrain, mutating only a handful of undesirable patches. Within a single generation this produces candidate terrains with an uninterrupted canyon. Between generation seven and nine, the user continues to use gene selection in order to form a narrower canyon.

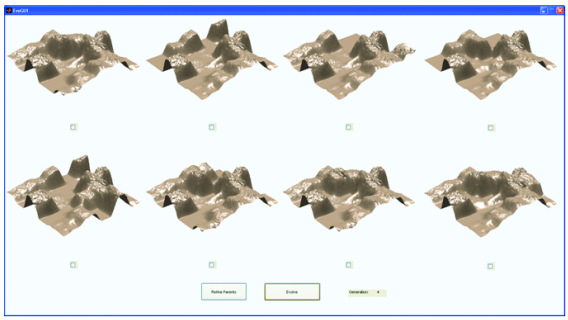
Figure 7.7 shows how the ETT can generate multiple terrains with similar feature layouts but with different details. Each of the terrains shown in Figure 7.7 was generated from a separate run of the ETT. The goal for all runs was the same as that in Figure 7.6. All three terrains exhibit a canyon from the bottom corner of the terrain to the top corner but with different widths, asymmetric ridges on either side, and different details on the canyon floor.

7.3 Video Game Map Generation

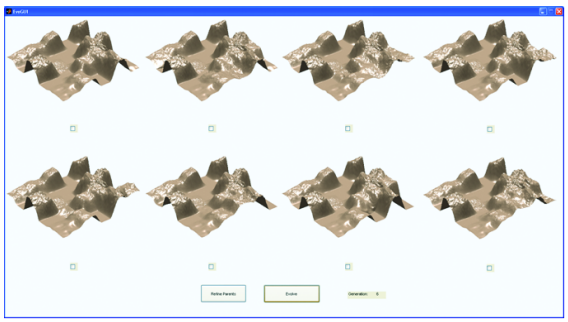
The purpose of the test in this section is to show how the proposed ETT can be used to generate terrain that is similar to a map in a top-selling game title. There are more required features of this type of terrain than in the terrains in the previous section, therefore increasing the complexity of the creative process that the ETT must allow for. However, it should be noted that the goal is not to replicate the game map but rather to produce terrains that are a variation of it; containing the same general features but different details.



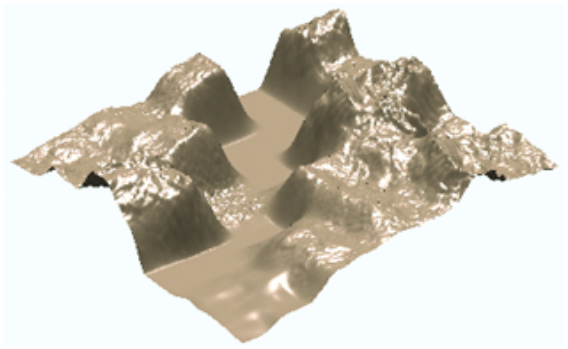
(a) Generation 1



(b) Generation 4



(c) Generation 6



(d) Result from generation 9

Figure 7.6: One run of the ETT with nine generations.

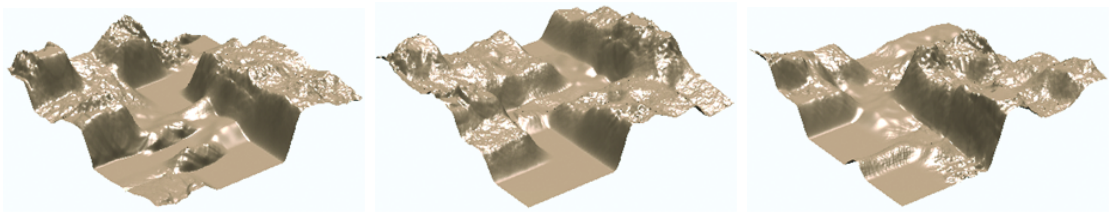


Figure 7.7: Results of three separate runs with a similar user goal as Figure 7.6.

7.3.1 Specific Example

The game chosen here is *Halo* (Bungie, 2001) and the map is titled 'Blood Gulch'. This is one of the most popular multiplayer maps of the game and, even though the game is now quite old, is still remembered by much of the gaming community for its simple design that still manages to deliver intense competitive gameplay. This map was chosen because it does not contain many other static objects, such as buildings and vegetation, which would obstruct the view of the underlying terrain.

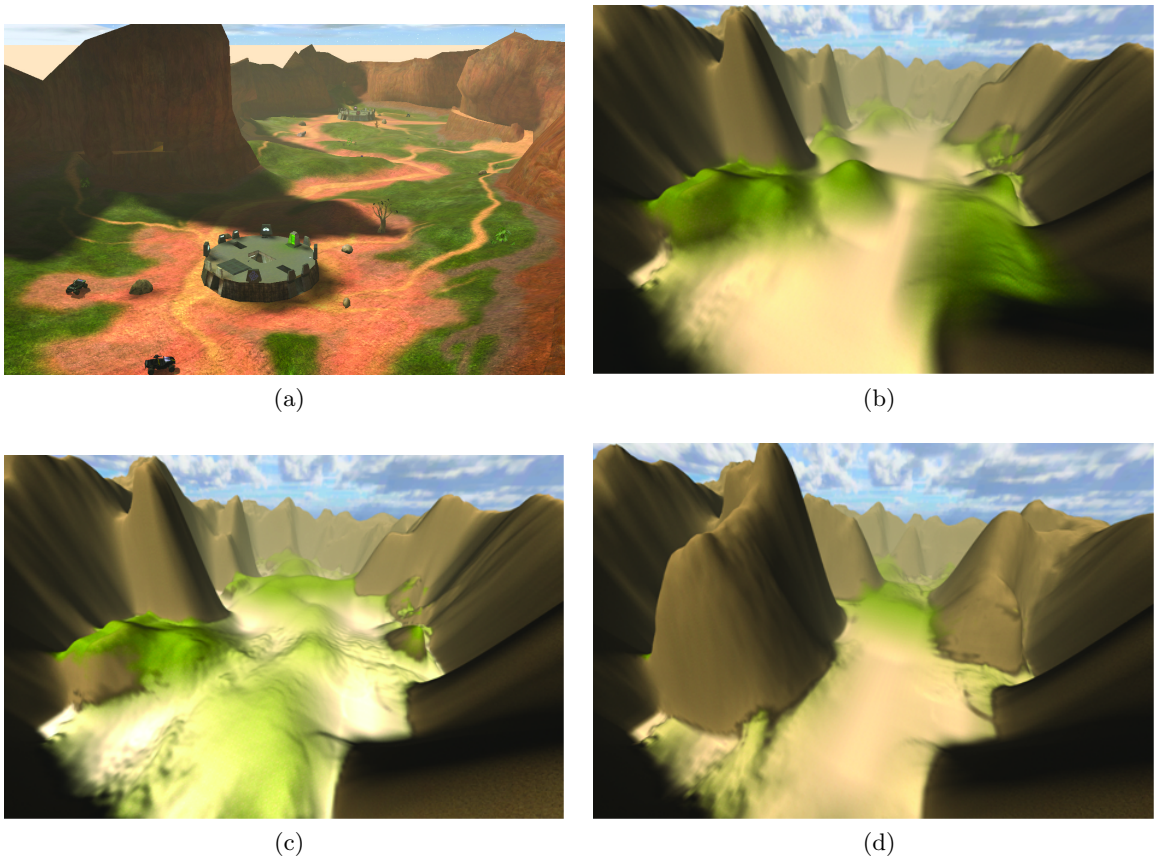
Figure 7.8 shows the results of attempting to create a similar terrain by using the ETT. Figure *7.8a* is the original game map that acted as an inspiration for the terrains in Figures *7.8b*, *7.8c*, and 7.8d. All terrains have been roughly textured and rendered in the *Unity*[1] game engine. It should be noted that our renderings lack the same attention to texturing that the original game map has received and do not have any static virtual objects or dynamic content to turn the terrain into a complete, playable game map.

Figure *7.8b* is the closest approximation to the original game map and was generated before the others. Figures *7.8c* and *7.8d* were modified from the terrain in Figure 7.8b to produce a variety of terrains. Figure *7.8c* shows how the playable area can be changed slightly, which would cause players to make minor changes to their in-game strategies. Meanwhile, Figure *7.8d* shows more drastic changes to the borders of the playable area, which would cause an even greater change in gameplay and possibly even require players to re-explore the map.

7.3.2 Multiple Runs with Different Parameters

As there are multiple terrain features being attempted here, using a single set of parameter values is not possible. To achieve the terrains in Figure 7.8, multiple runs were conducted, each with different parameter values. To do this, the first selected parent from every generation is saved to file. When the ETT is closed and launched again with new parameters, the last saved terrain is loaded along with all the other sample terrains for patch extraction. Additionally, this terrain acts as the parent for a round of mutation to produce the rest of the initial population. This allows the user to change parameters such as patch-matrix dimensions and overlap ratio between runs to achieve different terrain effects. The last parent terrain must be reloaded as an additional sample terrain because when the aforementioned parameters are changed the quantity and resolution of the extracted patches will change,

[1] http://unity3d.com/



(a) (b)

(c) (d)

Figure 7.8: (a) is a map from the game Halo *(Bungie, 2001). (b), (c) and (d) were all generated by our algorithm.*

affecting identifiers in the patch database. Without reloading the last parent as a sample terrain, it would be impossible to completely re-construct it with the new patch database.

In the example in Figure 7.8, the first and second runs used small patch-matrix dimension to generate a rough outline of the cliffs around the playable area with large patches. The third run used slightly larger patch-matrix dimensions and a small overlap ratio to form steep transitions between the patches to create the hard edges of the cliff face and protruding peaks. Finally, in all successive runs, the playable area was manipulated with large patch-matrix dimension to allow for more control over specific problem areas. A large overlap ratio was used at this stage to ensure smooth inclines and playable surfaces.

7.4 Discussion

The qualitative demonstrations in this chapter have shown how the parameters of the ETT offer a suitable level of control over the terrain creation process without requiring the user to handcraft the terrain features. This makes the ETT a potentially powerful tool for novice game designers who have little experience with map design. Though the designer will still need to put further structures and vegetation in the map to make it more immersive, simplifying the process of creating the base terrain will aid in the express creation of the geometry of most game maps.

Unfortunately, the ETT fails to address the goals of this thesis directly. The ETT can be used by a game designer to create a vast amount of maps in a shorter time frame than traditional methods and therefore produce an array of experiences for the players of the game. However, the objective of this research is to personalize game maps for an individual player with little effort from the player. The only way for the ETT to be used to personalize game maps is to require each player to utilize the tool themselves, rather than the game designer. This is an inappropriate expectation of the player and distracts from the core gameplay experience.

In order for this patch-based search-based procedural content generation (SBPCG) technique to be converted into an appropriate experience-driven PCG (EDPCG) solution, one of the following two changes must be made to the fitness evaluation method:

1. Evolutionary operations must be constrained such that every terrain is playable and appropriate to the gameplay. If this is the case, then IEC can continue to be used as it would not require excessive time commitments from the player.

2. Interactive evolution must be completely replaced with automated fitness functions that assess the quality of the terrain with regards to the player's preferences.

Additionally, regardless of which option is chosen, an obvious requirement for the terrain to be utilized as a game map is for content to be distributed on its surface to make it playable. Static objects should also be present to enhance the visual appearance and further expand the strategic decisions that the player must make. The first option above is the one that was utilized for creating the geometry in the *PCG: Angry Bots* game. Here, every geometry that is presented to the player is valid and complete with static objects to enhance the quality of the geometry, thus the player's selection of geometry is purely based upon their instinctive preferences.

The second option, to use an automated fitness function, would be the most desirable solution. In using this approach, the terrain personalization process would be similar to that of the content layout system in *PCG: Angry Bots*; requiring very little subjective input from the player. However, this is not as simple as it may first seem. For some gameplay geometries, there may be noticeable correlations between the geometry and the user's experience. For example, the angle of corners in a track of a racing game [Togelius et al., 2007] or the distance between platforms in a 2D platformer [Shaker et al., 2010] can give a quantifiable measure that can be shown to affect the challenge that a player will face in the map. However, what quantifiable measures determine the quality of a free form terrain, especially in relation to an individual player's preferences?

There have been a few attempts to answer this question though we feel that none have yet shown to be strong solutions. The accessibility fitness functions proposed by Frade et al. [2010a] only ensure that a terrain is valid, not whether it will be enjoyable. The many fitness functions used by Togelius et al. [2010b] encourage balanced and enjoyable maps but most of these metrics are for content layout rather than the structure of the geometry. Hullett and Whitehead [2010] provide a qualitative analysis of design patterns in FPS games but how these patterns can be translated into quantitative metrics for terrain-based geometries is unknown. It also may be of benefit to examine how the player interacts with the map rather than the features of the map directly, such as how Cardamone et al. [2011b] generated maps in a FPS game. When combining a player's actions or feedback with known features, as is done with the content layout *PCG: Angry Bots*, this may result in geometries being found quickly. However, without known features, it will likely require the player to experience many geometries before receiving an enjoyable map.

Aside from promoting enjoyable gameplay for the player, the fitness function or a constraint must ensure the visual quality of the terrain features. For example, a simulation style game should have terrain features that look natural. An unrealistic terrain will spoil the immersion that the player may experience and hinder their suspension of disbelief. The 'asymmetry' function used by Togelius et al. [2010b] evaluates the aesthetics of the geometry but only in a limited scope. Walsh and Gade [2011] have begun a study on generating terrains that maximize a measure of aesthetics that is based in philosophy, making a trade-off between 'complexity and order' [Birkhoff, 1933]. However, it is not made clear to what degree this measure ensures sensible terrain features are generated. Rather, it is used by the authors to determine the difference in quality between valid terrains.

As a final aside note on the demonstartions in this chapter, it is worth considering the role that the two-level IEC setup has played. The use of the gene selection screen allowed for desirable terrains to be found by the user in a much more efficient time frame. Without it, the game map terrains generated in Section 7.3 would not have been possible. We have not identified any other IEC application that makes use of a similar gene selection mechanism. Therefore, it may be of interest to investigate how this technique could be applied in other fields.

For a gene selection mechanism to be used, however, a direct genetic representation must be used with a one-to-one link between the genes of the genotype and some feature of resulting phenotype that is shown to the user. In the ETT, each gene (patch) of the patch-matrix (genotype) represents a small area (feature) of the resulting terrain (phenotype). This conflicts with many other IEC applications that use indirect genetic representations, such as the use of genetic programs [Xu et al., 2007] or CPPNs [Stanley, 2007] for genetic art or the evolution of terrains in early work by Frade et al. [2009b].

7.5 Summary

- By changing the patch-matrix dimensions and overlap ratio parameters of the evolutionary terrain tool (ETT), various terrain features can be achieved.
- Smaller patch-matrix dimensions lead to smoother terrains. Larger patch-matrix dimensions lead to terrains with many peaks and ridges.
- Smaller overlap ratios can be used to create sharp cliff faces at the border of two patches. Larger overlap ratios can be used to smooth out or round off terrain features.

- Changing the crossover rate and mutation rate parameter of the ETT affects how many generations it will take the user to find a suitable terrain by providing trade-offs between the exploration of the solution space and the refinement of partially suitable terrains.
- Small and large crossover rates cause offspring candidate terrains to resemble one parent, promoting the refinement of a parent's existing terrain features. Mid-range crossover rates promote the exploration of new terrain features that result from the combination of features from both parents.
- Small mutation rates help in refining terrains but can also lead to offspring exhibiting barely any improvement over the parent. Large mutation rates cause the offspring terrains to show no recognizable similarity to the parent. A mid-range mutation rate provides a balance between these two.
- The use of gene selection within the two-level interactive evolution setup of the ETT improves the utilization of all the above parameters and expedites the process of finding a suitable terrain.
- By combining multiple runs of the ETT with different parameter settings, it is possible to create multiple terrain features in a single terrain. This is important in creating terrains that are to be used as maps in a game.
- While the ETT is a powerful tool for game designers to create a handful of maps in their game, it is less suitable when attempting to produced personalized geometry's that align with an individual player's preferences.
- For *PCG: Angry Bots*, the IEC process is simplified by ensuring that every geometry is valid. This allows players to find enjoyable geometries in even the first generation, using the IEC controls to determine the style of future geometries rather than their quality.
- In order to avoid the use of an IEC in the creation of the geometry of a map, an automated fitness function would be needed. This is possible when there is a known correlation between geometry features and the player's and therefore requires genre specific solutions.

Chapter 8

PCG: Angry Bots Experiment Setup and Metrics

In this chapter and the one that follows we discuss the results of a public user study into personalized procedural generation of game maps. In this experiment, the *PCG: Angry Bots* game, along with the entirety of the components discussed in Chapters 4, 5, and 6, was made freely available to play online[1]. In Section 8.1 we describe the setup of the public user experiment, including details on who was eligible to participate and how invitations to participate were distributed. Section 8.2 then details alternative solutions that we compare our solution to during evaluation, followed by a description of the quantitative metrics used to analyze the player data in Section 8.3. Section 8.4 then gives a detailed discussion of the newly proposed *learning trend* metric, which is introduced to solve issues associated with traditional RS evaluation metrics.

8.1 Experiment Setup

General invitations to participate in the *PCG: Angry Bots* experiment were distributed through various social networking channels. The experiment was open to anyone who desired to participate, as long as they agreed to the accompanying ethics guidelines[2] and identified themselves as being over the age of 18. Thus, the participant sample contained a mix of

[1]The *PCG: Angry Bots* game is available at:
http://http://goanna.cs.rmit.edu.au/~wraffe/ExperimentHome.html

[2]Ethics approval for the PCG: Angry Bots public user experiment was granted by RMIT University's Science, Engineering, and Health College Human Ethics Advisory Network (CHEAN). Application ID: BSE-HAPP 01-12 Zambetta

novice video game players, veteran video game enthusiasts, and players of experience levels anywhere in between. The game was only available to play on the Windows operating system.

The experiment was anonymous and unsupervised, meaning that participants could play the game for as little or as long as they wanted. Participants were required to create a new account before beginning play and were given a randomly generated four digit identifier and password. With these login details, player could stop play and continue at a later date. No mechanisms were put in place to stop participants from creating multiple accounts. For every account, information about the maps that were played, the player's map ratings, and the player's general activity within each map were recorded and stored on secure RMIT University servers. The period of data collection was between December 2012 and March 2013. The game is still available to play and is still collecting data, however it was decided to end the official data collecting period and begin analysis. Thus, the results in Chapter 9 describe the data as of the end of March 2013.

8.2 Comparative Solutions

In order to evaluate the performance of the personalized procedural map generation solutions implemented in *PCG: Angry Bots*, it's desirable to compare it with other solutions. Unfortunately, it's difficult to compare with other PCG solutions as they are implemented for different games, for different types of game content, and modeling different affective states of the player. Additionally, to the best of our knowledge, there is no standardized metric for comparison that other solutions provide results for. Thus, testing against these solutions would most likely require that we re-implement them, a process which may introduce errors into the existing solutions and therefore produce an unfair comparison. Finally, testing multiple solutions, in most cases, would require multiple user experiments to be conducted. As it was a lengthy process to attain the data presented in the next chapter for a single personalized PCG solution, it would be infeasible to conduct additional experiments within the time allowed for this candidature. All these issues are unfortunately present in much PCG research and games research in general [Smith, 2013].

Instead, we compare the implementations of the last three chapters for geometry generation, content layout, and player preference modeling with a random procedural map generator. Additionally, various classifiers are experimented with for use as the content-based RS player model; a test that is conducted offline and therefore not requiring any additional data. The following subsections describe both of these types of comparative solutions.

8.2.1 Random Baseline

In *PCG: Angry Bots*, after each map is completed and the player has rated it, they have the option of generating the next map through the use of the mechanisms described in Chapters 4, 5, and 6 or by completely randomly generating the map. The option to create randomized maps was included to allow the user to break out of local optimums in the evolutionary process. This was in anticipation of overspecialization in the evolutionary optimization or RS player model; an issue that eventuated during the experiment and is discussed later. While the player is most likely not going to understand this concept, it was expected that they would use this option when they felt they were repeatedly given similar maps to those they had already played or bad maps (both of which are the result of a poorly trained NB classifier).

If the player chooses to randomly generate a map, then a tree with a random number of nodes between the start and exit rooms is produced and each node is given a chance to have branches, controlled by a randomly generated branching rate variable. The tree is then validated in the same way that a mutated tree is, regenerating the entire tree if the validation fails. Once the tree is constructed, the content setting of each content type of each node in the tree is randomly assigned, with each setting having an equally likely probability of being chosen. Player ratings are also gathered for randomized maps in order to further train the RS player model. However, randomized maps do not act as parents to the next optimized generation. These randomized maps and their ratings act as a baseline comparison in the results in this chapter. By doing this we are able to compare our implementation to game genres such as *roguelikes,* in which it is typical for all maps in the game to be randomly generated.

8.2.2 Alternative Classifiers

In the results in this chapter we show the performance of the NB classifier that was used as the core of the RS player preference model (PPM) during gameplay along with a few additional classifiers that have been utilized in previous content-based RS research [Pazzani and Billsus, 1997; 2007]. The first alternative is a multilayer perceptron with one hidden layer containing 12 hidden nodes, trained through backpropagation. The second is the IB1 nearest neighbor algorithm [Aha et al., 1991] and the third is a J48 decision tree, which extends the C4.5 algorithm [Quinlan, 1993]. The fourth is a random forest classifier [Breiman, 2001]. Finally, two runs of a random classifier are provided to show a baseline comparison. This

classifier randomly predicts a class for each map of each player. If this classifier were to be used as the RS in the actual game, it would cause all maps to be randomly generated.

The user experiment was only conducted with NB and so the collected data is a result of its use. This may cause some undesirable effects in the learning capabilities of the other classifiers. However, these plots give a general indication of the performance of the game if one of the other classifiers had been used.

8.3 Quantitative Evaluation Metrics

In this section we describe the metrics used to quantitatively evaluate the data collected from the *PCG: Angry Bots* experiment. The results of these evaluation metrics being used on the player data are reported Chapter 9.

We first use the chi-squared and Mann-Whitney U tests on map ratings provided by players in order to make quantitative statistical inferences about the performance of our systems when compared to random map generation. Then, to give insight into the types of content layouts that players experienced, we calculate the balance of enemies and pick-ups in each room of a map to determine whether the map increased, decreased, fluctuated, or remained constant in difficulty. The results for the geometry and content layout solutions also present various qualitative analyses of global trends across all players but these evaluations are not described in this section. Rather, the approaches to qualitative analysis are described in their respective sections in Chapter 9, just before the actual results themselves are presented.

Finally, the performance of the RS player models is determined first through mean classification accuracy and mean 3-fold cross validation accuracy. These metrics are common to the RS field [Herlocker et al., 2004; Shani and Gunawardana, 2011] and they are described in detail here in the context of an RS being used as a PPM. The RS player model performance is then also evaluated through the *mean* and *majority learning trend* metrics. These two metrics, jointly referred to as simply the learning trend, are proposed here as a means of improving the evaluation and visualization of RS data by ignoring issues that affect most other confusion matrix based classifier performance measures.

8.3.1 Statistical Analysis Tests

Every map played during the experiment fits into one of two categories; it is either an *optimized* map or a *randomized* map (discussed in Section 8.2.1). The sets of these two types of maps are henceforth referred to as S_{Opt} and S_{Rand} respectively. Many players regularly

played a combination of both optimized and randomized maps, which resulted in a large sample of ratings for both. This allows us to statistically compare the ratings that the maps generated with our solutions received with the ratings of randomly generated maps.

To compare the ratings of S_{Opt} and S_{Rand}, a visual analysis is first provided to give an overview of the rating trends between the two sets. After this, statistical analysis is provided through both non-parametric and parametric tests [Sheskin, 2003]. Unless otherwise stated, these tests are conducted on the raw ratings provided by the players, as oppose to the binary ratings used by the NB classifier.

We first use the chi-squared (χ^2) test of independence, which compares the frequency of ratings in each set with a calculated expected frequency. This tests whether the rating of a map is dependent on the set (S_{Opt} or S_{Rand}) it belongs to. That is, the null hypothesis (H_0) states that the two sets are independent of the ratings received and the alternative hypothesis (H_1) is that they are dependent. By proving H_1, the statement can be made that the ratings in S_{Opt} were not a matter of chance.

To reinforce this, we then use the one-tail Mann-Whitney U test [Mann and Whitney, 1947] to test whether or not maps from S_{Opt} receive statistically better ratings than those from S_{Rand}. The Mann-Whitney U test ranks the two sample sets against each other and computes the sum of differences in rank comparisons. In the RS field, the Friedman test is traditionally used [Shani and Gunawardana, 2011]. These two tests are conducted similarly, however, the Friedman test requires that both sample sets are of the same size, such as when comparing the accuracy of two classifiers acting on the same data. As S_{Opt} and S_{Rand} have a different number of ratings, the Mann-Whitney test is used instead. In using this test, H_0 states that there is no difference between the two sets and H_1 states that S_{Opt} performs better than S_{Rand}.

To reinforce this result, we also conduct the parametric Student's T-test. While the rating system is ordinal, it is also on a symmetrical Likert scale [Likert, 1932] and the distance between each rating can be assumed to be equal. It has been argued that under these types of conditions, as well as when having large sample sizes, ordinal data can be enumerated and considered as interval data instead with little error introduced into the results of parametric tests [Labovitz, 1967]. Here, we use the one-sample T-test to compare the mean and standard deviation of the optimized set to the mean of the randomized set. This is performed as a one-tail test and so H_1 states that S_{Opt} outperforms S_{Rand}. It should be noted, however, that the treatment of ordinal data as interval data is a topic of contention [Jamieson et al., 2004] and so the non-parametric tests are the primary statistical analysis method used here,

with the results of parametric tests being shown as supportive material and for the sake of thoroughness.

8.3.2 Content Balance

Visualizing content layout trends among all players can be difficult when considering all six content types. Thus, for much of the evaluation of the content layout in this chapter, we summarize the six content types into their broader categories: enemies and pick-ups. The assumption made here is that increasing the number of enemies of any kind will increase the difficulty of a room or map equally and increasing any pick-up quantity will reduce difficulty equally. While this assumption isn't entirely accurate (e.g. ammo is more valuable than health in this game), it does give a clear impression of the types of content layout being experienced by players.

With this in mind, the change in difficulty throughout a map can be summarized by first enumerating each of the four content settings {None, Low, Medium, High} as $\{0, 1, 2, 3\}$ and calculating the content balance of a room as the sum of all pick-ups minus the sum of all enemies.

$$ContentBalance = \sum_{0}^{3} pickups - \sum_{0}^{3} enemies \tag{8.1}$$

Therefore, a reduction in the content balance value from one room to the next indicates that the difficulty is increasing and vice versa. A value of zero means there is a perfect balance between the content types but does not indicate the quantity of each.

If the content balance is calculated for every room on the direct path between the start and end rooms (not including these two rooms or any branches), we can categorize the linear experience as either increasing in difficulty, decreasing in difficulty, fluctuating between the two, or holding a constant difficulty across all rooms. For example, in a map that has three rooms between the start and end rooms, if the second room is more difficult than the first and the third is less difficult than the second, then the map has a fluctuating difficulty. Meanwhile, if the third room had the same content balance as the second or was again more difficult, then the map is said to be increasing in difficulty.

8.3.3 Mean Prediction Accuracy and Precision

One of the most prominent methods of evaluating an RS is to compare the predicted rating and the actual user rating [Shani and Gunawardana, 2011]. As the NB predictions of the maps are done on the binary class scale of {Dislike, Like}, a confusion matrix can be generated. After a player experiences a map, the rating they provide is compared to the predicted rating and the confusion matrix is updated. Note that the map indices are in the order in which the player experienced the maps. Thus, for example, the prediction accuracy at map index 3 is calculated from the confusion matrix so far from the first three maps. It is also worth reiterating that as there is a stochastic nature to the CPPN-NEAT evolution and due to each player having a separate RS player model, each player will experience a different map at each index and, therefore, the map index does not correlate to any style of map.

The average learning performance of the NB classifiers is reported here instead of mean player ratings because the raw player ratings tell us nothing interesting other than at each map index players were having a variety of positive and negative experiences. Most players experience fluctuations in positive and negative ratings (discussed later in Section 9.1.1 of Chapter 9), which makes taking a mean rating at each map index senseless. The mean classifier performance, on the other hand, identifies how well the system is learning and applying the preferences of players given the amount of training data that is available at each map index.

Accuracy is one of the simplest of measures that is typically used to report a classifier's performance and encompasses the entire confusion matrix. It states the proportion of ratings that the classifier predicted correctly. That is, if the map was predicted to be liked and the player gave it a good rating or if the map was predicted to be disliked and the player also gave it a bad rating. For an individual player, accuracy is calculated as:

$$Acc = \frac{TruePositives + TrueNegatives}{AllPositives + AllNegatives} \tag{8.2}$$

An alternative to this accuracy measurement is to simply count the number of maps that were positively rated. This is because most maps that are presented to the player are predicted to be liked (a positive prediction) and so the result is either a true-positive or a false-positive. However, while rare, it is possible for a negatively predicted map to be presented to a player when no positively predicted map has been found in the previous evolutionary cycle. Thus, by ignoring these results, the accuracy measure would be skewed.

Therefore, we use the traditional approach to calculating accuracy as defined in the above equation.

Once the accuracy for every player is calculated at every map index that they experienced, the mean accuracy of all players at map index m can then calculated as:

$$\overline{Acc\,(m)} = \frac{1}{P_m} \sum_{i=1}^{P_m} [Acc\,(i, m)] \tag{8.3}$$

where $Acc\,(i, m)$ is the prediction accuracy of player i at map index m and P_m is the total number of players who have played up to map index m.

Aside from accuracy, the most common performance measures that are reported are precision, recall, and the subsequent F1 measure. *Precision*, also known as the positive predictive value, relates how many of the items that were predicted to be positive were actually positive. It can be seen as an accuracy measurement whereby only the positive predictions are considered.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{8.4}$$

It may be seen that precision is an important measure here because we want our system to maximized the number of true-positive predictions, accurately providing the player with maps that they enjoy as often as possible. However, the measure is slightly redundant. As the RS player models typically make positive predictions in this implementation, the precision curves are quite similar in shape to those of the accuracy ones.

Recall, the F1 measure, and all other metrics that rely on negative predictions are also not reported in this thesis. The reasons for this are as follows: *recall* describes how many of the items in the entire population that are suitable for a user were found by the RS. In an offline experiment, if there is a sample of items that are known to be liked by a user, recall measures whether an RS algorithm would recommend those items. This measure is suitable when it's important that all the best items are found for the user. It is calculated as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{8.5}$$

Recall is not useful in our system because there may be millions of map variations in the solution space that the player may like. However, as long as one of these maps is recommended to the player then the system has been successful. Additionally, while the system is in use

by a player, a false-negative result for an optimized map is not common. This means that after the first true-positive result, the recall will typically remain at a perfect score of 1.0 for most players. This subsequently affects the F1 measure, which is the harmonic mean of precision and recall, and other reliant measures such as precision-recall curves, receiver operator curves, and associated variants of these [Schein et al., 2002].

8.3.4 Mean K-Fold Cross Validation Accuracy

A slight variation to the accuracy measurement above is to use k-fold cross validation. The accuracy measure described earlier updates the confusion matrix once per map index and tells us how the classifier was performing while it was being used by the player. For example, an increase in accuracy at any given map index for an individual player indicates that the classifier properly predicted the rating for that specific map. However, k-fold cross validation instead indicates the classifier's ability to generalize to unseen test data. This is seen as key performance measure because if a classifier is overfitting to the training data (the ratings of maps already played), then it is unlikely to provide accurate predictions for new maps that differ substantially from those that have already been played.

In k-fold cross validation, the ratings for all the maps played so far are divided into k partitions (folds), using $k-1$ folds as the training data for the classifier and the last fold as the test set. After using the trained classifier on all the samples in the test set, a confusion matrix can be constructed and accuracy calculated in the same way as earlier. This is repeated multiple times, allowing each fold to be the test set once, and then taking the mean accuracy of all the tests. This is done for each map index that is greater than k for each player. For example, on the $12th$ map index of a single player using 3-fold cross validation, 3 folds are created with 4 maps in each one. The classifier is trained and tested 3 times and the mean accuracy is calculated. Once the k-fold cross validation scores are gathered for each player at each map index, the mean value of all players is calculated similarly to Equation 8.3.

For the evaluations in this chapter we use $k = 3$ folds. In many machine learning applications it is typical to use $k = 10$. However, as k must be smaller than the number of maps played, a small k value must be chosen here as it is expected that many experiment participants will play only a few maps. This is also why the cross validation plots in this thesis do not show results for map indices less than 3.

8.4 Mean and Majority Learning Trend

There are three issues that affect the use of the prediction accuracy plots covered so far in this type of application:

1. Reducing player counts: There are a reducing number of players at each consecutive map index. Therefore, the mean calculation is conducted with fewer players at later map indices. Some fluctuations in the plots are due to the performance of classifiers that belong to a group of players that are discontinuing play being significantly different to those of the players that are continuing. This gets worse later where only a few players are playing and if even one player discontinues it can have a drastic effect on the average.

2. Fraction denominator: The very nature of the fractions used in the accuracy measurements means that a classifier's performance between two map indices will be more varied towards the start of play than towards the end. This also works in inverse to the reducing player problem, as the fraction denominator problem causes steep fluctuations early on and the reducing player problem will cause steep fluctuations later.

3. Late learners: A few players have a positive experience only after many players have discontinued playing. This results in the mean accuracy being greatly lowered, especially as the number of players reduces. This issue also compounds with the fraction denominator problem as a classifier may begin to experience all true-positive results after a dozen maps but this will not raise the mean accuracy the same as if they had occurred earlier on.

In order to address these issues and get a more stable representation of classifier performance we propose the use of a *learning trend* metric. Rather than using the raw values of a performance measure, the learning trend only detects whether there is a positive or negative change from the last accuracy calculation of a single classifier and increments or decrements the plot by one standard unit. For example, if the previous performance measurement is 0.6 and the current measurement is 0.65, then this is a positive change and the plot value is incremented by one unit.

While this can be done by looking at the change in calculated accuracy, it is better to detect the learning trend directly from changes to the confusion matrix as the accuracy measure may not change below 0.0 or above 1.0. That is, when the confusion matrix is

updated, the learning trend for a single player i at map index m is adjusted by:

$$LT_{Acc}(i,m) = \begin{cases} \frac{1}{MaxMapIndex} & if\ TP\ or\ TN \\ -\frac{1}{MaxMapIndex} & if\ FP\ or\ FN \end{cases} \tag{8.6}$$

where TP, TN, FP, and FN are the four possible confusion matrix results for a single prediction. The unit of change could simply be 1 for a positive change and -1 for a negative change. However, here we choose to increment or decrement with a unit of $\frac{1}{MaxMapIndex}$, where the maximum map index is where we choose to measure up to rather than the maximum maps played by a single player. This gives a ratio at each map index of how many times the classifier's accuracy has increased when compared to a perfect case. In the results in this chapter, we examine the learning trend of the first 20 maps and so a unit of $\frac{1}{20}$ is used.

It should also be highlighted that the above equation makes use of accuracy (as defined in earlier in Equation 8.2) as the basis of deciding whether to increment or decrement the learning trend. Precision, recall, or any other confusion matrix based metric could be used to give a different view of performance how the corners of the confusion matrix are being increased or decreased.

Once the learning trend for each player is calculated at each map index, the results of all players can be combined in two distinctive ways: the *mean learning trend* and the *majority learning trend*. These two variations provide different information and the combination of the two gives a deeper insight into the performance the classifier. The mean learning trend is calculated as:

$$\overline{LT(m)} = \overline{LT(m-1)} + \frac{1}{P_m} \sum_{i=1}^{P_m} [LT_{Acc}(i,m)] \tag{8.7}$$

This is similar to the mean accuracy calculation except that the learning trend for the current map index m is added to the value of the previous map index $m-1$.

The majority learning trend is calculated much the same way but instead of a mean average being used, it is instead the most frequent trend value at the current map index (the *mode*) that is used. Therefore, if the majority of players experience a positive change in accuracy, then the majority learning trend is positive as well and vice versa. If there is no majority, then the majority learning trend for that map index is set to 0.0, which neither increments nor decrements the plot.

Both learning trend metrics have linear best and worst cases. For a single player, the best case would represent an increase in classifier accuracy for every new training sample, while

the worst case would indicate opposite learning and therefore it may be better to simply switch the output from the classifier. For the mean learning trend, the best case would represent all players' classifiers experiencing an increase in accuracy for every consecutive map index. The best case for the majority learning trend, however, states that for every map index, only a majority of players had an increase in accuracy.

Learning trend addresses the issues of traditional mean performance calculations due to there being a maximum increase or decrease between each map index. Thus, as the number of players reduces, it is less likely that there will be a substantial increase or decrease in performance due to the performance of the remaining players. While the mean learning trend is still likely to have larger changes towards the end with the mean being taken from fewer players, it is now bounded to maximum of one unit change. This problem does not exist at all for the majority learning trend as it will always be either a one unit increase, one unit decrease, or remain the same.

The fraction denominator problem also has no effect on either of the learning trend calculations as the raw accuracy values are no longer used. Additionally, late learners do not have such a significant impact on learning trend as the accuracy of all players' classifiers now carry equal weight and the low accuracy values at later maps don't skew the mean and majority learning trend. Another benefit of the learning trend metric is that it can capture values below 0.0 and above 1.0. For example, consecutive false-positive results at the start of play would cause the learning trend to appropriately drop below zero while the raw accuracy score would remain at zero.

8.5 Generalizing the Methodology

The user experiment in this thesis was administered in an unsupervised public format. This was done to evaluate the proposed solutions in a natural, relaxed environment. Participants could play for as long as they wanted, at a convenient time, in anonymity, and without the pressure of performing in front of a research team. This simulates the way in which commercial games may be beta tested within the community and avoids unintentional experimental biases where possible. Such experiments are better suited to evaluating the commercial viability of a solution over structured, supervised user experiments. They also allow for data to be gathered from a larger and more diverse audience than could otherwise be recruited to on-site lab testing, especially for small research teams.

Unfortunately, this approach does reduce the amount of experimental control and leads to noisy player data. Controlled environments allow for easier evaluation of the resulting data and for more definitive conclusions because variables can be fine-tuned during participation and users can be probed for further information. However, the data is not gathered from the entirety of the user experience, which, for games, involves player's enjoying the activity as a form of entertainment rather than as a task to be completed under supervised conditions.

In the unsupervised setting, proper statistical analysis of the resulting data becomes an important means of discerning meaning from the noisy player data. In a personalized PCG solution, we are primarily concerned with detecting whether the system is providing a better experience over a random baseline, whether the players' preferences are being properly captured and utilized, and the generative range of the various PCG components. These are evaluated here by the metrics described in Section 8.3. Conveniently, the subjective player feedback can be used to not only construct a player model but also as a means of evaluating a system's performance.

Additionally, there may be an issue associated with allowing the player's to choose whether the maps were optimized or randomized. As oppose to having one group of players play only optimized maps and another group playing randomized ones, this mixes the sample populations. This has the benefit of ensuring that overly positive or negative players provide ratings and feedback on both types of maps, rather than directing their ratings solely to one or the other. This is important where the number of sample points (e.g. participants and map ratings) is not enough to even out these types of biases. We do lose a little bit of control over the experiment by allowing the player to choose what type of map they play next and so can't ensure that all players experience an even number of each type of map. However, this option is also a vital part of the systems operation by giving the player the ability to break away from a repetitive series of maps due to the optimization processes being stuck in local optimums.

Finally, administering post-play surveys in an unsupervised public experiment can be difficult, especially when the period of play is indefinite (i.e. there is no end to the game). The survey can be forced onto the player after a specified time but this invasive act may encourage a premature stop in play. On the other hand, making the survey optional avoids interruption to play but makes it easier for the player to ignore or forget it. As a design choice, we used the latter setup as the survey data was of secondary interest compared to the interaction data. Again, there is also an issue of no control group for the the survey results to be compared against because the players experienced both optimized and randomized maps

before filling out the survey. This means that we could not directly compare the survey results of the optimized maps with those of the randomized maps and so instead look at the survey data as an indicator of the entire systems performance.

8.6 Summary

- The *PCG: Angry Bots* game was released as a public user experiment. The game included the following features that were described in the last three chapters: player driven interactive evolution of interior geometry, content layout via CPPN using the player selected geometry as input, and content layout optimization through NEAT evolution and RS based player models.
- After each map, the player was able to choose whether to optimize the next map by using the solutions described in this thesis or through random generation. The mixture of optimized and randomized map ratings provided by the players of the *PCG: Angry Bots* experiment allowed us to compare our solution with a random baseline.
- To compare our deployed RS based PPM with other potential solutions, we test the prediction accuracy of six comparative classifiers that are commonly used in RS literature: a neural network, a nearest neighbor classifier, a decision tree, and a random forest classifier.
- Statistical analysis of rating data is conducted through the chi-squared measure (testing whether the rating a map received was dependent on whether it was optimized or randomized) and the Mann-Whitney U test (testing whether optimized maps received statistically better ratings than randomized maps).
- The content balance metric is the sum of pick-ups minus the sum of enemies in a room. A negative number indicates that there are more enemies than pick-ups (a more difficult map) and a positive number indicates that there are more pick-ups than enemies (an easier map). Examining how the content balance changes from room to room of a map shows the flow of challenge throughout the map.
- Mean prediction accuracy and mean 3-fold cross validation accuracy calculations are used to determine the average performance of all players' PPM and thus the performance of the RS approach in general. Precision, recall, and subsequent measures are not reported as they do not tell us anything worthwhile about the system.

- The learning trend metric was introduced to deal with three identified shortcomings of taking the average of traditional prediction accuracy measures: the reducing player count, late learner, and fraction denominator problems. This metric works by detecting a change in prediction accuracy from the previous map to the current one and then either increments or decrements a running total by one standard unit. Taking the mean or mode (majority) of this value for all players gives a more stable visualization of classifier performance in this system.

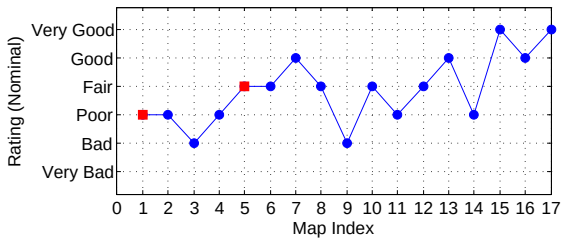
Chapter 9

PCG: Angry Bots Experiment Results

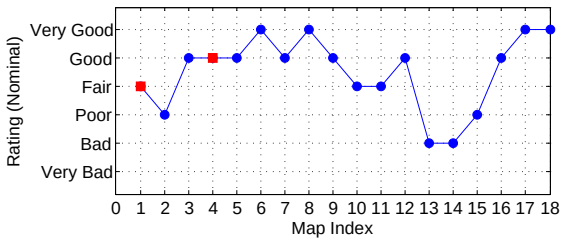
With the experiment setup and the quantitative evaluation metrics described in the previous chapter, this chapter presents the results of the *PCG: Angry Bots* experiment. Section 9.1 first gives a qualitative analysis of three example players to show the types of experiences that individual player's had. Sections 9.2, 9.3, 9.4, and 9.5 then provide the results of applying the quantitative metrics on all the collected player data to show an aggregate view of the performance of each of the solutions included in *PCG: Angry Bots*. Additionally, Section 9.6 contains results from an optional post-play survey that accompanied the *PCG: Angry Bots* game. Finally, a centralized discussion of the implications of the results is located in Section 9.7.

9.1 Qualitative Case Study Results

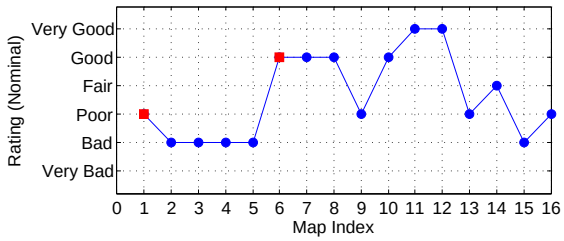
The first set of results presented here provides a qualitative analysis of the types of experiences that players had. The results in the remaining sections of this chapter attempt to provide a holistic view of the performance of the system by quantitatively analyzing the average experience of all players. However, the finer details of players' experiences cannot be averaged in such an objective way. Therefore, this section provides a qualitative view of the type of experiences that individual players had. This is done by selecting three sample players, presenting the ratings that these players provided and the performance of their respective RS player models, and discussing these in relation to renderings of sample maps experienced by each player.



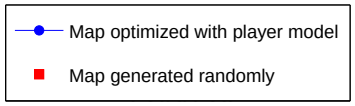
(a) Player 1



(b) Player 2



(c) Player 3



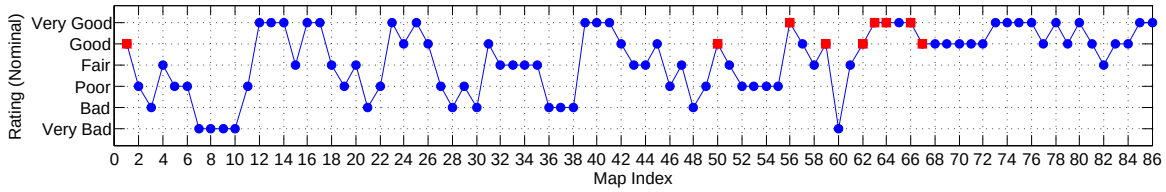
(d) Legend for rating plots.

Figure 9.1: Plots of the ratings provided by each of the three chosen sample players for every map that they played.

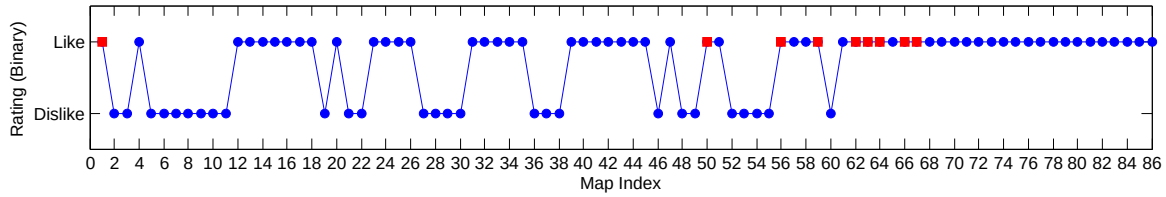
The results in Figures 9.1, 9.3, and 9.4 show three of the longest playing participants, henceforth referred to as Player 1, Player 2, and Player 3, who played 17, 18, and 16 maps respectively. These three players also demonstrate quite distinct preferences of gameplay, with Player 1 enjoying easy maps, Player 3 preferring challenging maps, and Player 2 desiring a level of challenge somewhere between that of the other two participants. It should be pointed out that as participation in this experiment is anonymous, we cannot truly know the preferences of the players and below are only our estimations. However, this highlights the strength of the system: the game designer does not need to know why a player likes or dislikes a particular map, as long as the map features used by the RS are able to encompass at least some of the reason behind a player's rating.

9.1.1 Raw Ratings

Firstly, Figure 9.1 shows the rating that each player gave to each map they experienced. The plots show the nominal rating scale that is presented to the player during the game, as oppose to the weighted binary scale used by the NB classifier. The plot for Player 1 shows a steadily improving rating over time, suggesting that the system is properly identifying his



(a) Nominal ratings



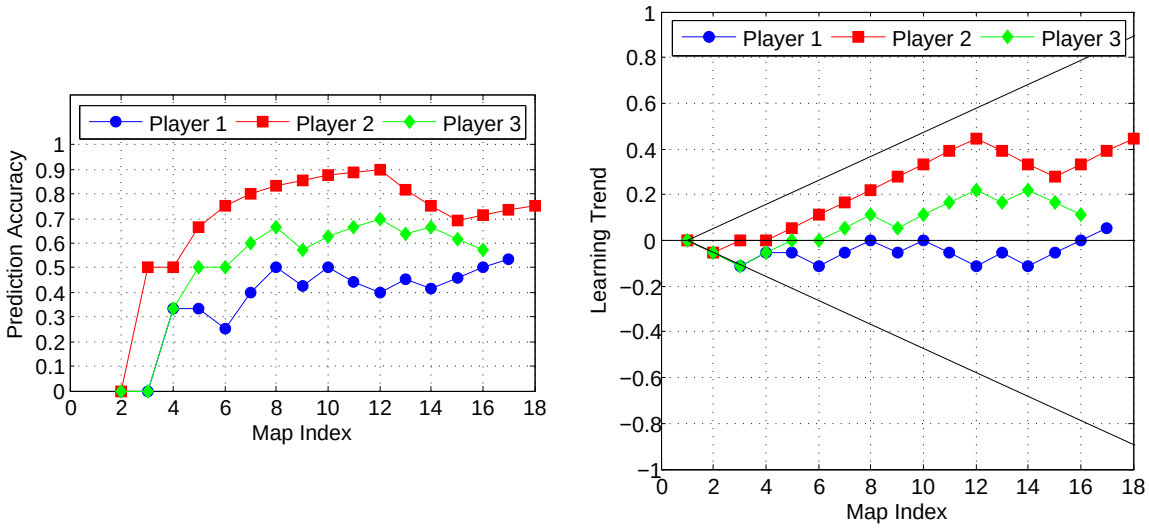
(b) Binary ratings

Figure 9.2: (a) The ratings provided by the player who experienced the most maps. (b) The same ratings converted to the binary scale used by the Naive Bayes classifier.

preferences. Both Player 2 and Player 3 show an increase in rating after a few games but a sudden drop towards the end of play. It is believed that this trend occurs for one of the following reasons.

- The NB classifier is pushing the NEAT evolution in a specific direction (e.g. adding more enemies) but has pushed the boundaries of what is acceptable too far (e.g. too many enemies have now been added).
- The geometry tree grows in depth and the CPPN population is not optimized for the lower levels of the tree.
- The player has become bored with the repetitive content layout that is a result of either the NEAT algorithm being stuck in a local optimum or an overfit RS. Alternatively, the player's skill or preferences have changed and what previously interested them no longer does.

Regardless, if a player is to continue to play beyond these few bad maps, the ratings appear to improve again shortly after the sudden drop. The most maps played by a single player is 86. Figure 9.2a shows the ratings provided by this player. When such a large number of maps are played, the trend of fluctuating ratings appears to re-occur over the lifetime of play, before remaining constantly positive after map 62. This trend is more easily noticed in Figure 9.2b, which depicts the binary ratings used by the NB classifier.



(a) (b)

Figure 9.3: (a) Prediction accuracy and (b) learning trend of the three classifiers of the chosen players.

9.1.2 Individual Player Model Performance

Figure 9.3 shows the performance of each of the three players' RS player models at each of the map indices that they experienced in terms of prediction accuracy and learning trend. The learning trend metric was predominantly designed to deal with issues involved with taking the average accuracy of many classifiers and therefore here the two plots look similar, except that the learning trend is more linear due to eliminating the fraction denominator problem discussed when the learning trend was defined.

A quick glance at these plots reveal that the classifiers of all three players make bad predictions at first but have positive learning later on. Both Player 1 and Player 3 experience fluctuations in prediction accuracy. For Player 3, this is due to a lack of a positive training example early on in play. The accuracy drops again later on in play in correlation to the drop in Player 3's ratings, indicating that the classifier may have been provided with conflicting training examples.

Player 1 has the worst performing classifier, which does not perform much better than a random classifier. This is most likely due to many of the provided ratings being either "Fair" or "Poor" and without strong sample of either "Very Good" or "Very Bad" the classifier is

struggling to learn. Player 1 does not experience a solid series of positive predictions until after Map 14.

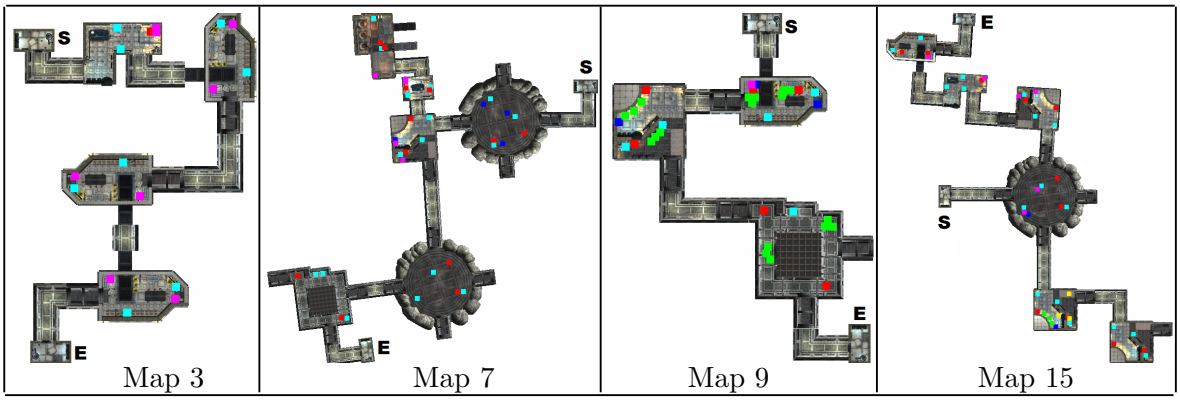
The classifier for Player 2 shows the greatest strength with a maximum accuracy close to 90% and a consistently positive learning trend early on. However, there is a drop in accuracy between Map 13 and 15 and by comparing this to the ratings in Figure 9.1 it can be deduced that the classifier had predicted that Player 2 would enjoy those maps but they didn't, introducing confusion into the training data. By Map 16, however, the classifier is beginning to learn from these new data samples and is improving in accuracy again.

9.1.3 Map Analysis

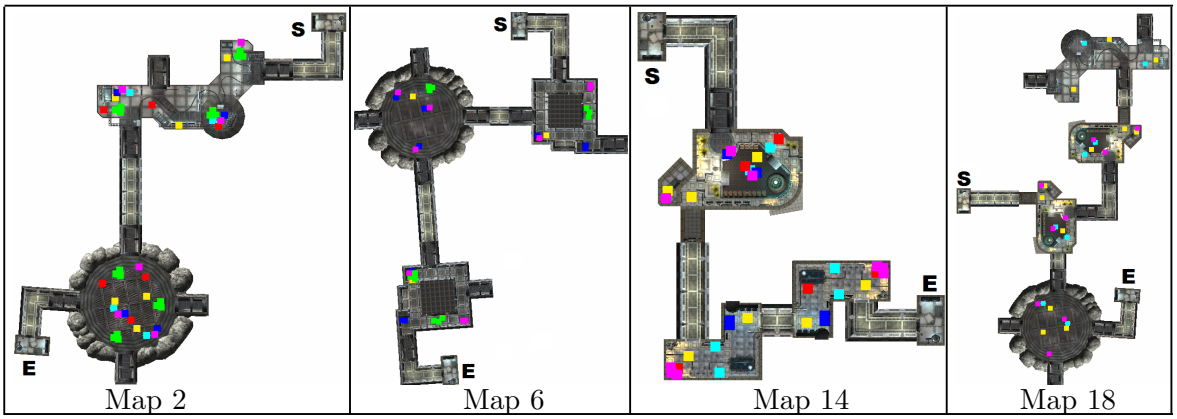
This section describes the experience of the three chosen players by showing a sample of maps that were generated for each of them. Figure 9.4d shows the color legend that is used in Figures 9.4a, 9.4b, 9.4c. Each colored square in the images of the maps represents a single piece of content. The room that the player starts in is marked with an "S" while the exit room is marked with an "E". The map index numbers match those shown in Figure 9.1.

Figure 9.4a shows four maps that Player 1 experienced. Player 1 appears to be a novice and so prefers maps with few enemies. At Map 3 there is not enough classifier data and the maps are being optimized in a negative direction. Map 3 has only a single 'Spider' enemy in the first room and while Player 1 wants an easy map, this appears to be too easy. Map 5, which is generated randomly, gathers a positive rating to balance the knowledge of the NB classifier. This is built upon in Map 7, which now has a few more 'Spider' enemies in each room. However, this trend of optimizing towards more enemies continues and as a result Map 9 is overly dense with enemies. By Map 15 the system has returned to an appropriate enemy quantity.

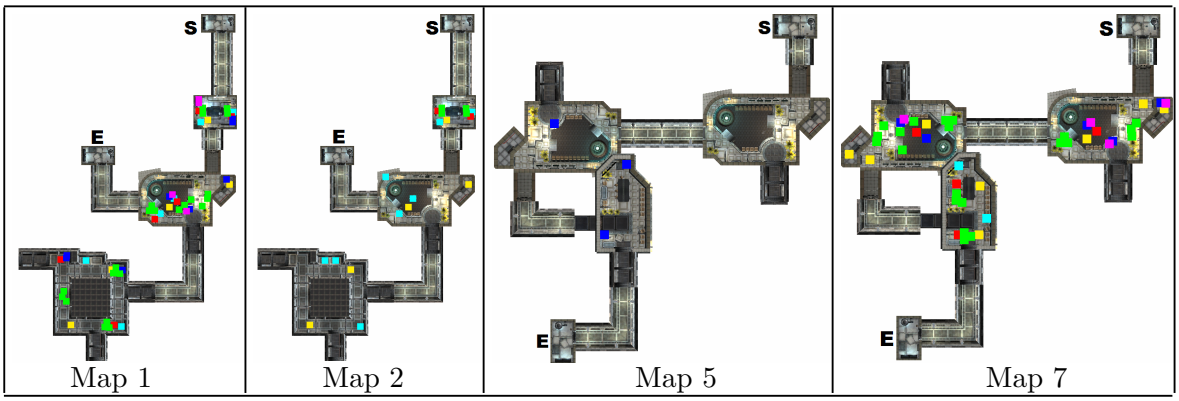
Some of the maps that Player 2 experienced are shown in Figure 9.4b. Map 2 has a high density of all content types but it is likely that there were too many enemies for the player to handle. Map 3 is optimized away from Map 2, receives a good rating, and thus continues to optimize towards fewer enemies. Map 6 shows the pinnacle of this optimization, which contains one 'Mech' enemy in each room, a handful of 'Buzz Bot' enemies, and plenty of 'Ammo'. However, this balance is broken in Map 13 and 14, which both contain higher numbers of 'Mech' enemies and 'Spider' enemies. In Map 18, the 'Spider' enemies have been removed, leaving only the 'Mech' enemies and plenty of 'Ammo' and 'Weapon' pick-ups, which seem to provide an appropriate challenge for this player.



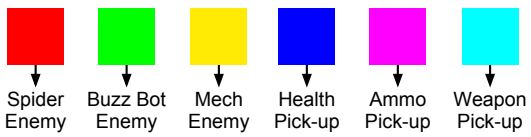
(a) Player 1



(b) Player 2



(c) Player 3



(d)

Figure 9.4: A sample of maps played by (a) Player 1, (b) Player 2, and (c) Player 3. Each colored square indicates a single piece of content, corresponding to the color legend in (d). In each map, the room marked 'S' is the starting room and 'E' is the exit room.

Player 3 appears to be a more experienced player. However, the plot for Player 3 in Figure 9.1 and the maps in Figure 9.4c demonstrate how the map optimization process can move in the wrong direction and get stuck in a local optimum early on. The first map that is shown in Figure 9.4c, which is randomly generated, is dense with enemies but comparatively scarce with ammo and health. In Map 2, a few of the enemies have been removed but so has all the ammo and health and is rated even worse than the first. By Map 5 almost all content has been removed from every room. This behavior is a result of there being no positive examples for the NB classifier to learn from. This is why the option to completely randomly generate a map was provided to the players. Map 6 is randomly generated and is rated highly. The NEAT evolution and NB classifier utilize this positive rating and the result is Map 7, which has returned to higher densities of enemies and pick-ups.

9.2 General Results

With individual case studies presented in the previous section, this section and the three that follow it present a primarily quantitative analysis. The results presented throughout the rest of this chapter show a broad view of the experiences that players had by taking the average of the full set of player data that was collected during the *PCG: Angry Bots* experiment. In this section we present general results that compare the entire personalized procedural map generation solution with random map generation. Then, in Section 9.3 we discuss findings that pertain to the geometry generation of the maps and the use of interactive evolutionary computing (IEC). The content layout results provided in Section 9.4 and the performance of the RS player models is analyzed in Section 9.5.

Table 9.1 shows general results of the experiment. In total, 147 users played one map or more. However, many users only played a handful of maps with the median number of maps played being 4. Meanwhile, the number of users that played ten maps or more was 20. A total of 893 maps were played, of which 570 were optimized and 323 were randomized. Users successfully completed 502 of those maps by reaching the exit, while the rest were skipped over. Players completed 67.89% of optimized maps and only 35.60% of randomized maps, a difference that is significant at $p < 0.01$ and with a medium Cohen effect size [Cohen, 1992] of 0.31, determined from a Mann-Whitney U test where H_0 states that the number of completed optimized maps is less than or equal to the number of completed randomized maps. This suggests that maps that were optimized were more appropriate for each player's skill and therefore more likely to be completed.

Table 9.1: General results from the user study. S_{Opt} is the set of all maps that were optimized through the combined use of interactive evolution, CPPN-NEAT, and RS. S_{Rand} is the set of all maps that were randomly generated.

Players (>= 1 Map)	147	Players (>= 10 Maps)	20
Maps Played	893	Maps Completed	502
Median Maps Played	4	Mean Maps Played	5.7
Optimized Maps	570	Randomized Maps	323
Liked maps in S_{Opt}	62.46%	Liked Maps in S_{Rand}	60.37%
Mean S_{Opt} Rating	3.87	Mean S_{Rand} Rating	3.73
Median S_{Opt} Rating	4 (Fair)	Median S_{Rand} Rating	4 (Fair)

9.2.1 Statistical Analysis

While the median of S_{Opt} and S_{Rand} are the same and the mean rating and percentage of 'Likes' appear to be similar between the two sets, there is in fact a significant statistical difference, which is discussed below.

Visual Evaluation

The two graphs in Figure 9.5 indicate the quantity of each of the ratings (on the nominal scale) within both S_{Opt} and S_{Rand}. Figure 9.5a shows that both sets have similar distributions of ratings. However, the difference between the two sets is that the ratings in S_{Opt} show a slight positive bias, with the 'Fair' and 'Good' ratings being selected most often. Meanwhile, the ratings in S_{Rand} are slightly more skewed to the negative side, with 'Fair' and 'Poor' being the most used options.

Figure 9.5b shows the percentage of maps that received a specific rating out of all maps played in S_{Opt} or S_{Rand}. More formally, as there is only a single trial of this experiment, Figure 9.5b shows the probability distributions of both sets, using the sample mean of the respective set as the population mean. This further highlights the skewed binomial distributions of both sets. Additionally, what is made clearer by this figure is that more 'Good' and 'Very Good' ratings were given to S_{Opt} than S_{Rand}, as well as slightly more 'Bad' and 'Very Bad' ratings but less 'Poor' and 'Fair'. This suggests that the optimization algorithms are generally performing better than a random baseline but that they also polarize ratings. That is, typically the system is performing well but when it is under-performing, the maps are worse than the random baseline.

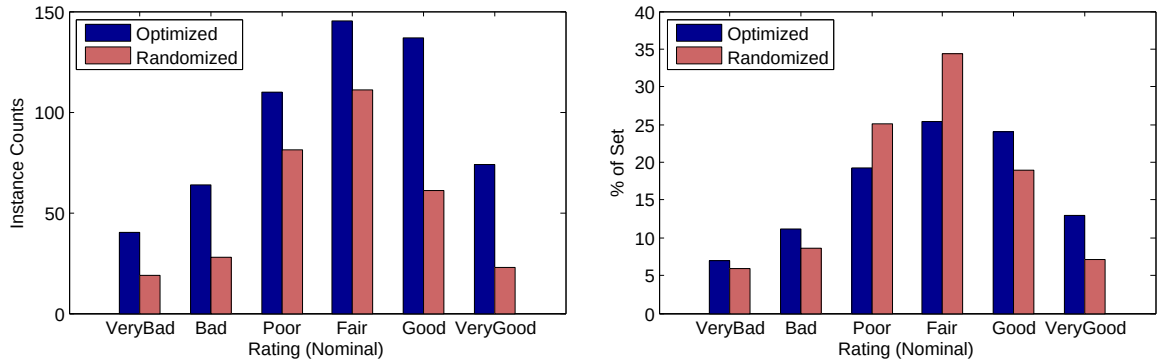


Figure 9.5: Histogram of ratings for optimized (set S_{Opt}) and randomized (set S_{Rand}) maps. (a) Count of each rating in each set. (b) The percentage that each rating makes up of each set.

Significance Testing

The following are results of statistical tests computed on the S_{Opt} and S_{Rand}. For a description of the tests and the hypotheses used, see page 145. In the χ^2 test, H_0 (the two sets are independent of the ratings they received) was rejected with $p < 0.01$, which suggests that the two sets are not related and that the ratings are dependent on whether the map was optimized or not. As one of our sample sets is a random baseline, this tells us that our algorithm is not simply generating maps at random and that the ratings they are receiving are not a matter a chance. For the Mann-Whitney U test, H_0 (S_{Opt} is worse or equal to S_{Rand}) was rejected with $p < 0.05$, indicating that our algorithm is receiving better ratings than a random generation process. The parametric one-sample Student's T-test also gives $p < 0.01$, reinforcing the same conclusion as the non-parametric U test.

Unfortunately, in both of these tests the effect size is small. For the χ^2 test, the ϕ coefficient [Cremér, 1999] is only 0.17, a small effect size when judged by Cohen's effect size bands [Cohen, 1992]. Meanwhile, using the Mann-Whitney effect size estimate suggested by Grissom and Kim [2005], if an optimized map and a randomized map were chosen from their sets at random, there is a 0.54 probability of the optimized map having a better rating; only slightly better than an equal chance. Thus, there is room in future work to improve upon these implementations.

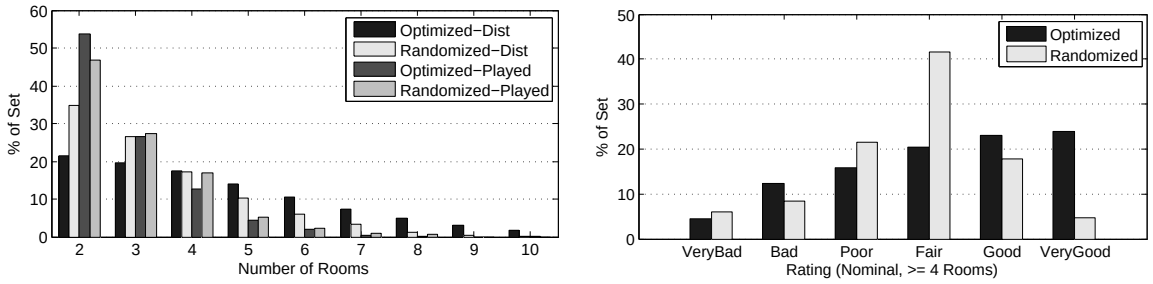
Additionally, it is possible to perform statistical analysis on the binary rating system. Here, the χ^2 test was used but it failed to reject H_0 with $p > 0.05$. By looking at the

probability distributions in Figure 9.5b, it can be seen that this is due to the large proportion of 'Fair' ratings for randomized maps. This result confirms what we believed when we designed the rating system; that a binary rating system of {Dislike, Like} does not properly capture the finer details of a player's preferences.

9.3 Geometry Results

Figure 9.6a shows the distribution of the number of rooms in maps created from mutation ('Optimized') and random generation ('Randomized'), taken from 100,000 valid maps generated through each technique. Small maps are favored during both mutation and random generation because validation fails more often as maps get bigger. However, mutated maps are more likely to be successfully validated. For random generation, validation succeeded 37.07% of the time for maps with only 2 rooms, 7.06% of the time for maps with 5 rooms, and only 1.8% of the time for maps with only 10 rooms. The success rate was higher for mutation with 83.28%, 43.42%, 6.59%. Mutation is also slightly more inclined towards growth due to the branching process of the addition operator and so when combined with the validation success rate produces a more even distribution in small map sizes than random generation. If more large maps were desired, the mutation probability could be adjusted to favor the addition operator even more or the branching rate could be forced to higher ranges.

Figure 9.6a also shows the distribution of geometry sizes that were actually played during the experiment. Despite being given opportunities for larger maps in most IEC cycles, players chose shorter maps more often, with a strong emphasis on two room geometries. Oddly though, this result conflicts with the responses to a survey question that asked how large a respondent's ideal map would be. Only 4.76% of respondents said their ideal map would be small (2 to 4 rooms), while the remainder of players' ideals were close to evenly split between medium (5 to 7 rooms) and large (more than 7 rooms) maps. Figure 9.6*b* supports this survey results by showing the rating distribution for maps with four or more rooms (113 optimized maps and 84 randomized maps). Larger optimized maps were more likely to be rated well rather than badly and received a higher proportion of 'Good' and 'Very Good' ratings than randomized maps. For maps with three rooms or less, the rating distrubtion closely resembles that of the histogram earlier presented Figure 9.5.



(a) (b)

*Figure 9.6: (a) *-Dist: The percentage of maps with the specified number of rooms given* $100,000$ *valid mutated and randomly generated geometries. *-Played: The percentage of maps with the specified number of rooms that were played. (b) The rating distribution of maps that had* 4 *or more rooms.*

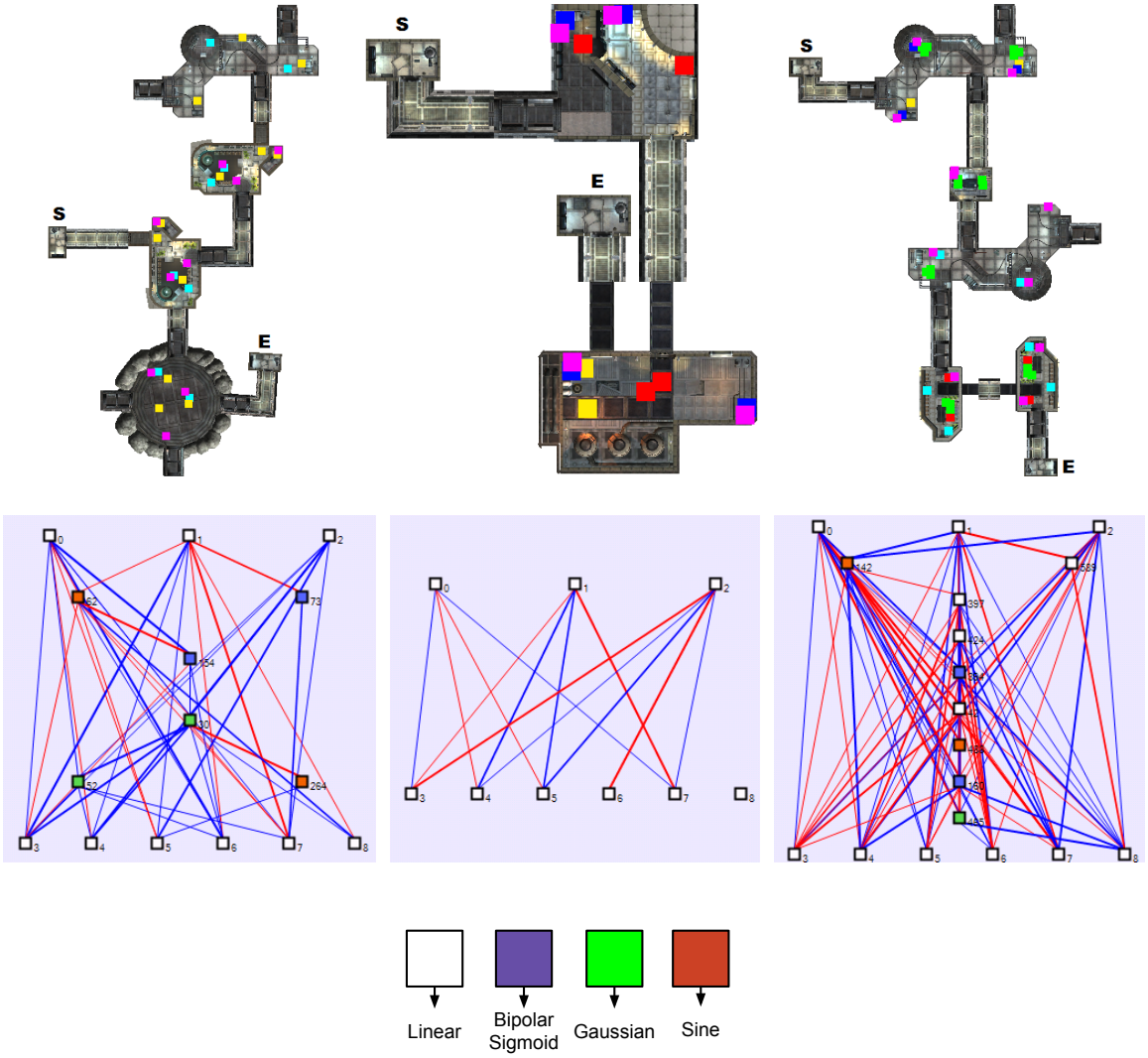
9.4 Content Layout Results

This section provides both qualitative and quantitative findings regarding the CPPN-NEAT approach to content layout. We first show examples of the CPPNs that were used to generate the content layout and look into any statistical significance between the CPPN content layout representation and the fixed n-ary tree geometry representation. Then, we examine how much of the feature space of enemies and pick-ups the CPPN-NEAT approach was able to explore. Some of these results are due to the CPPN-NEAT approach, while others are due to the performance of the RS player models; however, in both cases, the process of content layout optimization is being analyzed. Finally, patterns of content throughout all maps are examined to see if this approach created a variety in gameplay pace for the players.

9.4.1 CPPN Examples

Figure 9.7 shows three examples of the kind of maps that were generated for players and the CPPNs used to calculate the content within them. Additionally, all these maps were also generated for a different player and were rated 'Very Good'.

The similar structure of the geometries and their corresponding CPPN in Figure 9.7 is a coincidence and there is no statistical correlation between the complexities of the two representations. Similarly, there is also no correlation between the player's rating of a map and the complexity of the CPPN used to generate it. Thus, it is just as likely for a basic CPPN to generate an appropriate content layout in a large geometry as it is for a complex CPPN to be evolved for a small geometry. What this implies is that the complexification



(a)

Figure 9.7: Three example maps that were rated 'Very Good'. Shown with each map is the corresponding CPPN that was used to generate the content layout. A red line is a positive weight, a blue line is a negative weight, the thickness of a line indicates the magnitude of the weight, and the color nodes indicate the activation function (see (a) for legend).

process of CPPN-NEAT is not being utilized to its fullest potential, which may be a result of the discrete input and output of the CPPN representation.

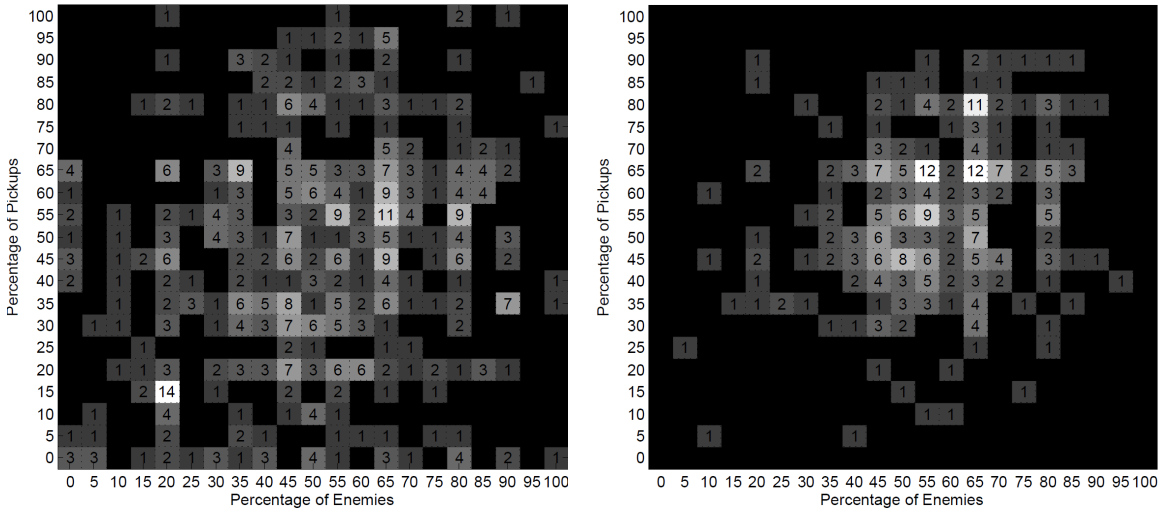
There is a statistical correlation between the length of play (map index) and the complexity of the CPPN, with a Pearson's linear correlation coefficient of 0.73, a Spearman's rank correlation coefficient [Hogg and Craig, 1994] of 0.58, and both having a statistical significance of $p < 0.001$. It could be hypothesized that the processes of complexification is responding to the complexity of the player model rather than the complexity of the geometry. However, it may also signal an unaddressed issue of bloat within the NEAT evolution if the complexity of the CPPN is not warranted. Bloat is the process of rampant growth in evolutionary candidate genomes that is a result of addition crossover and mutation operators being inadvertently favored over subtraction operators.

9.4.2 Simplified Feature Space Coverage and Convergence

The figures in this section reduce the original 18 features used for the RS player model learning and prediction of content layout preferences down to just two features: the percentage of enemies in a map and the percentage of pick-ups. Due to the fact that a maximum quantity of each content type is set in each room template, it is possible to calculate the maximum quantity for a given map geometry. Therefore, the axis values in the plot are the percentages of those maximums for a given map.

Figure 9.8 shows heat maps for all optimized maps and for all randomized maps in the feature space of percent of enemies versus percent of pick-ups, grouped into 5% bins. These are the percentages of the number of enemies or pick-ups in a map out of the maximum allowable number of enemies or pick-ups in that map. These images are similar to the expressive range metric proposed by Smith and Whitehead [2010] in that they show how much of the feature space was explored, though we only show maps that were experienced by players rather than all maps found during evolution.

What is immediately obvious from these heat maps is that the CPPN representation is allowing for more exploration of this solution space. With randomized maps, while the setting of each content type in each room of a map is uniformly random and independent, the sum of the settings across the map results in the Gaussian distribution shown in Figure **??**. The CPPN representation, however, is generating more maps that have consistently lower or higher levels of content in every room, thus producing more consistent patterns of content throughout a map. To improve the random baseline in future work, it would be beneficial



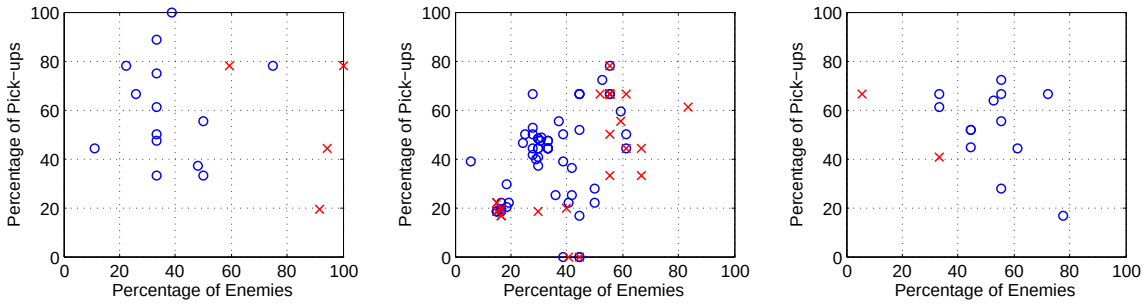
(a) Optimized maps (b) Randomized maps

Figure 9.8: Heat-maps of all (a) optimized and (b) randomized maps that were played, plotted in the feature space of percent of enemies against percent of pick-ups. All black areas indicate that no maps were played with the corresponding enemy and pick-up percentages. All values indicate the number of maps played with the corresponding content percentages.

to first randomly generate a maximum content density for enemies and pick-up and then randomly activate content in the rooms of geometry until this maximum density is reached.

For a more local view of the feature space exploration, Figure 9.9 shows the types of maps that were generated for a sample of players. The three plots each represent a different player. Each point in a plot represents a map, with a blue circle indicating a map that was liked by the player and a red cross representing a map that was disliked. Some players show a disliked map and liked map occupying the same feature coordinates. This may be due to the features being simplified for visualization or may represent an actual contradiction in the player's ratings for similar maps.

Despite these occasional rating clashes, the plots show clear clusters of liked maps or cut-off points for which maps became unacceptable. For example, the player represented by Figure 9.9b shows a strong cluster of liked maps around the 50% pick-ups to 30% enemies region. Meanwhile, the player represented by Figure 9.9a appears to not enjoy most maps with more than 60% enemies, regardless of the amount of pick-ups in the map. For the maps played by the player in Figure 9.9c it is possible to linearly separate positive and negative maps; whether this characteristic would hold if the player had experienced more maps is unknown.



(a) (b) (c)

Figure 9.9: Each plot represents a single player. Each point in the plot is a map that the player experienced in terms of the percentage of enemies versus the percentage of pick-ups within the map. Blue circles are maps that we liked while red crosses are maps that were disliked.

The fact that these types of clusters of liked maps are forming, albeit in a simplified feature space, indicates that the RS is learning the preferences of the player and is pushing the CPPN-NEAT evolution toward content layouts similar to those that were previously enjoyed by the player. Although it is harder to visualize, it is expected that these types of clusters may appear in the original 18 dimensional feature space that the maps are classified on.

Figure 9.10 shows the simplified feature space, calculated the same way as above, but only for Player 1 from the earlier case studies. Instead of plotting the feature space as before, Figure 9.10 instead plots each feature over time (map indices). By doing this, an example of feature convergence, due to the NB classifier learning, is made apparent. The quantities of the two content types begin to stabilize in the regions between Maps 5 and 7 and then again between Maps 10 and 14, especially with respect to enemy quantities. This indicates that the classifier has learned an enemy to pick-up ratio that is believed to be suitable for the player. However, the features begin to fluctuate again in Map 15 and this is due to the sudden drop in rating of Map 14. A re-rendering of Map 13 and 14 (not shown) indicate that the content settings were nearly identical but, for their own reasons, the player chose to rate Map 14 much worse. This contradicted the classifiers prior knowledge of the player and therefore causes it to promote adjustments in feature values.

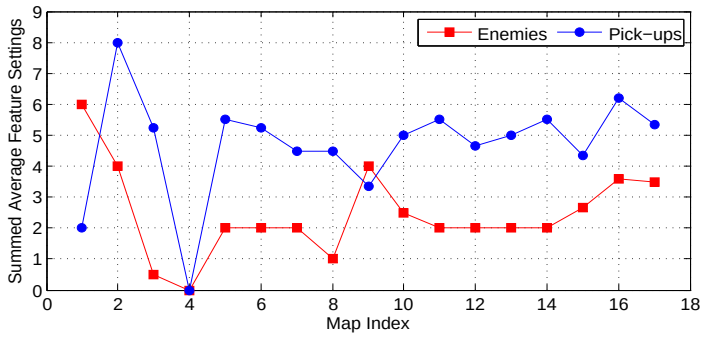


Figure 9.10: Feature values for the maps played by Player 1. The feature values are summed into enemies and pick-ups for clearer viewing.

9.4.3 Patterns of Content

When considering the room composition features used by the RS, there is an even balance of maps that were generated with all low enemy rooms, all high enemy rooms, or a mix of low and high enemy rooms. A similar balance exists for pick-ups. Maps that were rated 'Very Good' typically had all low enemy rooms and a mix of low and high pick-up rooms, indicating that players enjoyed having more pick-ups than enemies and challenge was introduced by reducing the amount of pick-ups rather than increasing the amount of enemies. Maps that were rated 'Very Bad' were less likely to have a mixture of low and high rooms for both enemies and pick-ups, suggesting that maps that had a constant difficulty were not enjoyed.

The brief analysis above uses high level features that indicate that there were not many maps that had fluctuations of content. However, more distinct patterns can be exposed by looking at the lower level features. This is done through the use of the content balance metric described in Chapter 8.

Table 9.2 shows the four mentioned difficulty trends cross tabulated with the ratings given to each map. This only includes optimized maps that had more than two rooms between the start and end of the map, as the fluctuating difficulty trend is not possible with only two rooms. Thus, a total of 229 maps were evaluated in this table. The final column shows the ratings given to maps with only two rooms. Additionally, the final row of the table shows the percentage of maps with the given trend in S_{Opt}.

Of interest in these results is that 77.96% of maps with fluctuating difficulty were liked, with a rating of 'Fair' or better. Additionally, 23.73% of fluctuating maps were rated 'Very Good'. Both of these are higher proportions than any other difficulty trend. In a Mann-Whitney U test where H_0 is that fluctuating maps receive worse or equal ratings than the

Table 9.2: Difficulty trend of all maps cross tabulated with the rating they received. Values are shown as a percentage of all maps in the respective difficulty trend.

	Increase	Decrease	Fluctuate	Constant	2 Rooms
Very Bad	5.13%	7.02%	5.08%	11.43%	7.33%
Bad	6.41%	22.81%	8.48%	14.29%	10.56%
Poor	23.08%	19.30%	8.48%	14.29%	20.82%
Fair	29.49%	24.56%	22.03%	11.43%	26.69%
Good	20.51%	12.28%	32.20%	40.00%	23.75%
Very Good	15.38%	14.03%	23.73%	8.56%	10.85%
% of S_{Opt}	13.68%	10%	10.35%	6.14%	59.83%

other three difficulty trends, the null hypothesis was rejected at $p = 0.003$ and if one map of each was drawn at random there is a 0.62 probability of the fluctuating map having a better rating; this was not the case for equivalent tests for any of the other difficulty trends. This indicates that maps that have fluctuating difficulty are more likely to be enjoyed.

Following from this, maps with increasing difficulty are the second most likely type to be enjoyed, with a total of 65.38% of those maps being liked. Next were maps with only two rooms, with 61.29% of these small maps being liked. Also, as was highlighted in the previous section, despite the moderate ratings and the responses to the user survey, maps with only two rooms were played much more often than larger maps, making up nearly 60% of all ratings. Out of those maps with three or more rooms, the distribution of difficulty trends that were played is relatively even. Finally, while the constant difficulty trend had the highest proportion of 'Good' ratings, it also had the lowest percentage of 'Very Good' ratings and had the second highest proportion of maps that were disliked (59.99% liked), with maps with decreasing difficulty having the worst ratings overall (50.87% liked).

9.4.4 Applying a CPPN to Multiple Geometries

Finally, Figure 9.11 shows the effect of applying an optimized CPPN to many geometries. Each box represents a room and the coordinates at the top of each box are the node coordinates from its respective geometry tree. As the first room will always have coordinate $(0.0, 0.0)$, all maps have a common start. The first box states the settings for each of the six content types. The other rooms indicate which content types changed (abbreviated to the first two characters) and how they changed since the previous room, either increasing in quantity or decreasing. Also listed at the bottom of each box is the content balance of the room, calculated as mentioned earlier.

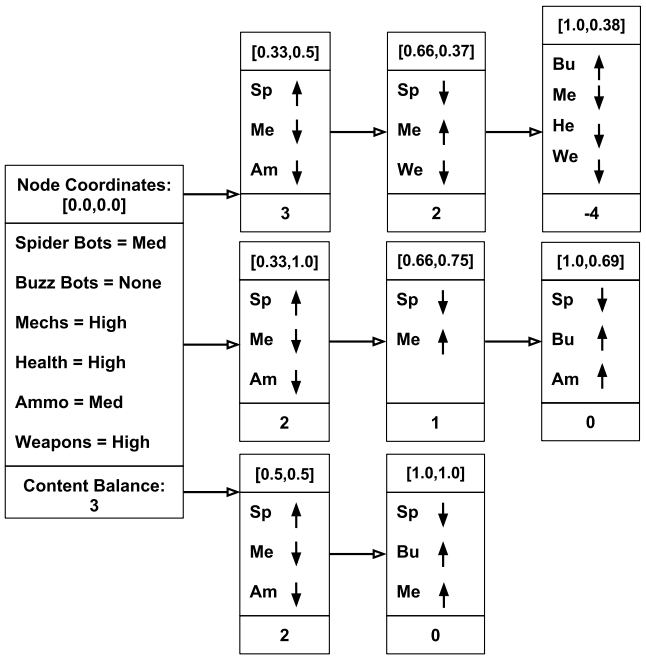
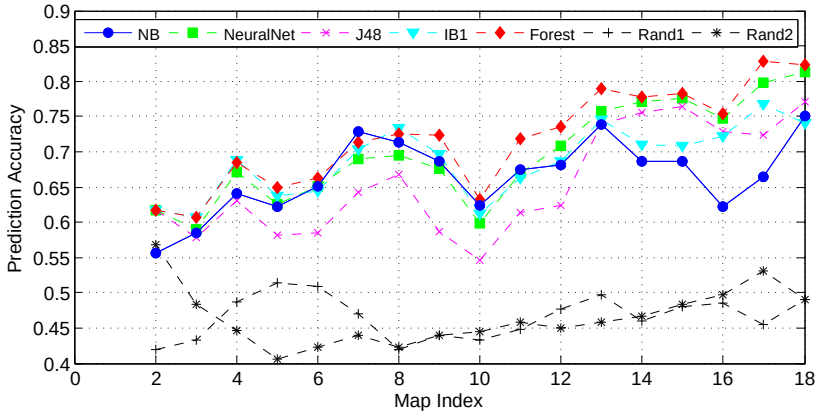


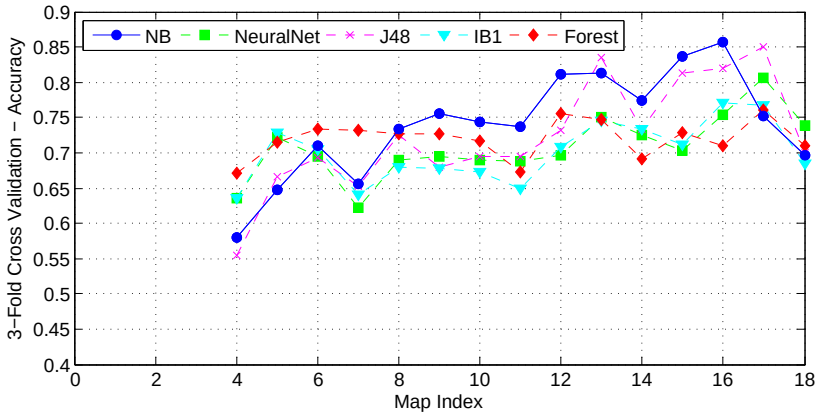
Figure 9.11: The results of applying the same CPPN to three different geometries. Each box represents a single room, with the node coordinates, changes to content from the last room, and the content balance of the room displayed. All three maps have an increasing difficulty.

What this figure shows is that applying the same CPPN to different geometries can give similar experiences due to its pattern-producing properties. The first map at the top of Figure 9.11 was rated 'Very Good' by a player, while the geometries of the other two maps were randomly generated. All the maps have a similar change between the first and second rooms, with an increase in 'Spider Bots' and a decrease in 'Mechs' and 'Ammo'. All the maps also have a decrease in 'Spider Bots' and an increase in 'Mechs'. Also, the last room of all three maps has an increase in 'Buzz Bots'. This shows that despite the different coordinates, a similar experience is being generated for all three maps, with slight variations especially towards the ends of the maps.

It is typical for a CPPN to produce a similar difficulty trend on most geometry trees, as shown here with all three maps having an increasing difficulty. However, this is not always the case and the CPPN used in Figure 9.11 also occasionally produced maps with a fluctuating difficulty, though the change in content settings were much the same and the content balance values were in a similar range. Finally, this diagram only shows the direct path between the start room and exit room and not any branching paths. Assuming two geometry trees have the same maximum depth, branches of these two maps would be similar to each other.



(a)



(b)

Figure 9.12: The mean classification scores of NB and comparative classifiers at each map index. (a) Mean prediction accuracy (S_{Opt} only) and (b) mean accuracy using 3-fold cross validation (both S_{Opt} and S_{Rand}). Data from all players was included except for that of Player $\varepsilon 1$ and Player $\varepsilon 2$.

9.5 Player Model Results

In this section we review the performance of the RS per-player preference models. This is done by evaluating how well the trained models were able to predict the players' preferences. This is done using the accuracy, 3-fold cross validation, and learning trend metrics described in Chapter 8. Additionally, we discuss why two outlier players were excluded from the accuracy and 3-fold cross validation calculations, which further motivates the need for the learning trend metrics.

9.5.1 Mean Prediction Accuracy and Cross-Fold Validation

Figure 9.12 shows the mean accuracy and mean 3-fold cross validation values for the first 18 maps for the various classifiers. The means are calculated from all players who played more than two maps, except for two. The reason for excluding two players from these plots is detailed in the next subsection. The accuracy calculations also only take into account S_{Opt}. Thus, we are only measuring the accuracy of the maps that were recommended to the player. As the first map that is generated is always randomized, these plots show prediction results starting from the second map index. For 3-Fold Cross Validation, however, both S_{Opt} and S_{Rand} were used in order to properly represent a classifier's complete knowledge at any given map index.

Figure 9.12a shows that all classifiers experience a steep improvement in accuracy early on. The NB classifier compares well with the other classifiers at this stage. However, the mean accuracy of NB drops after 15 maps while the other classifiers generally continue to improve. The classifiers are all performing better than both runs of the random classifier, which both appropriately have accuracy around 0.5 for all map indices. This suggests that if the random classifier were used as the RS then the user would have a positive experience only 50% of the time.

Figure 9.12b shows the results of the 3-fold cross validation. The NB classifier typically generalizes better than the other classifiers for most training set sizes. This is similar to the results in Figure 6.2 in Chapter 6 that lead to the decision to use the NB classifier in this experiment. However, in comparison with the earlier discussed accuracy plot, it may be that generalization is not needed in this system and a classifier that is overfitting is not necessarily a bad outcome. However, this would be in contrast to other results in this chapter that suggest that repetitiveness and the subsequent lack of variety in player experience is an issue within the system. The random classifier is not included here as unrelated predictions would be made in each fold of a single test.

9.5.2 Filtering Skewed Data

This is a declaration and a discussion of why two players were removed from data set before the average prediction accuracies were calculated. These two players are in fact the two players that experienced more maps than any other player, with the longest having played 86 maps (henceforth referred to as Player $\varepsilon 1$), and the second longest having played 23 maps (henceforth Player $\varepsilon 2$). However, both of these players experienced a poor early game,

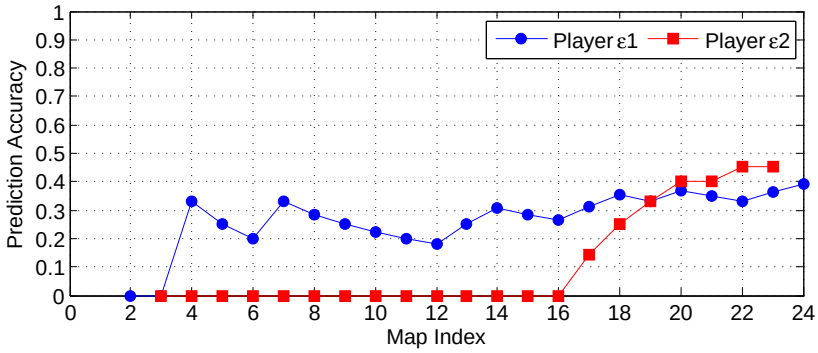


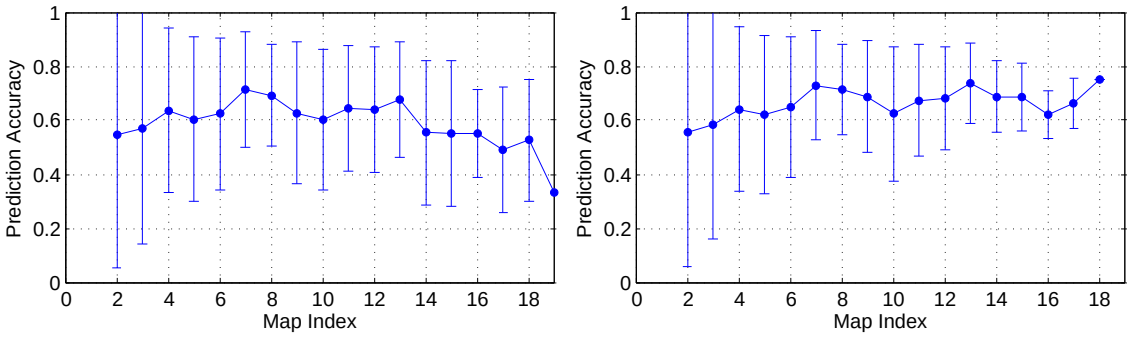
Figure 9.13: The prediction accuracy of the first 24 maps of the two players that caused a skew in the data.

with Player $\varepsilon 1$ rating most early maps negatively and Player $\varepsilon 2$ over utilizing the option to randomize maps, thus not allowing CPPN-NEAT and the PPM to effectively explore the solution space. The two next longest playing participants both played 18 maps, hence many of the prediction accuracy plots reported in this chapter have a maximum map index of 18.

Additionally, both players $\varepsilon 1$ and $\varepsilon 2$ responded to the optional survey and both identified themselves as being inexperienced game players. Upon re-rendering the maps that the two players experienced (not shown here), it appears that they have both provided a few contradictory ratings at the start of play. That is, similar maps were given opposing ratings, leading to contradictory learning samples for the classifiers. This may be due to these novice players developing their preferences and skills in the first dozen maps.

This results in a poor prediction accuracy of both players' classifiers early on in their play time. This can be seen in Figure 9.13, which plots the NB classifiers' accuracy for both players. This is especially true for Player $\varepsilon 2$ in which the classifier predicts the exact opposite of the player's actual rating for all S_{Opt} maps in the first 15 maps. A similar effect occurs with all other comparative classifiers used in the accuracy plots, suggesting that the players are in fact supplying contradicting ratings that are causing learning troubles for all classifiers in varying degrees of severity.

Despite this, in their survey responses, both players stated that their experience did in fact improve over time, which can be seen through the improvement in prediction accuracies towards the end of the plot in Figure 9.13. However, as all other players experienced 18 maps or less, this late improvement does not stop the data of these two players from skewing all other player data. Thus, these two players are extreme cases of the late learner problem



(a) (b)

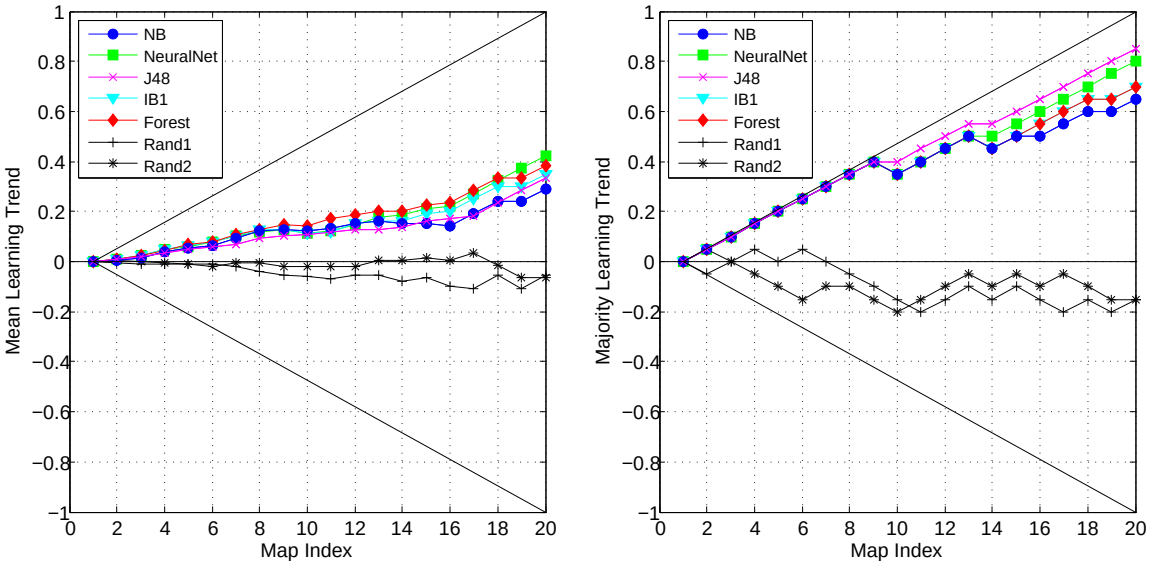
*Figure 9.14: The mean prediction accuracy between map index 10 and 17 of: (*a*) all players that played 10 or more maps and (*b*) the same plot excluding players $\varepsilon 1$ and $\varepsilon 2$.*

that motivates the use of the learning trend metrics. Therefore, the data from both of these players is included in the learning trend results in the next subsection.

Figure 9.14a shows the mean prediction accuracy among all players, including players $\varepsilon 1$ and $\varepsilon 2$, as well as the standard deviation of each map index. Figure 9.14b shows the same plot but without the two earlier mentioned players. This second plot is the same as that in Figure 9.12a but only showing the NB accuracy. Notice that in Figure 9.14b the standard deviation begins to narrow as more maps are played, indicating that the prediction accuracy of each player that played for that long is converging to the 60% to 70% region. However, this is not the case in Figure 9.14a, where the standard deviation remains large and therefore showing that the prediction accuracy of players $\varepsilon 1$ and $\varepsilon 2$ differs greatly from the general population.

9.5.3 Learning Trend

The results in Figure 9.15 show the mean and majority learning trends respectively. In these figures, the diagonal black lines represent the best and worst case scenarios. Again, only predictions made on S_{Opt} maps are included in the calculations. Map index 1 will always have a learning trend value of 0.0 because the first map was randomized for every player and so the first change in prediction occurs at map index 2. These figures reinforce that the NB classifier performs well when compared to other classifiers with low numbers of sample data. However, both plots show that as more training data becomes available, almost all the other classifiers are performing better. Thus, in future work, it may be beneficial to adopt one of the other classifiers as a baseline comparison for more advanced RS techniques. In both



(a) (b)

Figure 9.15: Results of the learning trend metric. (a) Mean learning trend. (b) majority learning trend. Both plots include players $\varepsilon 1$ and $\varepsilon 2$.

figures, the two runs of the random classifier stay close to 0.0, indicating that the majority of predictions are only accurate half of the time.

An interesting result from comparing both of these figures is that the clearly best performing classifier for the majority learning trend in Figure 9.15b is the J48 decision tree. However, Figure 9.15a shows that the mean performance of the J48 decision tree is typically below that of the other classifiers. This indicates that while it is more likely for the J48 to improve in accuracy for the majority of players, the other classifiers typically have an increase in performance for more players (that is, the majority set is larger).

Both the random forest and the neural network classifiers are performing well in both these learning trend plots and the earlier accuracy and cross validation plots. However, the downside to the use of the neural network is that time requirements for training the classifier sharply increase with the number of training samples. As the classifiers are re-trained with every new map rating, if a player is to experience more than 15 maps in *PCG: Angry Bots*, the neural network training time becomes unacceptable for the real-time response expected by the players. Thus, our recommendation for future work is to use a random forest classifier in the RS player model and not the current NB classifier.

Table 9.3: The results of the survey question asking respondents to rate their experience towards the end of their play time as compared to the start.

	Number of Respondants	
Answer	All Responses	Played >= 7 Maps
Much Worse	0	0
Worse	4	0
About the Same	10	1
Better	22	9
Much Better	6	6
Total	42	16

9.6 User Survey Results

As well as playing the game, participants of the *PCG: Angry Bots* experiment were given the option to complete a survey[1] on their experience. A total of 42 players responded to the survey and the results are presented here as additional evidence of the system's performance.

One of the most important questions of the survey asked participants to rate their experience towards the end of their play time as compared to their experience at the beginning. The first column of Table 9.3 shows the possible answers to that question. From this group, 68% of respondents said that their experience got either better or much better. The majority of the remainder said that their experience didn't change much and only four respondents said it got worse. No respondents said that their experience got much worse.

The positive bias of these responses matches that of the rating histogram for optimized maps in Figure 9.5a and the responses themselves are statistically significant. With the responses converted to a numeric scale of 1 to 5, the non-parametric one-sample Wilcoxin test [Wilcoxon, 1945] is used to test against an expected median. For this test, a median of 3 is used, with H_1 stating that the median of the responses is greater than this hypothesized median. Here, $p < 0.001$, indicating that the responses are significantly better than 'About the Same'. The ϕ coefficient here is 0.65, indicating a strong result when compared with Cohen's effect size bands.

It is also worth noting that many of the respondents that answered that their experience got worse or stayed the same did not play many maps. The third column of Table 9.3 includes only respondents who played seven or more maps. All the respondents that indicated that the game got worse have been eliminated and all but one of the "About the Same" respondents

[1] A copy of the survey questions can be found in Appendix C

Table 9.4: The reported change in difficulty vs. the reported change in enjoyment.

	Harder	Same	Easier	Total
Much Worse	0	0	0	0
Worse	2	2	0	4
About the Same	0	7	3	10
Better	4	5	13	22
Much Better	1	1	4	6
Total	7	15	20	42

have also been removed. Alternatively, none of those that stated that their experience got much better have been removed.

Participants were also asked to describe the difficulty of the maps towards the end of play as compared to the maps at the beginning. Table 9.4 shows the results of this questioning, cross-tabulated with the results of the experience question discussed before. Of interest is that most of the participants who said there was no change in enjoyment also said that there was no change in difficulty. Meanwhile, the majority of participants that said the maps became better also said that they became easier.

When asked about the variety of the maps that were played, 75.62% of respondents indicated that the variety was fair or worse. There are two reasons why this may be; firstly, it may simply be due to the re-use of the same enemies, pick-ups, and objectives throughout the game. Alternatively, the maps that the respondents experienced were much the same due to the RS being stuck in a local optimum, thus highlighting a deficiency in the RS approach. Once a player's classifier has enough training data to make accurate predictions on candidate maps, it is quite easy for the CPPN-NEAT evolution to get stuck in a local optimum and thus continuously provide maps that are similar to one another. This is what causes the clusters of liked maps in the feature space representations in Figure 9.9. This is a common issue in the RS field and is referred to as overspecialization [Adomavicius and Tuzhilin, 2005].

Respondents were also asked to describe their ideal map in terms of the amount of each content type in the map. Many respondents desired a high density of the easy to kill 'Spider Bots' and 'Buzz Bots' but wanted very few 'Mechs'. For pick-ups, almost all players wanted large quantities of 'Ammo' but the same desire wasn't shown for 'Health' and 'New Weapon' pick-ups. These results match our expectations from the game design process. It was expected that most players would struggle with the 'Mech' enemies and it was a purposeful design choice to make 'Ammo' a scarce commodity. This was done as an attempt to see if

maps would be generated to provide high quantities of ammo to meet the requirements of the player.

When the respondents were asked whether the best maps they played matched their described ideal map, 45.95% of respondents said that they were similar. Meanwhile, 32.43% said that the best maps that they experienced were neither similar nor too dissimilar to their ideal. This indicates that the system is coming close to generating maps that align with a player's preferences but still falls short quite often. However, when comparing this to earlier results that suggest the experience improved for many players, it may be the case that always giving the player their ideal map is not the best course of action and providing them with experiences outside of their comfort zone can improve the overall quality of the game.

9.7 Discussion

In this section we make a summary of the key findings and their implications regarding the geometry generation, content layout calculations, and player preference modeling techniques used in *PCG: Angry Bots*. Some of these insights are about the solutions used while others are curious observations about the players themselves. Each of these discussions leads into potential avenues for future work which are presented in the conclusion of this thesis in Chapter 10.

9.7.1 Fixed n-ary Tree Geometry

The combination of the fixed n-ary tree genetic representation of the geometry, the mutation operators used, the inverse mutation rate, and the validation process encourages a more appropriate distribution of map sizes in comparison to random generation, in relation to survey feedback that revealed players enjoy medium to large sized maps. However, a curious result of this experiment is that, despite this survey result and players having plenty of opportunities to choose larger maps through IEC, they instead mostly chose maps with only 2 rooms. It may be that players chose shorter maps because there was a better chance of the player completing a shorter map due to overall fewer potential enemies. This would hint at the players not trusting the optimization process to create appropriate content on a larger map, and therefore could be seen as a limitation of the system.

Alternatively, as players were aware that they were participating in an experiment, it may be that players wanted to complete as many maps as possible in a perceived sense of obligation to provide as much data to the experiment as possible. Playing smaller maps

would allow the player to do this in a shorter amount of time. When combined with the survey result that found that player's perceived a lack of variety in the game, it could be predicted that a more engaging game would incentives larger maps as the pleasure of playing the game is its own reward. However, both of these possibilities for the dependencies in map data and player self-reporting are just conjecture as we have no means of proving either at this time.

If the experiment could be repeated (or in future experiments) it would be worthwhile enforcing a bigger minimum size for the geometry. In the current implementation, mutation of geometries could not produce a geometry with less than two rooms between the start and exit room. However, as the number of rooms in a map expands, there is potential for more interesting patterns of content throughout the map.

9.7.2 CPPN-NEAT Content Layout

The CPPN-NEAT approach to laying out content explored more of the solution space of potential enemy and pick-up layouts than random generation by creating more coherent patterns of content throughout a map. Meanwhile, By cross tabulating the rating of a map with the change in content balance between the rooms in the map, it was possible to see that maps with fluctuating difficulty were preferred by players while maps with a constant difficulty were the least enjoyed overall. This aligns with map design theory that promotes sequences of high intensity and low intensity gameplay to set the pace of the game and prevents the player from becoming physically and psychologically exhausted [Adams, 2010].

However, while the constant difficulty was disliked the most overall and had the lowest percent of 'Very Good' ratings, it did have the highest percent of 'Good' ratings. This may because if a constant difficulty is assigned throughout the map and that difficulty matches the player's skill, then the map is acceptable, though maybe not as entertaining as a well-executed fluctuating map is. While increasing and decreasing difficulty trends give a variety of content throughout the map like the fluctuating difficulty trend does, the linear nature means that either the start of the map or the end of it will provide the highlight of the player's experience (intense action) and the progression through the rest of the map is seen as a tedious traversal of space. Meanwhile, in the best case of the fluctuating trend where two enemy rooms are punctuated by a pick-up filled room, the player is given multiple high intensity situations with the chance to prepare for the each encounter.

Additionally, once a suitable CPPN had been found it was capable of providing similar experiences over a variety of geometries. Essentially, after a player has experienced a few maps and played at least one map that they rate 'Very Good', there are two models of this individual player: the elite CPPN, which can be used as a generative model to produce additional suitable maps, and the content-based RS, which is an evaluative model that can be used to predict ratings on future maps, including randomly generated ones.

Finally, the current implementation uses a CPPN to calculate discrete settings in small sized maps. This is different from the majority of existing applications that use a CPPN to calculate continuous values from either continuous or high resolution inputs [Hastings et al., 2009; Risi et al., 2012; Secretan et al., 2008]. In any of those applications, if either the CPPN input or output had been further discretized, the patterns visible in the resulting content may not have been as evident. This may be alleviated by increasing the minimum size of the geometry, therefore increasing the resolution of the CPPN input. How the use of a CPPN on discretized inputs and outputs affects the patterns of content distribution in *PCG: Angry Bots* or how the content layout approach can be made continuous are topics for future investigation.

9.7.3 RS Player Model

The content-based RS used in this experiment performed well, but there is room for improvement. The statistical analysis of the ratings validates that the system is not simply randomly generating maps and that the optimized maps are, on average, receiving better ratings. However, unfortunately the effect sizes of these tests were small. It is not clear whether this small effect size is due to poorly performing player models or a lack of variety in potential maps caused by the game design and constrained discrete generative processes.

Despite this, we predict that any game that uses a random map generation procedure could be enhanced by using a content-based RS as a PPM in an experience-driven PCG (ED-PCG) framework. The extent of this enhancement, though, may be restricted by the means in which the maps are generated, as the results here have shown. While the generative processes here are discrete and constrained by the room template approach, random generation of maps in mainstream games is typically even more so controlled. For example, roguelike games such as *Diablo* (Blizzard Entertainment, 1996) have stronger constraints to ensure the player's experience is within a certain difficulty range, reducing the size of the solution space even further than was done in the *PCG: Angry Bots* experiment. While our experiment

cannot be directly compared to these games due to them having more advanced gameplay mechanics and their own unique means of map generation, the results of this experiment lead us to hypothesize that the use of the RS player modeling approach would provide more tailored experiences to the player.

While the mean values of the prediction accuracy measurements can be improved upon, the results are what we expected and represent a success for this experiment. With accuracy values of the NB classifier hovering between 0.60 and 0.75 and the 3-fold cross validation extending into the 0.80 to 0.90 region, these results are comparable with the original studies on using NB classifiers for content-based RS [Pazzani and Billsus, 1997]. However, there have recently been significant advances in recommender algorithms [Ricci and Shapira, 2011] which may provide a significant improvement.

There are a few simple possible solutions to improving accuracy, though they each have their downsides too. To start with, it was shown that a simple change of classifier from the NB to a random forest classifier could give better performance in most cases but not all. The IB1 nearest neighbor classifier may also provide steady improvement for all players. However, this may be risky as Pazzani and Billsus [1997] identify nearest neighbor classifiers as being the poorest performers in an RS designed to recommend web-pages to users.

The way in which the experiment is started could change to provide better initial performance. The first few maps could be manually designed to get feedback from players about very specific experiences. Making sure that a variety of experiences are provided to the player would prevent the optimization from getting stuck in a local optimum early on. However, considering that the median number of maps played in this experiment was 4, this alteration may risk not giving the participants a chance to experience maps produced by the RS. Alternatively, the player could be required to input an initial difficulty setting or state some other form of preferences and the RS could optimize from this starting point. This may be a good solution for a commercial game as it ensures that the player's exact preferences and past experiences with games are taken into account. However, again, for the purposes of this thesis, this approach would have lessened the impact of showing that a traditional, unmodified RS could be effectively used to capture a player's preferences without prior knowledge or assumed characteristics.

As further evidence of the success of this experiment, the user survey results presented in Section 9.6 show that the majority of players are able to notice an improvement in the suitability of the maps they are playing. These results also show that a participant is more likely to have a positive experience as they played more maps, correlating to the player's

classifier receiving more training data and being able to make better predictions about the player's preferences.

One of the major limitations of the current system at this time is a reported lack of variety in the maps that were played. While this may be due to the simple game mechanics of *PCG: Angry Bots*, it may also be dues to the overspecialization of either the CPPN-NEAT evolution or the RS player model (or both in conjunction). Case studies showed that not only did PPMs fluctuate in accuracy but also players provided fluctuation ratings. It was determined that these example players were providing conflicting ratings to consecutive similar maps, further indicating the presence of overspecialization and the negative effects it has on a player's experience. In the future work discussed in Chapter 10, we provide potential avenues of investigation to address this issue.

9.8 Summary

- During the *PCG: Angry Bots* experiment, a total of 147 players played 893 maps. Out of these, 570 were optimized for a player using the techniques described in this thesis (the set of which was labeled S_{Opt}) and 323 maps were generated completely randomly (the set of which was labeled S_{Rand}).

- Using the chi-squared test of independence, it was determined that there was a statistically significant relationship between the set that a map was in (S_{Opt} and S_{Rand}) and the rating it received. Thus, ratings in S_{Opt} were not just a matter of chance.

- Using the Mann-Whitney U test, it was shown that the ratings in S_{Opt} were statistically greater than those in S_{Rand}, providing evidence of the optimization process performing better than random generation.

- Unfortunately, the effect size of both of the above statistical tests was not high and so there is clear room for improvement of all included solutions in future work.

- The fixed n-ary tree representation and evolutionary operators produced a more even distribution of geometry sizes and caused geometries to be successfully validated more often when compared to randomly generated geometries. However, a curious result is that while player's reported a preference for medium to large sized maps in the survey, they frequently chose smaller geometries through IEC.

- Using a CPPN to produce patterns of content throughout maps gave players a variety of difficulty trends. Patterns of content that caused fluctuations in difficulty were most enjoyed by players. Meanwhile, maps with consistent difficulty were the least enjoyed overall.
- The CPPN representation also explored more of the feature space of enemies and pick-up than randomly generated maps did. This is most likely due to the CPPN producing logical patterns of content from room to room of a map.
- A CPPN can be applied to multiple geometries to give a variety of experiences that are within similar limits of difficulty.
- Analysis of individual classifier results showed that many players experienced fluctuations of positive and negative experience throughout their playing time. This is hypothesized to be due to either an overfitting classifier or the player's preferences changing.
- The mean prediction accuracy of the players' classifiers showed that the classifiers were learning from the player provided ratings and in turn producing more appropriate maps for the players. While the NB classifier performed well in 3-fold cross validation, most other classifiers performed better in one-off predictions.
- From the learning trend results, the NB classifier made true predictions for the majority of players on most map indices. Overall, the random forest classifier performed the best.
- Results from an accompanying survey suggest that participants who played more maps reported better experiences. This correlates with the increase in training data available to the RS, resulting in better predictions.
- The main disadvantage of the techniques used here is the lack of variety in the generated maps. This was reflected in the user survey results and plots of feature space coverage and convergence plots, which showed that, for a single player, maps were continuously generated to have features similar to maps that had been rated well in the past.

Chapter 10

Conclusion and Future Work

The aim of this research project was to investigate personalized procedural map generation via evolutionary algorithms (EA). This was conducted by dividing the map generation process into three core components: *geometry* (the non-interactive elements), *content* (the interactive elements), and *player preference modeling* (PPM).

As a preliminary investigation into geometry generation, we produced a search-based procedural content generation (SBPCG) solution for creating virtual terrain that could be used in a diverse range of game genres. This solution, referred to as *patch-based* terrain generation, involved combining small sample terrains together in order to create larger terrains. An *evolutionary terrain tool* (ETT) was created by combining the patch-based approach with interactive evolutionary computation (IEC). Through IEC, the user was presented with eight candidate terrains and asked to choose those that appealed to them. These terrains would then become the parents of the next generation. As well as this parent selection, a unique *gene selection* mechanism was introduced to allow the user to focus mutation on to specific patches of the candidate terrains. Through IEC and a variety of terrain and evolutionary parameters, the ETT gave users a high level of control over the types of terrains that were produced without requiring them to manually design and build the terrain themselves. This tool was evaluated by demonstrating the effects of each of the parameters and showing how the tool could be used to create detailed game maps.

While the ETT was a promising solution that could be used by even novice game developers, it required an investment in time and effort from the user. Thus, it was more suited as a development tool rather than being utilized by players during gameplay. For a solution to be acceptable for use by the player, it would need to be as unobtrusive as possible while

still capturing the player's subjective preferences. Additionally, the lack of a specified game genre made developing an automated fitness evaluation method difficult for the ETT as it could not be determined what exactly separated a high quality terrain from a low quality one.

Therefore, we chose to move forward with developing a solution within a test bed game that would give context to the map evaluation process and allow for any included solutions to be experimented on through actual player interaction. Thus, the *Procedural Content Generation: Angry Bots* (*PCG: Angry Bots*) game was designed and developed. While the ETT only incorporated geometry and only allowed for users preferences to be expressed through IEC, *PCG: Angry Bots* represented a complete solution by incorporating the three core components of a personalized procedural map generator.

The paradigm of using smaller segments of geometry to form larger geometries that was used in the ETT was adopted again for generating the geometry of *PCG: Angry Bots*. Interior spaces were created by connecting pre-made room and corridor templates together in a *fixed n-ary tree* structure. The geometry was again evaluated through IEC but this time each candidate geometry that was presented to the player was validated to ensure it was playable. Thus, the player could intuitively select from previews of the candidate maps based upon size and shape and be assured a standard of quality. This reduced the effort needed from the player to produce a playable map.

Once a geometry was chosen, the coordinates of each node (room) of the fixed n-ary tree were used as input to a *Compositional Pattern-Producing Network* (CPPN) [Stanley, 2007]. The output of the CPPN determined the quantity of enemies and pick-ups in each room of the map. CPPN candidates were evolved using *NeuroEvolution of Augmenting Topologies* (NEAT) [Stanley and Miikkulainen, 2002], in which fitness was determined by a PPM trained to the preferences of each individual player.

This PPM was framed as a *content-based recommender system* (RS). After each map was played, the player was asked to rate the map on a five point scale. Features of the map were extracted and combined with the player provided rating (converted to weighted binary scale) and used as a training sample for a *Naive Bayes* (NB) classifier. During NEAT evolution, the player selected geometry would be combined with each CPPN candidate and the same map features would be extracted. These features were then classified and the fitness value of the CPPN was set to the probability of the map being liked by the player.

The *PCG: Angry Bots* game and all the included solutions were made available to play online with an accompanying optional post-play survey. The data from this large scale,

unsupervised public experiment was then used to evaluate the performance of each of the included solutions. Statistical analysis showed that, as a whole, the maps that were optimized for the players' preferences were receiving better ratings than randomly generated maps. However, the effect size of this relationship was not large. To reinforce this result though, responses to the post-play survey showed that a statistically significant number of players stated that their experience got better over time. This highlights the potential of the included solutions but also the need for refinement.

Other findings included the CPPN approach to content layout searching more of the solution space by creating maps that had coherent patterns of content from room to room rather than random changes. Also along these lines, players appeared to favor maps with a fluctuating difficulty, supporting game design theory that states that players should experience alternating periods of low intensity and high intensity activity to maintain engagement [Adams, 2010]. Additionally, applying the same CPPN to different geometries produced content layouts that, while different, were within a similar range and created similar difficulty trends.

The mean prediction accuracy of the RS player models was comparable to the results of previous work into using NB for a content-based RS [Pazzani and Billsus, 1997]. However, these results may have been distorted by participants playing different amounts of maps, classifiers with poor performance early on, and the nature of the accuracy measurements themselves. Therefore we introduced the mean and majority *learning trend* metrics to reinforce the results and provide a clearer analysis of performance. From these results, it was deduced that a simple improvement to the current implementation would be to use a random forest classifier rather than a NB classifier, though overall the performance of all comparative classifiers was fairly similar.

The major limitation of the current implementation is a lack of variety in the produced map. This was identified through the post-play survey in which an overwhelming majority of players indicated that the variety of the maps that they played was poor. This could be due to poor game design, the discretized nature of the geometry and content layout generative processes, or overspecialization of either the RS player models or the CPPN-NEAT evolution. The latter possibility is supported by case studies that showed some players had repetitive experiences and provided conflicting ratings for similar consecutive maps.

Finally, as an overall statement, we consider the *PCG: Angry Bots* experiment a success in terms of providing us with the knowledge that we sought to collect in this investigation. It has shown that the ideas contained within the solution improve the player's experience

over random generation of game maps. However, probably more important is that it has also highlighted the weakness of the current implementation and the possibilities for improvement in future work.

10.1 Research Questions

This section reviews the research questions that were stated in the introduction of this thesis. Each research question is first re-stated and is then followed by a discussion of what was done to address them and what findings were unveiled.

10.1.1 Geometry

1. *How can the base terrain of a game map be represented in an SBPCG solution to provide a high level of control over the features in the generated maps?*

We developed an approach to terrain generation called patch-based terrains. The idea behind this approach was to combine smaller patches of sample terrain together to form a larger terrain that contained all the terrain features present in the samples. This was incorporated into an evolutionary cycle and a two-tiered IEC was used as the fitness evaluation approach. Due to the control allowed over numerous evolutionary and generative parameters, as well as allowing both parent selection and gene selection during IEC, the user was able to successfully generate terrains to their specifications.

2. *How does this genetic representation compare to existing SBPCG techniques for generating terrain?*

The patch-based approach, combined with IEC, allowed for more user control over the types of terrain features and their location within the larger terrain over existing evolutionary terrain techniques. Previous techniques have had only a small variety in the types of terrain features, little control over their location or combination in a single candidate terrain, or simply take an existing terrain and only alter the extent of the terrain features. The limitation of our approach is that it requires effort from the user to generate an appropriate terrain, therefore causing this system to be a developer tool rather than a means of effortlessly creating new experiences for a player. In order to make the system more appropriate, either the IEC process would need to be greatly simplified to be more user-friendly to players or it would need to be removed and replaced with an automated fitness function..

3. *Can the ideas behind the genetic representation be abstracted and used to generate interior spaces in a SBPCG manner as well?*

The patch-based terrain approach was abstracted to the idea of combining smaller map segments to generate larger maps. For generating the interior spaces of the *PCG: Angry Bots* game, pre-made room and corridor templates were joined together to form a map in a tree like structure. The tree had exactly one node (room) where the player started (above the root of the tree) and exactly one leaf node of the tree that was an exit room that the player needed to reach to complete the map. Because the room templates are all logically constructed, the only way that a map could be unplayable is if the sections of the map overlapped with each other. However, a quick validation process could detect these types of invalid maps and discard them. IEC was used to gather player preferences again here but, unlike with the patch-based terrains, was able to be utilized by the player efficiently because the player was always being presented with valid maps of a minimum playable quality.

10.1.2 Content Layout

1. *Given an example game, how does content affect a player's experience as compared to the geometry?*

This research question is not as strong as the others as it does not produce any generalizable knowledge. However, it was an important step within this thesis in order to frame some of the design choices made in our solution and to show how to analyze the content of a game in order to build an effective PCG system.

In the context of *PCG: Angry Bots*, the geometry of a map provides exploration opportunities to the player and allows for basic strategies such as hiding behind furniture and doorways. It also provides aesthetic qualities as a backdrop to the gameplay. Meanwhile, the content of the map (enemies and pick-ups) is the primary determinant of the challenge that the player will face. If there are too many enemies, the map will be difficult; if there are too many pick-ups, the map could be too easy. Furthermore, the layout of content affects the pace of the map. For example, successive rooms of a map with high quantities of enemies produce a high intensity map with little chance for the player to rest and replenish in-game resources. According to existing literature, an optimal player experience ccan be encouraged by interleaving high intensity and low intensity gameplay to maintain player engagement. However, in the commonly small and independent maps in *PCG: Angry Bots*, consistent

intensity, reducing intensity, or increasing intensity may all also provide a desirable pace of gameplay, depending on the player's preferences.

2. *Is there a genetic representation for an evolutionary procedural map generation algorithm that can encompass the needs of content layout in the example game?*

As the challenge and pace of a map in *PCG: Angry Bots* is determined by the layout of content, a good map should have a logical pattern of content throughout the map that is either interesting to the player or that is suitable for their skill level. Therefore, we investigated the pattern-producing properties of a CPPN; a neural network that can use various activation functions at each node of the network, the combination of which typically create a pattern in the output. In this application, the CPPN takes as input the coordinate of each room in the geometry tree representation and outputs the quantity of each type of content in a single room in the form of one of four discretized settings. A population of CPPN candidates was evolved through the NEAT algorithm.

While the CPPN-NEAT approach has been used in the past to produce patterns in game content, to the best of our knowledge this is the first time that they have been used in procedural map generation. Also of interest is that CPPN-NEAT has traditionally been used with high resolution inputs and produced continuous output values, which is not the case here. Through experimental player data, it seems that the CPPN representation explored more of the solution space of the number of enemies versus the number of pick-ups in a map. This is because the CPPN produces more logical sequences of content, with the adjacent rooms having content quantities within a similar range rather than random quantities. The approach also produced all types of difficulty trends described in the previous research question and indeed it was the fluctuating challenge that was most enjoyed by players. However, we believe these patterns were hindered by the low resolution input (i.e. the small number rooms in a typical *PCG: Angry Bots* map) and that the discretized output restricted the variety of the maps that the player could potentially experience.

3. *How can content layout be evaluated to determine whether it matches the player's preferences?*

A player's preferences regarding the geometry of a *PCG: Angry Bots* map could be captured through IEC because it is likely that the only considerations the player would have would be the size of the map, the number of branching paths, and the types of rooms that

were present. All of these can be easily visualized through a quick top-down preview of the geometry and the player could decide on their preferences with little effort. However, this is not possible with the content. With six different types of content positioned in many areas of a single room, there is not a simple visualization of an entire map's worth of content that would not require the player to carefully scrutinize each candidate. We felt this was too invasive of the gameplay flow and so we turned to player preference modeling techniques to evaluate CPPN candidates with little input required from the player.

10.1.3 Player Preference Modeling

1. *Is there a player preference modeling technique that can be generalized to multiple game genres for the purpose of procedural map generation?*

Through our literature search we could not find a suitable player modeling approach. Much of the existing PPM literature for the PCG field focuses on creating 'universal' or 'class' player models, attempting to generalize the preferences of a sample of players to all players who may be similar. However, we feel that these approaches ignore the fact that each player is a unique individual with their own reasons for playing and enjoying a game. Additionally, many existing PPM solutions are tightly bound to the problem domain or the genre of game that it was tested in, such as those that use agents to mimic player behavior and evaluate a map in their stead.

Thus, we established our own PPM framework by drawing a correlation between the fields of RS and experience-driven procedural content generation (EDPCG). If the EDPCG solution is built around a SBPCG approach, then the solution space of all possible content can be thought of as the item set of an RS. This allows for the utilization of existing knowledge in the RS field. In this thesis, we tested this idea through implementing a content-based RS as PPM for each individual player. While this approach does take map features specific to *PCG: Angry Bots* as input, the approach is easily generalizable to other genres with little effort through using expert knowledge to decide upon a new set of map features.

2. *For this project, which source of player data is the most suitable and why?*

A player's preferences can be determined by examining direct player feedback (subjective), their actions in game (gameplay), or their bodily functions (objective). We chose to use the first of these sources, subjective player data. Monitoring a player's bodily functions is not appropriate as it would require every experiment participant to use a specialized device to

monitor these signals. This is not a viable solution for a commercial game as consumers typically do not wish to purchase extra peripheral devices to play a single game and so it is not examined either. From the remaining two options, subjective player data was chosen because it aligns better with existing RS implementations and studies and because we believe that players are better able to communicate their preferences than we are able to understand through their gameplay data. Understanding how gameplay actions correlate with map features and how they jointly indicate a player's preferences requires finding trends in player data. This is typically a difficult process and is done offline, creating universal or grouped player models and allowing room for the misinterpretation of the player data by the research team or the insertion of our own beliefs on what causes a map to be enjoyable. While subjective data allows for error through players incorrectly self-reporting their current emotional state, we feel it is a much more stable means of capturing an individual player's preferences.

3. *When combined with an EDPCG application, does the player modeling technique provide an improved experience for players over random PCG?*

Through the *PCG: Angry Bots* experiment, in which the content layout of candidate maps was evaluated with an RS player model, we evaluated the prediction accuracy of the content-based RS and compared the player provided ratings for optimized maps and randomized ones. The mean prediction accuracy of the player models was as we expected, properly predicting the player's preferences for the majority of maps. This accuracy could be improved by using different classifiers at the core of the content-based RS. Additionally, a statistically significant ratio of players reported in a post-play survey that they felt the maps improved over time, suggesting that the RS player models were properly learning preferences as more training data became available. Meanwhile, statistical analysis of the map ratings showed that the ratings that a map received were dependent on whether the map was optimized or randomized and that the optimized maps were receiving better ratings. However, the effect sizes of these relationships were small, encouraging future investigation into the use of RS techniques as PPMs.

10.2 Limitations

In this section we briefly discuss some of the limitations and misgivings that we have about both the work of this thesis and of the PCG field in general. These are mainly about the

methodology used rather than the shortcomings of the solutions that were developed. This is because the investigation into personalized procedural map generation is ongoing and there will always be room to improve the presented solutions. Thus, these topics are discussed with regards to future work in Section 10.3.

Instead, the critiques here involve things that could have been done better if the investigation was repeated, as well as issues that are prevalent among the broader PCG community that we do not currently have a solution for. In general, these are observations that have arisen from our investigation into the PCG field and are left as open thoughts to the community.

- Despite limiting the scope of this research project in the introduction to this thesis, the scope still remained quite large. The aim of this thesis was to investigate all aspects of personalized procedural map generation, including geometry, content layout, and player preference modeling. Each of these components required separate lines of investigation and different solutions, each requiring a substantial investment of time. We considered all of these components to be required in order to build a complete personalized map generator; however, it is regrettable that each component could not have received more time for further investigation.

- After the investigation of patch-based terrains, we decided that investigating each component of a personalized procedural map generator individually would not be practical because a deployable solution would require all the components to be interacting with each other. Thus, the *PCG: Angry Bots* game was created with all the included solutions, all being simultaneously evaluated. However, this meant that there were a lot of different player interactions and system processes all occurring and being evaluated at once. This may have introduced conflicts in the systems. For example, the ability for players to choose a geometry through IEC may have led them to rate the subsequent map more highly due to a subconscious need to validate their own choice. Thus, the IEC process is interfering with the accuracy of the PPM.

- Two findings hint at issues in game design. Firstly, survey respondents indicated that the variety of maps was lacking, which may be due to a lack of engaging gameplay; every map has the same enemies, the same pick-ups, the same basic objective, no narrative, and no social engagement. Secondly, while a majority of survey respondents said that medium to large sized maps were preferred, player's overwhelming chose shorter maps

with only 2 rooms through IEC. This may also be due to a lack of engagement; if the game is not engaging enough and the players are aware that it is an experiment, do they make non-standard decisions and gameplay actions in a rush to complete their perceived obligations of participation? Did players choose smaller maps because they would be able to complete more maps in a shorter period and provide more rating data for the experiment? It could be deduced that in order for any user study in games research to be representative of real world conditions, the game must be as engaging as a commercially successful game. However, this has only rarely been the case in past research [Hastings et al., 2009] as game design and creation is a challenging, time consuming, and multidisciplinary task. Even though most PCG researchers will compromise the game design process in favor improving research contributions, the lack of engaging gameplay could in fact be harming the research outcomes as well.

- In the *PCG: Angry Bots* experimental evaluation, we compared our solution to a random baseline as well as testing the accuracy of different classifiers used in the content-based RS player models. However, the ideal would have been to compare our solution to that of others within the PCG field. Unfortunately, the diversity of game genres used as test beds, the varied research objectives, methodologies, and solution requirements makes this nearly impossible. With no standardized PCG performance metric yet established, solutions are typically compared on their features and capabilities rather than whether they can achieve better results than other solutions. Smith [2013] provides observations on this limitation of PCG research and through our experimental design we attempted to avoid as many of Smith's 'sins' as possible but we were still unable to provide a comparison to other existing solutions. There are a growing number of works that generate maps for the game *Super Mario Bros.* (Nintendo, 1985), due in part to the level generation track of the Mario AI Championship sponsored by the IEEE Computation Intelligence Society [Shaker et al., 2011]. However, platform games such as these only represent a small market share [Reeves, 2012] and even within the genre Super Mario Bros. is, while a classic game, an aging example that is being replaced with faster paced platform games such as *Super Meat Boy* (Team Meat, 2010). Thus, we question the significance and applicability of these studies on the current games industry.

10.3 Future Work

This section describes some of the specific limitations of the implementations discussed in this thesis that could be addressed through a clear line of future work. As much of this thesis was an investigation into the yet infant fields of SBPCG and EDPCG, it is natural that as any answer or potential solution is found during the exploration of the topic, it is replaced with even more questions and curious avenues of future study. Each one of the following topics represents a minor or major thesis in themselves, displaying the potential that the numerous PCG subfields still hold.

10.3.1 Combating Overspecialization

The results of the *PCG: Angry Bots* player survey highlighted a major limitation of the current system. When asked about the variety of the maps that were played, 75.62% of respondents indicated that the variety was fair or worse. There are two reasons why this may be; firstly, it may simply be due to the re-use of the same enemies, pick-ups, room templates, and objectives throughout the game. Addressing this issue would either require better game design when conducting such experiments or a means of procedurally generating the enemies, pick-up, narrative, or the objectives of the game. One such improvement may be to replace the current room and corridor templates used in the fixed n-ary tree geometry representation with procedurally generated rooms and corridors, such as the building interiors created by Tutenel et al. [2011].

Alternatively, players may have reported a lack of variety due to either the RS or the CPPN-NEAT evolution being stuck in a local optimum, thus providing the player with similar content layouts in consecutive maps and therefore repetitive experiences. This is a common issue in the RS field and is referred to as *overspecialization*; the case in which the user model is overfit to their preferences and they are only ever recommended items similar to those they have liked in the past, neglecting to recommend new experiences that user had not considered. There is a growing interest in the RS field into solving this problem [Adomavicius and Tuzhilin, 2005]. Billsus and Pazzani [2000] suggest that items (maps in this case) should not only be disqualified from recommendation if they are too dissimilar to items that a user has rated well in the past but also if they are too similar to any item the user has experienced in the past. Thus, in this case, maps would only be recommended to players if they were predicted to receive a good rating and were not too similar to maps that

have been played in the past. The difficulty in this approach is determining when a map is to be classified as 'too similar' to previous maps.

The solution may also reside in how the neuroevolution of CPPNs is conducted. Lehman and Stanley [2008] argue the benefits of avoiding an objective function in machine learning and instead look for novelty and diversity in the solution space. Instead of using the probability of a map being liked by a player as the fitness value for NEAT in *PCG: Angry Bots*, there may instead be a better objective measure that encourages diversity. That being said, simply looking for novelty may result in many maps that a player does not enjoy. Liapis et al. [2013a] have recently explored improvements to novelty search for procedural map generation. Their solutions utilize feasible-infeasible dual population approaches [Kimbrough et al., 2008]. It may be possible to utilize their approach in *PCG: Angry Bots* by placing any maps that are predicted to be disliked into the infeasible population and explore novelty only in maps that are predicted to be enjoyed.

10.3.2 Collaborative Data

In Chapter 6 we briefly discussed the difficulties of applying a collaborative filter to a domain in which there are no co-ratings in the user-item matrix. We predicted that, due to the size of the solution space (the item set), it would be unlikely that there would be many co-ratings. Indeed, even after over eight hundred maps were played by over a hundred players, there is no map that was played by more than one player.

Data sparsity is a prevalent topic of study in the RS field [Su and Khoshgoftaar, 2009], especially in new item or new user scenarios where there is no existing data about an item or a user in order to begin recommendations [Schein et al., 2002]. Solutions involve using a classifier to populate the user-item matrix with predicted ratings [Park et al., 2006], using demographic information to draw similarities between users based on personal traits rather than ratings [Rich, 1979], or by combining both content-based RS and collaborative filtering [Melville et al., 2002; Yoshii et al., 2008]. Additionally, dimensionality reduction [Sarwar et al., 2000], latent variable discovery [Popescul et al., 2001], and matrix factorization [Koren et al., 2009] techniques have all been investigated to reduce the size of the item set. However, while these solutions work well when there are only a few co-ratings between users, to the best of our knowledge there has been no research conducted on scenarios where there are no co-ratings between any users of the system, where the item set is as large as the one in *PCG: Angry Bots*, and where external personal information isn't available.

In order to compare two users with no co-ratings, similarity calculations would need to be done between the features of items that the two users have rated, rather calculating the similarity in rating of co-rated items. This process then resembles a content-based RS except that there would need to be a final score that aggregates the similarity of features and ratings between each item experienced by one user and each item of the other user, which would then be a single similarity measure of the two users. Not only is the algorithm to accomplish this not clear at this time but it would be a computationally expensive task as the number of users increases and the number of items that each of them rates increases.

This is an uncommon scenario for typical RS applications but it would be a key problem in further research into the use of RS technique for player preference modeling in PCG. CF would continue to build per-player preference models to capture the individuality inherent in each player, as was done with the content-based RS in this thesis, but it would also allow for the utilization of data generated by other similar player's to improve recommendations, especially for the first few maps where there is not enough data to train an effective content-based RS.

10.3.3 Additional Test Games

Throughout this research project, the ideas have only been tested within one game; *PCG: Angry Bots*. Testing within other games was not appropriate for this project as it would require that more time be spent on learning to modify specific games or on designing and developing new games rather than on investigative research. However, the successes and insights of the *PCG: Angry Bots* experiment encourage the testing of these solutions within other games.

Towards the end of Chapters 4, 5, and 6 we discussed how the presented solution could be generalized to other game genres. Of highest interest of these would be to test the RS player preference models in other games. The solid prediction accuracy of the player models in the *PCG: Angry Bots* experiment and its improvement in map ratings and completions over randomly generated maps leads us to believe that a SBPCG approach to map generation combined with an RS player model would produce consistently better experiences over equivalent random generation. For example, mainstream roguelike games, such as *Diablo* (Blizzard Entertainment, 1996) already contain randomly generated maps. A good measure of performance of the RS player models would be to attempt to modify such a random generation process to use RS and see if it offers a better experience for the player. This

would be straight forward process for the player model design as it would only require expert knowledge to decide on classifier features; instead much of the implementation time would be spent on modifying the existing game to accommodate the changes.

It may also be beneficial to test these implementations in the *Super Mario Bros.* test bed established by Shaker et al. [2011]. While we believe that platform games should not be the sole focus of PCG research due to their limited presence in the current games market, adapting our solutions for this domain would allow them to be more comparable with other EDPCG solutions. Of interest here would be using the CPPN-NEAT approach to content layout to determine the location of various enemies, coins, and power-ups throughout existing Super Mario Bros. map geometries. When combined with the RS player preference model, this approach should provide maps that are more suited to the skill level of an individual player. However, there may still be an issue with diversity of the generated content as the game mechanics of this game do not allow for it and the established player base may not accept maps the differ from the establish norm of the existing maps.

10.3.4 A Return to Patch-based Terrains

Due to the initial aims of this thesis and the fact that the ETT was more of a developer tool rather than a system that could be used directly by a player, we were required to discontinue that line of investigation and move onto the *PCG: Angry Bots* study. However, we believe that there is merit in the patch-based approach to terrain generation. It follows the concept of tiling, which has become popular for procedurally generating expansive 2D textures [Kelly and McCabe, 2006] and it could hold as much value for generating 3D terrains, either through developing it further as a design tool or through investigating it in a more automated terrain generation environment.

As in the real world, many virtual worlds use terrain as the base of the world geometry on which architecture, infrastructure, and environmental elements are placed [Smelik et al., 2009]. Therefore, simplifying the process of generating the terrain of a map would aid in reducing development time in the same way that the *SpeedTree* application (Interactive Data Visualization Inc., 2013) has simplified the process of populating a map with vegetation.

Some of the current research questions that we have for the patch-based terrain work include: 1) how does the choice of sample terrains (from which patches are extracted) affect the terrain features present in the candidate terrains that are constructed? 2) What other seam removal techniques could be used to join terrain patches together and how would these

affect the resulting candidate terrains? And 3) can the sample terrains contain details such as surface texture and static objects and still be decomposed into patches? How can these patches then be re-combined to make a coherent and logical candidate terrain?

10.3.5 Narratives, Missions, and Objectives

In the introduction to this thesis, we narrowed the scope of this work by removing narratives, missions, and objectives from the map creation process. This was necessary in order to make the research project achievable within the allowed time of the candidature. However, each one of these components play a fundamental role in the experience that a player has, especially in single-player games where the allure of social engagement with other players isn't present.

The narrative motivates the player to discover more about the virtual world that they are in. This was missing from *PCG: Angry Bots* and instead players were only given the single task of completing a map by getting from one side to the other. Procedural narrative is an ongoing field of work [Thue et al., 2007a] but how it influences the requirements of the map around the player is rarely investigated. From the opposite perspective, rather than just using procedurally generated environments as a backdrop to the narrative, they themselves can actually be used as a means of communicating the narrative as was done by Nitsche et al. [2006] who used the environment to reflect the protagonist's mood.

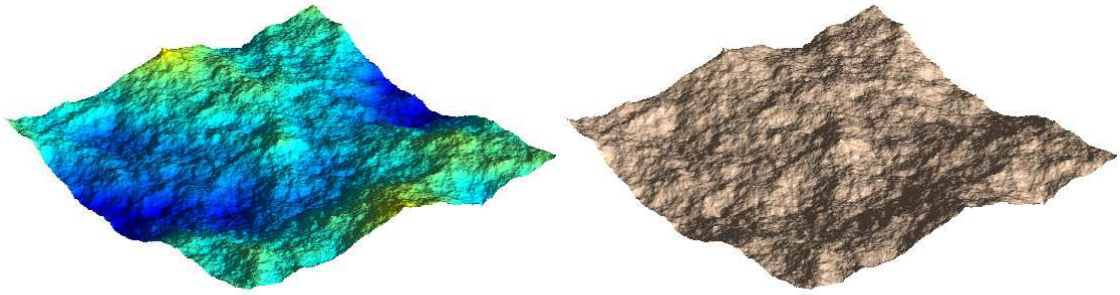
Here, we define missions as the overarching goal of a map and objectives as the individual steps needed to successfully complete the mission. Dormans and Bakkes [2011] provide a discussion and motivation for the inclusion of missions and objectives (succinctly referred to as missions) during the map design process. We believe that objectives can be thought of as a form of content, and thus the placement of objectives throughout a map can be combined with the content layout process. However, the major difference here is that objectives typically have to be completed in a certain order and so this constraint must be taken into account during procedural generation. For example, a key cannot be located after the door that it is meant to unlock, otherwise it will be impossible for the player to complete the map.

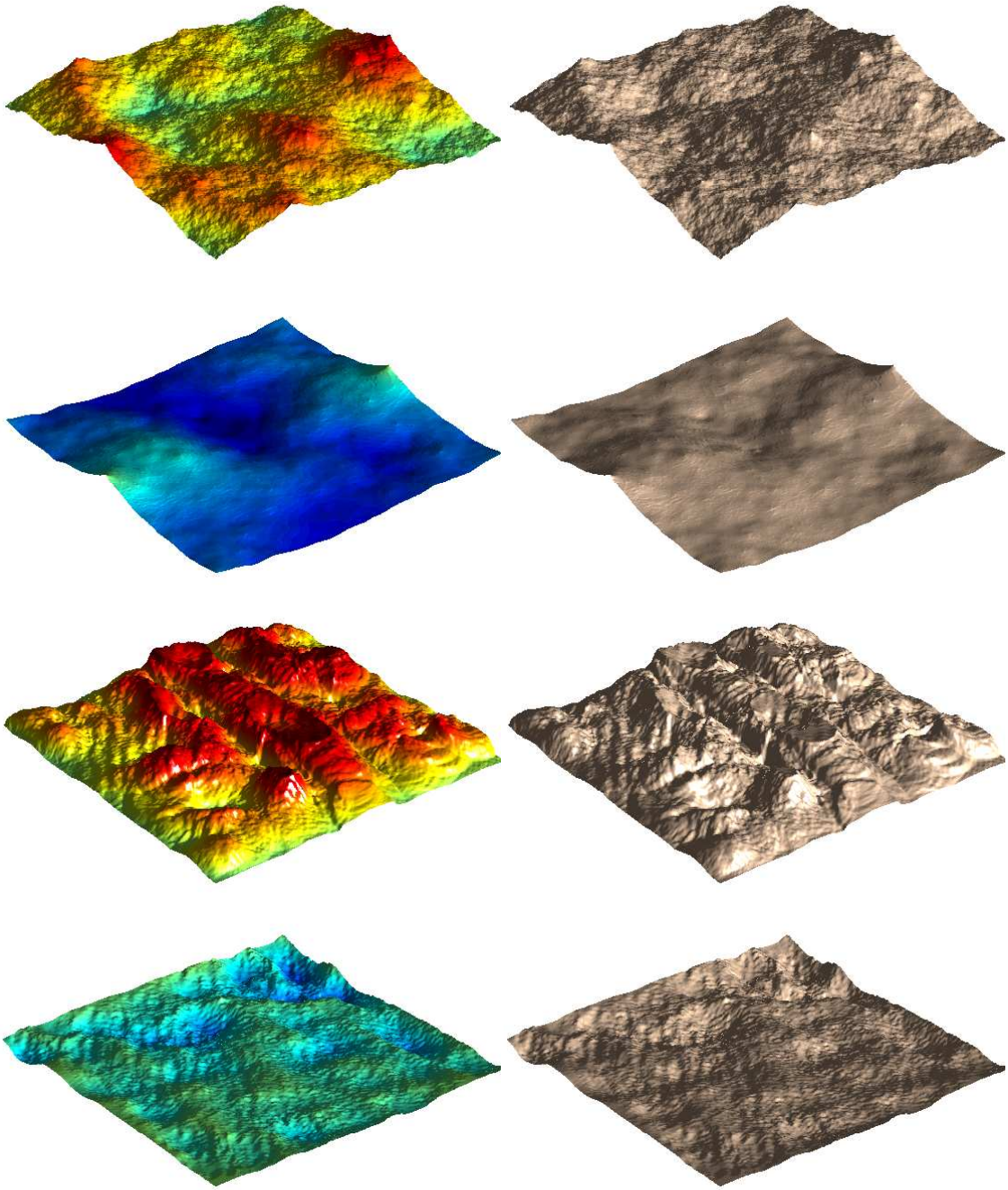
Meanwhile, generating missions may be considered as part of the narrative generation. Alternatively, it could be a process of deciding which objectives should be included in a map; much in the way a 'game master' would in a role playing table-top game [Tychsen et al., 2005]. Finally, it could even be a process of designing the rules of the game, an approach that has recently been investigated in an SBPCG approach by Togelius and Schmidhuber [2008].

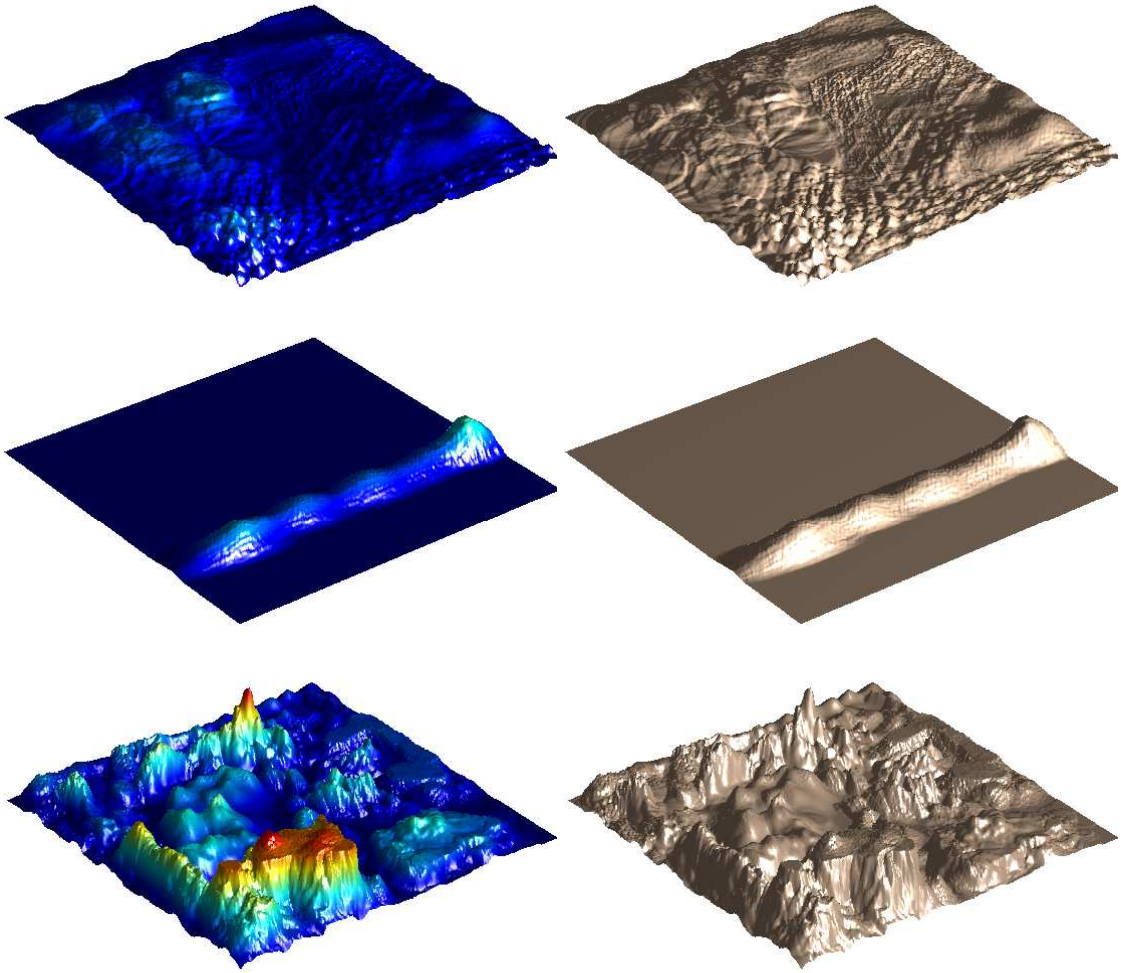
Appendix A

Evolutionary Terrain Tool - Sample Terrains

The eight sample terrains provided as input to the evolutionary terrain tool (ETT) for all experiments in Chapter 7. Each terrain is shown twice; firstly with height-dependent color values at each vertex to give a clear view of the height changes of the terrain and then with a simple sandy color accentuated by a virtual light source to show how the terrain may appear in a game. For the height-dependent color, dark blue indicates a height value of 0.0 (the minimum height value of this system) and dark red indicates a height of 1.0 (the maximum height value).





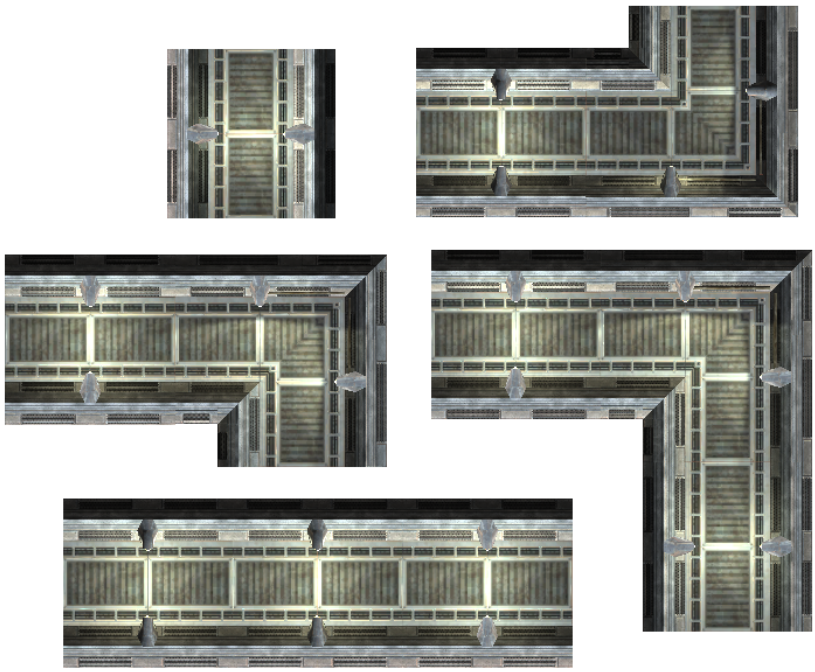


Appendix B

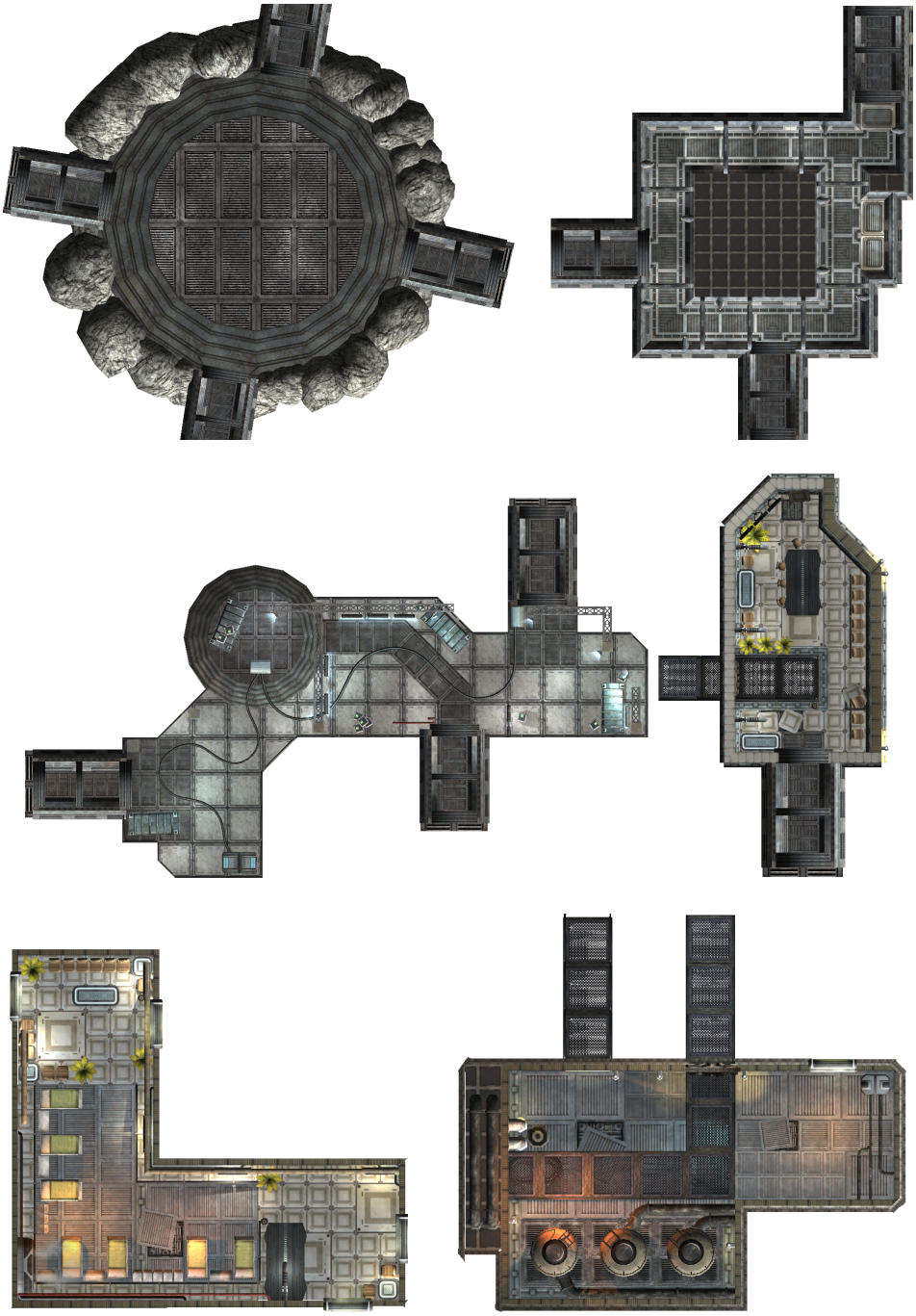
PCG: Angry Bots - Room Templates

The five corridor templates and ten room templates used for constructing the geometry in PCG: Angry Bots.

Corridors



Rooms



Appendix C

PCG: Angry Bots - Survey Form

This form can also be found at http://goanna.cs.rmit.edu.au/~wraffe/QualtricsSurvey.pdf

How would you rate the difficulty of the first few maps you played?

Very Difficult	Difficult	About Right	Easy	Very Easy
○	○	○	○	○

How would you rate the difficulty of the last few maps you played?

Very Difficult	Difficult	About Right	Easy	Very Easy
○	○	○	○	○

How would you rate your experience after playing for a while, as compared to when you started?

Much Worse	Worse	About the Same	Better	Much Better
○	○	○	○	○

How would you rate the variety of the maps you played?

Very Poor	Poor	Fair	Good	Very Good
○	○	○	○	○

Imagine your ideal map. What would be a good quantity of each item below in your ideal map?

	None	Little	Some	A Lot
Spider Bot enemies	○	○	○	○
Buzz Bot enemies	○	○	○	○
Big Mech enemies	○	○	○	○
Ammo pick-ups	○	○	○	○
Health pick-ups	○	○	○	○
Weapon pick-ups	○	○	○	○

How large is your ideal map?

Small (2-4 rooms)	Medium (4-7 rooms)	Large (>7 rooms)
○	○	○

How many branching paths does you ideal map have? (i.e. corridors and rooms that lead to a dead-end)

None	Little	Some	A Lot
○	○	○	○

Think of some of the maps that you enjoyed the most while playing. How closely did they match your ideal map?

Very Far from the ideal	Far from the ideal	Neither Far nor Near the Ideal	Near the ideal	Very Near to the ideal
○	○	○	○	○

Were you trying to get to the top of any of the leaderboards?

- ○ Yes
- ○ No

Which leaderboards?

- ☐ Rooms Visited
- ☐ Maps Completed
- ☐ Spider Bots Killed
- ☐ Buzz Bots Killed
- ☐ Big Mechs Killed
- ☐ Ammo Picked-up
- ☐ Health Picked-up
- ☐ Weapons Picked-up

Did you get to the top of any of these leaderboards at any time?

- ○ Yes
- ○ No

How did you achieve this?

Why do you feel you couldn't achieve this?

Is there anything that you would change about the game?

Is there any other feedback you would like to leave?

Bibliography

E. Adams. *Fundamentals of game design*. New Riders, 2010.

G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, Jun 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.99.

D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, Jan 1991.

D. G. Aliaga, C. A. Vanegas, and B. Beneš. Interactive example-based urban layout synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 160. ACM, 2008.

D. Ashlock, S. Gent, and K. Bryden. Evolution of l-systems for compact virtual landscape generation. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2760–2767. IEEE, 2005.

D. Ashlock, S. Gent, and K. Bryden. Embryogenesis of artificial landscapes. In *Design by Evolution*, pages 203–221. Springer, 2008.

D. Ashlock, C. Lee, and C. McGuinness. Search-based procedural generation of maze-like levels. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):260–273, 2011.

L. B. Baker. Factbox: A look at the $65 billion video games industry. Reuters, 2011. URL http://uk.reuters.com/article/2011/06/06/us-videogames-factbox-idUKTRE75552I20110606,Accessed:01/06/2013.

S. Bakkes, C. T. Tan, and Y. Pisan. Personalised gaming: a motivation and overview of literature. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, page 4. ACM, 2012.

W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. Genetic programming: An introduction: On the automatic evolution of computer programs and its applications (the morgan kaufmann series in artificial intelligence). 1997.

C. Basu, H. Hirsh, W. Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.

N. J. Belkin and W. B. Croft. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992.

J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

S. Berkovsky, J. Freyne, M. Coombe, and D. Bhandari. Recommender algorithms in activity motivating games. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 175–182. ACM, 2010.

H.-G. Beyer. Toward a theory of evolution strategies: The (μ, λ)-theory. *Evolutionary Computation*, 2(4):381–407, 1994.

P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by example. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 41–44. ACM, 2004.

M. Biggs, U. Fischer, and M. Nitsche. Supporting wayfinding through patterns within procedurally generated virtual environments. In *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 123–128. ACM, 2008.

D. Billsus and M. J. Pazzani. User modeling for adaptive news access. *User modeling and user-adapted interaction*, 10(2-3):147–180, 2000.

G. D. Birkhoff. *Aesthetic measure*. Cambridge, Mass., 1933.

M. Booth. The ai systems of left 4 dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE09)*, 2009.

B. Bostan and S. Ogut. Game challenges and difficulty levels: lessons learned from rpgs. In *International Simulation and Gaming Association Conference*, 2009.

E. Boyes. Are casual games the future? Gamespot UK, 2008. URL http://uk.gamespot.com//news/gdc-08-are-casual-games-the-future-6186207?tag=result%3Btitle%3B0,Accessed: 01/06/2013.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, Oct 2001.

R. Broughton and T. Howard. Introducing clutter into virtual environments. 2006.

P. Burelli and G. N. Yannakakis. Global search for occlusion minimisation in virtual camera control. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

M. Buro and T. Furtak. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, pages 63–70. Citeseer, 2004.

E. Byrne. *Game level design*. Delmar Thomson Learning, 2005.

L. Cardamone, D. Loiacono, and P. L. Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 395–402. ACM, 2011a.

L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*, pages 63–72. Springer, 2011b.

G. Chanel, C. Rebetez, M. Bétrancourt, and T. Pun. Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In *Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era*, pages 13–17. ACM, 2008.

J. Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, 2007.

M. Chiang, J. Huang, W. Tai, C. Liu, and C. Chang. Terrain synthesis: An interactive approach. In *International workshop on advanced image tech*, 2005.

J. Cohen. A power primer. *Psychological Bulletin*, 112(1):155–159, 1992. ISSN 0033-2909. doi: 10.1037/0033-2909.112.1.155.

C. Cotta and A. J. Fernández-Leiva. Bio-inspired combinatorial optimization: notes on reactive and proactive interaction. In *Advances in Computational Intelligence*, pages 348–355. Springer, 2011.

H. Cremér. *Mathematical Methods of Statistics (PMS-9)*, volume 9. Princeton university press, 1999.

M. Csikszentmihalyi and I. Csikszentmihalyi. *Beyond boredom and anxiety: The experience of play in work and games*. Jossey-Bass San Francisco, 1975.

S. Daly. The costs of aaa games development. Leviathyn: The Gamer's Chronicle, 2013. URL http://leviathyn.com/games/opinion/2013/04/12/the-costs-of-aaa-games-development/.

C. Darwin. On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. *New York: D. Appleton*, 1859.

G. J. de Carpentier and R. Bidarra. Interactive GPU-based procedural heightfield brushes. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 55–62. ACM, 2009.

K. Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, pages 13–46, 2001.

P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classi er. In *Proc. 13th Intl. Conf. Machine Learning*, pages 105–112, 1996.

J. Doran and I. Parberry. Controlled procedural terrain generation using software agents. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):111–119, 2010.

J. Dormans and S. Bakkes. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):216–228, Sep 2011. ISSN 1943-068X. doi: 10.1109/TCIAIG.2011.2149523.

A. Doull. The death of the level designer. *ASCII Dreams: http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html*, 2008.

A. Drachen, A. Canossa, and G. N. Yannakakis. Player modeling using self-organization in tomb raider: underworld. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 1–8. IEEE, Sep 2009. ISBN 978-1-4244-4814-2. doi: 10.1109/CIG.2009.5286500.

A. Drachen, L. E. Nacke, G. Yannakakis, and A. L. Pedersen. Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*, pages 49–54. ACM, 2010.

R. O. Duda, P. E. Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*, volume 2. Springer Berlin, 2010.

ESA. The Entertainment Software Association: Industry Facts. Entertainment Software Association, 2012. URL http://www.theesa.com/facts/index.asp,Accessed:1January2013.

D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.

M. Frade, F. F. De Vega, and C. Cotta. Modelling video game landscapes by means of genetic terrain programming-a new approach for improving users experience. In *Applications of Evolutionary Computing*, pages 485–490. Springer, 2008.

M. Frade, F. F. de Vega, and C. Cotta. Adding zoom feature to terrain programmes. In *VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEBï¿œ09), Málaga, Spain*, pages 293–300, 2009a.

M. Frade, F. Fernandéz de Vega, and C. Cotta. Breeding terrains with genetic terrain programming: the evolution of terrain generators. *International Journal of Computer Games Technology*, 2009, 2009b.

M. Frade, F. F. de Vega, and C. Cotta. Evolution of artificial terrains for video games based on accessibility. In *Applications of Evolutionary Computation*, pages 90–99. Springer, 2010a.

M. Frade, F. F. de Vega, and C. Cotta. Evolution of artificial terrains for video games based on obstacles edge length. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010b.

C. Green. Phased searching with NEAT: Alternating between complexification and simplification, 2004.

S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation ofpseudo infinite'cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 87–ff. ACM, 2003.

S. Greuter, N. Stewart, and G. Leach. Beyond the horizon. *Image Text and Sound Conferance*, 2004.

R. J. Grissom and J. J. Kim. *Effect sizes for research: Univariate and multivariate applications*. Psychology Press, 2005.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the Galactic Arms Race video game. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 241–248. IEEE, 2009.

R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

R. Herbrich, T. Minka, and T. Graepel. Trueskill: A bayesian skill rating system. *Advances in Neural Information Processing Systems*, 19:569, 2007.

J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22 (1):5–53, 2004.

P. Hingston. A new design for a turing test for bots. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 345–350. IEEE, 2010.

H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, volume 29, pages 2179–2186. Wiley Online Library, 2010.

R. V. Hogg and A. Craig. Introduction to mathematical statistics. pages 338, 400, 1994.

I. D. Horswill and L. Foged. Fast procedural level population with playability constraints. In *AIIDE*, 2012.

K. Hullett and J. Whitehead. Design patterns in fps levels. In *proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 78–85. ACM, 2010.

R. Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM Press, 2005. ISBN 1595931104. doi: 10.1145/1178477.1178573.

R. Hunicke and V. Chapman. Ai for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91–96, 2004.

S. Jamieson et al. Likert scales: how to (ab) use them. *Medical education*, 38(12):1217–1218, 2004.

M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 11. ACM, 2010.

G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

J. John. Pandora and the music genome project. *Scientific Computing*, 23(10):40–41, 2006.

D. Johnson and J. Wiles. Effective affective user interface design in games. *Ergonomics*, 46 (13-14):1332–1345, Oct 2003. ISSN 0014-0139. doi: 10.1080/00140130310001610865.

K. R. Kamal and Y. S. Uddin. Parametrically controlled terrain generation. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 17–23. ACM, 2007.

A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, 26(7):51–64, 1993.

G. Kelly and H. McCabe. A survey of procedural techniques for city generation. *ITB Journal*, 14:87–130, 2006.

S. O. Kimbrough, G. J. Koehler, M. Lu, and D. 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.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

B. Kuchera. Why its time to grow up and start ignoring the monthly npd reports. The Penny Arcade Report, 2012. URL http://www.penny-arcade.com/report/article/its-time-for-the-gaming-press-to-grow-up-and-ignore-the-npd-group,Accessed: 01/06/2013.

V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 277–286. ACM, 2003.

S. Labovitz. Some observations on measurement and statistics. *Social Forces*, 46(2):151–160, 1967.

A. Lagae, O. Dumont, and P. Dutre. Geometry synthesis by example. In *Shape Modeling and Applications, 2005 International Conference*, pages 174–183. IEEE, 2005.

J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11:329, 2008.

A. Liapis, G. N. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, Sep 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2192438.

A. Liapis, G. N. Yannakakis, and J. Togelius. Enhancements to constrained novelty search: Two-population novelty search for generating game content. In *Proceedings of Genetic and Evolutionary Computation Conference*, 2013a.

A. Liapis, G. N. Yannakakis, and J. Togelius. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, 2013b.

R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

A. Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

R. L. Linn and N. E. Gronlund. *Measurement and assessment in teaching*. ERIC, 2000.

R. Lopes and R. Bidarra. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):85–99, 2011.

M. C. Machado, E. P. Fantini, and L. Chaimowicz. Player modeling: Towards a common taxonomy. In *Computer Games (CGAMES), 2011 16th International Conference on*, pages 50–57. IEEE, Jul 2011. ISBN 978-1-4577-1451-1. doi: 10.1109/CGAMES.2011.6000359.

T. W. Malone. What makes things fun to learn? a study of intrinsically motivating computer games. *Pipeline*, 6(2):50–51, 1981.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

A. Martin, A. Lim, S. Colton, and C. Browne. Evolving 3d buildings for the prototype video game subversion. In *Applications of Evolutionary Computation*, pages 111–120. Springer, 2010.

P. McClintock. Global box office hit $32.6 bil in 2011, fueled by exploding international growth. The Hollywood Reporter, 2013. URL http://www.hollywoodreporter.com/news/global-box-office-china-international-growth-326-303324,Accessed:01/06/2013.

B. Medler. *Using recommendation systems to adapt gameplay*, pages 64–77. Information Science Reference, May 2011. ISBN 9781609605650. doi: 10.4018/978-1-60960-565-0.ch005.

P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 187–192. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

P. Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 105–112. ACM, 2007.

P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. *Procedural modeling of buildings*, volume 25. ACM, 2006.

F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *ACM SIGGRAPH Computer Graphics*, volume 23, pages 41–50. ACM, 1989.

M. Nitsche, C. Ashmore, W. Hankinson, R. Fitzpatrick, J. Kelly, and K. Margenau. Designing procedural game spaces: A case study. *Proceedings of FuturePlay*, pages 10–12, 2006.

J. Olsen. Realtime procedural terrain generation. *Department of Mathematics And Computer Science (IMADA) University of Southern Denmark*, 2004.

T. J. Ong, R. Saunders, J. Keyser, and J. J. Leggett. Terrain generation using genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1463–1470. ACM, 2005.

J. Orkin. Three states and a plan: the ai of fear. In *Game Developers Conference*, volume 2006, page 4. Citeseer, 2006.

R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller. User-centered design in games. *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pages 883–906, 2003.

S.-T. Park, D. Pennock, O. Madani, N. Good, and D. DeCoste. Naïve filterbots for robust cold-start recommendations. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 699–705. ACM, 2006.

M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine learning*, 27(3):313–331, 1997.

M. J. Pazzani and D. Billsus. *Content-based recommendation systems*, volume 4321 of *Lecture Notes in Computer Science*, chapter chapter 10, pages 325–341. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72078-2. doi: 10.1007/978-3-540-72079-9_10.

C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience in super mario bros. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 132–139. IEEE, Sep 2009. ISBN 978-1-4244-4814-2. doi: 10.1109/CIG.2009.5286482.

P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on Graphics (TOG)*, 22(3):313–318, 2003.

A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. Arches: a framework for modeling complex terrains. In *Computer Graphics Forum*, volume 28, pages 457–467. Wiley Online Library, 2009.

M. J. Ponsen, H. Muñoz-Avila, P. Spronck, and D. W. Aha. Automatically acquiring domain knowledge for adaptive game ai using evolutionary learning. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 20, page 1535. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

A. Popescul, D. M. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 437–444. Morgan Kaufmann Publishers Inc., 2001.

J. R. Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

W. L. Raffe, F. Zambetta, and X. Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *Evolutionary Computation (CEC), IEEE Congress on*, pages 1–8. IEEE, 2012.

P. Rani, N. Sarkar, and C. Liu. Maintaining optimal challenge in computer games through real-time physiological feedback. In *Proceedings of the 11th International Conference on Human Computer Interaction*, pages 184–192, 2005.

B. Reeves. Tracking video game genre popularity. Game Informer, 2012. URL http://www.gameinformer.com/b/features/archive/2012/12/27/tracking-the-popularity-of-videogame-genre.aspx,Accessed:01/06/2013.

S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009.

F. Ricci and B. Shapira. *Recommender systems handbook*. Springer, 2011.

E. Rich. User modeling via stereotypes. *Cognitive science*, 3(4):329–354, 1979.

M. O. Riedl. Scalable personalization of interactive experiences through creative automation. *Computers in Entertainment (CIE)*, 8(4):26, 2010.

S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley. Combining search-based procedural content generation and social gaming in the Petalz video game. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 63–68. AAAI, 2012.

N. Rodrigues, M. Frade, and F. F. de Vega. Development of chapas an open source video game with genetic terrain programming. In *VII Congreso Espaol sobre Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB), Valencia, Spain*, pages 1–8, 2010.

B. Rusnell, D. Mould, and M. Eramian. Feature-rich distance-based terrain synthesis. *The Visual Computer*, 25(5-7):573–579, 2009.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

R. L. Saunders. *Realistic terrain synthesis using genetic algorithms*. PhD thesis, Texas A&M University, 2006.

J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: Collaborative interactive evolution of images. *Leonardo*, 41(1):98–99, 2008.

N. Shaker, G. N. Yannakakis, and J. Togelius. Towards automatic personalized content generation for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press*, 2010.

N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, et al. The 2010 mario ai championship: Level

generation track. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3 (4):332–347, 2011.

G. Shani and A. Gunawardana. *Evaluating recommendation systems*, chapter chapter 8, pages 257–297. Springer, 2011. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_8.

D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman and Hall/CRC, 2003.

R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A proposal for a procedural terrain modelling framework. In *Poster Proceedings of the 14th Eurographics Symposium on Virtual Environments EGVE08*, pages 39–42, 2008.

R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen. A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*. Citeseer, 2009.

R. Smirke. IFPI 2012 report: Global music revenue down 3%. BillboardBiz, 2012. URL http://www.billboard.com/biz/articles/news/global/1098245/ifpi-2012-report-global-music-revenue-down-3-sync-pro-digital,Accessed:01/06/2013.

A. M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan. An inclusive taxonomy of player modeling. *University of California, Santa Cruz, Tech. Rep. UCSC-SOE-11-13*, 2011a.

G. Smith. The seven deadly sins of pcg research. PCG Discussion Group, 2013. URL http://sokath.com/main/the-seven-deadly-sins-of-pcg-papers-questionable-claims-edition/,Accessed:10/12/2013.

G. Smith and J. Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 4. ACM, 2010.

G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha. Launchpad: A rhythm-based level generator for 2-d platformers. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):1–16, Mar 2011b. ISSN 1943-068X. doi: 10.1109/TCIAIG.2010.2095855.

N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. In *Applications of Evolutionary Computation*, pages 131–140. Springer, 2010.

K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, Jun 2007. ISSN 1389-2576. doi: 10.1007/s10710-007-9028-8.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen. Real-time evolution of neural networks in the NERO video game. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1671, 2006.

X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4, 2009.

G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9. Morgan Kaufmann Publishers Inc., 1989.

H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

J. Taylor and I. Parberry. Computerized clutter: How to make a virtual room look lived-in. Technical report, Tech. rep. LARC-2010-01, University of North Texas, 2010.

D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Interactive storytelling: A player modelling approach. In *Artificial Intelligence and Interactive Digital Entertainment conference, Stanford, CA*, pages 43–48, 2007a.

D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Learning player preferences to inform delayed authoring. In *AAAI Fall Symposium on Intelligent Narrative Technologies, Arlington, VA*, 2007b.

J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 111–118. IEEE, 2008.

J. Togelius, R. De Nardi, and S. M. Lucas. Towards automatic personalised content creation for racing games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 252–259. IEEE, 2007.

J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 265–272. IEEE, 2010a. ISBN 978-1-4244-6295-7. doi: 10.1109/ITW.2010.5593346.

J. Togelius, M. Preuss, and G. N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–8. ACM, 2010b.

J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation. *Applications of Evolutionary Computation*, pages 141–150, 2010c.

J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 3. ACM, 2011a.

J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, Sep 2011b. ISSN 1943-068X. doi: 10.1109/TCIAIG.2011.2148116.

S. Tognetti, M. Garbarino, A. Bonarini, and M. Matteucci. Modeling enjoyment preference from physiological responses in a car racing game. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 321–328. IEEE, 2010.

T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra. Generating consistent buildings: a semantic approach for integrating procedural techniques. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):274–288, Sep 2011. ISSN 1943-068X. doi: 10.1109/TCIAIG.2011.2162842.

A. Tychsen, M. Hitchens, T. Brolund, and M. Kavakli. The game master. In *Proceedings of the second Australasian conference on Interactive entertainment*, pages 215–222. Creativity & Cognition Studios Press, 2005.

P. Walsh and P. Gade. Terrain generation using an interactive genetic algorithm. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

P. Walsh and P. Gade. The use of an aesthetic measure for the evolution of fractal landscapes. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1613–1619. IEEE, 2011.

B. G. Weber, M. Mateas, and A. Jhala. Applying goal-driven autonomy to starcraft. In *AIIDE*, 2010.

L.-Y. Wei, S. Lefebvre, V. Kwatra, G. Turk, et al. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117, 2009.

F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, Dec 1945.

Q. Xu, D. D'Souza, and V. Ciesielski. Evolving images for entertainment. In *Proceedings of the 4th Australasian conference on Interactive entertainment*, page 26. RMIT University, 2007.

G. N. Yannakakis and J. Hallam. *Ranking vs. preference: a comparative study of self-reporting*, volume 6974 of *Lecture Notes in Computer Science*, chapter chapter 47, pages 437–446. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24599-2. doi: 10.1007/978-3-642-24600-5_47.

G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, Jul 2011. ISSN 1949-3045. doi: 10.1109/T-AFFC.2011.6.

K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno. An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):435–447, 2008.

H. Yu and M. O. Riedl. A sequential recommendation approach for interactive personalized story generation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 71–78. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

J. P. Zagal, C. Fernández-Vara, and M. Mateas. Rounds, levels, and waves the early evolution of gameplay segmentation. *Games and Culture*, 3(2):175–198, 2008.

H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, pages 834–848, 2007.

A. E. Zook and M. O. Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI (2012, to appear)*, 2012.