# How can a procedurally generated world improve a game's replayability

GDEV60001 GAMES DEVELOPMENT PROJECT

Sam Ritchie
SUPERVISOR: CATHERINE FLICK
SECOND SUPERVISOR: LUKE BARSBY

# Contents

# Abstract

Procedural content generation is a commonly used tool within video games to increase its replay value, either by aiding development or by randomising content within the game itself, such as generating completely new worlds and maps for a given playthrough. This paper aims to investigate the reasons as to why procedural content generation is used for the purpose of improving replayability and the extend as to which it's use successfully does so. This paper describes a brief history of how procedural content generation has been utilised within games, some commonly used methods of implementing procedural generation as well as the processes specific games use to create procedurally generated maps and worlds. The paper also describes the reasons why and why not developers use procedural content generation in their games. To further research the extent in which procedural content generation supports replayability, a game was created that included two level types, a pre-build hand crafted level and a randomly generated level. These levels were then tested against various replays to see how people's attitudes and playstyles changed in replays between the two-level types. This investigation found that people would find more worth in replaying a procedurally generated map in order to experience something new rather than play the same level again when there are no further incentives to play the pre-built level. However, the investigation also showed that most people prefer a more authored experience in the case where the procedurally generated level is unpolished or unbalanced.


# Introduction

Replayability, the potential and capacity for a game to be played past the point of completion, is an important concept when considering game design (Oxford Learner's dictionary, n.d.). Many video game developers aim to maximise or at very least create a high level of replayability in their games to further encourage their play. This is done through many different methods such having branching story paths to encourage players to experience the alternative (Kaiser, 2012), or performance grading that encourages players to engage with a game and its mechanics to the point of mastery.

One method that aims to extend the replay appeal of a game is to have a map and starting position be different every time the game is started. One method this could be done is to simply hand create multiple maps for the player to play in, but this might take a large amount of time depending on how big the map itself is (Grey, 2017), and even then, this extra content will eventually be exhausted.

To help with this, video game developers use procedural content generation to create fully random maps and worlds, resulting in what could be near limitless permutations of how the map or world is laid out. Because of this, players who wish to play the game multiple times will never run out of different maps and games to play in.

However, procedural content generation is algorithmic in nature and often lacks the refinement of a handcrafted world. Resulting in an experience that may be shallower than a single world hand created specifically around maximising the player's experience (Grey, 2017). The leads into the subject of investigation: How can a procedurally generated world improve a game's replayability and to what extent procedurally generated worlds do this.

## Aims and Objectives

The purpose of this study is to determine how procedural content generation can be used to increase the replay value of a video game, specifically in the generation of level maps and worlds. And to determine the extent in which the use of procedural content generation encourages players to replay games. To answer these questions the following objectives must be pursued:

1. Define procedural content generation and conduct research into how it has been applied in video games in the past and why it has been applied in such ways, taking into account the limitations of using procedural content generation for games.
2. Conduct research into specific procedural content generation methods and algorithms using examples of how they have been applied and implemented in games to aid in the development of an artefact that can generate a fully procedural and pseudo-random level to conduct tests with.
3. Conduct a study using this artefact to determine how people respond to replaying a procedurally generated level rather than replaying a pre-built level, taking into account various factors such as playtime and how fun they find replaying the generated level to ultimately determine if there is a specific reason people would or would not replay a game with a procedurally generated level map over a game that lacks these.

## Literature Review

### Defining procedural generation

Procedural generation can be defined as the process of creating data algorithmically with limited or indirect user input rather than through manual means (Shaker, et al., 2016). Procedural generation is used in various industries such as Animation, Visual Effects and Video Game development (Autodesk.com, n.d.). Procedural generation uses randomisation and other mathematical algorithms under a series of predefined parameters and rules in order to create unique and varied data and content. In video game development this content could be maps, levels, quests, characters, textures, etc.

Procedural generation is used in video game development as a way to remove the necessity of human developers spending large amounts of time manually creating content which can be used to reduce the time and cost required to develop aspects of games with a large scope, such as algorithmically placing foliage across a large open world instead of manually, or by smaller independent developers aid a potential lack of budget or skillset in certain areas such as level design (Shaker, et al., 2016).

Procedural content generation also allows for games to be designed in ways that would be unfeasible to be created by human developers. If algorithms are in place to generate game content while playing the game instead of exclusively during the development process, then players could repeatably generate more content instead of exhausting the content in the game (Shaker, et al., 2016). This allows for games to be designed around the player experiencing the game's content randomly, such as the Roguelike genre of games, or for games to other a completely unique world and experience to the player in games like *Minecraft* (minecraft.wiki, n.d.) and *No Man's Sky* (nomanssky.fandom.com, n.d.).

History of procedural general in games

In order to determine how procedural content generation can be used to increase a game's replay value, how and why games have used it in the past should be examined.

One early example of procedural content generation in video games was the 1979 role playing game *Akalabeth: World of Doom*. Here the dungeon layout and player stats are randomised according to a set seed the player inputs at the start of the game, known as a lucky number (Maher, 2011). This allowed the player to choose whether to play with a set dungeon layout and stats or to randomise these variables using the random function in BASIC (Tauber, n.d.). The idea of letting the player randomise the world or to pick a seed would become common in games that highly utilise procedural generation after this.

Another of the earliest examples of procedural generation in video games was *Rogue* (Procedural Content Generation Wiki, n.d.), which released in 1980 and served as the inspiration and basis for the Roguelike genre which would go on to heavily feature procedural generation techniques. In *Rogue,* the player descends a dungeon layer by layer with each floor having a random arrangement of rooms. This is done through a simple system that maps the connections between rooms out before drawing and populating the rooms themselves (Hughes, 2003). At this point, procedural content generation is being used to make games more replayable, offering unique experience each time the game is played, though this is also to allow the games to actually work, since these games were built to have too small file sizes to store game content that isn't generated live.

*Elite*, released 1984, uses procedural generation to generate a large explorable universe of eight galaxies each with 256 stars. In *Elite*, various aspects of a system or planet is stored as a series of numbers that is generated using an algorithm similar to the Fibonacci sequence from a single 4-digit seed. There is also no random element to this generation since the seed is hardcoded into the algorithm, which means that players will all play in the same universe despite the games world being generated at runtime (Spufford, 2004). The method of using a preset seed to procedurally generate a massive world that is consistent with every copy would continue to be used in several games in later years, such as *The Elder Scrolls II: Daggerfall*, which boasts a map the size of great Britain (Bethesda Softworks, 2004) and *No Man's Sky*, in which the procedural generation algorithm can in theory generate up to 18 quintillion different planets to explore (Khatchadourian, 2015).

*The Sentinel*, released 1986, uses a procedural content generation algorithm to generate 10 000 levels. Completing each level rewards the player with the code that will be used to generate the next level. With the correct code the player can generate any level at any time (C64 Wiki, n.d.). This use of a procedural content generation algorithm allows for a large number of unique levels while maintaining a game size of under 100 Kilobytes (C64 Games, 2018). These are example of games that use a non-random form of procedural content generation to have massive worlds while keeping file sizes down. Despite the fact that these worlds are not random, the scale of them means that players will likely never exhaust new content.

Procedural generation continued to evolve during the 90s with clear grid based maps such as *Sid Meier's Civilization* (Karth, 2015), *Master of Magic*, *and SimCity 2000* (Retro Gaming Discourse, 2021). Other examples of 90's games that include procedural generation include Age of Empires, which uses a tile based system typical of the Real Time Strategy genre (Karth, 2015); Worms

Armageddon, which features a random terrain generator to create the maps for each game (Worms Knowledge Base, 2008);

And Diablo, which uses an isometric tile grid system 40x40 tiles large in order to generate its dungeon maps (Newgas, 2019). Diablo generates its dungeons in two stages, first generating the walkable map in an array before more complex terrain and visuals are generated after the map is converted to a grid map. This allows the game to check that the map is valid before adding visuals, ensuring that every map is visually and atmospherically consistent.

Another step forward for procedural generation in video games was *Dwarf Fortress*, released in Alpha in 2006 (procedural-generation, 2015). Rather than just using procedural content generation for its map layout, *Dwarf Fortress* measures climate vegetation, the moral temperament of the world as well as centuries of history in order to create a backstory for the cultures and places in the map. This is all in service of creating a unique experience for each player and for each playthrough of the game, a leading contributor to its replayability.

The most popular example of procedural content generation in video games is *Minecraft*, which started development in 2009 (Zucconi, 2022). Inspired by the game Infiniminer, which itself uses procedural content generation for its map, *Minecraft* creates a completely random 3D world for players to explore spanning 60 000 000 x 60 000 000 blocks. *Minecraft* combines and layers many different procedural generation techniques on top of each other in order to create this expansive world, such as various noise functions and stochastic cellular automata. *Minecraft* is one of the most popular games of all time and arguably the most famous game to include procedurally generated worlds. These massive unique worlds make each players experience unique, often leading players to replay the game to have a different experience, Procedural content generation, alongside total player freedom, is one of the main reasons as to why *Minecraft* is so replayable.

Procedural content generation has been used from the earliest days of video games for various purposes, from reducing a game's file size to extending how long players spend on a single playthrough. While common in modern games, the idea of using procedural content generation to incite players to replay the game has also been around for a long time.

## Different procedural generation techniques

For the purpose of investigating how procedural generated levels and worlds can increase the replay value of a game, different procedural content generation techniques should be looked at for the purpose of creating an artefact that can be tested.

One common procedural generation technique is the use of Perlin Noise, which was introduced in 1985 and has been used across a variety of different games as part of their procedural generation method, such as Civilization 6 (Kowalski, et al., 2018) and *Minecraft* (Minecraft Wiki, n.d.). Noise is a randomly generated and unstructured pattern, defined by Perlin and Hoffert as "*an approximation to white noise band-limited to a single octave."* (Perlin & Hoffert, 1989) That is, a pattern of frequencies within a specified range with random phase, often represented as a pattern for use in computer graphics.

Procedural noise functions are an algorithmic and programmed process for generating noise. These functions have several benefits to being used in procedurally generating game content (Lagae, et al., 2010). Some potential advantages include that the function itself is much smaller than a pre constructed noise image or volume, which saves space in the program. They are non-periodic, never

repeating itself, thus is unlimited in size and can fill large areas without repeats. Procedural noise functions are continuous rather than discreet, meaning that values are given at every resolution. The output of the procedural noise function can also be affected by different parameters that could be included in the function, granting the developer much more control over the output of the noise function.

Perlin noise is a Lattice gradient noise, which is a type of procedural noise function characterised by manipulating the random values and/or gradients defined by points in an integer lattice. Specifically, Perlin noise determines the noise at any point in space by first hashing the points of an integer cubic lattice around the point that is being looked for to give a series of pseudo-random gradients and then doing a splined interpolation with a quintic polynomial. The use of gradients in this algorithm allows for a smooth transition between random values instead of completely discreet integer results. This means that, if the Perlin noise function were applied to a height map the result will show a series of hills and valleys connected smoothly (Dawnosaur, 2023).

This algorithm has been used across a variety of computer graphics and games since its conception and may be a useful algorithm to use as part of a greater algorithm to create a procedurally generated world.

*Figure 1 A 2D view of Perlin Noise*

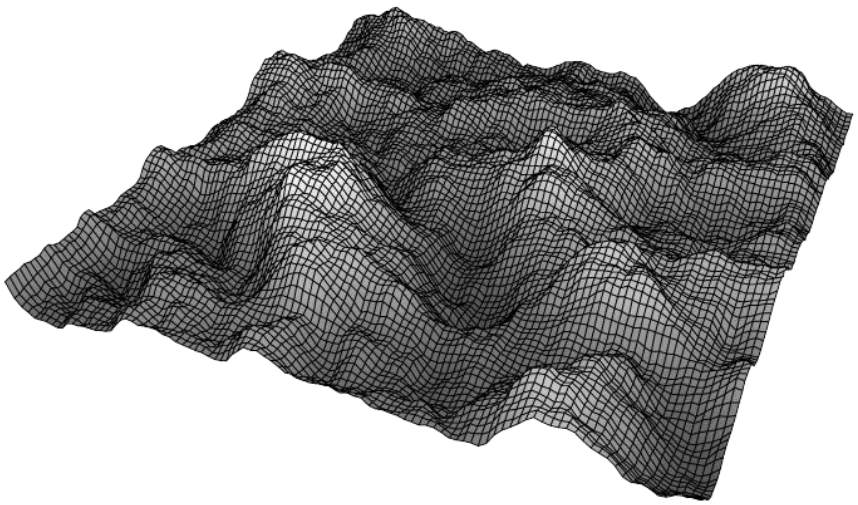

*Figure 2 Perlin noise applied to terrain generation (Scratchapixel.com, n.d.)*

Another technique that could be employed for use in grid based procedural generation is Cellular Automata. A Cellular Automaton consists of a regular and discreet grid of cells in which each cell is designated one of a finite list of states. Each cell is also designated neighbouring cells relative to its position. At discrete and regular time intervals each cell, in parallel, updates its state to reflect the result of a mathematical computation involving its current state and the states of the neighbouring cells (Berto & Tagliabue, 2023). This computation, or rule, is what defines the behaviour of the automata.

The method of using Cellular Automata to procedurally generate a level map is discussed in a conference paper by Daniel Ashlock and Matthew Kreitzer (Ashlock & Kreitzer , 2019). Ashlock and Kreitzer experiment with using cellular automata for map design by encoding the automata rule as a square matrix of size equal to the number of available cell states. The encoding as matrices allows for the rule to be easily morphed across different areas of the map using multiple matrices.

Their testing used 2 6x6 matrices with each matrix element being stored in a vector data structure of length seventy-two. They then created child structures the matrix data is copied into but with a certain segment swapped between the matrices as a two-point crossover. Following this a random number of elements in the child matrices are further randomised as a simulation of mutation. From here both the parent and children matrices are applied to different population members or starting cells in the grid. From here the cellular automata begins to work following the rules encoded by both the parent and children matrices and are further influenced by a value they call the “quality measure” which assigns a score to how well the evolved automata resemble a playable and enjoyable level map.

Ashlock and Kreitzer describe the basis of the quality measure to maximise the traversable path between the centre of each the four boundaries of a quadrilateral map. Beyond this basic measure they employ three different modifiers so that the final evolution resembles a good map. These include the density modifier, which ensures that there is not too much or too little empty space; the

diversity modifier, which ensures that each state is evenly distributed; and the angular modifier, which uses the angle between vectors to calculate and avoid evolution in which the rule has not updated thanks to overlap between parent and child matrices.

The conclusion Ashlock and Kreitzer came to was that while cellular automata could not create perfectly optimised levels unless run for an unlimited amount of time with good crossover and mutation elements when the children matrices are created, allows for a programmer to procedurally generate maps for a war game or video game that are all different, low cost and rapidly created while still being tactically equivalent.

Another reported use for cellular automata in procedural generation is to generated maze patterns that could be applied onto a level map (LifeWiki, n.d.). An example of this ruleset is *Maze*. In this automata ruleset that cells are born if they have exactly three neighbours and die if they have no neighbours or more than five neighbours. From a random starting arrangement this will grow into a large pattern in a maze structure with defined walls and corridors. A similar automaton is *Mazectric*, which kills cells if they have above 4 neighbours rather than five, this results in a maze with longer corridors than ones generated with *Maze*.

The outcome of the generation is deterministic, with each pattern being determined by the starting arrangement. This allows the pattern to be entirely replicated provided that the initial conditions are also replicated. This may be useful for a level designer to implement if they want a random maze that can be set by using the same seed, by using a seeded noise function to replicate the initial conditions for the automaton.

However, it should be noted that the Maze ruleset generates a pattern that is maze-like but is not complete in the sense that every empty corridor is connected to each other, instead featuring a series of disconnected pockets of corridors that resemble a maze pattern overall. An algorithm and ruleset for a cellular automaton that generates a completely connected and traversable maze is described by Justin A. Parr (Parr, 2018). This method develops corridors from connecting cells to each other, resulting in an array of cells with descriptions for which cell they are connected to, if maze generation logic is applied then this could be rendered as a fully connected maze by rendering walls between cells in which there is no connection.

*Figure 3 Maze generated with the Maze cellular automaton ruleset. Note how while appearing maze-like, this does not generate a fully traversable map (LifeWiki, n.d.)*

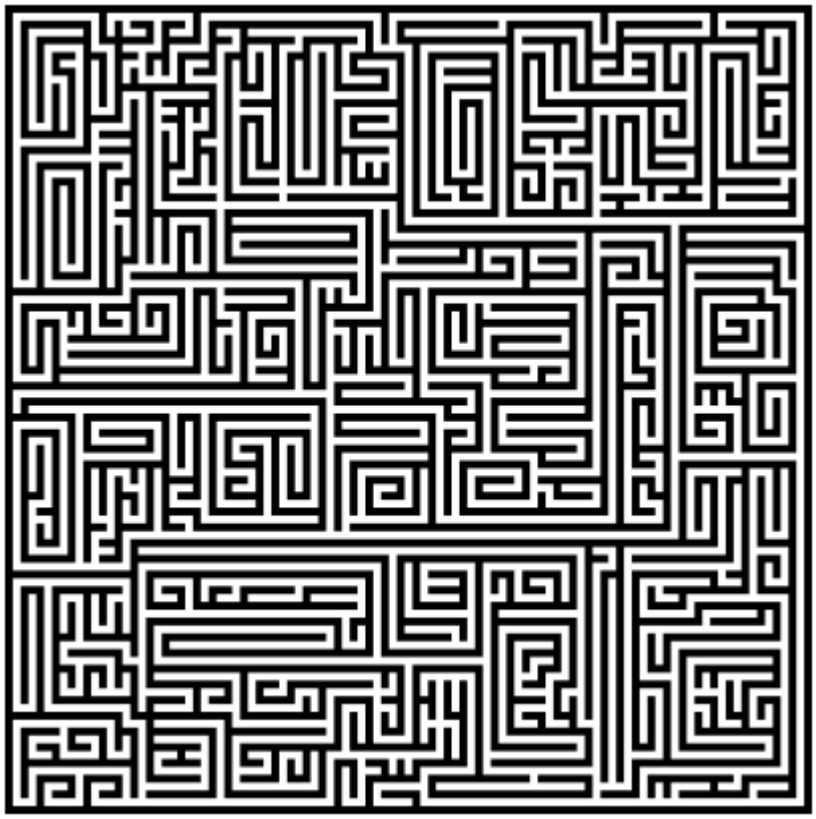

*Figure 4 Fully traversable maze generated using Justin Parr's cellular automaton ruleset (Parr, 2018)*

Mixed-initiative content creation, or MICC, combines the best aspects of procedural content generation with human driven design to create a method that still gains the benefits of procedural generation while requiring significantly more human input in order to generate a more predictable and "designed" content (Liapis, et al., 2016). There are different extents as to how much influence a human developer has on a MICC, with computer-aided design being an example in which humans have most of the influence over the design with the software supporting and facilitating this process and interactive evolution in which the software takes most of the initiative to autonomously generate content with human input acting as a guide steering the generator in the desired direction but without any direct control.

Interactive Evolutionary Computation, or IEC, can be applied to a system in order to develop procedurally generated content for games. In this process humans select which generation to survive and create offspring according to a criterion until the result fulfils the desire of the developer or user. This is different to a completely automated evolutionary algorithm, such as Ashlock and Kreitzer's Cellular automaton (Ashlock & Kreitzer , 2019) in that the generations are manually approved or rejected, which allows developers greater flexibility and control over the final outcome but with the issue that human input is subject to the eventual fatigue of the developer.

An example of MICC is *Tanagra*, a design assistant for 2D platformers (Smith, et al., 2010) which functions by allowing human developers to specify certain specific areas of a level's geometry and have the software fill the gaps in a way that always results in the level being playable given that the inputs are valid.  Many games in the roguelike genre use MICC, employing a combination of premade content, such as room content and smaller arrangements, and algorithmic processes that arrange this content in a pseudo-random pattern (e Silva, et al., 2020).

A MICC method proposed by Rafael C. e Silva et al. (e Silva, et al., 2020) for a 3D game involves manually placing down connecters containing pins on a piece of level geometry. This is then fed to an algorithm which is also given certain parameters specified by the developers. The algorithm then produces a playable map with these in mind. Geometry pieces are premade and are given connectors where they would connect to another piece, such as doors. The pins of the connecters dictate which connectors can be connected, like puzzle pieces. Starting with a single piece of geometry the algorithm places valid pieces of geometry down according to the different mode of generation, with different pieces being preferred in different circumstances. This repeats until certain conditions are met, such as running out of valid pieces or connections or reaching a limit to the number of rooms.

This algorithm is designed for a 3D game, but similar methods could be used in a 2D game as well. Overall, a MICC algorithm is an effective way to mix predesigned content with procedurally generated layouts and experiences, giving the developers more control over the gameplay at the cost of involving them more in the development process itself.

These procedural content generation techniques should be used in the creation of the artefact, a procedurally generated level that will be tested to determine if procedurally generated world and levels do increase a game's replay value.

Examples of procedural generation in games

To further support development of the artefact, research is done into how specific games handle its procedural generation, these methods and algorithms could be applied to the creation of the artefact but also inform about the specific ways procedural content generation techniques influence the design of a game to improve a game's replayability.

According to software developer Mark Damon Hughes (Hughes, 2003), *Rogue* handles its randomly generated map by dividing the map into a 3x3 grid and giving each cell a room with a Boolean value checking if the room is connected to the rest of the map and an array containing all of the room's connections. Starting at a random room, connect to a neighbouring unconnected room and mark it as connected and as the current room before repeating. The algorithm will cycle again if there are no neighbouring unconnected room until all rooms are connected in a random pattern. Additionally, a random number of extra room connections are made. The rooms are then drawn onto each grid cell and corridors are drawn to each connected room with doors being drawn where the corridors intersect with the rooms. Sometimes *Rogue* chooses not to draw a room in a cell but keeps connection, resulting in a long generally L shaped corridor.

Some games such as *Spelunky,* released 2008, uses procedural generation to create smaller stages rather than larger worlds. According to programmer Darius Kazemi, each stage in Spelunky is divided into 16 rooms in a 4x4 arrangement with each room being 10x8 tiles large (Kazemi, n.d.). Each room fits into four types depending on where the room can be exited from and whether the room is on the main path from the stage's entrance to its exit. The first stage of the stage's map generation is to set out which room type each of the sixteen rooms are by randomly picking room types until there is a path from the top lay of the stage to the bottom. Rules are put in place here to make sure the randomness does not break the intended game loop, such as forcing a room that allows the player

to descend to the next layer if at the stage boundary. After this path is formed each room not on the path is given a room type without specific connections.

After each room is given a type, they are randomly assigned a room template from a premade list with each template having tiles that are static and “probabilistic” tiles that are themselves given a random chance to generate as several distinct types of tiles. Also, on these room templates are “obstacle blocks,” groups of 5x3 tiles that themselves have a randomised template and contain probabilistic tiles to create extra help or obstruction for the player and further randomises the room away from the template.

*Spelunky* has been praised for its choice to include procedural generation in its stage design as it creates levels that cannot be practised, which benefits the game by making each run is intense as the others (kestrel, 2014). This supports the idea that procedurally generated stages are beneficial to games in the Roguelike genre or similar.

An example of a larger scale work of procedural generation in a 2D game is the world generation in *Terraria*, released 2011. *Terraria* is a sandbox game with a heavy emphasis in exploration and thus uses a complex procedural generation algorithm to create a detailed world for the player to explore. This process begins by allowing the player to select different parameters for the world generation (Terraria Wiki, n.d.). These parameters include World seed, which resolves every random element of the algorithm; World size, which tells the algorithm how large the generated world should be; Difficulty, which determines certain elements of the world generation such as what loot is found within chests; and “evil biome”, Which is the choice between two mutually exclusive biomes that appear in a world with unique design and geometry.

*Terraria* uses a system it calls Generation passes (or GenPass as it is written in the code) to handle the order in which different elements in *Terraria’s* world are generated (Relogic, 2011). These passes are placed in a specific order since content generated in newer passes will overwrite content in earlier passes. For example, the pass that handles ores takes place before the pass that generates cave structures so that the ore is not drawn on top of the cave structure (GabeHasWon, 2022). *Terraria*’s process starts with the main world terrain before progressing to other large terrain features such as ocean biomes for map borders and different biomes in the mainland before progressing smaller features such as ores, traps, and treasures (Terraia wiki, n.d.). Individual segments of the generation process use Perlin noise, which are overlayed over each other (rubby, 2022).

The process of dividing the procedural generation process into many different passes is especially useful when developing an algorithm to generate a custom map with many overlapping elements.

While the *Binding of Isaac*, released 2011, uses premade rooms in its level maps, the layout of the map itself is procedurally generated. This process is described by Florian Himsl, one of the developers of the game (Himsl, 2020). Himsl’s algorithm begins with a singular square starting room in the centre cell of a nine-by-eight grid and randomly chooses which of the four walls to put doors on. For each wall with a door an adjacent room is placed, with the algorithm repeating until the desired number of rooms have been met. After this there will be several rooms with only one connection, which are designated as “end rooms” representing their status as dead ends. These rooms are given special properties such as being designated as a shop, item room, or boss room. The boss room is always the end room furthest from the starting room. From here the rooms are filled with object, terrain and enemy layouts chosen from different lists of premade room depending on the desired difficulty. Secret rooms are placed with an entirely different method that searches for an

empty cell next to three other rooms and no end rooms. If no suitable location has been found after every 300 attempts the criteria will loosen to ensure that the secret room will always be placed but with bias to certain conditions (Newgas, 2020).

This algorithm, while simple, has the potential to run into some issues. For example, if the door randomiser decides to not generate doors, then every branch of the map will reach an end room before the desired number of rooms have been met, and the map must be thrown out and regenerated. Another problem is that the grid the map generates onto is only nine-by-eight cells. If the algorithm attempts to place a room outside of this boundary, then the current generation is also scrapped. This becomes more difficult on later floors which have a higher desired number of rooms and thus it is more likely that the branches of the map will approach and attempt to overstep what is allowed. This results in the possibility of extended loading times between floors as the game attempts to generate a useable map.

The 2014 remake of *Binding of Isaac*, entitled *Binding of Isaac: Rebirth*, alongside its subsequent expansions alter the level layout generation algorithm (Binding of Isaac: Rebirth Wiki, n.d.). much like the original algorithm, the *Rebirth* algorithm starts by calculating the desired number of rooms however now it starts by calculating a desired number of end rooms and their locations rather than them being placed as a consequence of the map reaching its expansion limit. The end rooms are also filled with special rooms at this point with the boss room being assigned the end room furthest from the starting room, much like Himsl's original algorithm.

From here layouts and positions of normal rooms are selected to fill out the transition between end rooms and the desired number of rooms. These normal rooms consist of premade rooms of variable difficulty. In *Rebirth* each room is built with the connections in mid rather than the room's content being independent of its connections like in the original (Himsl, 2020). *Rebirth* also makes use of rooms that occupy more than a single cell on the grid, such as rooms in L shapes or longer thin corridors. Secret rooms are placed at this point by grading each viable space depending on how many rooms its adjacent to and picking the highest.

The evolution of the *Rebirth* algorithm over the original shows how a procedural generation algorithm can be improved overtime to expand what is possible to be generated in the map. The original algorithm, however, is still useful to look at as it represents a way an effective procedural generation algorithm can be designed without the resources to design the more complex features nor the larger array of prebuilt rooms of *Rebirth*.

The use of procedural generated levels and worlds in each of these examples each show how Procedural content generation is used to increase a game's repalyability

<u>When and why to use procedural generation</u>

In the book Procedural Generation in Game Design, Darren Grey of Roguelike Radio writes that the reasons to use procedurally generated content can be sorted into either basic logical reasons that support the development or gameplay loop of a game or more unique "fun" reasons in which PCG can be implemented for a more unique effect (Grey, 2017). For the more pragmatic uses Grey includes:

- Time Saving, in which games with large amounts of geographic content are procedurally generated rather than manually created due to the excessive workload that manually

created this work might have, though care should be taken that the work done to create and tweak the generator does not exceed that of what be done to create the world manually.

- Making a game more repayable by generating content that is unique to the current game. Used a lot in Roguelikes but also used in large survival games such as *Minecraft* and *Terraria*.
- Generation of content on a scale or difficult to create by hand, in cases like *Elite* or *No Man's Sky*, the games are defined by their scale, which vastly exceeds what is realistically, or physically, possible with a limited timeframe and budget.
- Making a game of such a large scale fit onto a smaller storage device, or simply to reduce file size, as in most cases an algorithm will be smaller in size than prebuilt terrain meshes. The more terrain generated at runtime is the less terrain stored in the game file.
- If a new feature is added to the game, it can be easier and more time saving to have a content generator to implement it into the game rather than manually implementing it. For example, if a new weapon is added to a large open world game, it is much easier to added it to a list of items that can be generated inside chests than placing it manually inside a large number of chests.
- Some content generators can be used across different projects, significantly reducing the work that need to be done.
- Content generators will always stick to desired rules and conventions where manmade content may accidentally break these rules.
- Procedural content generation can also be used in some games to simulate behaviours in game based on given rules.

These are the reasons most developers will employ procedural generation techniques. But for more personal reasons someone might use these methods, Grey suggests ideas that include how procedural generation can give each player a completely unique experience that is perhaps more personal. Another idea is how procedural generation inspires new interactions with a game, such as how players can strategize to manipulate the algorithm that controls the game's generations. Grey also suggests that procedural generation can be used to create content and ideas that the developers might not think of, resulting in interesting and unique results.

Not only is it important to understand the uses of procedural generation, but also the risks associated with their use. A generated game may not be guaranteed to function correctly 100% of the time, but locating these issues can be difficult before release. The less constraints applied to the generator increases the chances of a serious issue generating, but while having more constraints may results in the generated content being seen as repetitive. The Mixed initiative content creation approach can sometimes fall victim to this, if each room of a dungeon is prebuilt with a generated layout, then there is little to no risk of the rooms themselves being broken, but players may see the limited number of room layouts as repetitive.

While Procedural content generation may save a lot of time for generating large maps over handmaking the maps, the time and effort put into designing, programming, tweaking and testing the generator itself may come close to or even exceed the time that would otherwise be spent on hand making the level. A PCG system may be more unpredictable in how much time and resources it takes to develop than handcrafting, and much harder to adapt if the scope of the game changes. If a game's map halves in size over the course of development, this cuts the effort to handcraft this map in half but makes little difference to the time and resources spent on a PCG system.

PCG is also unsuitable for parts of games that require a more authored approach, such as games in which the level design works to support the story or games in which the who gameplay loop in to

beat the challenges presented in a specific room. The best platformers have level design that takes advantage of every present mechanic to build an engaging gameplay experience. In this case it will almost always be more suitable to handcraft these levels than generate them.

Procedurally generated content, especially levels and items, can cause issue with game balancing if not correctly polished by hand. PCG in multiplayer maps, seen in RTS games, can result in an unfair advantage being placed upon a particular player.

Overall PCG should not be relied on too heavily and the novelty should not be a substitute for the game experience. When making a game with PCG it is a good idea to limit the number of systems that utilises it and to carefully consider the effect that the specific PCG method has on the player experience. If the generation is too restrictive then certain content might repeat itself, robbing a particular part of the game of its uniqueness, while too unrestricted a method my result in content the player dismisses as just random. Both of these extremes undermine the meaning the player absorbs from a particular point of the game (Johnson, 2017).

## Research Methodologies

This investigation will utilise the design Science research methodology, which focuses on the creation of an artefact to better understand the involved technology and systems, such as how and why procedural content generation is used in games. This artefact should undergo testing to attain a better understanding of the problem and how a more suitable artefact could be created.

To investigate this question, volunteers will be asked to spend time playing a level from a game prototype developed specifically for this investigation. There will be two different versions of this prototype, one will contain a handcrafted level that takes into account level design best practices, and the other will contain a procedurally generated level that is generated at runtime using a random seed so that the level is completely different each time the level is generated.

The participant will be asked to play each version of the prototype twice, experiencing the prebuilt level twice and 2 randomly generated levels. After each level the participants will be asked to express on a scale from 1 to 5 how much fun they had exploring each map. After all four playthroughs participants will then be asked if the pre-built level or the generated level felt more worth replaying and if playing the pre-built level twice felt repetitive. After this, participants will be asked to give general comments about how they would compare the pre-built level vs the Procedurally generated level.

The amount of time each participant spends on each playthrough of the pre-built and generated level will also be recorded and compared with each other to determine if there is a trend in which the second playthrough of each level differs from the first or if the time taken to complete the pre-built level significantly differs from the time taken to complete the procedurally generated level.

Given the available timeframe of this project the scope of the prototype itself should be limited, visuals will not be focused on and while basic game mechanics will be implemented, the gameplay itself should not be much more than a means to enable and encourage the exploration of the levels, which is the focus of the investigation.

The prototype will be built in Unity using the 2D Tilemap system, and both versions of the level will contain an overworld area connecting a maze and a dungeon area inspired by binding of Isaac and the Legend of Zelda. These were chosen as the scope of the level as anything more complex would become difficult to program an algorithm to generate within the timeframe.

The overworld should be generated by applying a Perlin noise texture onto the tilemap, which should result in a natural looking terrain due to the values on the Perlin noise texture being continuous rather than discrete. The maze can be generated using an adaptation of Justin A. Parr's method of using a cellular automaton to create a maze. The dungeon can be generated using Florian Himsl's method of dungeon generation created for the binding of Isaac (Himsl, 2020), in which the layout of the rooms is generated first and then populated with preset room interiors using mixed initiative content creation. The algorithm to generate the room layout can be adapted from the algorithm that will be used to generate the maze.

## General Gameplay

The general gameplay of the game is 2d with a top-down perspective. The objective of the game is to collect 2 stars found in the dungeon and the centre of the maze, forcing the player to explore both these areas and the overworld area between them. To allow players reasonable ability to complete the objectives and to prevent players getting lost, a compass is given on the UI that points towards the closest star so that players always have a direction to head towards.

To encourage exploration beyond simply following the compass, the player is given limited health and some hazards to avoid within the level. Enemies will chase the player and deal damage when next to the player, these enemies are located in the overworld and dungeon areas but can be defeated by the players attack or avoided by the player dashing away.

Another hazard is spinning blades in the dungeon area that deal damage on contact and cannot be destroyed by the player, but do not move.

If the players health is depleted, they will be returned to the same level (with the same seed if procedurally generated) and will have to collect both stars again.

## Generation Algorithms

The procedurally generated level is created from an array of integers with each integer corresponding to a different tile either on the floor layer or the wall layer. Some tiles also have a corresponding object to be spawned on top of it. The array is converted into tiles at the very end of the process, with each individual procedural generation method affecting the array rather than the tilemap itself.

The first stage of the process is to set the seed. This is either randomly generated or set from the main menu. All random functions or methods requiring randomness use this seed. This will result in an identical map every time the level generation algorithm is run with the same seed.

This seed is used as the starting offset for Unity's built-in Perlin noise function which is applied across the map. Each point on the array is given a corresponding value based on the value of the noise function. This translates the continuous function to one with discrete values that can each be

given a tile. The Perlin noise function is incremented by $1/10^{th}$ for every cell in the array, which results in more natural terrain. Low values of the noise function are given tiles that resemble water and the high values are given land. The highest and lowest values are designated as deep water and rock formation respectively and placed in the wall tilemap so that the player will have to navigate these features.

After the overworld is set in the array the maze and dungeon are generated using Cellular Automata.

The maze uses Justin A. Parr's method for maze generation which involves a smaller array of structs containing a cell's state, the cell that the current cell was activated from, the cell the current cell is trying to activate, and an array of the four orthogonal neighbouring cells. Using the level seed a "seed cell" is selected from the deactivated cells in the array and given the "seed" state.

From here the algorithm cycles through each cell in the array several times following the instruction set given to each state that the cells are in, these include:

State 0 (disconnected/deactivated):

1. no instruction.

State 1 (seed):

1. set neighbour array to include each disconnected cell orthogonal to this cell.
2. If the seed cell has information of what cell activated/connected it, decide using a random function if the next invited cell should continue the straight line or pick a random direction.
3. Note the cell that the current cell is trying to invite and change the current cell's state to 2.
4. If there are no available neighbours, change state to 3.

State 2 (invite):

1. Change the invited cell's state to 1 and its connected cell to the current cell.
2. Using a random function, determine if the current cell's state is set to 3 (connected with no further process), or state 1 (seed state, for branching pathways).

State 3 (fully connected):

1. If there are no active seed cells and this cell has at least 1 unconnected orthogonal neighbour, then using a random function this cell can return to state 1 (this is so the entire array is connected by the end of the automaton).

The result of this is an array of cells connected together. These are then transferred onto a larger integer array with the connected cells and the cell in the invite direction being set as empty corridor cells. This will result in a complex maze in which every point is fully accessible from any other open point. To finalise the maze an empty room was cleared at the centre of the maze that will hold a gameplay objective which is given a designated cell in the centre of this room. Then the final array is mapped onto the main level array, overwriting the existing data in that smaller area.

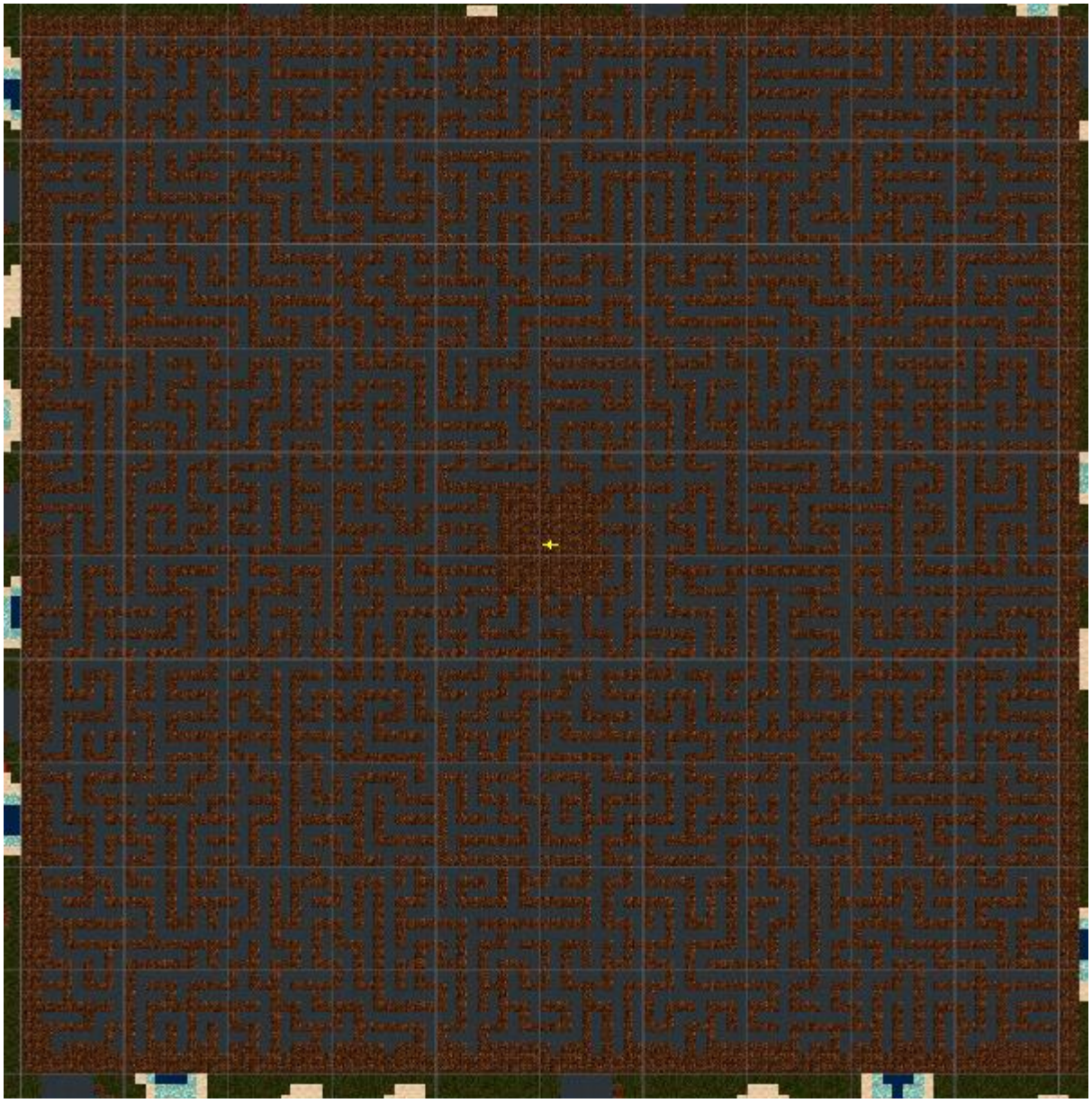

*Figure 5 The procedurally generated maze used in the prototype, adapted from Justin A. Parr's cellular automata algorithm*

The dungeon generation algorithm was inspired by Florian Himsl's process for dungeon generation from Binding of Isaac and uses the same cellular automaton method as the maze with the distinction that a limit is given to the number of cells that can activate before the automaton is stopped. The array is also a fifth of the size of the maze array and does not include a random seed like the maze automaton, instead the centre cell starts in the seed state.

Here, each cell represents a room, and the automaton generates which rooms are generated in the level and how each room is connected together. These are written as large empty rooms 10 times the scale on an integer array. These empty rooms are populated with a random handmade room arrangements following a mixed initiative content creation framework. This is so each room is completely traversable in game. Spelunky's method for room generation could have been applied here, this method also uses preset room arrangements but also includes set areas withing these arrangements that are further randomised but given the timeframe it makes more sense to limit the scope of the project. Certain tiles in the dungeon are designated to spawn traps and a gameplay objective in the centre room. Doors are created for rooms that are connected together and each

room is given a random chance to spawn more doors so that the dungeon can be accessed from outside and to make the room arrangement less linear.

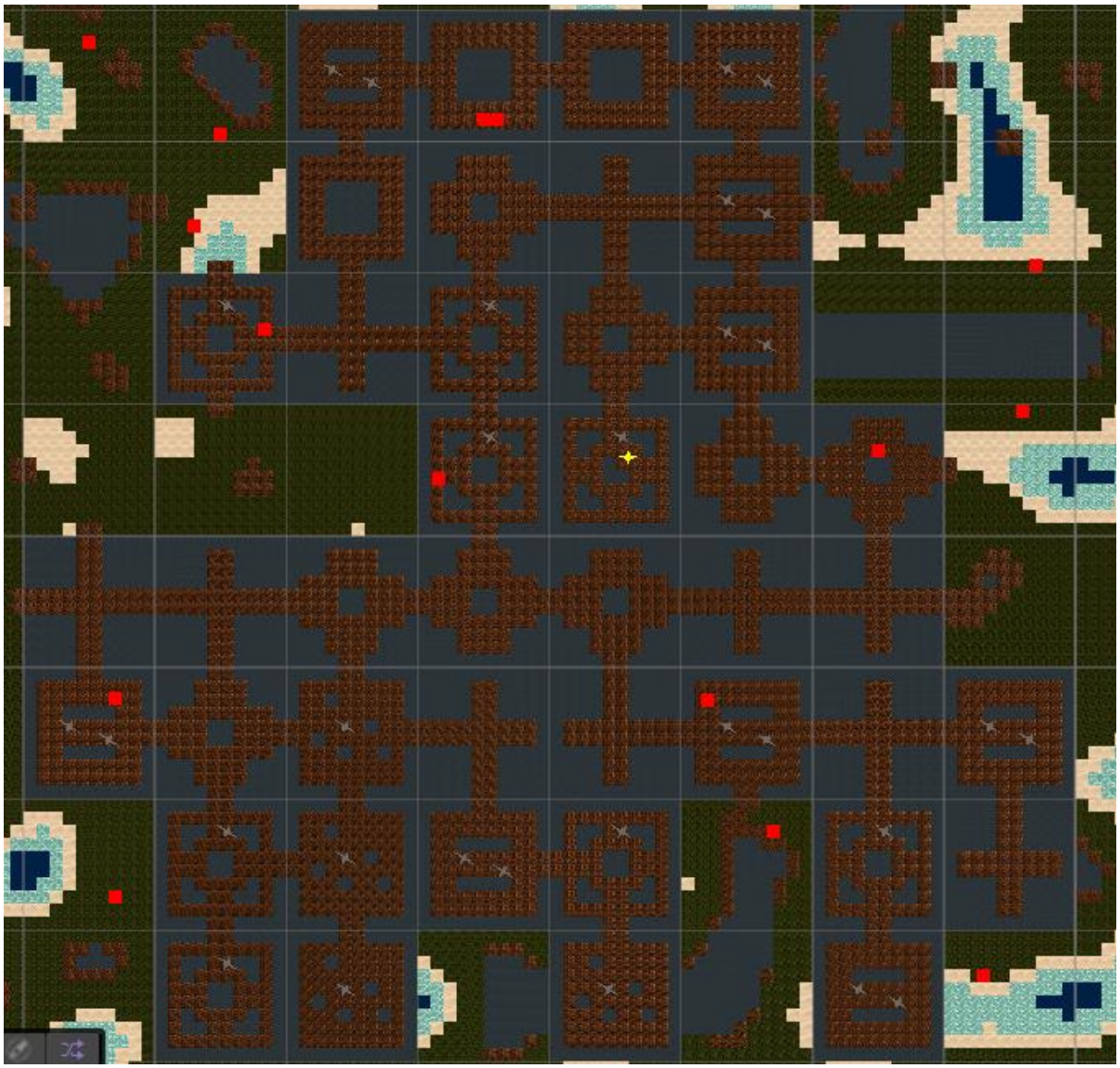

*Figure 6 The procedurally generated dungeon used in the prototype, inspired by Florian Himsl's MICC framework for Binding of Isaac*

After the maze and dungeon arrays are mapped onto the main world array the values in the array are each converted to their respective tile on the wall or floor tilemaps. Cells designated to spawn traps and objectives do so by creating an instance of these on top of a generic floor tile. A number of enemies are placed on floor tiles in any place except for the maze. Enemies are placed using a simple random number generator and will fail if the generated spawn point is inside a wall or in the maze area. The enemy spawner will retry until a suitable spawn location is found. This means that the same number of enemies will always spawn during the level generation. The Enemies use a navMesh to navigate the level, which is generated last after the rest of the map. This navMesh uses the NavMeshPlus plugin by h8man (h8man, n.d.) to work with the tilemap system.

Pre-built level design

The pre-built level takes into account various game design methods to form a rudimentary alternative to the procedurally generated level. This includes using the position of the enemies and the arrangement of wall tiles representing rock formations in the overworld to guide players following the compass towards the main entrances of the dungeon and the maze.

While the overall map size is the same between the pre-built level and the procedurally generated level, the dungeon and maze areas are smaller in the pre-built level compared to their generated counterparts. This was done as manually creating the dungeon and maze to match these sizes would have taken an unreasonable amount of time and effort given the scope of this project. This is in line with one of the more common reasons a game developer would employ procedural generation, to save time and effort in cases when creating large amounts of repetitive content would be excessive and/or unreasonable.

The dungeon layout in the pre-built level has more variety both in room shape and size, and the arrangement of the room interiors compared to the generated dungeons and are arranged as to require the player to explore a large amount of it before reaching the star. Room interiors in this version are created to follow the rooms connections rather than having to be generalised to allow for entrances to be placed on all 4 walls. While the compass and terrain lead the player to the main entrance, that which has the longest path to the star, entrances have been placed around the dungeon for those who miss the main entrance. These entrances are often closer to the star, rewarding the player for their exploration.

*Figure 7 Examples of the more unique dungeon room arrangements used in the pre-built level of the prototype*

The maze in the pre-built level is arranged with some affordance to allow for it to be solved without too much struggle without being too easy. As with the procedurally generated level, there are entrances to the maze all around the outside, giving the players many choices with how to approach.

*Figure 8 The maze from the pre-built level of the prototype. Noticeably much smaller than the procedurally generated maze due to time limitations*

## Results and Findings

The primary investigation yielded 6 sets of responses. The prototype performed as designed, procedurally generating a level exactly as specified and the design of the pre-built level worked to guide players down an optimal path, that is the path of experiencing the most content out of a given playthrough.

There were some bugs with the prototype during the testing stage, however. In one case the menus were altered partway through the testing stage so that a player could return to the main menu after completing a level, this was to streamline the testing so that the protype did not have to be reset after each level completion, but removing the active seed from memory to facilitate level regeneration caused issues where players would fail to respawn in the correct level after defeat. This bug was fixed by keeping the seed in memory until level completion and not just following the generation of the level. This bug was present during one of the tests, effort was made to correct the participants time following this bug.

A common experience from the results is that there is a disparity in the difficulty between the pre-built level and the procedurally generated level that went unnoticed until the testing stage. While this was to be somewhat expected as a procedurally generated level is created with less oversight, the significance of this disparity was noted repeatedly, especially in the number of enemies. The best use of procedural generation applies rules and heuristics to create content that fits the rules that the developer puts in place, such as the intended difficulty of a level. Balancing this would be as simple as adjusting the number of enemies the system attempts to spawn, but doing so would alter the prototype in a way that would subsequently alter the results, requiring a full restart of the testing. Instead, it was decided that the increased difficulty of the procedurally generated level would remain in the prototype for the duration and would be noted with the results of the tests.

Finally, there was a potential issue in which the star that would be placed in the dungeon would spawn with a room interior arrangement that would render the star impossible to pick up and hence leave the level impossible to complete. This is a significant issue but one rare enough to never appear in the actual tests before the bug was fixed.

The first question asked of the participants was to rate how “fun” they found playing through the pre-built level the first time on a scale from 1 to 5. 5 out of 6 participants reported 4/5 for this question while 1 participant reported 3/5 for an average of 3.83. The second question shared the same premise but for the second playthrough of the pre-built level. Here, 3/6 participants reported a relative fun score of 2/5, 2 participants reported a score of 3/5, and 1 participant reported 4/5 for an average of 2.67. Nobody reported a higher score for the second playthrough over the first, though 2 participants rated the first and second playthroughs the same score. Otherwise, half of the participants gave the second playthrough of the pre-built level a score 2 below their initial playthrough while 1 reported the second playthrough as 1 below the first. From the data on average the relative “fun” amount of the second playthrough of the pre-built level will be scored 1.17 below the first.

The next questions asked participants to rate the first and second playthroughs of the procedurally generated level against each other on a scale from 1 to 5, much like the previous questions for the pre-built level. For the first playthrough of the procedurally generated level 4 of the participants gave a relative “fun” score of 3/5 and two participants gave a score of 2/5 for an average score of 2.67. For the second playthrough of the procedurally generated level, 3 participants gave a score of 2/5, two gave a score of 3/5 and one gave a score of 4/5 for an average of 2.67. Unlike with the pre-built level results, the second playthrough scores are reported by some of the participants to exceed the first playthrough. Overall, there is an average of no difference between scores of the first and second playthroughs in the procedurally generated level.

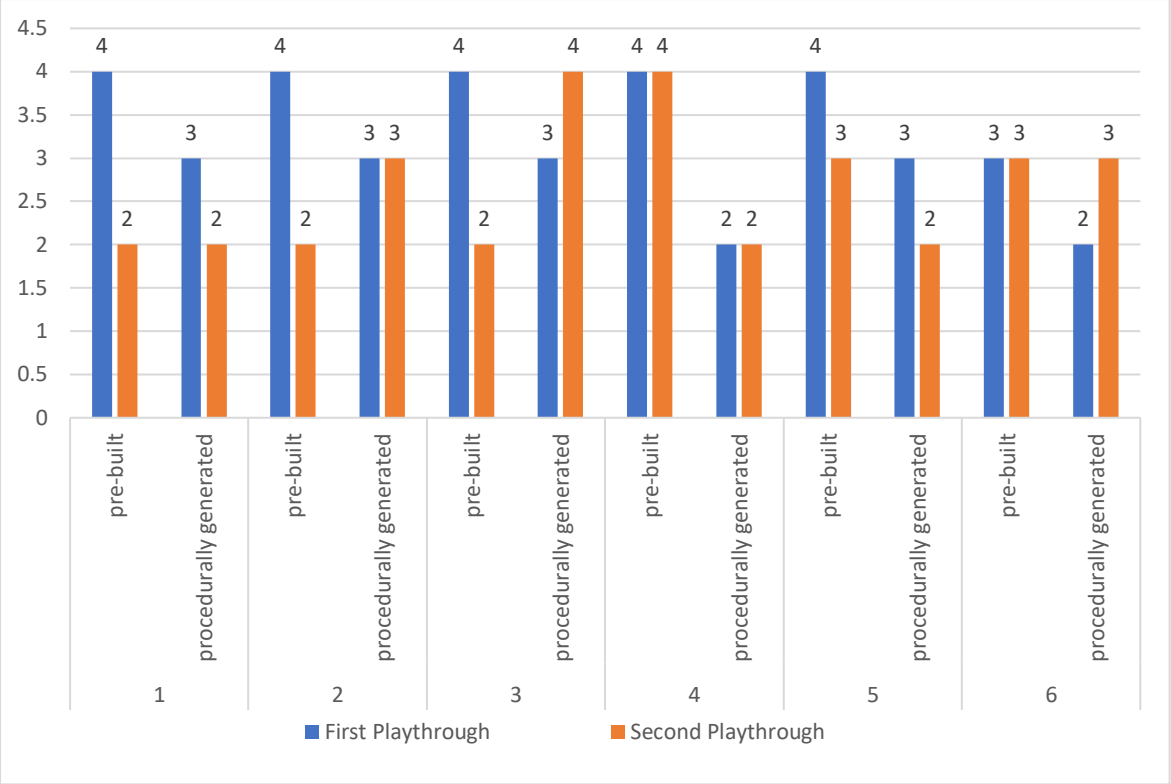


*Figure 9 Graph showing the difference in how each participant rated the first and second playthroughs of each level type*

The following question asked which of the two versions participants felt was more worth replaying. All participants answered the procedurally generated level.

The next question asked if replaying the pre-built level feel repetitive or offer new challenges. 4 of the participants expressed the lack of new challenges with one mentioning that they had been made familiar with the map layout and enemy placement from the first playthrough. 2 of the participants who mentioned the lack of new challenges would express that the level did not yet feel repetitive on only the second playthrough and would only get repetitive on even more playthroughs than what was tested. It should be noted that it was one of these 2 that reported no difference in the amount of fun they had on the first and second playthroughs of the pre-built level.

1 participant called the second playthrough of the pre-built level “slightly repetitive”, citing knowing the solution to the maze as part of this. But this participant does note the option of exploring different entrances and parts of the dungeon of successive replays. Finally, the last participant stated that there was in fact a new challenge presented on replaying the pre-built level, that being to beat the level as quickly as possible.

The final question directly asked to the participants was about What made the procedurally generated level more or less fun to explore. Participant 1 stated that there were too any enemies and that their spawns were unfair. Participant 2 stated that the level was more luck dependent, the dungeon is less interesting, but the maze is larger. Participant 3 stated that everything felt bigger and more challenging though once again noting that the dungeon layout was less interesting. Participant 4 stated that the maze was too complex and difficult compared to the pre-built level’s maze. Participant 5 stated that they liked the larger maze and how it has a different solution each time but again notes the large number of enemies. Finally, participant 6 noted how the difficulty seemed to be subject to randomness.

On top of the questions asked to the participants, they were also timed on how long it took to complete each of the 4 playthroughs that make up the test. A participant's times will not be compared to that of another participant. Everyone experiences games differently and how long a person took on a certain segment compared to another is not an indication on how they felt about that segment compared to the other. Times, however, can be used to compare the first and second playthroughs of each level type and compared against the other level type, so long as a participants times are only compared with times from the same participant.

| Participant | Level type | First playthrough time | Second playthrough time | Average time | Time difference |
|---|---|---|---|---|---|
| 1 | Prebuilt | 6:46 | 1:27 | 4:07 | -5:19 |
| | PCG | 3:43 | 3:31 | 3:37 | -0:12 |
| 2 | Prebuilt | 6:07 | 1:28 | 3:47 | -4:39 |
| | PCG | 2:51 | 2:02 | 2:27 | -0:49 |
| 3 | Prebuilt | 4:25 | 1:34 | 2:59 | -2:51 |
| | PCG | 2:18 | 3:57 | 3:07 | +1:39 |
| 4 | Prebuilt | 4:10 | 2:01 | 3:05 | -2:09 |
| | PCG | 7:45 | 5:04 | 6:24 | -2:41 |
| 5 | Prebuilt | 5:41 | 2:02 | 3:52 | -3:39 |
| | PCG | 3:57 | 5:30 | 4:44 | +1:33 |

Due to issues with recording data, participant 6 does not have times for their playthroughs and will be ignored for this area of results.

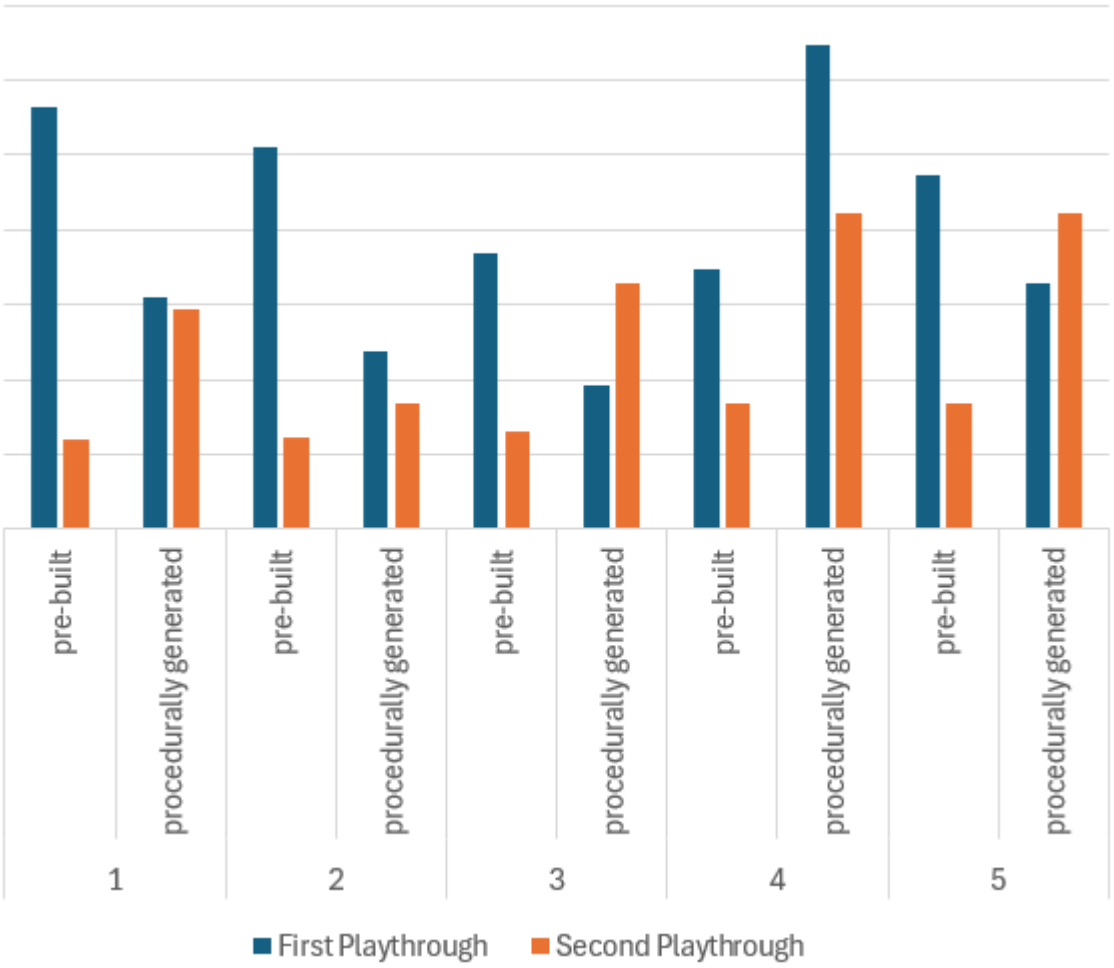


*Figure 10 Graph showing the difference in how long each participant took to complete the first and second playthroughs of each level type*

Overall, for most participants the procedurally generated levels took less time to complete than the first pre-built level playthrough but more time than the second pre-built level playthrough. On average, participants completed the second playthrough of the pre-built level 3 minutes and 43 seconds faster than the first playthrough. However, on average participants completed the second playthrough of the procedurally generated level only 6 seconds faster than the first, with a large amount of variance.

While it is of note that the first playthrough of the pre-built level might have extended times due to it being the participants first experience with the game, the data confirms that subsequent playthroughs of the procedurally generated level does not significantly decrease playtime like the pre-built level does.

## Discussion and Analysis

Overall, the investigation has shown that on average people prefer the pre-built level over the level created with procedural content generation, owing to its more curated balancing and structure. The criticisms of the procedurally generated level that contribute to this preference include the poor balancing of enemy spawns and the number of enemies, the lack of complexity in the dungeon area and that the maze was too big. All of these factors are not inherent to procedural content

generation but are in fact the product of the limitations and oversights of the algorithm used in the investigation. While an expansion of the algorithm to allow for more complex dungeons would take an amount of time and effort, the number of enemies and size of the maze could be adjusted with relative ease. The fact that the pre-built level was received with none of the criticisms about balance supports Darren Grey's writing that procedurally generated content, specifically items and level, may have balancing issues if not polished by hand (Grey, 2017).

A possible criticism of this investigation could be that the balancing oversights negatively affected the reception of the procedurally generated level and that if the level had balancing more similar to the pre-built level, in terms of the number of enemies and size of the maze, then the two level types would have been met with similar enthusiasm. Despite this, it is doubtful that people would consider the quality and fun factor of the procedurally generated level to exceed that of the pre-built level, at least on the pre-built level's first playthrough as time spent balancing the procedural generation method is also time spent refining the prebuilt level. This is shown through the dungeons, in which the rooms and layout are much more complex and balanced in the prebuilt than in the procedurally generated level due to limitations in the generation algorithm. A more advanced algorithm could be implemented, indeed more complex dungeons of this style can be procedurally generated such as in Binding of Isaac (Binding of Isaac: Rebirth Wiki, n.d.), But the time and effort necessary to create this more advanced algorithm would be too much given the scope of the investigation.

It can be seen from the results that the second playthrough of the pre-built level was rated, at least by the participants, as similar to that of the procedurally generated level even with the latter's balance criticism. Many of the participants reported no new challenges with this playthrough and cited knowledge of the solutions to the maze and knowledge of other areas of the map layout and enemy spawns as reasons as to which this playthrough was seen as repetitive. Some of the participants stated that the pre-built level had not gotten repetitive yet as of the second playthrough, with one stating that the different entrances to the dungeon could serve as a way to mix up first playthroughs. It is likely then that after even more successive playthroughs of the pre-built level, once every dungeon entrance and alternate path has been exhausted by its players, that the theoretical ranking given to a playthrough the pre-built level would fall even more, markedly below that of the procedurally generated levels.

While the average ranking for the procedurally generated level was below that of the first playthrough of the pre-built level, there was no significant drop or change in how the level was rated on the second playthrough. This was most likely the result of the level being different in design to the first playthrough due to the random nature of the procedural content generation used to create the levels. Each playthrough offers a new solution to the maze and a new series of rooms to explore for the dungeon, even if the rooms in this dungeon are less unique in their own design, the random arrangement was found to be more preferable to replaying the pre-built dungeon several times. On top of this, despite some criticism towards the size of the maze, some people preferred it to the comparatively smaller maze in the pre-built level.

The times taken for each participant to complete each of the four playthroughs of the investigation confirm that replaying the pre-built level does offer a significant difference in how the level is played, with the second playthrough for all players tasking significantly less time to complete. The most probable explanation for this is that players know the solutions to the maze and how to find the star in the dungeon as quickly as possible which is confirmed by the participants' comments on the pre-built level and replaying it. On the other hand, there is no significant difference between the average time taken to complete the first and second playthroughs of the procedurally generated level. This shows that no such familiarity or strategy exists that the participants could form from the

first playthrough and take into the second. Despite this average there is some marked variance in the times of the two playthroughs. For one participant the second playthrough was two minutes faster than the first, and for another participant the second playthrough was two minutes slower. This wide range of deviations between playthroughs show that luck has a large factor into how difficult or complex the procedurally generated map for any seed. This fact is supported by some of the comments made by the participants.

There was 1 participant who stated that there was a new challenge in replaying the pre-built level, that of trying to beat the level as quickly as possible. This, while not officially encouraged by the prototype, does offer insight as to one of the reasons why people replay games, the intrinsic motivation of the gratification that comes from mastery over a given gameplay system, perhaps encouraged extrinsically by the game itself. For example, some games in the *Sonic the Hedgehog* series will include a time attack mode or grade players over their times (Sonic Wiki Zone, n.d.). And certain games in the *Metal Gear* series will reward players with items for beating the game without getting caught, encouraging both replaying the game to practise and beat the challenge, but also to make use out the reward (Metal Gear Wiki, n.d.). For games without explicit reward for mastery or for mastery beyond what the game rewards, the pursuit of proving a player's skill or knowledge becomes a Metagame. The competition of surpassing their own times / challenges or competing against other players is in itself a game that drives the replayability of the target game (Paez, 2020).

When each of the participants were asked which level type felt more worth replaying, each one answered the procedurally generated level. This question does not only evaluate their experiences about their four playthroughs but also how they will predict further paythroughs would go. The participants who responded that the pre-built level had not yet gotten repetitive and that, because of poor balancing, the pre-built was simply more fun, despite this experience believed that the procedurally generated level was more worth replaying. The most plausible explanation for this is that the idea that a person's experience with a game will be different and fresh is a significant contributor to what leads people to replay games in the first place. That, if given the choice rather than guided by the investigation, a significant amount of the participants would have chosen not to replay the pre-built level at all but would have done so for the procedurally generated level.

Beyond the self-imposed gratification that comes with replaying a game to the point of mastering its mechanics, reasons to replay games often include the idea of consuming a new experience out of a game. Whether this manifests as replaying an Rpg with a new playstyle or character build, replaying a story-based game to experience the choices and outcomes not made previously, players are drawn to experience that which has not yet been experienced in a game or any media. For this reason, procedural content generation offers infinite replayability. The results of the investigation, which state that each participant feels the procedurally generated level is more worth replaying even if they believe the pre-built level was more fun, show us that the idea that a game will offer a new experience, even if the experience is lesser than a curated one, is significant enough to be more appealing to replay than a game that does not offer any new experiences to replay.

While The methodology used in this investigation was sufficient enough to draw these findings and conclusions, there are some criticisms one could make with certain aspects of the investigation. The main issue was with the balancing of the procedurally generated level, which had an impact on the results. If this investigation were to be repeated, a control tester would likely be used to make sure the level types are of equal difficulty before the testing stage begins properly. Another change that would increase the value of the data received in the testing stage would be to expand the investigation to a higher scope. Some of the comments made by participants said that two playthroughs were not enough for the pre-built level to feel repetitive, so introducing more replays

would allow participants more time to draw their opinions. Another method would be to increase the extent in which the pre-built level uses proper level design methods and ideas, which could require a completely different approach to the artefact itself, perhaps introducing a 3D environment rather than a low resolution 2D environment. However, both of these ideas, while possibly yielding more accurate or significant data, extend the investigation far beyond the scope that is possible in this project.

# Conclusion

The primary aim of the project was to investigate how procedurally generated worlds can be used to increase a game's replayability. For this research was conducted into how developers use procedural content generation in games, one major facet of this is using procedural content generation to randomise the content in a game with the purpose of creating a unique experience for players, producing highly replayable gameplay. Procedural generation is used here since it would be impractical, or even impossible to hand craft enough varied elements and content to a game for it to feel endlessly replayable which is necessary for games that require replaying the game within its gameplay loop. Roguelikes often require completing certain content over and over again, requiring a unique experience each time to not feel stale. Competitive games that include the environment or other random factors, such as *Sid Meyer's Civilisation*, and request players play different matches within the lifespan of the game use randomised environments to make sure no two games feel exactly the same. Games with fully procedurally generated environments, such as *Minecraft* or *Terraria*, while may not include replaying the game as part of the core gameplay loop, use procedural content generation to create a completely unique experience for each player and each game, which in turn allows these games to never feel stale to players even on successive replays.

The replay value of a game is dependent on many factors. In some cases, nostalgia (GameCentral, 2016) or a fun core gameplay loop is enough to inspire a player to replay the game, sometimes the intrinsic motivation of mastering a gameplay loop is enough to inspire players to continue on (Cochran, 2025), maybe it's about getting the best time, or the best score, maybe this is a self-imposed challenge or a competition between other players (Paez, 2020). Sometimes the game itself encourages these replays, rewarding players for all the time spent mastering the game's mechanics and using these rewards to further incentivise playing the game again. Some online games are endlessly replayable because every match is different because every player is different. In story driven games replaying is inspired by alternative choices, other pathways, different dialogue options and different endings. Some players will be satisfied with one route, but others may play the game again and again until every piece of content has been consumed. But after this point, after every piece of hand-crafted content has been experienced, every reward unlocked, then there is no new content to the game that inspires them to replay the game outside of players going back because of nostalgia, or how engaging they find the core gameplay loop.

But use of procedural content generation allows developers to create games in which players cannot experience everything the game has to offer, leaving the player with a fresh experience each time they play. A completely unique dungeon or world helps prevent the game getting stale and repetitive, to the point in which the idea of replaying a game with a completely new world to explore and experience to have become more inciting to replaying the game than mastering the gameplay or other built in incentivisation would.

This comes with a warning, however. Procedural content creation can be overused. Too much randomness in a games experience can leave a game lacking in immersion, worlds like *Minecraft* and *Terraria* need cohesion and believability or else players will not see the unique experience as something to actually experience, but rather a product of calculations and randomness. Each procedurally generated world needs either a hand to polish the content or strict rules and guidelines that allow these worlds and levels to be generated in such a way to still be meaningful despite the randomness.

The use of procedurally generated worlds does not guarantee that the game would be more enticing to players or that it will not get stale eventually. Some people will get bored of the gameplay or will stop playing after a single playthrough. A procedurally generated world must be built to a set of rules and structure to work as the game's world, and a player could simply get bored of this structure.

Despite the issues with the difficulty balancing with the prototype that was test, which in of itself speaks to the difficulties that might occur with less than direct oversight over map creation that comes with utilising procedural content generation, most of the participants agreed that the procedurally generated level was more consistent with how fun it was over the pre-built level, which for most participants had a significant drop in terms of fun factor as a result of already knowing the layout and solutions to any challenges that it might provide.

Each participant, including those who both claimed that the pre-built level was more fun but also that they had not yet found the pre-built level repetitive, said that they felt that the procedurally generated level was more worth replaying, showing that despite the fact that more fun might be had replaying a familiar experience, more worth is found in a new experience, that procedural generated levels and worlds will always provide.

That said, procedural map generation is not the only method in which to increase a game's replay value. If a hand-crafted map is significantly more suitable for the game the developer is trying to make, then it may be better to employ a different method of encouraging players to play the game again.

Ultimately then, the Aim to answer the questions of how procedurally generated worlds increase a game's repalyability and the extent to which procedurally generated worlds encourage replaying these games has been achieved. The research is able to show that procedural content generation can be used to improve a game's replayability and the extent to which it does so; though there were issues with the primary testing between the difficult balancing issues, lack of scope in the testing that prevented some participants from forming developed opinions on level replays and the overall low amount of participants that were able to be tested, the extent to which these results reflect a general population's position on if procedurally generated world improve a game's replayability is unknown.

## Recommendations

Procedural content generation is a complex and versatile tool for game developers to use in both the development and gameplay of their games as a way to increase the playtime and reply factor. One primary reason people replay games is to have a new experience than last time, methods of facilitating this include creating a large enough world in which it would be difficult or unlikely for any player to experience all of the game's content in a single playthrough, another method would be to

create a completely randomly generated map or features so that each playthrough of a game is entirely unique. Both of these methods can be optimised or facilitated by procedural content generation.

For large static worlds procedural generation can be used to speed up development, creating large landscapes and placing small details so that the developers do not have to expend the time, effort and resources to do so themselves. For randomising the game to create a unique experience, procedural content generation can be used to change the layout of a game's map, items or even used to create a fully custom world in which any players experience would be entirely unique.

Various methods of procedural content generation such as noise functions, cellular automata, or Mixed initiative content creation are effective when being applied to games and can be layered together to customise the product further.

However, there are reasons to not use procedural generation. If using it to speed up game development, a developer should avoid spending more time and resources on the tool to help design a world than they would otherwise spend on the world itself.

For using procedural content generation to generate a completely randomised map, time and effort must be spent ensuring that there is no potential for errors or that the map does not generate with an unbalanced difficulty. Overall, if a more authored experience would result in a higher quality of game, then developers should consider using this authored approach rather than relying on procedural content generation to extend the playtime of a game. If developers still want to increase a game's replay value, then other methods to encourage this could be considered such as branching story paths or rewards for skill at the gameplay loop.

To further research, there could be an investigation of similar style to the one taken above, but on a larger scale, this could include using 3D maps to better incorporate common level design paradigms for the pre-built level and to make each generated level more unique. This investigation would also benefit from a larger scope, investigating reactions to a greater number of replays and allowing participants to rest rather than replaying the level immediately.

# References


Ashlock, D. & Kreitzer , M., 2019. *Evolving Diverse Cellular Automata Based Level Maps. Proceedings of 6th International Conference in Software Engineering for Defence Applications.* s.l., s.n.

Autodesk.com, n.d. *Procedural generation: Creating infinite algorithmic realities.* [Online]
Available at: https://www.autodesk.com/uk/solutions/procedural-generation
[Accessed 4 November 2025].

Berto, F. & Tagliabue, J., 2023. Cellular Automata. In: E. N. Z. a. U. Nodelman, ed. *Stanford Encyclopedia of Philosophy.* s.l.:Metaphysics Research Lab, Stanford University.

Bethesda Softworks, 2004. *The Elder Scrolls 10th Anniversary: Daggerfall.* [Online]
Available at:
https://web.archive.org/web/20040407020037/http://www.elderscrolls.com/tenth_anniv/tenth_anniv-daggerfall.htm
[Accessed 14 November 2025].

Binding of Isaac: Rebirth Wiki, n.d. *Level Generation.* [Online]
Available at: https://bindingofisaacrebirth.fandom.com/wiki/Level_Generation?dlcfilter=3
[Accessed 16 November 2025].

C64 Games, 2018. *The Sentinel.* [Online]
Available at: https://www.c64games.de/phpseiten/spieledetail.php?filnummer=836
[Accessed 13 Novmeber 2025].

C64 Wiki, n.d. *The Sentinel.* [Online]
Available at: https://www.c64-wiki.com/wiki/The_Sentinel
[Accessed 13 November 2025].

Cochran, M., 2025. *These Are The Hardest Challenge Runs To Try In Video Games.* [Online]
Available at: https://www.thegamer.com/speedruns-nuzlockes-best-gaming-challenge-runs/
[Accessed 9 February 2026].

Dawnosaur, 2023. *How Minecraft generates Worlds you want to explore.* [Online]
Available at: https://dawnosaur.substack.com/p/how-minecraft-generates-worlds-you
[Accessed 11 Novmeber 2025].

e Silva, R. C., Fachada, N., Códices, N. & de Andrade, D., 2020. *Procedural Game Level Generation by Joining.* Lisbon, Lusófona University.

GabeHasWon, 2022. *Modding Tutorial: World Generation.* [Online]
Available at: https://forums.terraria.org/index.php?threads/modding-tutorial-world-generation.47601
[Accessed 12 November 2025].

GameCentral, 2016. *The psychology of replaying games - Reader's Feature.* [Online]
Available at: https://metro.co.uk/2016/05/01/the-psychology-of-replaying-games-readers-feature-5850175/
[Accessed 9 February 2026].

Grey, D., 2017. When and Why to Use Procedural Generation. In: T. X. Short & T. Adams, eds. *Procedural Generation in Game Design.* Boca Raton: Crc Press, Taylor & Francis Group, pp. 3-13.

h8man, n.d. *NavMeshPlus.* [Online]
Available at: https://github.com/h8man/NavMeshPlus
[Accessed 27 December 2025].

Himsl, F., 2020. *Binding of Isaac: Room Generation Explained!.* [Online]
Available at: https://www.youtube.com/watch?v=1-HIA6-LBJc
[Accessed 16 November 2025].

Hughes, M. D., 2003. *Game Design: Article 07: Roguelike Dungeon Generation.* [Online]
Available at:
https://web.archive.org/web/20080612035939/http://kuoi.com/~kamikaze/GameDesign/art07_rogue_dungeon.php
[Accessed 5 November 2025].

Johnson, M. R., 2017. Meaning. In: T. X. Short & T. Adams, eds. *Procedural Generation in Game Design.* Boca Raton: A K Peters/CRC Press, pp. 301-313.

Kaiser, R., 2012. *Opinion: How Mass Effect challenged my definition of 'RPG'.* [Online]
Available at: https://www.gamedeveloper.com/design/opinion-how-em-mass-effect-em-challenged-my-definition-of-rpg-
[Accessed 14 February 2026].

Karth, I., 2015. *Age of Empires II: Age of Kings.* [Online]
Available at: https://procedural-generation.isaackarth.com/2015/12/09/134882754122.html
[Accessed 19 November 2025].

Karth, I., 2015. *Sid Meier's Civilization (1991).* [Online]
Available at: https://procedural-generation.isaackarth.com/2015/12/07/civilization.html
[Accessed 19 November 2025].

Kazemi, D., n.d. *https://tinysubversions.com/spelunkyGen.* [Online]
Available at: https://tinysubversions.com/spelunkyGen
[Accessed 25 October 2025].

kestrel, i., 2014. *Why Spelunky is the best game I've ever played.* [Online]
Available at: https://medium.com/@iznaut/why-spelunky-is-the-best-game-ive-ever-played-50cc265e37ab
[Accessed 25 October 2025].

Khatchadourian, R., 2015. World Without End. *Annals of Games*, 11 May, pp. 48-57.

Kowalski, J. et al., 2018. *Strategic Features and Terrain Generation for Balanced Heroes of Might and Magic III Maps.* Wrocław, Institute of Computer Science, University of Wrocław.

Lagae, A. et al., 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum,* 0(1981), pp. 1-20.

Liapis, A., Smith, G. & Shaker, N., 2016. Mixed-initiative content creation. In: *Procedural Content Generation in Games.* s.l.:Springer Cham, pp. 195-214.

LifeWiki, n.d. *OCA:Maze.* [Online]
Available at: https://conwaylife.com/wiki/OCA:Maze
[Accessed 15 November 2025].

Maher, J., 2011. *Akalabeth.* [Online]
Available at: https://www.filfre.net/2011/12/akalabeth
[Accessed 13 November 2025].

Metal Gear Wiki, n.d. *Stealth Camouflage.* [Online]
Available at: https://metalgear.fandom.com/wiki/Stealth_Camouflage
[Accessed 9 February 2026].

Minecraft Wiki, n.d. *Noise generator.* [Online]
Available at: https://minecraft.fandom.com/wiki/Noise_generator
[Accessed 11 November 2025].

minecraft.wiki, n.d. *World generation.* [Online]
Available at: https://minecraft.wiki/w/World_generation
[Accessed 5 November 2025].

Newgas, A., 2019. *Dungeon Generation in Diablo 1.* [Online]
Available at: https://www.boristhebrave.com/2019/07/14/dungeon-generation-in-diablo-1
[Accessed 19 November 2025].

Newgas, A., 2020. *Dungeon Generation in Binding of Isaac.* [Online]
Available at: https://www.boristhebrave.com/2020/09/12/dungeon-generation-in-binding-of-isaac
[Accessed 16 November 2025].

nomanssky.fandom.com, n.d. *Procedural generation.* [Online]
Available at: https://nomanssky.fandom.com/wiki/Procedural_generation
[Accessed 5 Novmeber 2025].

Oxford Learner's dictionary, n.d. *replayability.* [Online]
Available at: https://www.oxfordlearnersdictionaries.com/definition/english/replayability
[Accessed 14 February 2026].

Paez, D., 2020. *How "speedrunning" became an Olympic-level gaming competition.* [Online]
Available at: https://www.inverse.com/gaming/speedrun-meaning-definition-origin-gaming-coined
[Accessed 9 February 2026].

Parr, J. A., 2018. *Generating Mazes Using Cellular.* [Online]
Available at: https://justinparrtech.com/JustinParr-Tech/wp-content/uploads/Creating%20Mazes%20Using%20Cellular%20Automata_v2.pdf
[Accessed 20 December 2025].

Perlin, K. & Hoffert, E. M., 1989. Hypertexture. *Computer Graphics,* Volume 23, pp. 253-262.

Procedural Content Generation Wiki, n.d. *Rogue.* [Online]
Available at: http://pcg.wikidot.com/pcg-games:rogue
[Accessed 5 November 2025].

procedural-generation, 2015. *Dwarf Fortress.* [Online]
Available at: https://www.tumblr.com/procedural-generation/135866825581/dwarf-fortress-when-were-talking-about-procedural
[Accessed 8 February 2026].

Relogic, 2011. *Terraria/WorldGenerator.cs (Version 1.4.4.9) [Source code].* s.l.:s.n.

Retro Gaming Discourse, 2021. *Infinite Games - 90s Procedural Generation.* [Online]
Available at: https://www.youtube.com/watch?v=s6x2C6cGIRQ&t=98s
[Accessed 19 November 2025].

rubby, 2022. *How does Terraria world generation work?.* [Online]
Available at: https://terraria.wiki.gg/wiki/World_generation
[Accessed 12 November 2025].

Scratchapixel.com, n.d. *Using Perlin Noise to Create a Terrain Mesh.* [Online]
Available at: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise-terrain-mesh.html

Shaker, N., Togelius, J. & Nelson , . M. J., 2016. *Procedural Content Generation in Games: A textbook and an overview of current research.* s.l.:Springer.

Smith, G., Whitehead, J. & Mataes, M., 2010. *Tanagra: An Intelligent Level Design Assistant for 2D Platformers.* Santa Cruz, University of California Santa Cruz.

Sonic Wiki Zone, n.d. *Time Attack.* [Online]
Available at: https://sonic.fandom.com/wiki/Time_Attack
[Accessed 8 February 2026].

Spufford, F., 2004. *Backroom Boys: The Secret Return of the British Boffin.* Main ed. s.l.: Faber & Faber.

Tauber, J., n.d. *Akalabeth Main Game File.* [Online]
Available at: https://jtauber.github.io/game-hacking/akalabeth/main_game.html
[Accessed 13 November 2025].

Terraia wiki, n.d. *World generation.* [Online]
Available at: https://terraria.wiki.gg/wiki/World_generation
[Accessed 12 November 2025].

Terraria Wiki, n.d. *World.* [Online]
Available at: https://terraria.wiki.gg/wiki/World
[Accessed 12 November 2025].

Worms Knowledge Base, 2008. *Worms Armageddon manual.* [Online]
Available at: https://worms2d.info/files/Worms_Armageddon.pdf
[Accessed 19 November 2025].

Zucconi, A., 2022. *The World Generation of Minecraft.* [Online]
Available at: https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/
[Accessed 8 February 2026].