**An investigation into the possibility of using Procedural Level Generation to create more replayable 2D platformers**

Callum Myers

School of Design and Informatics

Abertay University

DUNDEE, DD1 1HG, UK

# ABSTRACT

With the games industry constantly progressing and expanding, many players are spending hundreds or even thousands of hours playing games. As games become more impressive and on a larger scale, some simpler game genres may be somewhat overlooked as they simply cannot offer the content and playtime that these larger games can. One such genre is 2D platformers, which are often created to give the player one or possibly two in-depth playthroughs before being put down since they do not offer much opportunity to change beyond the content released upon the game's release. This could be improved using Procedural Level Generation (PLG), where the level is created dynamically by algorithms and mathematical functions, as opposed to being restricted to levels created manually by a team of level designers. These pseudo-random levels can be changed on the fly, which could allow for platformer games that offer extensive replayability throughout multiple playthroughs.

**Aim**

The aim of this project is to investigate and analyze the potential benefits of procedural generation when creating levels for 2D platformers and consider how this could be used to improve the game’s replayability, then to use this information to develop and implement a level generator for a simple 2D platformer.

**Method**

Using the Unity game engine with C# scripts, an application will be created containing a simple player character controller, as well as a level manager which will use a variety of programming techniques such as Noise Functions and Markov Chains to procedurally generate unique levels for the player. There will also be an in-game menu that will allow the player to set parameters for the level generator, changing the way levels are generated according to the player's preference.

**Results:**

It is expected that the level generator will be able to create a large number of levels which are all different and unique, as well as always having a path from the player's spawn point to the end goal objective, meaning each level must be completable.

**Keywords**

Procedural Level Generation; Replayability; 2D Platformers; Perlin Noise; Markov Chains.

**1. INTRODUCTION**

In recent years, many game studios have started to put a focus on creating a longer gameplay experience for players. This puts pressure on development teams, particularly level designers, artists, and programmers, who must create a huge array of unique levels manually which is not only time-consuming but also resource-intensive, as each level needs to feel unique and challenging to maintain player interest over extended playtimes. This can significantly increase their workload, leading to longer development cycles and higher costs for game studios.

One solution to this problem is Procedural Content Generation (PCG), which refers to "the algorithmic creation of game content with limited or indirect user input" (Shaker, N., Togelius, J. and Nelson, M.J., 2016, pp. 30)

By using PCG, games can create a wide range of unique levels automatically, reducing the manual effort needed from designers and allowing for more content to be created in a shorter timeframe. For instance, in a study by Balim Alpay, it was found that creating a level using PCG took only 30 minutes on average, compared to the 60 minutes required for a manually designed level (Alpay, B., 2024, pp. 45) which is a good example of how the use of PCG could help reduce this workload and allow teams to turn their focus to other aspects of the game, such as gameplay mechanics or more impressive graphics.

PCG also offers a major advantage in terms of replayability. In games designed with a set number of manually created levels, the player’s experience tends to remain the same over multiple playthroughs. However, procedural generation allows for an almost endless variety of combinations. In the same study, a game with six levels designed by a human would only give the player a unique gameplay experience for the first playthrough. At the same time, the PCG version had the potential for 720 unique sets of levels (Alpay, B., 2024, pp. 41). With this huge variety of levels available, players will be able to play through the game many times and each run will feel different from the one before.

Despite its advantages, PCG does come with challenges, particularly regarding the balance between variety and emotion which only a talented designer could add by hand to the level. To address this, designers can implement PCG in two ways: “online” or “offline.” Online PCG creates levels whilst the player is playing through the game, such as generating level segments or “chunks” just before the player encounters them (Smith, G., 2014, pp. 4). This method works well for games where replayability is the main goal, but it may lack the specific design elements or narrative touches the designer could add. Offline PCG, on the other hand, generates levels before the game begins (Smith, G., 2014, pp. 4), either at the start of a new playthrough or even before the end of development. This approach allows for more human input, as designers can handcraft certain features or add custom elements like easter eggs or specific collectibles.

This paper will look to use an “online” approach since the primary focus is to demonstrate how the use of PCG levels can be beneficial in extending the potential playtime of platformers, and dynamically created levels seem like the most appropriate way to tackle this goal.

**2. BACKGROUND**

PCG began in the 1980s with games like Rogue and Elite as a solution to the limited storage capacity of older computers (Shaker, N., Togelius, J. and Nelson, M.J., 2016, pp 20). By procedurally generating content at runtime, developers could avoid requiring external files for levels, art, and other assets, significantly reducing file sizes.

With the improved storage in modern computers, game developers can instead focus on using PCG to improve gameplay in a variety of ways. One vital advancement is the use of procedural level generation where algorithms can create a huge number of unique levels, allowing players to experience something new in each playthrough. An example of a game that has achieved this is Minecraft (Mojang, 2011), which uses a combination of procedural generation techniques to create vast open worlds for the player to explore and has up to 18 quintillion unique possible world generations.

Procedural generation is most regularly used in sandbox or rogue-lite games, such as No Man’s Sky (Hello Games, 2016) or Enter the Gungeon (Dodge Roll, 2016). One genre that does not often utilise this potential however is 2D platformers. One of the few 2D platformers which attempt this is Spelunky (Derek Yu, 2008), and as a result, has many more levels than most games in the genre. However, as opposed to being entirely procedurally generated, it is a collection of pre-defined level "templates" that are stitched together semi-randomly by an algorithm. Whilst this does provide a larger volume of unique levels than could realistically be created manually by a team of designers, it offers significantly less variety than Minecraft, as it is restricted by how many templates the team can feed the algorithm to use for level generation.

**2.1 Current Work**

Minecraft (Mojang, 2011) is a 3D sandbox survival game that involves players spawning into a new world during each playthrough and is a strong example of the effectiveness of using PCG for level generation to create a vastly replayable game. Each world is created using multiple PCG algorithms, with the main ones being noise functions, randomization, and rules-based generation. (A Gamer Girl, 2024)

Spelunky (Derek Yu, 2008) is a 2D platformer that also utilizes PCG, however unlike Minecraft it is not entirely random. Each level in Spelunky first chooses from 1 of 4 different basic room choices, and from here it picks from between 8 and 16 templates available for each room. Finally, some of these templates have spaces within them which can allow for a random tile to be placed, for example, there may be an empty tile or block, or it may be an obstacle, but the type of obstacle is random, and when all these templates are stitched together randomly, it can create a huge range of potential level layouts. (Shaker, N., Togelius, J. and Nelson, M.J., 2016, pp. 64).

Looking at the differences between the two games and how they utilize their respective PCG algorithms, it is clear where their priorities are. Minecraft, with the use of noise functions and randomization, will allow for a much larger variety of potential level layouts than Spelunky, where there are still many different possible layouts however this is restricted based on how many rooms and templates the team has time to create to supply to the algorithm. If the level generator used the same algorithms as Minecraft used, this could allow for a much larger array of potential levels and allow the player to play through the game even more times before they encounter the same level twice.

**3. METHOD**

This project will be using a combination of the Unity game engine with a variety of C# scripts to create the software required for a practical demonstration of the ideas outlined in this paper.

The project will have a set of requirements and features, which include: a basic player controller that will allow the player to move the character horizontally and jump over gaps in the level, the ability to zoom in or out of the level to give the player a better look at the procedural level generator in action, and a menu which is available to the player allowing them to set parameters such as the maximum ground height of each level, the length of the level, the number of collectibles or enemies generated, and toggles to decide if the level should contain platforms in the air or gaps in the ground. Each level also has its own set of requirements which will decide if the level generator is effective. These include: every level should have a player spawn point and an end goal objective, there should always be a completable path between these two positions, and every level should look unique from the one generated prior.

The level generator will be built using two separate stages: Perlin Noise to create the terrain followed by a rule-based script that will randomly place obstacles, enemies, and collectibles throughout the level. This will be done using an online method of procedural generation since a key requirement of the project will be giving the player the ability to generate a new level on demand while setting their parameters for the generation. After researching available algorithms or techniques frequently used in procedural generation, a combination of Wave Function Collapse and Markov Chains is most appropriate.

Perlin Noise is a noise function that creates a smooth random noise and is appropriate since it creates a more gradual gradient than other noise functions. Using this gradient, the level generator will sample the noise map and use the value at a certain point to decide the height of the ground at a point in the level. The player will have the ability to choose the extremity of the height differences using sliders in the parameter menu, which will be combined with the noise to create fractal noise, causing the level generator to create more drastic differences in height with an increase in the amplitude setting, or more gradual if the amplitude is set to a lower value.

Markov chains are mathematical models that use the current state to decide on the likelihood of what the next state should be. This, for example, could mean that if the ground tile being generated is rock, this may increase the chance of the next tile being a gap in the ground, whilst decreasing the chance of this next tile being grass. This will be useful when improving the flow of the level, ensuring the levels do not look haphazard and too random, whilst also potentially allowing for the implementation of biomes if it may improve the project later down the line.

The Wave Function Collapse algorithm is a newer technique in procedural generation and uses sets of rules and constraints to generate content. In this project, it will be used to allow the player to set their parameters such as the level length, level maximum height and the number of collectibles and enemies spawned. It can also be useful to ensure all levels have a plausible path from start to finish or ensure certain aspects cannot be generated in certain situations, such as making sure collectibles do not spawn on top of enemies.

**3.1 Schedule**

|  |  |
| --- | --- |
| **Months** | **Tasks** |
| October to December | * Implement a basic level generator. * Implement player movement. * Allow players to generate new levels. |
| December and January | * Add a menu to edit level generation parameters. * Begin writing the dissertation. |
| January and February | * Test if all levels generated have a complete path to the goal. * Begin distributing for user testing and set up a feedback channel. * Fix any major bugs found during user-testing |
| February and March | * Evaluate user feedback and see if the project goal was completed. * Continue writing the dissertation. |
| March to Submission | * Finish writing dissertation |

**4. Summary**

There is an increasing number of games beginning to use Procedural Level Generation to create vast, open landscapes. This project hopes to take all the research and knowledge used for this, to apply it to a more niche game genre, 2D platformers, to investigate the possibilities of creating more replayable games in this area in the future. The aim is to research and implement an efficient level generator that could be used in a larger scale game to create an infinite number of levels, and drastically improve the replayability and potential playtime.

**5. REFERENCES**

A Gamer Girl (2024). *How Minecraft generate worlds? - Gamers Wiki*. [online] Gamers Wiki. Available at: https://gamerswiki.net/how-minecraft-generate-worlds/ (Accessed 6th Oct. 2024).

Alpay, B. (2024). *A comparison of procedural-generated and human-designed two-dimensional platformer game levels.* Available at: <https://gcris.ieu.edu.tr/handle/20.500.14365/5265> (Accessed 24th Sept. 2024).

Egana, B.M. (2018). *Procedural level generation for a 2D platformer.* Available at: <https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1156&context=cscsp> (Accessed 20th Sept. 2024).

Shaker, N., Togelius, J. and Nelson, M.J. (2016). *Procedural content generation in games.* Available at: <https://link.springer.com/content/pdf/10.1007/978-3-319-42716-4.pdf> (Accessed 29th Sept. 2024).

Smith, G. (2014). *Understanding procedural content generation: a design-centric analysis of the role of PCG in games.* Available at: <https://dl.acm.org/doi/abs/10.1145/2556288.2557341> (Accessed 19th Sept. 2024)