PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

Procedural Content Generation in Role Playing Games

Rhys Llywelyn Davies

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor of Science
in Computer Science

Supervisor Dr. Llyr Ap Cenydd

4th March 2022

Acknowledgements

I would like to give thanks to Stephanie Evans, my dearest partner and the one who kept me motivated through the course of this project. She is truly a shining light in my life and I would be remiss to not note her importance to myself.

I would also like to thank my personal tutor Dr Llyr Ap Cenydd, his mastery of the Unity platform and C# has provided much needed help during the course of this project, his creativity in guiding me to create the project was invaluable.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Rhys Llywelyn Davies

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Rhys Llywelyn Davies

Abstract

This project explores the uses of procedural content generation in the context of role playing games. The project examines available research on procedural generation methods and the games which utilise them. The overall goal of the project is to provide game developers a resource for understanding and using procedural content generation. The project accomplishes this goal by providing a detail overview of the design, implementation, and evaluation of a role playing game created using Unity and C# throughout the course of this project.

Contents

List of Figures

Chapter 1

Introduction

Over the past 40 years the video game industry has quickly grown to be one of the largest entertainment sectors in the modern world, with there being an estimated 2.3 billion active players in 2018, responsible for an industry worth of over $130 billion [3]. With so many people reliant on video games for entertainment around the world it is no wonder that the industry is always seeking new ways to keep their players engaged and improve replayability of their games, in addition to keeping costs low and profits high. Procedural Content Generation (PCG) is the process of using algorithmic means in order to generate content for a video game, with or without input from a human source, for example a player or developer [24], and can provide solutions for some of the problems the modern games industry faces.

The utilisation of PCG in video games is a practice that goes back to the 1980s with games such as Rogue(1980), which used PCG to generated rooms within a dungeon for players to explore, and Elite(1984), which used PCG to generate entire planets and star systems based on parameters kept as integer values, allowing large scale generation using small amounts of memory. This practice has been continued and refined throughout the years, including mass scale generations using pseudo-random values known as seeds which instantiate generative algorithms [20], such as the Perlin Noise algorithm used in games such as Minecraft (2011) to generate playable worlds. In addition, games such as Shadow of Mordor (2014) and Shadow of War (2017) use PCG to procedurally generate enemies and use PCG techniques to dynamically alter the challenge of the game and improve player experience by improving replayability.

PCG is a key area of study in respect to video games involving story based immersion, such as Role playing games (RPGs) or adventure games where the player's choices are the driving force behind the progression of the game's story and narrative. In cases such as RPGs, PCG can be utilised in a number of different ways such as map generation, NPC and AI generation, or loot generation, these applications will be further expanded upon in Section 3.2 when discussing other applications of PCG.

Chapter 2

Aims and Objectives

2.1 Aims

This paper aims to be a resource for game developers seeking to learn about PCG, particularly in the context of RPGs, including providing information pertaining to all aspects of PCG such as its history (Section 3.1), its applications (Section 3.2), techniques involving PCG (Section 3.3), and discussions about other papers that delve into PCG (Section 3.4). In addition, this paper aims to provide game developers a basic guide about the creation of an RPG which utilises PCG as a key portion of the player experience and will have its development and creative processes documented in Chapters 4 and 5, and a finished showcase will be provided as part of this paper.

2.2 Objectives

In order to complete the aforementioned goals several tasks will need to be undertaken:

Firstly, in order to provide developers with a clear view of PCG as a concept a background of the process will be given. Within the background chapter this paper will first explore the history of the process, this will include examples of video games that utilised PCG, and a more in-depth look at what PCG accomplished for said games. The history section will also explore other historic uses of PCG, such as in analog games, and will briefly discuss other fields of study such as generative art. Secondly, the background chapter will look more broadly at the applications of PCG within the context of video games, this will include looking at what can be procedurally generated, and

what is best left to the developer to hard-code into the game. Thirdly, within the background chapter this paper will discuss the techniques for using PCG within video games, be they through generated nav-mesh algorithms, or maps generated using a noise algorithm. Finally, the background section will review other related literature and how each of these works pertain to the fulfilment of this paper, this section will also be the basis for the idea of the RPG showcase the projects aims to create, and how previous games created by these works helped mould the game imagined;

Secondly, a design phase for the creation of the game will be utlised, this chapter will document the creative process for the RPG proposed by the paper. In addition, this chapter will explain the types of PCG used within the game, and the reasoning behind those choices. The chapter will not discuss the algorithms employed within the game, as this will be covered in a the implementation chapter. In total, the chapter will contain explanations of plans for the environment, the non-player characters (NPCs), the narrative, and any other information pertaining to the imagined player experience;

Thirdly, an implementation chapter will be employed to document the implementation of the design phase ideas into a suitable game engine, for repeatability and simplicity, this project will be undertaken using the Unity engine, and will aim to produce a 3D playable area with PCG elements that will be coded using C#. This chapter will explain the details of the algorithms used, the reason for their use, and what they will be used for. In addition this chapter will provide visual aids to developers, including screenshots of the rendered environment in Unity 3D, or snippets of code with an explanation of the techniques used;

Finally, an evaluation of the finished product will be given, this will take into account the effectiveness of PCG within the game, and how replayability, and player experience is affected by the presence of PCG in the game.

When all aims of the project are met this paper will conclude whether the proposed project was a success or a failure by analysing the game produced,

and the usefulness of the information provided within the background chapter. The paper will end by providing a look at any future work that could be made pertaining to the project, be that improvements to the game, or creating a more in-depth overview of the supporting literature and the background of PCG.

Chapter 3

Background

3.1 History of Procedural Content Generation

PCG can trace its history back to the mid 1970s with the invention of Dungeons and Dragons in 1974 by Garry Gygax and Dave Arneson giving players the chance to create endless worlds and narratives using a set of rules and a random number generator in the form of dice, this table-top phenomenon is enjoyed by millions [6] for its uniqueness and replayability. Across the years many attempts have been made to replicate the success of Dungeons and Dragons using digital systems, some of the first were in 1975 using the PLATO system at the University of Illinois [6]. These early games paved the way for more well-known titles such as 1980's *Rogue* which, as mentioned in Chapter 1, was one of the first commercial uses of PCG, and spawned an entire sub-genre of RPGs known as '*Roguelikes*' which aim to provide replayability and ease of storage through use of PCG [5]. PCG is often considered to be random generation, and while this is the case for some games, games can use parameters to direct randomness and provide constraints on generated content to ensure it is playable [21] as PCG can sometimes generate content that is unusable which is referred to as a *catastrophic failure*[24].

Most PCG based games of note utilise procedural map generation as this is perhaps the most well-documented and easiest PCG technique to employ when designing and creating a game. Use of PCG within gaming has very often relied on similar principles, from Rogue in 1980 to Diablo in 1996, the chief form of PCG was procedural dungeon generation, however, as methods and techniques were improved and refined new PCG based games provided

new and interesting ways to utilise PCG. Games such as Terraria (2011) and Minecraft (2011) both use PCG to generate the worlds the players inhabit, both incredibly popular with Minecraft selling over 100 million copies [6], all while having been originally developed by a single person [2]. No Man's Sky (2016) took PCG to further lengths, procedurally generating whole galaxies of planets, in a similar vein to Elite (1984), but on a much grander scale.

The use of PCG in games has provided players with a more personalised experience, with games such as Middle Earth: Shadow of Mordor (2014) players with an endless number of procedurally configured enemies to challenge and defeat, and replayability, ensuring their longevity within the gaming market, with games such as Minecraft growing an ever increasing player base and having been acquired by Microsoft in 2014, has grown to new heights, including releases for Windows 10 and mobile. In addition to the success of certain games, the entire genre of Roguelikes still heavily influences the gaming market with games such as Spelunky (2008) and its recent sequel Spelunky 2 (2020) showcasing how PCG can be used to provide replayable, interesting content for the player.

3.2 Applications of Procedural Content Generation

The primary use of PCG in video has been as a generator for maps or terrain, these types of PCG typically use a pseudo-random string of values as an input to call upon algorithms to generate the terrain, or dungeon rooms [20], this topic will be further expanded in Section 3.3. PCG has also been used to provide challenges for the player, and to create content that is both non-repetitive, and exciting for the player. Infinite Mario uses PCG to generate the portions of the side-scrolling area for the player at runtime [23], and the Middle Earth: Shadow of games (2014, 2017) both implemented procedurally generated enemies with varying strengths and weaknesses to challenge and both further the player's narrative within the game. PCG can also be used as a more direct and less interpretive way of developing story and narrative as

Woldwalker Games' 2019 release Wildermyth demonstrates. Wildermyth is a tactical, turn-based RPG that utilises PCG in order to select pre-written stories to help a narrative between characters within the game unfold naturally [10], the game itself gives players a great deal of freedom and independence as it allows for players to make choices to decide how their adventure will pan out, and affect how the game itself is played. Story driven PCG is not only limited to fantasy genres as Orange Care, a serious, educational game proposed in the 2019 Brazilian Symposium on Computer Games and Digital Entertainment, shows. Orange Care utilises PCG to create characters to be diagnosed by the player, who will examine skin lesions and determine whether they are cancerous or not, the storytelling aspect is introduced via procedurally generated emails and feedback reports given to the player after the diagnosis, this helps to simulate a more realistic environment for the player as the character remains as part of the game even after diagnosis [18], papers like this help to demonstrate both that video games are a technology which can be utilised as an integral, and important part of modern life and also that PCG can help mould stories to create a more tailored experience, and a more emotional response from the player. Storytelling can also have an effect on the generation of terrain, and as such PCG based storytelling may also utilise PCG based terrain generation in particular for RPGs. For example, Matthews and Malloy in 2011 propose that map generation can be a good way to procedurally develop the narrative of the game, in the paper it is discussed that a large proportion of any RPG involves travelling, be it towards a town, or a bandit hideout, or a dungeon, and while most games actively create maps that seek to develop storytelling through this travel, using PCG to accomplish this can help to create a more unique, less derivative world for the player to explore, once again leading to a more tailored experience. The paper demonstrates that by providing parameters based on factors such as weather, distance from other towns, and other constraints by the developer it was possible to create entire continents full of towns that were connected to each-other via a series of interwoven narratives [15]. This paper demonstrates the usefulness of PCG by showing that an algorithm can be used to reduce the amount of work needed when creating a world that is designed from a storytelling perspective.

In addition to terrain generation, challenge and enemy generation, and story based generation, PCG is a technique that can be applied to almost any facet of a game. In the Hendrikx et al. survey in 2013 it was proposed that there were 6 primary areas where PCG could be utilised, being: Game Bits, Game Space, Game Systems, Game Scenarios, Game Design, and Derived Content [13].

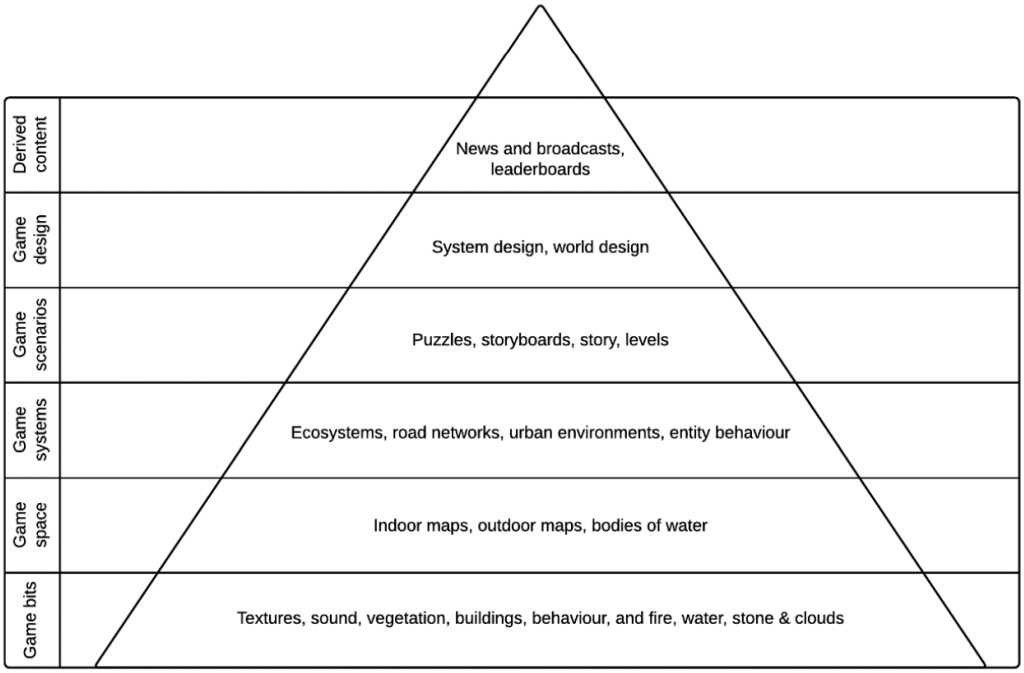

Figure 3.1: A demonstration of the hierarchy of the Hendrikx et al. content types, image from Bomström 2016 [5]

As Figure 3.1 demonstrates, each primary aspect of content is responsible for several other aspects of the game, and as such different aspects of PCG will focus on different sections of the hierarchy. For example, using PCG to develop terrain or maps would likely involve sections such as Game Bits and Game Space, but would not likely utilise Derived Content or Game Scenarios. Important distinctions such as the separation of the sections in the hierarchy help to provide developers with a better idea of where to focus their efforts when implementing PCG into their game.

Music is often used in video games to dictate both the tone of the game, and instruct the player of a challenge or a change in the world. In games such as Terraria (2011) and Dark Souls (2011) music is often used when a

player encounters a boss enemy within the game world, this "boss music" often has connotations to a challenge for the player and so often video games use scores of dramatic music to increase the adrenaline of the player and affect how they feel and play in that moment. Plans and Morelli's 2012 paper: "Experience-Driven Procedural Music Generation for Games" proposes the use of PCG to develop music for use within video games [19], in their paper it is demonstrated that by tailoring the musical experience to the player's actions, by controlling aspects such as the beats per minute (BPM) and novelty of the music played, the player is more likely to enjoy the gameplay experience, and be less frustrated when facing difficult challenges [19]. The paper itself affirms a primary benefit of the use of PCG: tailoring an experience to the player is more likely to make the game more enjoyable.

3.3 Procedural Content Generation Techniques

PCG can vary in the complexity of its implementation. A very simple form of PCG may be to introduce a random number generator (RNG) aspect to the game, contrarily PCG implementation may involve use of incredible complex algorithms that use developer input to generate elaborate systems or content. An important distinction that must be made for PCG is that procedural does not always imply random, constraints are often put in place when designing a game using PCG to ensure that a catastrophic failure does not occur and the content generated is playable and usable [5]. Even when utilising a pseudo-random number generator (PRNG) or seeds for use with algorithms to create maps and terrain, a given value will always provide the user with the same output from the algorithm [20], as such pseudo-randomness can be extensively used to create and refine certain types of content, as the output will not change due to the seed, which means that aspects of the seed can be changed to change the content itself.

As terrain generation is one of the most used aspects of PCG there are a number of approaches a developer may take when generating an aspect of the environment. For example, when using PCG to develop terrain of the

world of the game there are several algorithms that can be used, as is shown in Table 3.2.

TABLE I. NOISE FUNCTIONS

Algorithm	Speed	Quality	Memory Requirements
Diamond-Square Algorithm	Very Fast	Moderate	High
Value Noise	Slow - Fast*	Low - Moderate*	Very Low
Perlin Noise	Moderate	High	Low
Simplex Noise	Moderate**	Very High	Low
Worley Noise	Variable	Unique	Variable

*Depends on what interpolation function is used
**Scales better into the higher dimensions than Perlin Noise

Figure 3.2: A table displaying the various algorithms the could be utilised to procedurally generate terrain for a video game, table from Rose and Bakaoukas 2016 [20]

Noise generation is generally favoured for generating terrain as it can be difficult to represent nature using Euclidean shapes, as such fractal generation such as noise helps to provide a more realistic output of nature, by creating more chaotic and varied terrain, that has the benefit of also providing a heightmap which adds an additional layer of depth to the game world [20]. However, when using PCG to create a road system or village use of noise is not favoured as roads and settlements tend to focus more on using Euclidean geometry in planning, and as such algorithms that take into account Euclidean distance are more widely used for generating villages and road systems [4]. A common method is to use triangulation to connect vertices in order to create a road system that does not change in elevation too rapidly, this is accomplished by looking at the terrain map and assigning height values, the road will then be given a beginning and end point, and the algorithm will work through the grid of the terrain to provide the shortest, most logical route possible, that involves the small differentials in height possible [4].

PCG can also be implemented into RPGs by using pseudo-random number generation to generate loot for the player at runtime by utilising adaptive content generation (ACG), this can be accomplished through asking the system several questions about the player and their actions and assigning values to the answers to affect the formulae used to calculate loot for the player to influence what kinds of weapons would be given, their damage output, and other such factors [16]. Dynamic Difficulty Adjustment (DDA) is another form of adaptive content which is used to create new challenges to the player by analysing the state of the game and determining whether or not to make the game more or less difficult. This could be accomplished by changing the number of enemies, or certain aspects of the enemy such as amount of health or damage. DDA is often used as it keeps the player engaged, this arises from the fact that the concept of "Flow", a term used by psychologist Mihaly Csikszentmihalyi to describe the player's mindset when facing challenges [16]. Flow dictates that if a player finds a game too easy they will lose interest, and if a player finds a game too challenging they will lose motivation of every completing the game. The key to proper DDA is therefore to strike a balance between difficulty, thus ensuring a more constant level of enjoyment for the player [14] this balance is demonstrated in Figure 3.3.

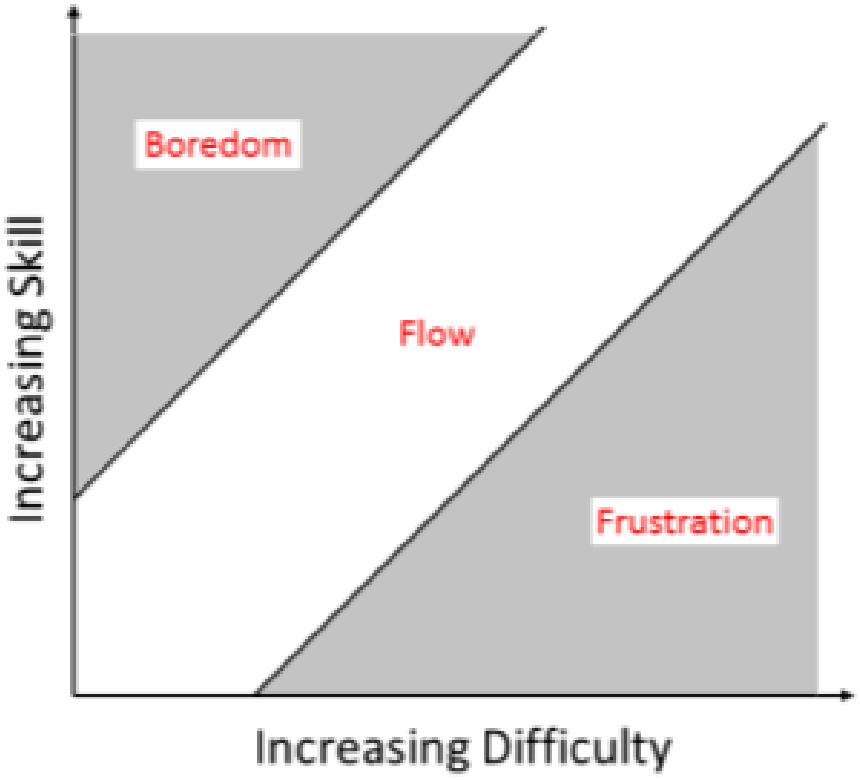

Figure 3.3: A visual demonstration of how DDA maintains a fine line between boredom and frustration by utilising a *Flow* [17]

A third example of PCG implementation is experience driven (ED) and is often used to prioritise the player's own enjoyment by taking information about the player's actions and, similarly to ACG, uses it to generate content for the player in an effort to provide a more enjoyable experience. One such example of EDPCG is to create, or change the game's music based on how the game is being played, this helps to keep the player invested and stops the game from becoming repetitive [19], and can be accomplished by using PCG algorithms to feed information about the player's movement or the current challenge, including boss enemies or different puzzles.

Finally, there is Search Based PCG (SBPCG) which is a more evolutionary approach to PCG and helps to create more branching and varied experiences for the player. SBPCG tends to work on a cyclical methodology to constantly use the player to provide constructive criticism to game mechanics or challenges that are then changed for future challenges or mechanics. SBPCG

utilises what is known as a test function which will grade certain aspects of a game based on the evaluation by the player which can be provided by considering the player's enjoyment of the level and other factors such as the time taken to complete or frustration, this grade is called a fitness value. Content with a higher fitness value is more likely to be generated than content with a lower fitness value, as such the player is provided with content that is more enjoyable and the content can be further refined by the algorithm [24] shown in Figure 3.4.

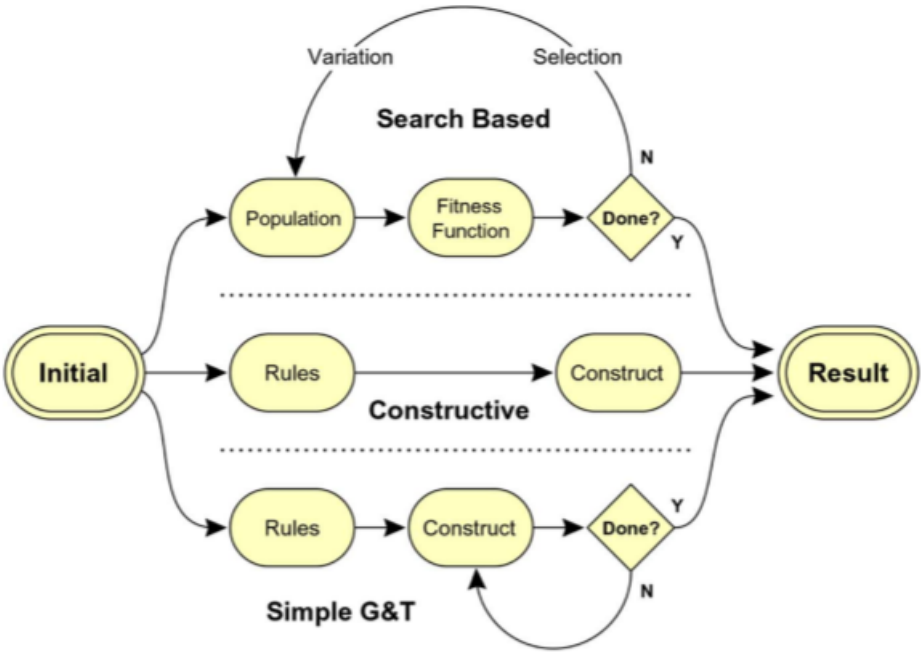

Figure 3.4: A visual representation of the three major types of content generation algorithms, with Search Based techniques, Constructive techniques, and Generate and Test techniques shown [24]

As is seen in Figure 3.4 Search Based methods allow for more variation when generating content as opposed to Constructive generation, which provides content based solely on rules and parameters, and the Generate and Test method which will only reiterate content if there is a fault during construction or the content is unfinished, i.e. a *catastrophic failure* [24]. Search Based therefore allows for more varied and interesting content to be generated as there are a number of factors upon which it relies such as player experience, which as discussed previously can help improve the enjoyment and is more likely to ensure the player continues playing the game [16]. SBPCG can also

reduce discomfort from having to deal with a challenge or mechanic that the player does not enjoy or does not want to replay, this is valuable in PCG as it provides players with the most optimal and enjoyable gaming experience possible [22].

3.4 Related Work

Utilising PCG to either create or improve the experience in RPGs is an ongoing field of study with a number of exciting and varied project currently in development to demonstrate how PCG can be of use.

In *Adaptive Content Generation for Games* Oliveira and Magalhães seek to lay the foundations of a new approach to utilisation of PCG for a wave based RPG that generates large aspects of its game using PCG techniques such as DDA, ED, and rule-based pseudo-random algorithms. The game itself also takes into account different aspects of the player's behaviour such as their play style, their class choice and weapon choice to create a more immersive experience for the player. Oliveira and Magalhães conclude by proposing that using PCG that is tailored to the individual is a clear way to improve the player experience, and outline potential for improvements that could be made to the existing models and data collection methods [16].

Lara-Cabera et al. explore the value of evolutionary algorithms in *Procedural Content Generation for Real-Time Strategy Games* by evaluating two basic Real-Time Strategy (RTS) games called *Planet Wars*, in which players accumulate ships to invade and control other planets, and *RobotWars*, which was a strategy based game in which the goal was to eliminate the enemy general. *Planet Wars* takes into account any imbalances such as growth, ship number, and even game length to determine how the AI will allocate resources and even how subsequent maps will be procedurally generated. *RobotWars* uses a "Hall of Fame" method and co-evolutionary cycle to develop AI strategies to challenge the player. In their conclusion Lara-Cabera et al. discuss how using PCG in the context of an RTS game can yield significant

results and thus improve the player experience by creating a more balanced game enviornment and AI [7].

Globalized Random Procedural Content for Dungeon Generation by William Forsyth attempts to formulate a new algorithm for the generation of dungeons by taking inspiration from the algorithms used in 1980s *Rogue*, Forsyth improves on the original design by removing the dimensionality of the world, allowing dungeons to be unrestricted by the physical space of the game world by utilising non-Euclidean geometric methods. The algorithm proposed uses graph theory to create varied rooms and challenges for the player to explore and endlessly generated dungeon. In his conclusion Forsyth explains that the revisions made to the original algorithms used in *Rogue* had been successful in generation of non-dead-end corridors, but had drastically increased the time taken to generate the same number of rooms, which Forsyth concluded was due to the need to generate 5-levels, each with the same number of floors, as opposed to the original algorithm's singular level [9].

During the 10^{th} Doctoral Symposium in Informatics Engineering (DSIE'15), Ulisses et al. propose the use of PCG to integrate narrative into map and world design. By developing a prototype serious game called *Project Orion* Ulisses et al. were able to demonstrate the variety of techniques that can be used to improve the narrative of a procedurally based game. Techniques used for *Project Orion* include algorithmically based room placements, exploration based problems, and event based dialogue. *Project Orion* itself makes heavy use of developer input as the majority of gameplay mechanics and challenges are editable within an external XML file, and, while the game itself is built using Unity3D and C# code, the enemies provided utilise a JavaScript code which allows for a more dynamic enemy experience for the player. Ulisses et al. conclude that use of an editable XML helps to bring a greater depth to the world as the player has greater control over the type of challenges they may face, however, the procedural nature of the game ensures that the nature of the story is provided to the player by means of exploration, improving their experience and reducing boredom [25].

Chapter 4

Design

4.1 Original Concept

The original concept and design for the game developed as part of this project was a first-person Role-playing game utilising PCG. PCG would be integrated into the game by means of dialogue and quests, the player would interact with various NPCs which would react to the player's dialogue choices and also the player's actions. The overall aim of the game was to provide a unique repeatable experience for the player by creating procedural NPCs that react in real-time. This original idea, however was not feasible for the time limit, as some of the NPC mechanics proved to be too complex for the project scope.

4.2 Revisions to Original Concept

The first major revision to the original concept of the showcase was to exclude the use of procedurally generated and reactive dialogue in favour of using procedurally generated NPC jobs and quests. The quests would be provided to the player through interacting with an NPC of their choosing. The player themselves would only be able to hold a single quest at a time. The quests would involve talking to another NPC, upon which the player would be rewarded with a procedurally generated item. The item would depend on the NPC who original gave the player the quest, for example a Blacksmith NPC may provide the player a sword or piece of armour, whereas a Fletcher NPC would lean towards providing a bow. Once the item is determined several other statistics are considered, its rarity, attack value, and any other modifiers that could be added to it. Should a player speak to an NPC while on a quest the

NPC will direct the player towards the correct NPC. All NPCs will be assigned a colour which corresponds to their occupation, and upon being directed towards a certain NPC their name will be coloured in a colour matching the NPC in question.

Another concept that was considered was procedural weather or a day/night cycle, to add some ambience to the scene, the weather would either be clear or rainy, and would combine with the light levels gradually changing across time. Using outside lighting, such as point lights or area lights, would help establish a warmer feeling to the environment and will be discussed further in this chapter.

Finally, a concept that was considered was having an interact-able door which would lead to an area for combat, whereby the player could be tasked to fight enemies using the weapons they have accumulated through quests. For this to function properly the player would need a starting weapon as it would be possible to have a combat focused quest be the first quest allocated to the player.

A change in camera angle was considered, utilising a third-person fixed perspective, instead of the originally proposed first-person perspective, to provide a more cinematic and easier to focus perspective for the player, however, this was ultimately determined to be inadvisable as it would make the NPCs more difficult to interact with due to problems with depth perception and positioning.

4.3 Environment

The original concept was to place the player within an medieval tavern style environment, utilising free assets from the Unity store. The goal of this environment was to provide the player with a a comfortable and familiar space to allow the player to experience the procedurally generated NPCs and quests in a welcoming, small area to reduce the need for movement, while maintaining a certain level of atmosphere.

Originally there was a plan to have multiple rooms, to add a depth of space and a dimension of exploration to the game. While, this is still a possibility in the future for the current iteration of the game focus is maintained on a singular room. In addition to other rooms, as mentioned previously, there was to be a combat area for use within combat based quests. The combat area would have been designed as a mine of some kind, with the main door to the tavern being the access point to this new level. The mine itself would have had a much more natural look, and could have even been procedurally generated itself. The colour palette used would likely have been a number of shades of gray, due to the prevalence of what would mostly be rocks. However, colour could be added to the mine by using certain colours like purple, red, or green to indicate gems or rare minerals within the rocks. Lighting would have likely come from torches rather than natural lighting to imply depth and distance from the sun. As a combat system was not decided to be used within the first iteration of the game, a mine was not modelled within the Unity scene.

The final design of the environment was a small to medium sized tavern, consisting of a bar, complete with glasses, bar stools, and bottle behind the bar, a number of tables, barrels, and candlelight. These tavern environment was populated by Unity cylinders, which are the interact-able NPCs, these NPCs are placed in locations around the bar to give an indication of a busy tavern. The NPCs are not intended to move as it could prove frustrating for players to have to deal with moving entities, and the complexity of the code could lead to worse performance, as algorithms to ensure the NPCs don’t bump into each other and the player would need to be implemented.

Lighting in the scene will have two main sources. There will be point lights placed onto the candles on the bars and tables to provide a warm lighting around areas of congregation adding to the atmosphere. Originally there was to be a chandelier which would also provide lighting, however, this caused problems with player movement as the player would hit the ceiling when jumping, as such the chandelier was removed in favour of individual candles on tables. The second light source would come from a direction light which

would shine from left to right, and the intention is to give it a darker hue and lower brightness to indicate a late evening setting.

4.4 Inspirations

The original concept for a procedurally generated RPG draws from a number of inspirations from video game culture. RPGs such as Skyrim and other Elder Scrolls games, alongside games such as Runescape or the Zelda RPGs provided the inspiration for the setting and overall atmosphere of the showcase. While games such as Wildermyth and the Middle Earth: Shadow of series provided more inspiration towards the procedural nature of the project.

Utilising procedurally generated weapons was a concept that was originally added to the project inspired by the work done by Oliverio et al. in their paper "*Adaptive Content Generation for Games*" where weapons were generated procedurally based on a number of factors such as the number of waves reached by the player [16]. While the project is not intending to provide wave based challenges providing procedurally generated items/weapons as rewards for completing quests seemed to be a good incentive and interesting way to incorporate various PCG methods into the final game.

Using Procedurally generated NPCs was inspired by both Wildermyth and the Middle Earth: Shadow of series, both of which utilise PCG to generate NPCs that the player can interact with, whether it be through combat, or through the addition of a member to an adventuring party. The goal of the project was not to provide players with an enemy to fight or companions to recruit, but to provide a small immersive experience of a typical RPG. As such, the NPCs designed to be ordinary people within the RPG, those whom a player may interact once or twice with, with the aim of making them feel more immersive to the player by being a constant source of interest when playing, either by providing the player with a new quest or with a reward upon its completion. This area could be further expanded by having the NPCs sense changes in their enviornment and comment about it. For example, a day/night cycle or

weather could add more depth to procedurally generated NPCs by having them make certain comments based on the time, the place, or the climate.

Providing NPCs with a job was inspired by the villager system in Minecraft, where procedurally generated villages are populated by interactable NPCs that can be traded with if they are assigned a job. While Minecraft contains a number of villager jobs, for the project it was believed best that only some should be used to keep the complexity level of the showcase lower in order to avoid overwhelming the player.

4.5 Finalised Design

The final iteration of the design was a medieval inspired tavern containing a bar, tables, windows and lighting from candles, and five procedurally generated NPCs. Each NPC is to be assigned one of five jobs, with no repetitions, these being: Blacksmith, Fletcher, Farmer, Butcher, and Wizard. Each of these jobs is to be represented by a colour: Black for the Blacksmith; Light brown for the Fletcher; Green for the Farmer; Light red for the Butcher; Blue for the Wizard. Through providing the NPCs with distinct colours the hope is that players are able to easily distinguish between the NPCs which would prove useful for identifying the target of a given quest. In order to aid the ability for the player to locate the quest target, when the quest is given the name of the NPC in question will be provided and the name will be coloured the same as the NPC, for example: "*I need you to go and talk to the Farmer for me*". When the player has received a quest they will be unable to gain another until the previous is completed. The intention is to ensure that the player does not overload the number of goals they have. To ensure this when a player speaks to an NPC different to their target they would be directed towards their target, for example: "*I think you're looking for the Fletcher*". Upon completing the quest an item will be provided to the player, to be placed within an inventory, the item will be given a type, these being: Sword, Bow, Scythe, Knife, or Staff. Each weapon corresponds to a job, that meaning that a quest giver with the Blacksmith job will be more likely to provide a Sword, with a Fletcher being more likely to give a Bow and

so on. It is important to note that it is the quest giver not the target who is responsible for providing the item. Each item will be given a rarity which will be influenced by the number of quests the player has completed, this rarity will also affect the bonuses of the weapon attributes. These attributes will include weapon attack, range, critical hit chance, and the possibility of elemental damage. The weapons will be stored in an inventory, in which they will be able to be examined by the player, but not equipped, as the modelling and rendering of the items proved to be too time consuming and could cause the player’s view to become cluttered.

Chapter 5

Implementation

5.1 Environment

As the showcase itself is being built and coded using Unity's scene manager and Unity's C# compiler, the use of the Unity asset store was available for use within the project. When implementing the environment, it was imperative to provide the player with a depiction of a medieval tavern, this was accomplished using the Medieval Tavern Pack by 3DeLucas[1], which provided the prefabs such as the walls, windows, bar, and tables, while the flooring was handled by the Wooden Floor Materials asset pack by Casual2D[8], which gave the plane used for the floor within the scene a wooden texture.

The first stage when implementing the environment was deciding on lighting to suit the atmosphere within the scene, as the scene was within a medieval style tavern, somewhere which would typically be more populated during late evening, a lower light level was chosen to reflect this. At the beginning of the implementation lighting was provided in the scene using the Unity directional light, which acts as a global light level. Later on in development it would be possible to include point lighting on the tables to give the illusion of soft candlelight, but this is not integral to the overall showcase.

Figure 5.1: A screenshot demonstrating the initial implementation of the available area for the player to interact with NPCs

When designing the tavern to be implemented, it was important to keep player movement and mobility in mind, as such when looking at Figure 5.1 there are large areas of empty space, with tables and chairs focused mainly around the edge of the environment. This was done deliberately in order to not inhibit the player due to the primary function of the game being to move between NPCs. Having the primary area within the environment focus on movement, with decoration utilised on the outside ensures that players will not have to deal with obstacles, and does not diminish from the feeling of a tavern, as though the tavern is sparse, the abundance of tables and chairs make it also feel warm and populated.

NPC placement was another key feature of the environment, as it is the primary source of interaction for the player. Each NPC was placed at a position to ensure they are nearly equidistant, while also being somewhat sporadic to add a more natural feel to their placement. The aim of having NPCs placed at convenient locations in the tavern was in order to ensure that they were easily reachable and recognisable by the player, thus ensuring ease of movement between NPCs.

During the implementation it was important to keep parts of the scene separate from one another to ensure that the game ran smoothly without any misplaced objects, and that objects could be moved easily without fear of disrupting other objects. As such, creating a hierarchy of objects within the scene was imperative.

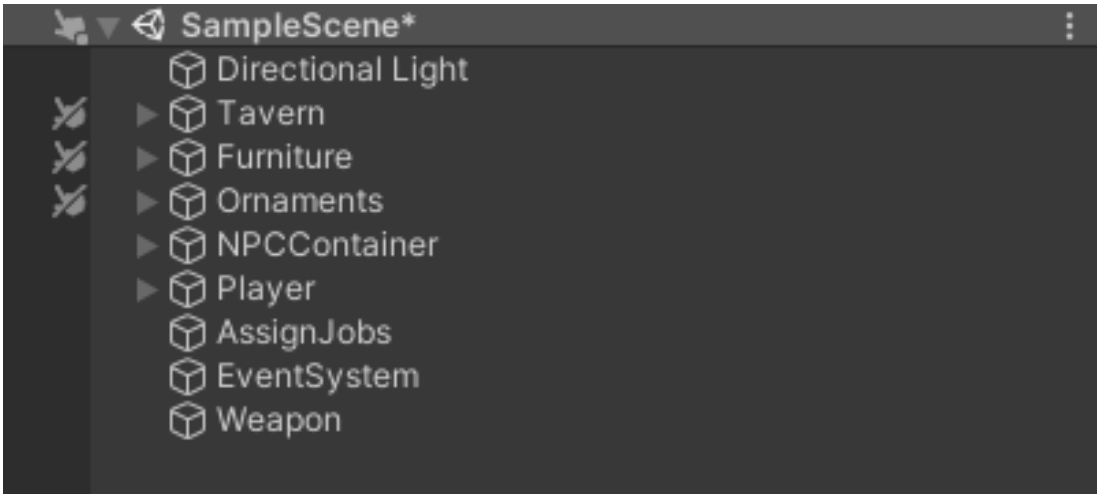

Figure 5.2: An overview of the hierarchy of objects within the game scene

As shown in Figure 5.2 each section of the tavern was given a container which ensured that when placing other objects, such as the NPCs or the player, no part of the environment was changed. The "Tavern" container held all objects and prefabs used in the building of the structure of the tavern, this being the walls, flooring, roof, and windows. "Furniture" held all objects and prefabs such as the tables, chairs, barrels, and the bar itself. "Ornaments" refers to all objects and prefabs which decorated the tavern, these included candles found on tables and barrels, and glasses, both on tables and on the bar shelves. Having these three containers helped separate sections of the

project and allow for editing of a specific section, in most cases "Ornaments", as they provide the most immersion in the scene while also not hindering player movement or mechanics.

5.2 Player

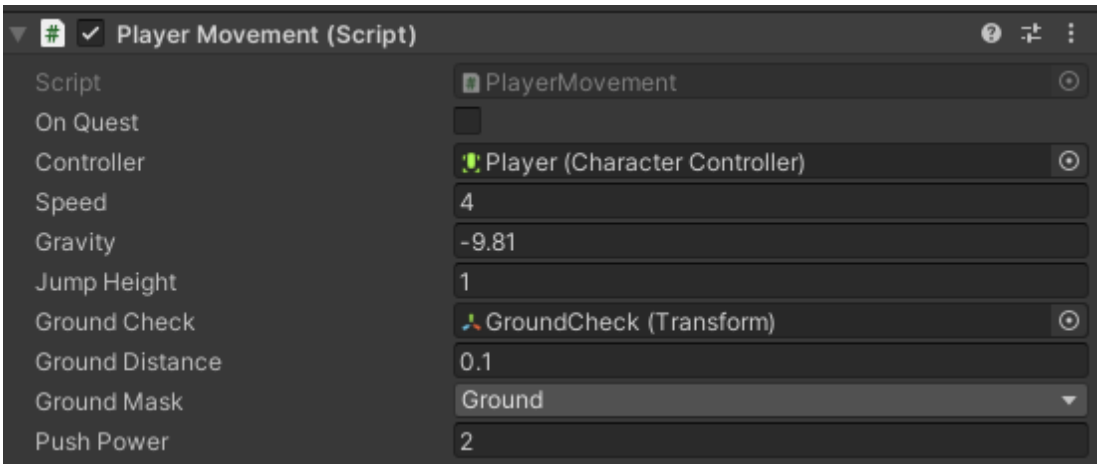

Figure 5.3: The player controller in the Unity inspector

The player has two main functions within the showcase, to interact with NPCs, and to move around. Using C# as well as the Unity character controller a player character was constructed that is moved using the W, A, S, and D keys on the keyboard, can use the spacebar to jump, and the F key to interact with NPCs to receive quests. All values for player movement, jump height, and gravity are adjustable, see Figure 5.3, as they are set to public values, allowing for greater customisability within the showcase, by increasing move speed of the player, quests can be completed faster to demonstrate the mechanics in a much more rapid fashion. Interactions with NPCs are handled using a Unity feature in C# called raycasting. Raycasting is the process of sending a line outwards from a point to a radius or distance depending on the kind of raycast, in the context of the player in the showcase there are two types of raycasts used. The first raycast used is to check the distance from the bottom of the player to the ground, this uses a CheckSphere() method which tests to see if an invisible sphere of size X, starting from the position of the bottom of the player, contains any objects with the tag "ground", if this is the case the player is considered grounded and can therefore jump again,

if this is not the case the player is assumed to be mid jump and so cannot jump again. The second raycast used is for interaction with NPCs, and is a traditional Unity Raycast() method, which sends a ray from a position out in a certain direction, in this case forward, to a certain range. It was imperative to provide a fairly restrictive range to the raycast to ensure players were encouraged to move around and explore the environment while interacting and completing quests.

5.3 Non Player Characters (NPCs)

The NPCs are the most crucial part of the showcase, as they exhibit all aspects of PCG within the showcase. Each NPC is equipped with an NPC script which handles its "Title", "Colour", and "Weapon". At runtime the NPC is assigned three attributes from the AssignJobs script, attached to an empty game object within the script. The attributes are kept using a List<> function within the NPCJobs script attached to the NPCContainer. When the game is begun the AssignJobs script uses a foreach() function to look through each NPC object within the NPCContainer and assign it a value of i, this value is a unique value between 0 and 5 which corresponds to the index value of the title, colour and weapon in the NPCJobs script. The attributes are then given to the NPC and its material is changed so that the colour is equivalent to the attributed colour using Unity’s render and material.color setter. Utilising these steps to generate attributed NPCs and given that there are 5 possible values for i this means that there are 120 different ways that the NPCs could be ordered, which would increase dramatically if there were more NPCs, for example 10 NPCs would give over 3.6 million different combinations of NPC order. Using a system such as this means that the more complexity added to the game the more replayable the game becomes as new combinations are essentially guaranteed.

The second aspect of PCG that NPCs exhibit is that of quests. Whenever the player interacts with an NPC it will select a random NPC target that is not itself as the objective of the quest, this was accomplished within the player script by creating a method called GiveQuest() which takes in an argument of an NPC

object. This method is called when a player who is not on a quest interacts with an NPC, and generates a random value between 0 and 5 once again and uses that to select an NPC. If the value corresponds to the current NPC and is less that 4 then its value is incremented by one, or if it is equal to 4 its value is decreased by one. This ensures that the target of the quest will never be the same as the giver of the quest. Questing is very simple in the showcase, an NPC provides the player with a quest to interact with another NPC, upon interacting with the NPC in question the quest is considered completed and the player is given a reward. This reward is also procedurally generated and uses a method, GiveWeapon(), in the script Weapon. This method also takes an argument of NPC which is equivalent to the NPC who is the target of the quest rather than the quest giver. The weapon itself has 5 attributes in total: Rarity, Attack, CritChance, Range, and Elemental. Each attribute is generated using a random number, which then applies a specific value to the weapon. Firstly a rarity for the weapon is chosen, these range from common to legendary, with each tier having a different percentage chance of occurring, for example legendary weapons are only provided if the value of j is greater than 94, while common weapons are given if j has a value lower than 44. These values act as a percentage chance for the rarity as the value of j is always between 0 and 101, non-inclusive, as such legendary weapons have a 5% chance of being rewarded while common weapons have a 45% chance, this helps to balance the weapons as each rarity offers a greater modifier, and also provides a source of replayability for the player as rarity would be sought after. Modifiers are multipliers dictated by the rarity of a weapon which then directly influence the attack and critical chance of the weapon, with modifiers ranging from 0.75x at common to 1.5x at legendary. These modifiers are multiplied by a random number between 100 and 500 for attack, and 0 and 50 for critical chance, with critical chance being the percentage chance for a attack to deal critical damage. This means a perfect weapon could have a critical chance of 75% and a damage output of well over 700 should the weapon generate in such a way. Range is the next attribute and is assigned differently depending on the kind of weapon, for example a bow and a staff will have a range of between 5 and 8, which would be the maximum distance a player could inflict damage from, while other weapons would have a range

of at most 2. This was done to add realism to the game as it wouldn’t make sense to have a bow with an effective range equal to a sword and visa versa. The final attribute is elemental and dictates what, if any, elemental damage the weapon will inflict. There are four elements at this point in time: Earth, Water, Fire, and Air. In order to calculate if a weapon were to have elemental damage attached to it a percentage chance was once again used, as such any value below 60 meant that a weapon was not elemental, while every interval of 10 from 60 and above the element changed, and with it the output of the method. The method returns a string based on the weapon generated, if the weapon is an elemental weapon the string returned will be: "You’ve been given a ’rarity’ ’weapon’ of ’elemental’! Attack: ’atk’ Critical Hit Chance: ’critChance’ Range: ’range’. While if the weapon has no element attached to it then the string will return a similar sentence without the "of ’elemental’" part. This was done to ensure that the output the player receives did not have any null values inside it to reduce the risk of errors.

5.4 User Interface

In its first iteration the output of quest dialogue and rewards were logged directly to the console on Unity. In order to improve gameplay and immersion a user interface (UI) was implemented. This UI utilised assets from the Unity asset store[11] in order to provide a simple wooden graphical user interface (GUI) to hold all text that was given to the player from NPCs or from quest rewards. This was accomplished using a Unity canvas and event listener, as well as using scripts to output the values of strings to appear as text on the GUI. A second use the UI had is that of an inventory system, where the rewards players had gather would be visible to view by pressing the I key, the inventory itself is an empty array of objects which a filled when the GiveReward() method is run. The inventory itself is in rows of 9 items, and allows the player to scroll if they have too many items to fit on screen at once. When an item in the inventory is hovered over, it will provide its rarity, name, and other attributes.

Chapter 6

Evaluation

Looking back at the project and how the final showcase has developed, it is clear to see that the aims and objectives have been accomplished. This evaluation will look through each individual objective and the overall aim to determine how they have been fulfilled.

The first objective was to provide a detailed background chapter on PCG, this was accomplished by performing a literature review on other sources available and using the information garnered in order to provide a clear overview on how PCG was first implemented and how it has evolved across the years, and how it is used today.

The second objective was to provide a design phase for the creation of a PCG inspired game showcase. This objective was accomplished by providing details on all aspects of the game before its implementation, including original ideas and reasons for including or not including certain aspects within the showcase.

The third objective was to implement the design and give an overview of coding the game itself. The conditions for this objective were that playable 3D environment with NPCs was created and that explanations of techniques and code was provided, both of which were discussed in Section 5, and can therefore be considered a success.

Finally, an objective outlining an evaluation of the finished product will be discussed now. The game showcase itself is a simple 3D RPG in which the player character can receive a quest from an NPC to talk to another NPC and

upon doing so will be rewarded. The PCG within the game is effective as there are a varying number of combination available with positions of NPCs, quests given, and rewards. The player experience is admittedly somewhat unexciting, however, the game itself provides a solid foundation to further expand on the game by implementing various more exciting applications which will be discussed in Section 8.

Overall, the aim of the project was to provide developers a resource for utilising PCG within the context of RPGs, from the above evaluations we can see that this condition has been met as it provides information on varying aspects from theoretical background knowledge to practical coding techniques. The project can therefore be labelled a success.

However, improvements could be made to the project, such as including information from sources which were outside of the scope of the project, or to provide greater detail on game that utilise PCG techniques. These factors were omitted from the final project iteration in order to ensure that the project fell within the purview of its requirements and that all information was given in its most concise form to make sure that the reader remains engaged in the reading of the project.

Chapter 7

Conclusion

PCG is an ever-expanding field of study with a huge potential application, both in gaming and further afield. PCG allows for the improved replayability of a game without the requirement of a developer needing to hard code new content. It provides players with the option to either explore a world that feels truly alive and natural, or experience a challenge that will never become too repetitive for them, it allows developers to create beautiful landscapes, and interesting rewards without needing to compromise on other aspects, giving them more time to develop narratives or character development to aid in user engagement. PCG is an important aspect of both the future and past of gaming and a valuable tool to developers looking to add depth to a project while being uncompromising on the core mechanics.

Chapter 8

Future Work

When discussing further work for the project this is referring to the PCG showcase developed in tandem with the project. The showcase itself does fulfil all objectives proposed by the project, however some aspects of the showcase could be improved upon, or expanded to further add to new areas for the player to explore and more mechanics for the player to master.

A primary piece of further work would be to implement a combat system, in order to give the player a use for the rewards they receive from quests. This combat system could include methods for attacking, blocking, or charging attacks, which would be used in regard to the staff or bow weapon types. When implementing a combat system, areas with enemies would be a priority to allow for players to utilise the combat mechanics. This could be accomplished by have the player interact with a door object to change the level to a new area. In keeping with a fantasy RPG theme, a dungeon or mine would be an appropriate design for this new level. The area itself could contain obstacles and ornament, and even multiple levels for added verticality during combat. Enemies could be procedurally generated to be placed in random areas and could use pathing systems such as A* pathfinding system[12], which calculates the minimum cost path for movement to ensure efficient movement of NPCs between points.

Building upon the existence of combat and enemies within the game, questing could be improved by providing players with tasks to defeat X amount of enemies, this would keep the game fresh for players, in addition adds a challenge to the player, which could utilise DDA[16] to keep the game

engaging and provide more longevity to a playthrough. When discussing enemies and questing it is also important to outline ideas for enemy drops, a possibility would be to provide the player with a type of currency available which the player could use to upgrade or buy new weapons from respective NPCs, this could also further improve player to NPC interaction by creating dialogue trees, and end up creating more personable NPCs using PCG techniques, such as implementing sensors to the NPCs to tell the time and comment on such events, or if the player completes a certain number of quests the NPC becomes more friendly towards the player. Small improvements like these could be difficult to implement due to the necessity of improving the base AI of the NPCs, however if successful such results could provide a game that is self-sustaining and completely replayable.

Finally, an ambitious piece of further work could be expansion of the environment where the NPCs reside, this could include creating an entire village with more interactable NPCs, that have daily routines and make a village feel alive. The environment could also create opportunities to add other skills into the game, such as the player creating their own weapons, armor or furnishings for houses. The benefit of the scope of this project is that there are a number of different directions that a procedurally based first person RPG can take, and the limitations to the genre are practically non-existent, the only true obstacle to any major further changes is the complexity and time it would take to implement.

Appendix A

Showcase Appendix

```
if(Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, range))
{

    if(Input.GetKeyDown(KeyCode.F))
    {
        NPC npc = hit.collider.GetComponent<NPC>();
        reward = GameObject.Find("Weapon").GetComponent<Weapon>();
        if(npc != null)
        {
            if(!onQuest)
            {
                GiveQuest(npc);
                onQuest = true;
            }
            else
            {
                if(npc.title == targetJob)
                {
                    Debug.Log($"Quest Complete, you found the {targetJob}");
                    Debug.Log(reward.GiveWeapon(npc));
                    onQuest = false;
                }
                else
                {
                    Debug.Log($"You are already on a quest, go find the {targetJob}");
                }
            }
        }
    }
}
```

Figure A.1: The interaction code used in conjunction with the questing method

```
public void GiveQuest(NPC npc)
{
    i = Random.Range(0,5);
    targetJob = GameObject.Find("NPCContainer").GetComponent<NPCJobs>().jobTitle[i];
    if(npc.title == targetJob)
    {
        if(i == 4)
        {
            i -= 1;
            targetJob = GameObject.Find("NPCContainer").GetComponent<NPCJobs>().jobTitle[i];
        }
        i += 1;
        targetJob = GameObject.Find("NPCContainer").GetComponent<NPCJobs>().jobTitle[i];
    }
    Debug.Log($"Go and find the {targetJob}");
}
```

Figure A.2: The questing method which finds a NPC to be the target of a quest

```
public class AssignJobs : MonoBehaviour
{
    private GameObject[] npcs;
    private int i;
    private List<int> iValues = new List<int>();
    public string title;
    public Color color;
    public string weapon;
    public GameObject npcContainer;

    private void Start()
    {
        npcContainer = GameObject.Find("NPCContainer");
        npcs = GameObject.FindGameObjectsWithTag("NPC");
        foreach (GameObject npc in npcs)
        {
            bool flag = false;
            while(flag == false)
            {
                i = Random.Range(0,5);
                if(!iValues.Contains(i))
                {
                    iValues.Add(i);
                    flag = true;
                }
            }
                NPC thisNPC = npc.GetComponent<NPC>();
                thisNPC.title = npcContainer.GetComponent<NPCJobs>().jobTitle[i];
                thisNPC.color = npcContainer.GetComponent<NPCJobs>().colours[i];
                thisNPC.weapon = npcContainer.GetComponent<NPCJobs>().weaponPref[i];
                npc.GetComponent<Renderer>().material.color = thisNPC.color;
        }
    }
}
```

Figure A.3: The algorithm to determine an NPC's job at runtime

```
        j = Random.Range(0,101);
        if(j > 90)
        {
            elemental = "Earth";
        }
        if(j > 80 && j <= 90)
        {
            elemental = "Water";
        }
        if(j > 70 && j <= 80)
        {
            elemental = "Fire";
        }
        if(j > 60 && j <= 70)
        {
            elemental = "Air";
        }
        if(j <= 60)
        {
            elemental = "None";
        }
        if(elemental == "None")
        {
            rewardedWeapon = $"You've been given a {rarity} {weaponName}! Attack: {atk} Critical Hit Chance: {critChance} Range: {range}";
        }
        else
        {
            rewardedWeapon = $"You've been given a {rarity} {weaponName} of {elemental}! Attack: {atk} Critical Hit Chance: {critChance} Range: {range}";
        }

        return rewardedWeapon;

    }
}
```

Figure A.4: The code which returns the weapon string from the weapon script

References

[1] 3DeLucas. ‘Medieval tavern pack.’ (), [Online]. Available: https://assetstore.unity.com/packages/3d/props/furniture/medieval-tavern-pack-112546 (p. 23).

[2] M. C. Angelides and H. Agius, ‘Procedural content generation,’ in *Handbook of Digital Games*. 2014, pp. 62–91. DOI: 10.1002/9781118796443.ch2 (p. 7).

[3] R. Baltezarevic, B. Baltezarevic and V. Baltezarevic, ‘The video gaming industry (from play to revenue),’ *International Review*, pp. 71–76, Jan. 2018. DOI: 10.5937/IntRev1804071B (p. 1).

[4] T. Boelter Mizdal and C. T. Pozzer, ‘Procedural content generation of villages and road system on arbitrary terrains,’ in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018, pp. 205–2056. DOI: 10.1109/SBGAMES.2018.00032 (p. 11).

[5] H. Bomström, ‘The application of procedural content generation in video game design,’ Ph.D. dissertation, PhD thesis, 2016 (pp. 6, 9, 10).

[6] N. Brewer, ‘Computerized dungeons and randomly generated worlds: From <italic>rogue to minecraft</italic> [scanning our past],’ *Proceedings of the IEEE*, vol. 105, no. 5, pp. 970–977, 2017. DOI: 10.1109/JPROC.2017.2684358 (pp. 6, 7).

[7] R. L. Cabrera, M. N. Collazo, C. C. Porras and A. J. F. Leiva, ‘Procedural content generation for real-time strategy games,’ *IJIMAI*, vol. 3, no. 2, pp. 40–48, 2015 (p. 16).

[8] Casual2D. ‘Wooden floor materials.’ (), [Online]. Available: https://assetstore.unity.com/packages/2d/textures-materials/wood/wooden-floor-materials-150564 (p. 23).

[9] W. Forsyth, 'Globalized random procedural content for dungeon generation,' *J. Comput. Sci. Coll.*, vol. 32, no. 2, pp. 192–201, Dec. 2016, ISSN: 1937-4771 (p. 16).

[10] W. Games. 'Wildermyth.' (), [Online]. Available: https://www.wildermyth.com (p. 8).

[11] B. Hammer. 'Fantasy wooden gui: Free.' (), [Online]. Available: https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811 (p. 29).

[12] P. E. Hart, N. J. Nilsson and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths,' *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136 (p. 33).

[13] M. Hendrikx, S. Meijer, J. Van Der Velden and A. Iosup, 'Procedural content generation for games: A survey,' *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, Feb. 2013, ISSN: 1551-6857. DOI: 10.1145/2422956.2422957. [Online]. Available: https://doi.org/10.1145/2422956.2422957 (p. 9).

[14] M. Hendrix, T. Bellamy-Wood, S. McKay, V. Bloom and I. Dunwell, 'Implementing adaptive game difficulty balancing in serious games,' *IEEE Transactions on Games*, vol. 11, no. 4, pp. 320–327, 2019. DOI: 10.1109/TG.2018.2791019 (p. 12).

[15] E. A. Matthews and B. A. Malloy, 'Procedural generation of story-driven maps,' in *2011 16th International Conference on Computer Games (CGAMES)*, 2011, pp. 107–112. DOI: 10.1109/CGAMES.2011.6000324 (p. 8).

[16] S. Oliveira and L. Magalhães, 'Adaptive content generation for games,' in *2017 24º Encontro Português de Computação Gráfica e Interação (EPCGI)*, 2017, pp. 1–8. DOI: 10.1109/EPCGI.2017.8124303 (pp. 12, 14, 15, 20, 33).

[17] R. Parekh, 'Staying in the flow using procedural content generation and dynamic difficulty adjustment,' Ph.D. dissertation, WORCESTER POLYTECHNIC INSTITUTE, 2017 (p. 13).

[18] Y. H. Pereira, R. Ueda, L. B. Galhardi and J. D. Brancher, 'Using procedural content generation for storytelling in a serious game called orange

care,' in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019, pp. 192–197. DOI: 10.1109/SBGames.2019.00033 (p. 8).

[19] D. Plans and D. Morelli, 'Experience-driven procedural music generation for games,' *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 3, pp. 192–198, 2012. DOI: 10.1109/TCIAIG.2012.2212899 (pp. 10, 13).

[20] T. J. Rose and A. G. Bakaoukas, 'Algorithms and approaches for procedural terrain generation - a brief review of current techniques,' in *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 2016, pp. 1–2. DOI: 10.1109/VS-GAMES.2016.7590336 (pp. 1, 7, 10, 11).

[21] G. Smith, 'An analog history of procedural content generation.,' in *FDG*, 2015 (p. 6).

[22] G. Smith, 'What do we value in procedural content generation?' In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ser. FDG '17, Hyannis, Massachusetts: Association for Computing Machinery, 2017, ISBN: 9781450353199. DOI: 10.1145/3102071.3110567. [Online]. Available: https://doi.org/10.1145/3102071.3110567 (p. 15).

[23] 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*, ser. PCGames '11, Bordeaux, France: Association for Computing Machinery, 2011, ISBN: 9781450308724. DOI: 10.1145/2000919.2000922. [Online]. Available: https://doi.org/10.1145/2000919.2000922 (p. 7).

[24] 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*, vol. 3, no. 3, pp. 172–186, 2011. DOI: 10.1109/TCIAIG.2011.2148116 (pp. 1, 6, 14).

[25] J. Ulisses, R. Gonçalves, A. Coelho, A. A. Sousa and E. Oliveira, 'Procedural generation of maps and narrative inclusion for video

games,’ in *Proceedings of the 10th Doctoral Symposium in Informatics Engineering - DSIE’15*, 2015, pp. 106–118 (p. 16).