|  |
| --- |
| Evaluating the use of procedural generation to create level designs for an action RPG  Marcus McGinn  BSc (Hons) Computer Games Applications Development 2021/22 |

|  |
| --- |
| School of Design and Informatics  Abertay University |

Table of Contents

[Table of Figures ii](#_Toc92987347)

[Table of Tables iii](#_Toc92987348)

[Acknowledgements iv](#_Toc92987349)

[Abstract v](#_Toc92987350)

[Abbreviations, Symbols and Notation vi](#_Toc92987351)

[Chapter 1 Introduction 1](#_Toc92987352)

[Chapter 2 Literature Review 4](#_Toc92987353)

[Chapter 3 Methodology 12](#_Toc92987354)

[Chapter 4 Results 21](#_Toc92987355)

[Chapter 5 Discussion 27](#_Toc92987356)

[Chapter 6 Conclusion and Future Work 33](#_Toc92987357)

[List of References 36](#_Toc92987358)

[Bibliography 40](#_Toc92987359)

[Appendices 41](#_Toc92987360)

# Table of Figures

Fig 1 genetic algorithm map

Fig 2 A short run of the stochastic, “blind” digger

Fig 3 A short run of the informed

Fig 4 the prototype model

Fig. 5 example of map representation

Fig. 6 Random walk map example

Fig 7 breeding example

Fig 8 genetic map example

Fig 9 Pseudo code for Agent-Based

Fig 10 impossible level

Fig 11 Map and level comparison

Fig 12 overall survey results

Fig 13 average graph

Fig 14 procedural generation graph

Fig 15 correct graph

Fig 16 familiarity pie chart

Fig 17 experience graph

Fig 18 Genetic algorithm problematic solutions

Table of Tables

Table 1 Fundamental Components

# Acknowledgements

Thank you to my supervisor Gareth Robinson for his continued support with this project.

Thank you to my sister and proof reader Dominique, who has been a real support throughout my time at University.

Thanks to my parents for giving me the opportunity to reach my goals in life.

# Abstract

Time and resource management is an extremely important aspect to consider when developing games therefore, any methods that could create more content in less time should be investigated. Level design is an area in games development which has the potential to be time consuming, new techniques are pertinent to consider in order to make the process more time efficient. This paper explores Procedural Content generation as a method for generating lots of content in little time and may be useful in speeding up development time.

The overall aim was to develop a prototype procedural generation system for a game to compare and analyse the effectiveness of procedural generation (PG) techniques against handmade content.

The project developed in several stages following a prototyping methodology. First a procedural level generation system was developed and revised until it could generate acceptable levels. Then, a simple game was to be developed for the generated levels to be used in. The last addition was to add several handmade levels to the game so they could be compared to the procedurally generated levels.

A group of volunteers were assembled to play through the game without being told which levels were procedurally generated and which levels were not. They were then asked to fill in a survey where they rated how much the liked each level and which levels they thought were procedurally generated.

The results of this survey indicated that the procedurally generated levels were rated just as well as the handmade ones and that players were only able to correctly guess if a level was procedurally generated fifty five percent of the time. These results show that Procedural Generation can be just as effective as designing levels by hand and should be considered in future as a substitute for level design in order to save development time.

# Abbreviations, Symbols and Notation

PCG – Procedural Content Generation

PGC- Procedurally generated Content

RPG – Role Playing Game

# Chapter 1 Introduction

**1.1 Background and context**

In games development there are few things more important than time management. The bigger the game the more important this becomes. “Deadlines are often stringent and inflexible – the bigger the game, the more intense the deadline” (Morrow, E, 2021;1). To address this, it is important to consider all the possible tools available which could increase effective time management and efficiency for generating content. The tool this project addresses is the use of Procedurally Generated Content (PGC) and how it compares to content made using manual methods.

Procedural Generation simply put, is the creation of new data by a computer (Van Brummelen and and Chen, 2021). Procedural generation has had several uses in games over the past few years, with some important examples being; “The Binding of Isaac”(2011) and “No Mans Sky”(2016). Using No Mans Sky (2016) as an example, PGC was used to create a massive explorable world. A pattern in the use of Procedural Generation is that it is more commonly used in large open world games like No Mans Sky (2016) or 2D dungeon crawlers, like The Binding of Isaac(2011). A genre that has seen less use of Procedural Generation is Action Role Playing Games (RPG).

The initial inspiration for this project was the action RPG “Bloodborne” (2015) which has levels called “Chalice Dungeons”. These “Chalice Dungeons” are dungeon crawler levels where the player must explore and fight enemies to reach the end and they were originally thought to be procedurally generated. However, BloodBorne-wiki (2021;1) states that “while the dungeons were presented as procedurally generated, there’s in fact no procedural generation and almost no randomization going on.” The levels are actually just handmade maps stored in the games files which makes it appear procedurally generated. This surprise is where the idea for this project originated, since a clear comparison could be made between the newly created PGC level system and the handmade levels. In essence, the project will try to answer the question "How effective are Procedural Generation Techniques for dungeon level design and are they more effective than manual level design techniques?" To answer this question research was done on procedural generation, procedural generation techniques were evaluated and chosen and an application was developed with a survey in order to evaluate the final result.

**1.2 Aims and Objectives**

The key objectives and aims of this project are as follows:

Aim: To develop a prototype system for a game to compare and analyse the effectiveness of procedural generation techniques against handmade content.

Objectives:

-Research level design theory, case studies and procedural techniques

-Select, with reasoning, a suitable procedural generation technique

-To design and prototype a procedural generation system for level design using relevant techniques and technology

-Build prototype levels using traditional level design techniques

-To obtain performance data of use of prototypes from testers.

-To analyse and evaluate the data to compare both types of level design.

-To make recommendations and suggest future work.

**1.3 What is procedural generation?**

The main idea behind procedural generation is the creation of new data through the use of an algorithm (Van Brummelen and Chen, 2021). In the development of games there are times where the amount of content necessary to complete a game isn’t feasible since it would take far too much time and application to be handmade as the scope of the game is significantly larger than the development time would allow. Times such as this are when PGC is typically used (Shivang, 2021).

When procedural generation is used there are two key aspects to consider; the randomness and the rules which control that randomness (Van der Linden *et al*, 2014). The rules make sure the content meets the requirements for its use and the randomness ensures that each piece of content is unique. When deciding whether or not to use handmade content there is a trade-off which must be considered because when using PGC the developer sacrifices control due to the randomness in exchange for creating much more content (Van der Linden *et al,* 2014). The problem with sacrificing control is that the developer gives up their ability to structure challenges that will engage and entertain the user

**1.4 What is a dungeon**

In computer games, a dungeon is typically a labyrinth, where the player must avoid obstacles, accomplish goals, and exist at another point (Shaker, 2018.) This definition of a dungeon is believed to have originated with the table-top roleplaying game Dungeons and Dragons and has become a staple of many games which have defined the RPG genre including; Legend of Zelda and Elder Scrolls V Skyrim (Shaker, 2018).

# Chapter 2 Literature Review

**2.1 Introduction**

In the introduction both PCG and dungeons were discussed. I will now go on to discuss how both of these have the potential to be combined to propagate Dungeon Design. Initially, Dungeon Design was researched so that it could be broken down and applied in a form suitable for procedural generation.

“Patterns, Dungeons and Generators” was a useful paper which deconstructed dungeon design by exploring dungeons from various different games, compiling a list of the most important elements which can be seen below in the table. (Dahlskog, Björk and Togelius, n.d.) Elements of this study were used to inform the procedural generation and handmade development of dungeon levels, this will be discussed in the methodology. While not all the information present in the table below was useful or carried on into the project, several aspects were. These key components of a dungeon that were used in this project are; the idea of using tiles to represent the dungeon and separating the tiles into different categories, which included separating the walkable space and walls of the dungeon. The bottom four components on the list were not applicable to this as the agent is not the same type of agent that is discussed later in the paper and the other three are functions that were specific for that project and are not part of the dungeon themselves.

|  |  |
| --- | --- |
| **Component** | **Definition** |
| Tile | The basic unit of space in a dungeon. Individual Tiles have Boolean attributes associated with them: Passable and Seethru |
| Level | A rectangular space of Tile. |
| Wall Tile | The base classification of Tiles in a Level. Tiles belonging to this category are not Passable and not Seethru. The Base Content of a Tile in Rogue is “Rock”. |
| Ground Tile | A classification for Tiles that are Passable and Seethru. |
| Item | A game object that can be in a Tile but which can also be picked up, carried, and dropped in other Tiles. |
| Agent | A game object that can perform actions, e.g. moving and attacking, and is located on a specific Tile. The player’s avatar is an Agent as are monsters. They typically hinder other from entering the Tile they are in, i.e. they temporary remove the attribute Passable from the Tile they are in. |
| Line-ofSight(A,B) | A Boolean function returning if all Tiles on a straight line between point A and B in a Level are Seethru. |
| Traversable(A,B) | A Boolean function returning if a player can move between point A and B in a Level. A Route is a traversal solution and the length of different Routes may be needed for some design patterns. |
| Sequenced(A,B) | A Boolean function returning if point A must be visited before point B in a Level. |

Table 1 Fundamental Components ((Dahlskog, Björk and Togelius, n.d.)

The key principles of applying procedural generation to dungeon levels are as follows:

*Procedural generation of dungeons refers to the generation of the topology, geometry and gameplay-related objects of this type of level. A typical dungeon generation method consists of three elements: 1. A representational model: an abstract, simplified representation of a dungeon, providing a simple overview of the final dungeon structure. 2. A method for constructing that representational model. 3. A method for creating the actual geometry of a dungeon from its representational model.*

(Shaker, 2018; 31)

In relation to this project these concepts were used to plan out how the end result would be implemented. The “representational model” in the case of this project, is the 2D map image used to represent design of the dungeon generated by the procedural generation. The “method for constructing that representational model” is the specific type of procedural generation that was used to create the maps. In the case of this finished project the “method” is the Agent Based procedural generation that will be discussed later in the literature review. Finally, the method of creating the geometry was by creating corridors and rooms in the unreal engine which can be placed together to match the map layout. Each of these aspects is covered in more detail in the methodology.

**2.2 Considered Techniques-**

During the initial research stage of the development process there were several procedural generation techniques that were considered. Generative Grammars were a method originally used to study linguistics and the structures of phrases. “The theory of generative grammar postulates a mechanical procedure by means of which preferred descriptions are chosen from among several alternatives” (Halle, 2015; 55). In short, linguists working in this field aim to develop rules and laws that dictate languages. The key aspects to using Generative Grammars that will be discussed are; Graph Grammars and Generative Grammars which have been used in the past to create similar dungeon levels (Van der Linden et al, 2014). Generative Grammars were considered but ultimately not used as they require a significant amount of developer input that other techniques don’t. (Van der Linden et al, 2014).

Constraint based PGC was also considered. “The Constraint Based PGC approach involves the declarative specification of properties of and constraints on the content that will be created” (Smith, 2021; 2). What this means is that in a constraints based PGC system, a set of rules have to be created for the system to follow. Since the system has these rules, it means that all of the data generated is usable for the game, provided the rules were set up correctly. However, to accomplish this, the rules must be carefully set up and constructed which can take more time than other methods to accomplish a similar result so ultimately it wasn’t chosen.

**2.3 Genetic Algorithms**

Genetic Algorithms are another type of Procedural Generation algorithm which searches for the most optimum solution to a problem. To accomplish this, the algorithm mimics the process of Darwinian evolution and how species adapt over time (Shaker, 2018). The first important aspect to consider is genetic representation, this is how the algorithm represents the data from the solutions it constructs, using strings so that the data can be manipulated in a similar way to cells in the process of evolution (Van der Linden *et al,* 2014). The other essential part of the algorithm to be taken into account is the fitness function. This is a function that takes in the string data and evaluates the quality of it. The Genetic Algorithm uses both of these individual parts in a cycle, constantly evaluating the data and taking the best parts of each solution (Van der Linden et al, 2014). These superior parts are then stitched together and used to create new and better solutions.

The final piece of the process to mention is mutation. Mutation is a process where there is a very small chance of randomly changing one of the characters in the string (Van der Linden *et al,* 2014). When a cell is mutated, it changes at random, this random change could improve the quality of the map, lower the quality of the maps or have no effect. The process works in such a way that if it was given an infinite amount of time, the perfect solution would potentially be found eventually due to combining better results and mutation creating improvements that breeding wouldn’t allow. Genetic Algorithms have been used to produce maps for dungeon levels in the past below is an example.

Qr code

Description automatically generated

Fig 1 genetic algorithm map (Van der Linden *et al,* 2014)

A map as seen above could be used to automatically generate a level using premade assets. For example, if the black parts represented where to use a wall asset and the white parts where to use a floor asset instead, some challenges regarding the practicalities of this may arise, for example; Wall + floor tile might work for a simple 2D game, but in reality you might need different sprites for corners and single blocks. In addition, representing it in 3D would get even more complex. There were also more problems that arose from the use of this approach that will be discussed in the methodology and discussion chapters.

**2.4 Lee Algorithm**

For the genetic algorithm prototype the original idea for the fitness function was to check the length of the levels. To accomplish this a pathfinder called a Lee algorithm was chosen to track the distance between the start and end of the level. The levels with a long path would be bred together to create longer and longer levels, this is covered in more detail in the methodology. The Lee algorithm was chosen over other pathfinders because it always finds the shortest path possible. (Lee's Algorithm Explained with Examples, 2022) The algorithm does have the downside of requiring a lot of memory for large and complex mazes but considering the small size and simplicity of the maps generated by the project it was decided this downside didn’t matter. (Lee's Algorithm Explained with Examples, 2022) The algorithm marks the starting point and checks the surrounding points in the maze, it then numbers the walkable tiles and checks the tiles surrounding those. This process continues in a loop, expanding through the map until the end point is found.

**2,5 Agent-Based Dungeon Growing**

In an agent-based approach a single agent is used in order to plot the layout of a dungeon by digging tunnels and placing rooms when appropriate (Shaker, 2018). An Agent-Based system is “a system is modelled as a collection of autonomous decision-making entities called agents. Each agent individually assesses its situation and makes decisions on the basis of a set of rules.” (Bonabeau, 2002; 1) These Rules will then define the sorts of levels generated from the agents.

General advantages of using an Agent Based solution include; it can represent problems naturally as it’s supposed to mimic human decision making and they are very adaptable which was very useful over the course of development (Bonabeau, 2002). One of the disadvantages Agent-Based modelling has is that it can be very computationally expensive when there are many agents, this will not be an issue in this project as only one agent is in use (Darrin, 2022). When compared to other PCG techniques the result created from an agent-based solution can be chaotic and random which can be both a benefit and a problem. The benefit being that it makes the layout generated more natural and believable while the problem that can arise being that the level may not be fit for purpose.

Taking these two aspects into consideration is important when designing an agent-based as the AI must have rules in place to control the randomness. “The appearance of the dungeon largely depends on the behaviour of the agent: an agent with a high degree of stochasticity will result in very chaotic dungeons while an agent with some “look-ahead” may avoid intersecting corridors or rooms” (Shaker, 2018). An example of a PCG system in action can be seen below.

Fig 2 A short run of the stochastic, “blind” digger (Shaker, 2018) Graphical user interface, text, application

Description automatically generated

This is a very simple example which results in a level were different rooms overlap with each other. This could be useful for procedurally generating something natural like a cave system but would not be effective for something more structured like a dungeon level. However, these problems can be addressed by making the AI that controls the agent more complex and adding more rules. The example below shows an example of a more complex Agent-Based solution in action which creatures a more structured and dungeon like layout.

Graphical user interface, text

Description automatically generated

Fig 3 A short run of the informed, “look ahead” digger (Shaker, 2018)

An Agent-Based method was chosen for the project because when compared to a genetic algorithm it can generate more than one result, unlike generative grammars it doesn’t require a designers influence to generate levels and its randomness can create more interesting results than most other PCG techniques. Agent-Based does have its own drawbacks but these will be discussed in the methodology.

**2.6 Unreal Engine 4**

To develop the game in order to test the levels the Unreal Engine 4 was chosen. “Unreal Engine (currently released as Unreal Engine 4) is a popular and widely-used game engine developed by Epic Games.” (Denham, n.d.) There are several reasons to use the unreal engine however, the main reason it was chosen for this project was the blueprint tool which allows developers to create a working prototype fast (NJIKI, 2019). As the game was intended to be very simple and easy to develop for the purpose of being used as a means to test the generated levels, the speed at which it could be developed was an important factor making the unreal engine very useful.

# Chapter 3 Methodology

**3.1 Paradigm**

A paradigm is essentially a thought process for how to conduct research. It was originally described by Kuhn as “a way of looking at or researching phenomena” (Kuhn, 1962: 23). There are many different ways to think when conducting research and therefore, many different types of paradigm to consider. The type of paradigm that was chosen for this research was a prototyping methodology. Prototyping is a methodology that was devised during the 1980s and was born out of the necessity for a design methodology that would help account for uncertainty (Daniel A. Fern Scott E. Donaldson). In the case of this project the uncertainty being managed was whether or not Procedural Generation Techniques for dungeon level design are more or less effective than Manual Level Design Techniques.

The prototyping model is made up of four main steps which are; identifying the basic requirements, developing a working prototype, implementing and using the prototype and revising and enhancing the prototype. (Justus, 1982) The diagram shows this process below.

Diagram

Description automatically generated

Fig 4 the prototype model (Justus, 1982)

The first step was identifying the basic requirements of the system. The specifics of this stage depend upon the project but it typically involves defining what the problem is and what would be required for the project to meet expectations. Then a very simple prototype is developed, this needs to be developed quickly in order for the methodology to work properly as it should only take a day or two at most (Justus, 1982). The prototype does not need to be perfect as the point, it can be improved upon later. Next the prototype is then implemented and tested before moving on to the evaluation stage. Here the prototype is evaluated and improvements are planned. The project then loops between the implementation and evaluation stages until a satisfactory solution is created.

In the case of this project, the identifying stage was primarily carried out during my research where it was identified what a dungeon is, what are the attributes required for a dungeon, what PCG techniques are available and how they can be used for dungeon generation. The definition of a dungeon used for this project was previously highlighted in the introduction as “a labyrinth where the player must avoid obstacles, accomplish goals, and exist at another point” (Shaker, 2018; 31).

**3.2 Random walk**

The next stage was to develop a very basic working prototype. To accomplish this a random walk algorithm was chosen as a starting point for the project. The random walk is a simple agent-based method which could be built on later in development. The random walk developed was loosely based on a tutorial by Ahmad Absoleb. To start the random walk, a twenty by twenty 2D array was created. This 2D array was used to represent the map generated with there being four types of possible value coordinates on the map. A point on the map marked with zero would represent an empty space, a point marked with one would represent a walkable space like a corridor, a point marked with three would represent where the player would start on the map and a point marked with four would represent the end of the dungeon that the player must reach. An example of how this looked can be seen below.

A computer screen capture

Description automatically generated with medium confidence

Fig. 5 example of map representation

To generate this map there were several stages. First the application chose a random coordinate on the map and marked it as the starting point and a random direction was chosen for it to travel in. The four directions were stored in a four by four 2D array which was used to move the agent up, down left and right. For the application to work a maximum size of tunnels and maximum number of tunnels had to be set. For the purposes of the prototype these were originally set to ten. The agent would then at random choose a tunnel size within the maximum and choose a random direction that wasn’t their current direction or previous direction. This ensured the agent didn’t get stuck. The agent would then move along their new direction and changed each value into a tunnel until the agent reached the edge of the map or had reached the maximum size they had chosen. The agent then simply repeated this process until it had reached the maximum number of tunnels. At this point the agent marked where they had gotten to as the end point.

The programme as described by the tutorial was not perfect and required amendments to work properly. The programme as originally described, would get trapped when it hit corners of the map, to prevent this extra rules had to be added. When the agent reached the corner of the map it would choose the only direction possible to advance rather than getting stuck by choosing the wrong one. In order to visualize the results of the prototype the application looped to generate twenty maps and output them to PPM files. The application took the 2D arrays and coloured each pixel of a twenty by twenty ppm based in the values in the 2D array. In the image created empty space was marked by red, the corridors were marked with green, the starting point was marked with blue and the end point was marked with purple. Below is an example of how these maps turned out.

Qr code

Description automatically generated

Fig. 6 Random walk map example

After finishing the prototype, the project moved to the evaluation stage. The key things that were decided upon during evaluation was that the start and end points were not always the most appropriate as they didn’t always effectively use the space available and would leave large areas of red. In addition, due to being exclusively made of overlapping tunnels, it didn’t really reflect the layout of a dungeon very well.

**3.3 Genetic algorithm**

To address some of the above issues genetic algorithms were chosen to be implemented. The goal of this new prototype was to address the issue of the start and end points as well as the issue of wasted space. The intention was to address the other problems after solving these ones. The plan was to have the application use the random walk algorithm to generate twenty maps. These maps would then be used as the starting data for the genetic algorithm. There are three key aspects required for a genetic algorithm to work which are; the breeding process, the fitness function, and the mutation process.

The fitness function is used to control what sort of result is to be created by the algorithm. In the case of this application the maps generated by the random walk aimed to be longer and make more effective use of space. To accomplish this goal, a pathfinder called a Lee algorithm was used as explained in the literature review. The idea was that the fitness function would return how long the shortest path from the start of the level to the end of the level was. In the event that the algorithm got stuck because the level was impossible it would return a value of zero.

The breeding process works as follows; it passes the maps into the fitness function and orders the maps from highest score to lowest score. The two maps with the highest score are then combined, this is done by taking the second-best map and adding random map points from the best map to it. An example of this in practice can be seen below.

parents



Child with random aspects of both



Fig 7 breeding example

This process would then loop until it had the longest possible path as all inferior maps would be outbred by the superior solutions. However, this would only get the best result using pieces from the other maps, in order to generate totally new maps the mutation stage is needed. The mutation stage only happens a fraction of the time. At the start of the function a variable called mutation chance is used to count how many times the genetic algorithm has looped. At the end of each loop a number between one and five is chosen at random, if the number is less than the mutation chance then the mutation process starts. In the mutation process a map is chosen at random, then five points on the map are assigned a new random value. To make sure there are only ever one start and end point each time a new start/end is placed the old start/end is given a value of zero making it an empty space.

The genetic algorithm was poorly suited for the problem and took considerably longer than planned to get working. The reasons for this will be covered in the discussion. Below can be seen an example of the sorts of maps generated.



Fig 8 genetic map example

After evaluation, it was decided that they were not structed well enough like a dungeon level as the levels generated by the genetic algorithm were very chaotic and winding making them more like a maze than a dungeon. Indeed, in following a Prototyping methodology based on accounting for uncertainty, the project reached an uncertain point which required a revaluation. After research and discussion with my mentor it was decided that the genetic algorithm would be shelved and instead, time would be taken to build on the random walk to create an Agent Based solution in order find out how effective Procedural Generation Techniques for dungeon level design are in comparison to Manual Level Design Techniques.

**3.4 Agent Based Dungeon Generator**

To address the problems mentioned above an agent-based solution was chosen as it would be a natural extension of the random walk generator and would create levels that would be structured much more like a dungeon level. To create the agent-based dungeon generation one of the examples described in chapter three of the book “Procedural Content Generation”. Below is the example described in the book.

Text

Description automatically generated

Fig 9 Pseudo code for Agent-Based (Shaker, 2018)

The version created works similar to the above with a few alterations. In the application, the agent picks a random starting point and places a starting room. The agent will then choose a new direction at random and place a corridor of a random size between one and seven before choosing a new direction. Each time the agent chooses a new direction it checks first if there is enough space for a room, if there isn’t it will check for a corridor instead and if that doesn’t work the agent will stop and output the map because it got stuck. This process loops until it has used up to the maximum amount of corridors or it gets stuck and cannot continue. When the agent stops it places an end point at its last position.

This method of generation was much more effective at creating something more akin to the structure of a dungeon as the combination of rooms and corridors looked much more structured than those created by the genetic algorithm. It was also much faster than the genetic algorithm, taking only a few seconds, however, it didn’t generate a usable result every time because “the uncontrollability of the algorithm may result in organic, realistic caves (simulating human miners trying to tunnel their way towards a gold vein) and reduce the dungeon’s predictability to a player. In addition, it may also result in maps that are unplayable or unentertaining” (Shaker, 2018; 41) If these unplayable maps were to be used, they would very likely be rated poorly as many had no challenges, were impossible to traverse or didn’t even have an end point. Examples of this are shown below.

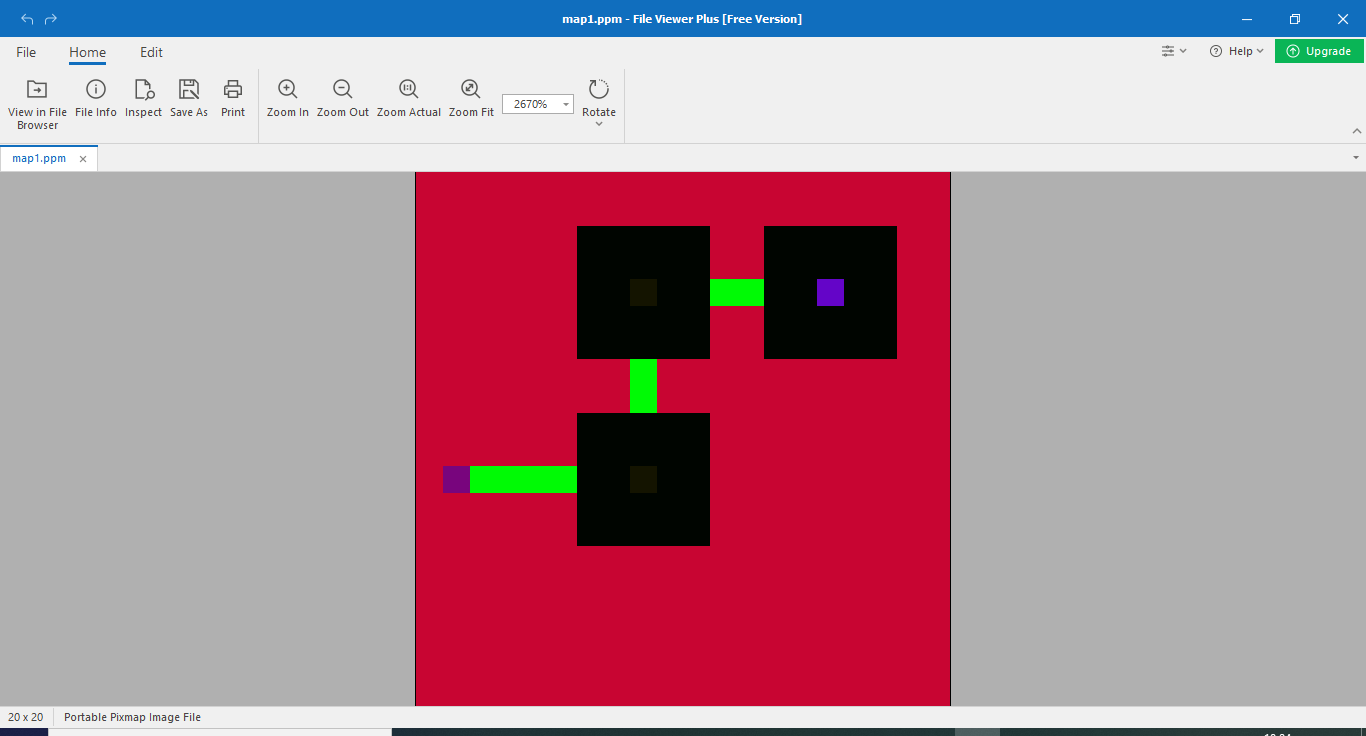


Fig 10 impossible level

To fix this problem a solution was created by using some of the ideas from the genetic algorithm. To ensure that the algorithm only generated usable levels, the fitness function was reworked to be used to check if its possible to get from the start of the map to the end. The algorithm loops until it has generated five usable levels with at least three rooms. To populate the rooms in the dungeon, a method similar to the game “Spelunky”(2008) was used. In Spelunky (2008) there are pre-set layouts of rooms which are chosen at random. Six room types were made, three different layouts with treasure and three different layouts with enemies.

**3.5 Building levels in unreal**

In order to test the levels generated, a simple game was created using the Unreal Engine Four. To build the levels, sections of corridors and rooms were created which could seamlessly fit together. These rooms and corridors were then placed to follow the map layouts. These level were created from maps designed by for the purposes of testing.

Diagram, engineering drawing

Description automatically generated

Fig 11 Map and level comparison

**3.6 Setting up the survey**

Primary research was undertaken to collect both quantitative and qualitive data through issuing a survey to those who volunteered to play the game. The survey was intended to be answered by 10 volunteers however, with greater time more volunteers may have been advantageous. Quantitative data, is usually numerical data “expressing a certain quantity, amount or range” (OECD, 2006; 1). In the context of this project, quantitative data was gathered through closed survey questions, for example when rating a level on a scale from 1-5. Although, predominantly quantitative, qualitative data was also used to enhance the findings. “Qualitative research involves collecting and analysing non-numerical data (e.g., text, video, or audio) to understand concepts, opinions, or experiences” (Bhandari, 2020; 1) An example of this is through open ended questions about players opinions on the different levels. Respondents could answer in as many words as they saw fit. Overall, the key questions considered; rating quality of the levels, which levels were procedurally generated or manually created and what sort of changes could be made to make the experience more positive for the player. The purpose of these questions was to verify if the volunteers could tell procedurally generated levels from hand made levels, if the had a preference for one over the other and what sort changes could be used for future work in order to answer the research question.

# 

# Chapter 4 Results

# 4.1 survey results

The ten volunteers were asked to play the levels and rate them out of five as they go. Below are all the ratings by each player. The end column shows an overall average rating for each level with 1 being a low rating and 5 being the highest rating. This information will be used to determine if the players had any sort of preference between procedurally generated and handmade levels.

Fig 12 overall survey results

For better clarity the averages have been put into their own table. The levels that were handmade were marked with orange and the levels marked with green were procedurally generated.

Key Orange Bar- Manally Generated

Green Bar – Procedurally Generated

Fig 13 average graph

From the data, two levels scored 3.3 out of 5 on average, which was the highest rating. This would suggest that players found both level 3 and level 6 equally enjoyable. It is important to note that level 3 was a manually generated level, whilst level 6 was a procedurally generated level. This would suggest that there wasn’t much between the two in terms of enjoyment for the player.

When looking at the players scores individually however, it should be noted that level six was rated a 2 by one player and a 5 by another, showing that there are inconsistencies in the ratings for the same level. This would suggest that personal preference may play a role in a players rating of a level and this can be a difficult concept to judge for the creator as to how well the game is going to be received by the audience. Level 3 on the other hand was more reliable with most scoring the level either a 3 or a 4.

It is also important to note that the lowest scoring level in terms of enjoyment was level 5 with a score of 2.6. Level 5 is a manually generated level, this would suggest the possibility of manually generated levels being less enjoyable however, this is not significant enough, particularly since the second lowest scoring level was level 7 which is a procedurally generated level. The data therefore suggests that in terms of enjoyment of a level, there is very little difference between manually generated and procedurally generated levels.

In answering the research question, it is important to consider whether players could identify if the levels they played were handmade or procedurally generated. The graph below shows the number of players who guessed correct and incorrect for each level.

Fig 14 procedural generation graph

Level 10 is significant in that 9 out of the 10 volunteers guessed correctly that the level was procedurally generated. This would suggest that the players were able to identify if the level was procedurally generated easily, however, this was not consistent across the ten levels. For example, level 1 scored the highest incorrect value and was a manually generated level. Players therefore assumed that this level was procedurally generated even though it was not. Overall, the results were inconsistent highlighting that the players could not easily identify whether or not a level was manually or procedurally generated.

When calculating a percentage of incorrect answers to correct answers, the result was that volunteers were right fifty five percent of the time as can be seen in the pie chart below.

Fig 15 correct graph

It is also important to consider the volunteer’s familiarity with procedural generation. The graph below indicates the eighty percent where aware of procedural generation and only twenty percent were not.

Fig 16 familiarity pie chart

This may have impacted the players ability to recognise procedural generation from manual generation and may account for the fact that fifty five percent of players were able to correctly identify manual from procedural generation.

The graph below represents the volunteers’ experience when playing games. This information was asked of them as it may indicate if their experience with games may have given them a better chance at detecting if a level of procedurally generated or not. As indicated in the graph most volunteers were very experienced with games and none of them considered themselves novices. This is likely due to the demographic of those asked to participate as described above.

Fig 17 experience graph

The volunteers were then asked three questions about the game after play which included:

In as many words as you wish please describe what you liked about the levels you played.

In as many words as you wish please describe what you didn’t like about the levels played.

In as many words as you wish please describe what you would add to the levels to make

them more interesting.

The most consistent aspect which the players liked about the levels was the varied layouts as sixty percent of volunteers commented on it. For complaints about the levels the most common complaint was the levels not feeling very organised however, despite being the most common complaint it was only written by twenty percent of volunteers. For improvements, the most common suggestion was that the game could be improved by adding more room types, obstacles and layouts as sixty percent of volunteers suggested it. This improvement makes sense as increased variety could always be an improvement for the level generation no matter how many objects it has to work with.

# Chapter 5 Discussion

# 5.1 Survey results

When it comes to which levels were preferred by the players, the data indicates that they enjoyed both procedurally generated and manually generated levels as they were rated similarly. Indeed, the highest rated levels had the same score of 3.3 and rather interestingly one was manually generated and one was procedurally generated. Therefore, it can be concluded that the players enjoyed both types of levels, this is positive since procedurally generated levels are efficient for the programmer to produce larger amounts of content. From the findings, something to consider is players personal preferences for a level. Indeed, even the highest rated levels showed some degree of variation in the scores.

In answering the research question it is pertinent to consider if the players were able to identify procedurally generated levels from manually generated levels. From the results it can be concluded that players guessed correctly 55% of the time. This suggests that players marginally identified the level correctly however, it was not a significant enough number to determine that the players could tell the difference. It is important to consider however, that when there are only two possible answers a volunteer would likely be right fifty percent of the time if they chose at random. Because there are only two answers, the odds are 1 out of 2 that the player would guess correctly. Therefore, while the pattern indicates that players were right 55% of the time, an answer of around 50% would be expected following the rules of probability. To get a more conclusive answer on this pattern it would possibly be necessary to do more testing with a significantly larger number of volunteers. In addition, it may also be advantageous to test more players who were less familiar with procedural generation in order to gain more reliable results.

While the data was based on a players ability to detect what levels were procedurally generated, it is important to consider the experience of the volunteers. All the volunteers were experienced gamers as they all rated themselves at least a three out of five on their experience with games. Also important to note was that eighty percent of the volunteers claimed to have some knowledge on procedural generation. It is plausible to suggest that this combination of knowledge made the group of volunteers significantly more likely to be able to notice the repeated patterns that could give away the use of procedural generation. Taking this into account, it is possible that a larger survey with more variances in experience would indicate that the volunteers would be less likely to detect procedurally generated levels. The reason for the volunteers being more experienced with games is likely due to the demographic of the volunteers. All the volunteers were either Abertay students from one of the games’ course or were simply people with an interest in games, it is therefore only natural that they wouldn’t consider themselves novices at games and that they would be aware of techniques used in the development of games like procedural generation.

When it comes to the answers to the other questions asked in the survey the aspects players like were closely connected. On one hand the main thing volunteers like about the levels was the variety of the levels and their structure while on the other hand the volunteers key complaint was a sense that the levels were very organised or thought out. Both of these points were to be expected as they are inherent to the agent-based solution. The randomisation used in this method creates random and varied levels that feel natural while at the same time that randomness sacralises the developers control to construct the challenges presented to the player. The improvement suggested by most of the volunteers was adding more possibilities to the levels like different types of rooms and more obstacles. This suggestion is only natural a adding more possibilities to the levels could be an improvement no matter how many options there are.

The survey was certainly a useful tool in answering the question- How effective are Procedural Generation Techniques for dungeon level design and are they more effective than manual level design techniques? The key issue with the survey is the small survey size, if further research is done on the topic then it would likely give more useful results to find a larger number of volunteers from more varied backgrounds.

**5.2 Discussing Genetic Algorithms**

From undertaking this project, the limitations of genetic algorithms were explored. Indeed, while the initial prototype for the genetic algorithm was finished it was ultimately scrapped for various reasons. The first reason was that the algorithms speed at generating content was lengthy, after being left for several days the genetic algorithm still hadn’t generated a final result. The algorithm was working as it generated maps with better and better fitness score, however, its main downfall was that it wasn’t producing a final outcome required for the next stage in the project. The second reason that genetic algorithms were placed on the back burner was because the design of the genetic algorithm was flawed as it was supposed to stop when all the members of the population matched. In the system created, there likely wasn’t only one solution with the maximum fitness value. Below is an example of two different results with a high fitness value. This means that the algorithm isn’t able to create one perfect solution for the problem and therefore, never stops trying to.

Qr code

Description automatically generatedQr code

Description automatically generated

Fig 18 Genetic algorithm problematic solutions

The third problem with using the genetic algorithm was that there was no way to expand on the prototype through genetic algorithms. The prototype was intended to be expanded upon by adding to the fitness function so that rather than just generating longer levels it would instead generate more fun and engaging levels. This was found to not be possible as it required quantifying what makes a level fun and engaging for the fitness function. The final problem found was that the level generation was too simple for the genetic algorithm to improve it adequately and it didn’t really resemble a dungeon at all in terms of structure, to improve beyond what was generated by that prototype meant adding more components like rooms to the level which would require essentially starting from scratch.

**5.3 Discussing the agent-based solution**

The Agent-Based solution was found to be much more suitable for generating dungeon levels than the Genetic Algorithm as it had none of the problems caused above. Indeed, it only took a few seconds to generate the dungeons required, the Agent-Based solution doesn’t have the issue with not being able to stop, the agent-based solution doesn’t require quantifying what a “fun” level is as it just generates a random one, and finally, when using the agent-based solution the process of adding more elements does not require as much change. Thus, making it much more efficient, in addition to generating results that were much closer to the traditional layout of a dungeon as it was much more organised looking and used a combination of rooms and corridors.

**5.4 Comparison with previous studies**

The paper Procedural Generation of Dungeons Roland van der Linden, Ricardo Lopes, and Rafael Bidarra brought up several interesting points which agree with much of what was found through this project. It was identified that PCG techniques were very useful at generating dungeon type levels since when using an Agent-Based solution the end result was made up of rooms and corridors which are typical of a dungeon. Roland van der Linden, Ricardo Lopes, and Rafael Bidarra would likely agree with this sentiment as they stated “Dungeon levels for adventure games and RPGs are probably among the few types of game worlds that have been generated very successfully in the past by applying PCG methods.” (Van der Linden *et al,* 2014, page 11) This suggests that the techniques chosen to generate the dungeon levels were advantageous.

In the same paper, it was also stated “These methods are fast enough, in the sense that their procedural generation is faster than creating the dungeon content manually, thus leading to spare time and money in the game development process. (Van der Linden *et al,* 2014; 11) This also identifies one of the benefits found throughout the project. The difference in speed relative to handmade level designs is likely the most valuable advantage that procedural generation has as. It allows for time and money to be spent developing other parts of the game rather than spending time and money on level design.

The paper also agreed with my findings on the quality and variety of levels created from procedural generation techniques as the most common thing volunteers liked about the levels created was their variety in layout and content. A survey was done by comparing various different PCG techniques to see any patterns in their development and capabilities, the points raised indicate that the processes used in this project deviate from the norm but in ways may be to the projects benefit. (Breno M. F. Viana and Selan R. dos Santos 2019)

The patterns found in the various approaches to PCG levels were as follows:

– Most dungeon level generators favour solutions that are game genre agnostic;

– The search-based approach is predominant;

– Few works presented combination of approaches;

– It may be useful to separate the generation process in steps, even for the search-based algorithms; – It might be helpful to differentiate rooms during the generation process, so that the method might assign different purpose and/or content to rooms.”

(Breno M. F. Viana and Selan R. dos Santos 2019: 37)

The point about most generators being “genre agnostic” means it is a system that generates a level without taking into account the type of game being made. This is interesting for two reasons, the first being that in this project, the entire point was to develop a level which took into account what was necessary to produce a dungeon for use in an action RPG similar to the game Bloodborne (2015). The second reason that is interesting is that the survey goes on to suggest that “it might be important to define the game genre a priori because it, in turn, might narrow down the contents and constraints involved in the level generation process.” (Breno M. F. Viana and Selan R. dos Santos 2019; 37) This implies that the approach taken to the development of this system was ahead of the standard as it took the games genre into account making it easier to decide what was necessary.

The second point about search-based approaches being predominant is interesting to consider when compared this projects development process as initially a genetic algorithm was considered for the solution which is a search-based algorithm. However, this was found to not be very effective, and an agent-based system was chosen instead which is a type of constructive method for level generation. This decision was made for many reasons discussed earlier in this section.

The third point is interesting because while it would be a stretch to call it a hybrid approach, the programme reworked some aspects of the genetic algorithm, mainly the fitness function in order to make sure that the levels generated were usable. The fourth point about making the generation process several steps was actually considered for the next step for the project but it had to be cut for time. If this project were to be expanded upon, this would have been the approach going forward. The final point about marking rooms as separate also indicates my project was successfully designed as in my project rooms were marked as separate during generation so they could then be given purpose later.

# Chapter 6 Conclusion and Future Work

**6.1 Answering the question**

The purpose of this study was to answer the question "How effective are Procedural Generation Techniques for dungeon level design and are they more effective than manual level design techniques?.” This study has been somewhat successful in answering that question. Through the development process it has been catalogued how various procedural generation techniques were considered for use, prototyped, evaluated and developed to create a suitable system for testing in order to answer the question. While the survey could have been improved in terms of how many volunteers were found and how much data was gathered, overall it still gave more than enough to indicate and answer to the question.

From the results of the survey, it can be concluded that the procedurally generated levels performed just as well, if not slightly better than their handmade counter parts, this indicates that PCG techniques can be used for dungeon generation to produce a dungeon level design that is just as good if not better than a handmade dungeon level design. The data also indicated that volunteers were accurate at guessing if a level was procedurally generated just over 50% of the time which indicated that players couldn’t confidently tell between the levels and even in cases where they could tell it didn’t seem to effect their enjoyment as both level types were rated favourably. Taking all of the data into account, the answer to the question could be interpreted as such. “Procedural Generation Techniques can be effective for creating dungeon levels for an action RPG as when compared to handmade techniques they are rated similarly, are had to tell a part and using PCG many levels can be designed in a short time, thus speeding up the development process.”

**6.2 Suggestions for improvement**

There are several ways in which this project could be improved upon. The improvement most mentioned by volunteers of the survey would be adding more varied content to the level generation, this would include things like more variations of rooms generated, more varied and different obstacles as well as components with different graphical aspects and themes in order to differentiate the procedurally generated levels from each other. Adding more varied content is likely a very important improvement to consider as it was mentions by sixty percent of participants.

There were also a few additions that were planned for the level generation which included; adding the functionality for levels to have branching pathways and the functionality for variable room sizes. As the project currently stands, the generated levels only include a single path that leads from the start to the end of the level, adding the functionality to add branch paths would increase the variety of the levels generated as it would encourage players to explore other rooms. The rooms in the current project are all uniform in size, adding variable room sizes would be beneficial as it would add more variety in terms of level layouts. Both additions were unfortunately cut due to time constraints, if this project was developed further these would likely be the main starting point of development.

There are also additions that would improve the project that are suggested by other studies. The study by Breno M. F. Viana and Selan R. dos Santos mentioned in the discussion, also talks about the aspects that haven’t been investigated properly or could be improved on in many approaches to procedural content generation for levels. There are areas that could be addressed in a more developed version of this project. Making the level generator create 3D layouts, which in the case of dungeon generation would mean creating multiple floors of a dungeon that connect to each other, developing a method to influence the difficult of the level by using a difficulty setting to determine how long levels are or how many enemies are present and implementing barriers to slow players progress through levels are all areas which could be further developed.

**6.3 Future work**

There are several ways in which this project could inform future study. The project suggests that Procedural Generation can be an effective tool for creating dungeon type levels so there is room for this to be researched further. Research on this subject would likely benefit from considering the gaps that have not been properly investigated like implementing 3D level designs, variable difficulty in levels or some sort of adaptive generation. The genetic algorithms approach showed promise early on that indicates that there may be some potential there worth investigating however, a different design methodology may be better suited to get the most out of that approach. The prototype methodology made the genetic algorithm harder to work with as it doesn’t lend itself to making changes mid development.

This study would suggest that in the future, procedural content generation will likely be a valuable tool in the development of games- particularly where dungeon type levels are concerned.

# List of References

Abdolsaheb, A. (2022) How to code your own procedural dungeon map generator using the Random Walk Algorithm. [online] Available at: <https://www.freecodecamp.org/news/how-to-make-your-own-procedural-dungeon-map-generator-using-the-random-walk-algorithm-e0085c8aa9a/?fbclid=IwAR1y15CKZhzlt-1jSW\_0K6CU6rki83A4FAbcp53BLne8xDpPfG7KiYxfLO4> [Accessed 15 May 2022].

Bhandari, P. (2020) What is Qualitative Research? | Methods & Examples (online) <https://www.scribbr.com/methodology/qualitative-research/> (last accessed 15th May 2022)

*Binding of Isaac*.2011. Windows [Game]. Nicalis: California

Bonabeau, E., 2002. *Agent-based modeling: Methods and techniques for simulating human systems*. [online] PNAS. Available at: <https://www.pnas.org/doi/10.1073/pnas.082080899#abstract> [Accessed 3 May 2022].

Bloodborne.2015. Ps4 [Game]. Sony Interactive entertainment: Tokyo

Bloodborne-Wiki (2021) Chalice Dungeon Generation System - How It Works. [online] Bloodborne-wiki.com. Available at: <https://www.bloodborne-wiki.com/2019/02/chalice-dungeon-generation-system-how.html> [Accessed 17 October 2021].

Dahlskog, S., Björk, S. and Togelius, J., n.d. [online] Diva-portal.org. Available at: <https://www.diva-portal.org/smash/get/diva2:1420263/FULLTEXT01.pdf%20use%20table> [Accessed 16 October 2021].

Darrin, D., 2022. *Pros & Cons of Agent-Based Modeling*. [online] educational research techniques. Available at: <https://educationalresearchtechniques.com/2020/08/07/pros-cons-of-agent-based-modeling/#:~:text=ABMs%20are%20focused%20on%20the,a%20computer%20seriously%20slowing%20down.> [Accessed 7 May 2022].

Denham, T., n.d. *What is Unreal Engine?*. [online] Concept Art Empire. Available at: <https://conceptartempire.com/what-is-unreal-engine/#:~:text=Unreal%20Engine(currently%20released%20as,like%20Psyonix's%20'Rocket%20League'.> [Accessed 6 February 2022].

freeCodeCamp.org. 2022. *Lee's Algorithm Explained with Examples*. [online] Available at: <https://www.freecodecamp.org/news/lees-

algorithm-explained-with-examples/> [Accessed 4 May 2022].

Fern, D. and Donaldson, S. "Tri-Cycle: a prototype methodology for advanced software development," in Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track, Kailua-Kona, HI, USA, 1989 pp. 377-386.

Halle, M. (1962) Phonology in Generative Grammar, *WORD,* 18:1-3, 54-72

Jansen, C., 2019. Benefits of using Unreal Engine for your next Game app development [online] Available at: <https://www.whatech.com/games/press-releases/622394-benefits-of-using-unreal-engine-for-your-next-game-app-development> [Accessed 19 October 2021].

Kuhn, T.S. (1962) The Structure of Scientific Revolutions. Chicago, IL. University and Chicago Press

Morrow, E., 2021. Never Enough Time: The Dangers of Crunch In The Game Industry. [online] Ritual Motion. Available at: <https://ritualmotion.com/articles/never-enough-time-the-dangers-of-crunch-in-the-game-industry/> [Accessed 17 October 2021].

Naumann, Justus D., and A. Milton Jenkins. “Prototyping: The New Paradigm for Systems Development.” *MIS Quarterly*, vol. 6, no. 3, 1982, pp. 29–44

*No Mans Sky*.2016. Windows [Game]. Hello Games: Surrey

NJIKI, E., 2019. *Prototyping workflow with Machinations and UE4*. [online] Game Developer. Available at: <https://www.gamedeveloper.com/design/prototyping-workflow-with-machinations-and-ue4> [Accessed 17 April 2022].

OECD (2006) Glossary of Statistical Terms. (online) <https://stats.oecd.org/glossary/detail.asp?ID=2219#:~:text=Quantitative%20data%20is%20data%20expressing,the%20height%20of%20a%20person>. (last accessed 15th May 2022)

*Spelunky*.2008. Windows [Game]. Mossmouth: Los Angeles

SHAKER, N., (2018) PROCEDURAL CONTENT GENERATION IN GAMES. [Place of publication not identified]: SPRINGER, pp.30-40.

Shivang (2021) Procedural Generation – A Comprehensive Guide Put in Simple Words [online] Available at: <https://www.scaleyourapp.com/procedural-generation-a-comprehensive-guide-in-simple-words/> [Accessed 18 October 2021].

Smith, G., (2021) Understanding procedural content generation | Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/pdf/10.1145/2556288.2557341> [Accessed 18 October 2021].

Valve Corporation (2021) Browsing Procedural Generation. [online] Available at: <https://store.steampowered.com/tags/en/Procedural+Generation#p=0&tab=ConcurrentUsers> [Accessed 17 October 2021].

Van Brummelen, J. and Chen, B., (2021) Procedural Generation. [online] Mit.edu. Available at: <http://www.mit.edu/~jessicav/6.S198/Blog\_Post/ProceduralGeneration.html> [Accessed 17 October 2021].

Van der Linden, R., Lopes, R. and Bidarra, R., (2014) Procedural Generation of Dungeons. IEEE Transactions on Computational Intelligence and AI in Games, [online] 6(1), pp.78-89. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6661386> [Accessed 16 October 2021].

# Bibliography

If required

# Appendices

Graphical user interface, application, Word

Description automatically generated

GDPR Data Sign Off Form

Graphical user interface, application

Description automatically generated

Survey page 1

Graphical user interface, application

Description automatically generated

Survey page 2