Evaluation of Procedural Generation to Create 3D Platforming Levels

Ewan Fisher
1900176

BSc (Hons) Computer Game Applications Development, 2023

School of Design and Informatics
Abertay University

Table of Contents

Table of Figures

Table of Tables

Table of Tables

Acknowledgements

I would like to thank my supervisor for this project, Gaz Robinson, for his consistent feedback and advice on any enquiry I had during the project. His feedback and examples proved incredibly useful and guided me throughout the whole development process. I'd also like to thank any lecturers who have supported my growth at Abertay – listening to my wildest questions, helping me find where I exist within the creative space, and pushing me to create things that genuinely expressed myself.

I'd also like to thank my friends for their love and support, continuously being there – cheering me on at my best moments and comforting me when I needed it the most. The experience and camaraderie of student life helped guide me throughout my years at Abertay – and my best work was achieved when I had people to share it with, collaborate with, or just chat and get distracted by.

Abstract

Context: Procedural Content Generation (PCG) has been used within games for decades and has led to the development of the roguelike genre, which in itself has become a vessel for a multitude of different games spanning a wide variety of gameplay styles.
Aim: However, PCG has had very little crossover with the 3D platforming genre, despite similar games utilising it in 2D. By developing and building a prototype system for a 3D platformer which utilises procedural generation, it could lead to new gameplay experiences for players.
Method: This project makes use of a system utilising Perlin worms and customised, rules-based generation that takes influence from both human-designed elements and artificially created layouts.
Results: This results in a system which effectively generates 3D levels that are successfully completable. User testing found that the levels were considered enjoyable to play, and the generated levels were as enjoyable to play as human-crafted levels of the same format.
Conclusion: The completed artefact showcases results in a new type of gameplay experience for platformers, showcasing a unique, endless gameplay pattern that is unattainable through standard level design. In addition, the final results provide a clear link between polished platforming controls and enjoyable level design – as audiences judge level design based on enjoyability of the gameplay experience as a whole.

Keywords

Procedural content generation, 3D platformer games, Perlin worms

Abbreviations, Symbols and Notation

Procedural Content Generation - PCG

Search Based Procedural Content Generation - SBPCG

Chapter 1 Introduction

1.1 Procedural Content Generation

Procedural Content Generation (PCG) is an algorithmic process used to computationally generate content at run-time. This process can be used within the field of video games to both create content at a much faster pace than possible through typical design, whilst also allowing for seemingly “infinite” game design for players using randomness. PCG has been used in a large variety of games in many genres, with games such as Minecraft (Mojang Studios, 2011) gaining extreme popularity through their randomly generated, seemingly infinite worlds; both adding variety to gameplay and creating content that would be impossible to design and store using standard systems.

One genre that has thrived through the use of PCG are “roguelikes”, named after the genre-defining game Rogue (Glenn Wichman et al., 1980). These games use PCG to generate top-down dungeon levels for the players which differ each time – allowing for each experience to be unique and to amplify the replay value for players. The roguelike genre has expanded over time, going beyond top-down tile-based dungeon crawling to crossover with a multitude of different genres. Examples of hybrid roguelikes include Risk of Rain 2 (Hopoo Games, 2020), a third person shooter; Spelunky (Derek Yu et al., 2008), a 2D Platformer.

One huge benefit of procedural generation is on the development side – as the games industry grows, the costs of game development are skyrocketing, and games are having longer and longer development cycles. Procedural generation allows the developers to provide players with far more content and replay value from a game through smart use of programmatic systems – rather than needing to design hours upon hours of content.

1.2 Procedural Content Generation in Platformers

Platformers are a genre of games in which the player has control of a character and is tasked with navigating them through an obstacle course of terrain by jumping and using other techniques. There are a huge number of facets to this genre, with some games focusing on 2D linear platforming – such as Sonic the Hedgehog (Sonic Team, 1991), whilst others focus on 3D, open-ended design with an end goal of collecting items, such as Banjo-Kazooie (Rare, 1998).

Another branch of platformers is the 3D linear platformer – providing linear obstacle courses that are more associated with 2D platformers in another dimension of depth and control. Primary examples of the genre include Super Mario Galaxy (Nintendo EAD, 2007) and Crash Bandicoot (Naughty Dog, 1996). Whilst games like Spelunky and Cloudberry Kingdom (Pwnee Studios, 2013) have tackled the concept of a 2D platformer using procedural generation, there has been very little

progress made towards utilizing these systems within a 3D environment. 3D Platformers have been having a resurgence in popularity in recent years, due to popular mainstream titles such as Super Mario Odyssey (Nintendo, 2017), and the rising success of independent titles in the genre such as A Hat in Time (Gears for Breakfast, 2017), showing that there is a desire to see games in this style expanded upon. Roguelikes and other games utilising PCG have also seen a surge in popularity, with the game Hades (Supergiant, 2020), which was named as Game of the Year by more than 50 publications. As such, there is merit to expanding upon both of these genres through creating a hybrid program.

1.3 Aim and Objectives

The aim of this project is to develop a prototype system for use in a 3D platformer which procedurally generates level design. In order to meet this aim, a set of objectives were devised:

- To review existing procedural generation examples in the industry
- To prototype systems which procedurally generate 3D terrain with a linear progression
- To build and implement gameplay mechanics and character controls for a 3D platformer
- To test the systems in order to improve level design
- To gather player data
- To analyse and evaluate the data

1.4 Research Question

One downside of procedural generation is that content can occasionally feel artificially created – where the aimless design of the procedural systems can create design that lacks the key components used to make the genre “fun.” The proposed question aims to explore how procedural generation can be fine-tuned and tailored to create designs that are repeatable but feature the same enjoyable traits as human-designed content:

How can procedural techniques be used to generate enjoyable levels for a 3D platformer?

Chapter 2 Literature Review

There are a multitude of methods for procedural generation which are applicable for different types of applications. The following section discusses existing research on these methods of generation within the industry, evaluating which procedures would be applicable for this project. This section also analyses platformer level design philosophies and character controllers through a cast study of industry examples to determine what enjoyable level design entails.

2.1 Custom Rules-Based Generation

A general and accessible method for PCG is to generate based on a defined set of rules. This method has been used in Spelunky to create 2D platforming levels – which would allow the system to be easily expanded into 3D. Spelunky procedurally generates a level by determining a start location, and building rooms to the left, right and beneath, until an end location is decided upon reaching the bounds of the map.

Figure 1: A visualisation of a generated level in Spelunky – showcasing the pathway through the level (marked in red). *(Kazemi, 2010), available at* http://tinysubversions.com/spelunkyGen/

In Spelunky, each level is built up of a grid of 4x4 rooms. The starting room is always placed on the top row – and as shown in Figure 1, the game then algorithmically builds out the mandatory pathway, generating rooms with 1, 2 or 3 exits. Upon reaching the bottom, if the game tries to

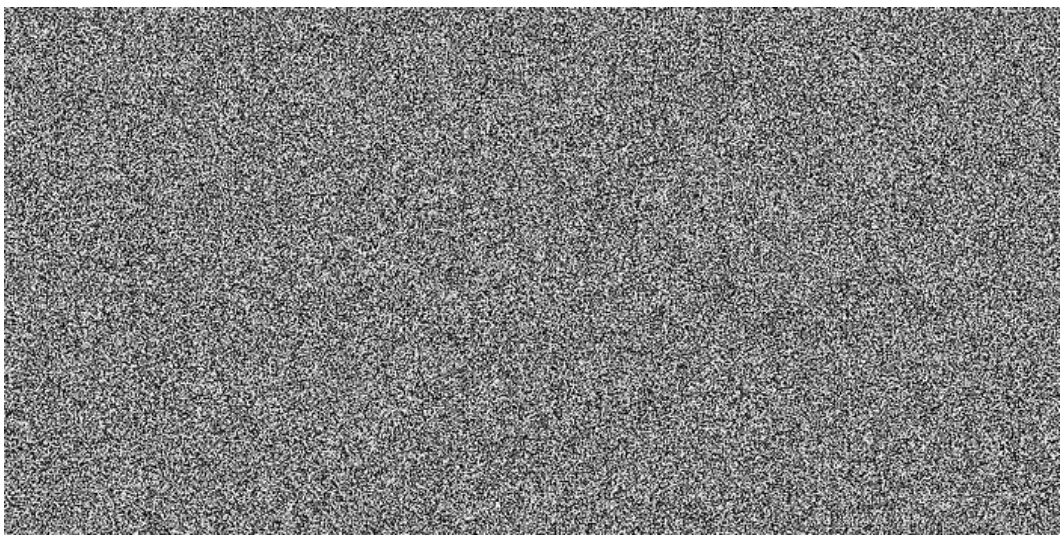
generate a room with a downward exit – it instead places the exit which leads to the next generated level. The remaining space is then filled with 0 rooms, that have no mandatory exits.

Each of these rooms are then randomised with differing obstacles in order to add variety to the platforming. This allows Spelunky to “generate human-feeling challenges” whilst still being random each time. (Kazemi, 2010). Whilst a 3D platformer would likely not use a “room”-based system, a similar system could be implemented by generating different sizes and shapes of platforms, with restricted lengths and sizes.

Another game employing a rules-based system specifically tailored to generating 2D maps is Dead Cells (Motion Twin et al., 2018). Lead Designer Benard discussed in 2017 the procedural methods used – which involved a hybrid of both hand-designed and programmatically constructed level design. By making use of themed biomes, it allows the game to have themed, narratively integrated challenges whilst maintaining the randomized elements. Whilst Dead Cells is a far more open ended game than this project intends to study, the lessons of using a hybrid of both types of design is valuable.

2.2 Perlin Noise

Perlin Noise is a pseudo-random texturing primitive, heavily used within computer games to create PCG. Developed by Ken Perlin, the primitive is used to generate a “pattern of randomness that is all at the same scale” (Perlin, 2002), meaning that a constant input into the Perlin function will produce the same randomised output (Ginting et al., 2019). Unlike traditional random algorithms, which have no coherent structure and simply generate random values irrespective of the values surrounding it – Perlin noise looks closer to patterns found in nature in order to generate smooth transitions between random values (Himite, 2021).



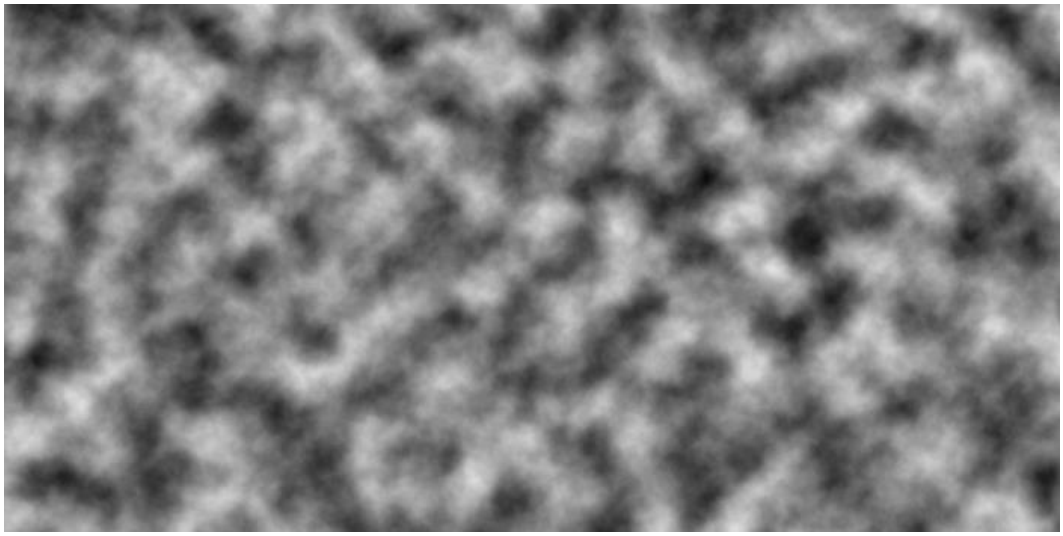


Figure 2: Visual comparison of completely random noise (above) compared to Perlin noise (below) *(Himite, 2021).*

As shown in Figure 2, Perlin noise allows for gradual and natural looking transitions between random values. Whilst originally created for realistic computer-generated textures, this has been extrapolated for a multitude of different randomly generated processes – most importantly being PCG for games, and as such can be extrapolated for use in generating natural feeling level design.

2.2.1 Perlin Worms

One of the most notable examples of a game making use of Perlin noise to generate a procedural world is Minecraft (Mojang Studios, 2011). When generating the caves and mountains in a Minecraft world – a variation of Perlin noise, known as Perlin worms, are used.

After the main terrain of the world has been generated, a sphere is placed at a random position. This sphere then turns – using Perlin noise to determine how far it should turn. The usage of Perlin noise allows these turning values to be more natural compared to turning in completely random directions – as seen in Figure 2. The sphere then moves forward a small increment – carving out the stone it overlaps with, replacing it with air.

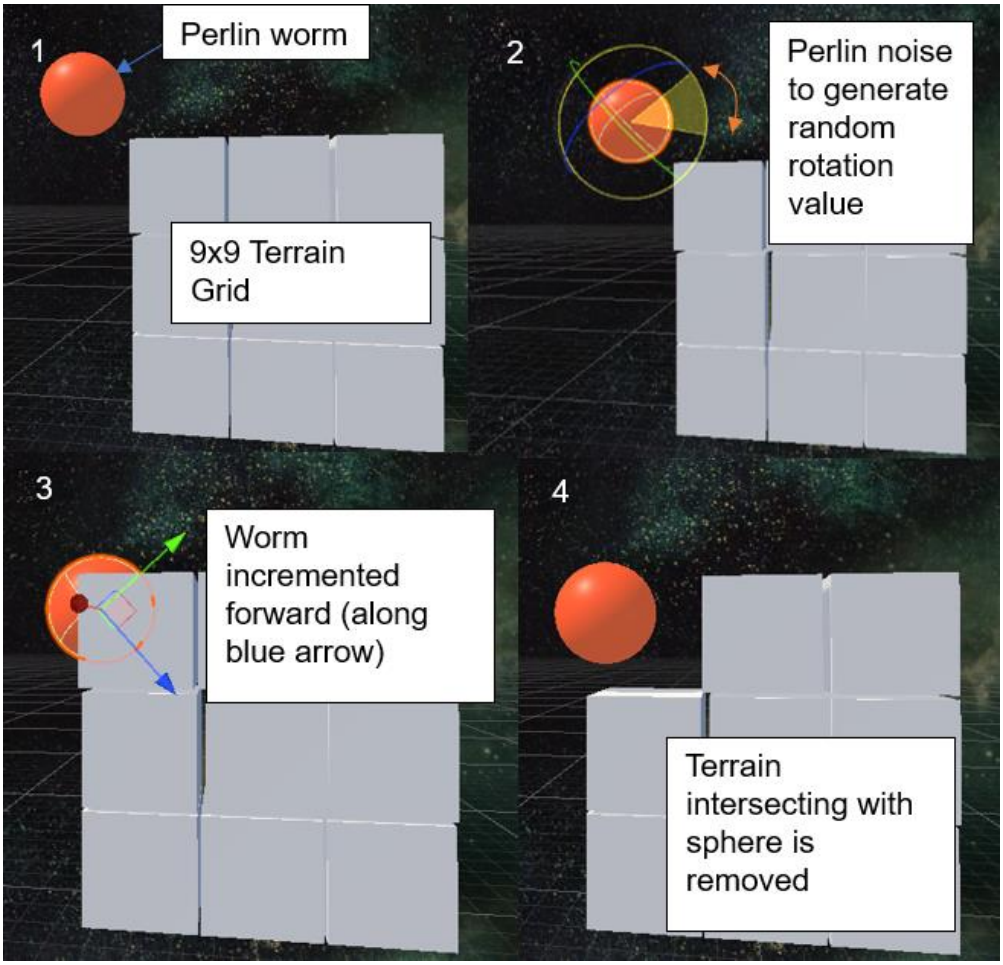


Figure 3: Diagram showcasing the process of using a Perlin worm to carve out generated terrain.

This process works in reverse for generating mountains, replacing air with stone. This method of generation could be used in generating natural feeling terrain that could be used for a platformer game.

Minecraft also makes use of fractal noise in the generation of its worlds – a variation of Perlin noise that stacks several differently sampled Perlin noise maps together in order to generate smooth, realistic feeling terrain. Minecraft specifically samples three fractal noise maps, in order to create terrain that is both realistic feeling but also has a wide array of variations. (Zucconi, 2022).

2.3 Level Generation Approaches

In addition to Perlin worms, there are a multitude of other industry standard PCG techniques that have been researched. Whilst Perlin worms are ideal for this project, due to the linear and smooth terrain generation lining up with the level design methodologies detailed in Sections 1.2 and 2.4.1 - it is worthwhile to discuss other potential methods of PCG, as they could prove relevant for future work.

One such method of PCG is Search-based PCG, a process that utilises artificial intelligence to calculate the fitness of a solution – how optimal the solution is at meeting the specified task (Togelius et al., 2010). The effectiveness of this fitness function directly correlates to the quality of the solution.

A subcategory of SBPCG worth discussing is Genetic Algorithms. This method is modelled on patterns of Natural Selection in nature, making use of a system of genetic operators in order to improve solutions over a number of generations and implementing systems of mutation and crossover in order to enhance the overall fitness of the solution. Mourato et al., 2011 analyses the use of genetic algorithms to create level design for 2D platformers. The system used involves a set of heuristic values used to calculate how appropriate a generated level was – including factors of completability and path structure. Baghdadi et al. in 2015 discussed using this method for the procedural generation of levels in Spelunky that utilised a difficulty and evaluation score to determine the fitness of the level. Whilst this study is promising, expanding the system into 3D would likely need to require huge adaptation, as both the inclusion of an additional access and the absence of traditional room-based design would result in issues. In addition, creating an objective fitness function to analyse subjective qualities of level design would likely result in level design that feels less enjoyable to human players.

Another algorithm for PCG that has increased in popularity in recent years is the Wave Function Collapse (WFC) algorithm. This method was discussed by Cheng et al., 2021, and defines a list of items with a set of rules, before generating a valid item for each position. If there is no possible valid selection for this item, the process is reset and regenerated until every item in the list is correctly generated. Whilst Cheng does provide significant improvements to the algorithm for use in games, adapting it to include multiple layers and global restraints, it is stated that this method would not work as effectively in 2D side-scrollers. As such, it would likely also struggle with 3D platformers, as terrain has to be built along a flat plane – leading to levels that would lack in verticality, one of the main traits of the genre.

2.4 Platformer Case Study

To answer the research question – a standard for enjoyable level design within a 3D platformer must be established. A case study was performed on several notable platformer games – to analyse the enjoyable qualities of these levels, and how feasible procedural generation of these levels would be. The case study also analysed platformer controls to determine the impact they have on how players enjoy the level design generated. The findings of this case study are shown below.

2.4.1 Platformer Level Design

The first game analysed was Super Mario Odyssey (Nintendo, 2017), due to its critical acclaim and offering of a wide variety of different platforming types and 3D level design ideologies. The primary level type in Super Mario Odyssey is a Kingdom – a large area with a lot of small objectives to complete. In these types of levels, the key component of enjoyability is the freedom the player has when exploring, and the intricacy of how each level is designed. Procedurally generating levels of this size and detail would be hugely out of scope for this project and would likely lead to levels that lack in focused direction.

Super Mario Odyssey also features smaller platforming challenges within its kingdoms known as “sub-areas”, which take the form of small, self-contained, linear platforming challenges. These sub-areas are often based around the exploration of the game’s mechanics, a good example of which being the sub-area “Breakdown Road”. This level is built around having a bridge for the player to platform across that is slowly destroyed by the “Bullet Bill” enemies. This challenge is well-designed, as it gives the player a multitude of ways to complete it, allowing them to stick to the top path, or capture a Bullet Bill to ride past the danger. The game does not present any one of these options as correct – it is up to the player’s own intuition to decide which route they take. The destructible platforms and enemies in this level act as “Dynamic Challenges” which involve the player adapting to the changing situation of the game to overcome them.

Another game with level design philosophies worth discussing is Super Mario 64 (Nintendo, 1996) - which similarly features a huge amount of variety in level design ideology. The first level, Bob-Omb Battlefield, has a wide space for the player to explore, whilst orbiting around a single focal point for the player to platform through – the central mountain. Whilst this style of level design is still quite open – it is a worthwhile endeavour to explore generation of this type of level design, as it could allow for procedurally generated levels that maintain coherent linearity whilst also featuring human-crafted landmarks.

Figure 4: A map showcasing the Bob-Omb Battlefield level, with an arrow added to show the main pathway.

2.4.2 Platformer Character Controllers

With regards to character controllers in 3D platformers, Super Mario Odyssey is very highly regarded. There is a wide array of actions at Mario's disposal, each chaining into the next. For example, the player can dive once to obtain further horizontal distance whilst airborne – which can be combined with a bounce from Mario's hat to create another jump whilst airborne. The consistency of this character controller lets players predict exactly how Mario will move and lets them solve the obstacle courses in a huge variety of ways. The consistency of the controller ensures that the player can effectively answer any platforming challenge without worrying about not successfully landing their jumps.

Whilst a character controller with this extreme level of depth is hugely out of scope for this project – it is a good inspiration for the direction to take the project. As movement itself in Super Mario Odyssey is intrinsically fun; from this it can be said that having a well-constructed character controller will naturally lead to more enjoyable levels as it will provide the player with further free choice in how they complete each level.

2.5 Summary

After looking over current literature on the project, it seems that using a system which makes use of Perlin noises, specifically Perlin worms,

would be most fitting for a project such as this. Perlin worms have been tested for generating linearly explorable areas in existing games such as Minecraft and combining them with a custom ruleset for procedural generation as seen in Spelunky would allow the generated levels to maintain the enjoyable aspects of traditional level design – such as dynamic challenges and elements of player creativity.

Chapter 3 Methodology

3.1 Overview

In order to answer the research question, a games application was developed using C# and the Unity3D engine with the aims to showcase the potential of 3D, procedurally generated, linear platforming levels. The program uses Perlin worms to generate a pathway from a central point outward, continuously generating and adjusting positions in order to create a functional level layout. Following this, spline-based platforms, platforming challenges and obstacles are placed along. The player navigates these using a simple platformer character controller. The software also features a tutorial so the player could get used to the controls without being inhibited by the random level design. Including this also helps with testing – as it allows the tester to thoroughly separate the level design and character controller components and more easily articulate distinct feedback on both.

3.2 Application Design

After writing a case study and analysing the literature review, a few factors were identified for both what makes functional and enjoyable platformer level design:

- Levels must have both a visible start and end point.
- Levels must have a mostly linear direction in order to prevent to avoid aimless levels.
- Levels must have gaps of varying sizes for the player to jump between.
- Levels must be completable, not having gaps that are too high or wide for the player to jump across.
- Levels should have multiple ways to complete each platforming challenge.
- Levels should include dynamic challenges of some kind – such as platforms that move or rotate.
- Levels should have some form of obstacle for the player to avoid.
- Levels will preferably have varying levels of verticality to make full use of the 3D space.

These factors were used to create a set of rules that the procedural system.

3.3 Level Generation

The level generation process was split up into a variety of different processes – beginning by using Perlin noise to generate the pathway of

the spline, before using more specific, customised rules to generate the platforms and obstacles.

The generation can be broken down into separate parts. This section is a breakdown of the process in simplified terms, before each section will go into further detail:

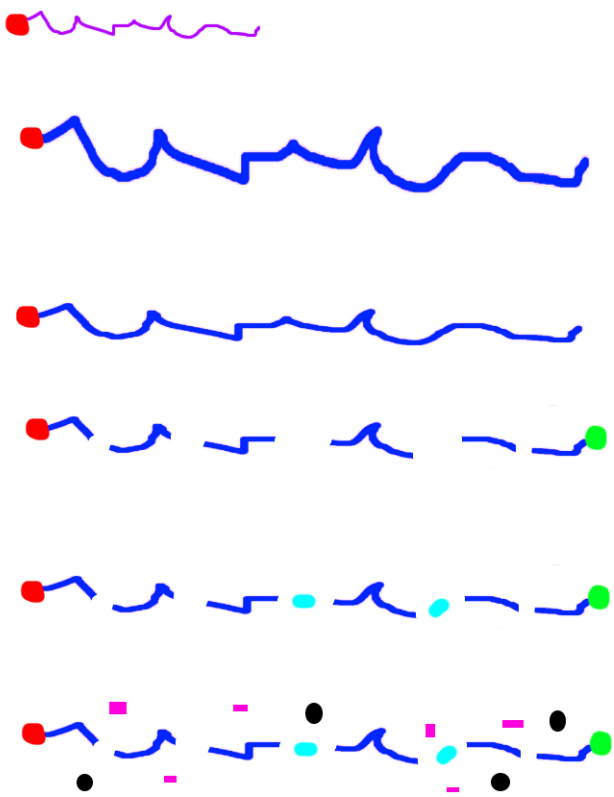


Figure 5: 2D Diagram showcasing the stages of generating a level, using an example Perlin Worm generation. Each of these stages is explained in further detail throughout this section.

The first step involves creating a Perlin worm at the starting point, and having it travel through a set number of points in the scene.

Figure 5.1: Diagram showcasing the a sample pathway from the world origin (red) of a Perlin Worm (trail represented in Purple).

After this, the transforms of the worm are then sampled – and used to generate a spline. This spline is amplified horizontally and vertically, to

better fit with the expected length of a platformer level. If the level generated is too tall, the level is then clamped to fit within a shorter height in an attempt to prevent levels that are too vertical for the player to jump through.

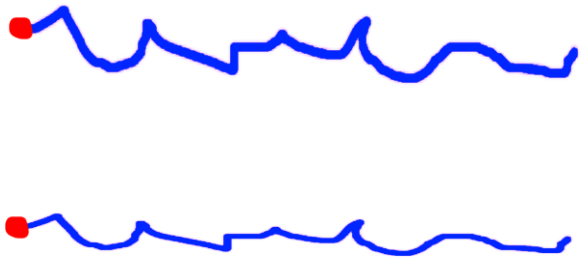


Figure 5.2: Diagram showing the transforms of the Spline (blue) after initially being sampled from the Perlin worm (above) and after being reduced to fit within an appropriate height (below).

Once the final points of the Spline have been decided, the terrain has to be generated. This terrain is split up into smaller sections of the Spline that become the platforms the player jumps between. At this point, the end of the level is determined, and the end goal is placed.

Figure 5.3: 2D Diagram showing the spline as it has been segmented and the terrain has been generated. The End Goal (green) has been placed.

After each segment is generated – a check is performed to determine the distance between the gap left in between. Depending on the size of the gap, a Dynamic Obstacle (such as a moving platform) is placed in the empty space.

Figure 5.4: 2D Diagram showing the extruded Spline, with Dynamic Obstacles (cyan) placed in the appropriate gaps.

To finish the process, static obstacles are instantiated along the pathway of the Spline – each having their position, rotation and scale randomised. These static obstacles include stationary platforms that the player can jump on, and spike ball that can damage the player.

Figure 5.5: 2D Diagram showing the final generated level, with static platforms (pink) and spike balls (black) placed.

3.3.1 Pathway Generation

For generating the level pathway, Perlin worms are used. This method was chosen due to its proven success with creating linear terrain in Minecraft. As Perlin noise is best at creating natural feeling generation – using it here allows for natural feeling levels with smooth curves, rather than levels generated with sharp turns and breaks between platforms. This program essentially adapts the existing process used to generate caves in Minecraft, placing terrain along the path of the Perlin worm to create a linear structure.

In a 3D scene, a Perlin Worm is placed at the origin, proceeding to generate a set number of points outward that roughly indicates the level length. This uses Perlin Noise to determine an angle at which the worm should turn, then moving forward in small increments. These worms are limited to only being able to turn either 90 degrees to the left or the right, in order to not make the worm curve back in upon itself. In terms of level design, this forces the levels generated to be linear in nature, exclusively travelling outwards from the centre point.

Figure 6: Vi*sualisation of the Perlin Worms with limited turning capacity in Unity Engine. Adapted from Perlin Worm code by Gaz Robinson.*

3.3.2 Spline Generation

The positions that the worm passes through are then sampled by the Pathway Generator as control points or “knots” in a Bezier curve – from which a spline is created. Whilst we could simply place platforms on each of the control points, this would create a very block-y, grid-based level. Using Splines allows us to generate terrain that feels natural, and human created. In addition, Splines in Unity are easily able to be extruded, which will be used in the following step to generate the level geometry.

For each point that is sampled from the Perlin worm, adjustments must be made to better tailor the scale of the level. If the raw positions of the Perlin worm were used to generate the Spline, the level geometry generated would be far too short horizontally, and too vertical in contrast – creating levels with odd layouts. As such, the X and Z values of each are multiplied by 3x, in order to give the level more horizontal movement rather than vertical, and to extend the length of the level. The Y values are each multiplied by 1.5x - this adds verticality to the platforming.

For each point adjusted, the program keeps track of whether it is the highest or lowest point generated. The lowest point generated is used to act as a “death plane” for the player – if the character falls below the lowest point, then the player has lost the level.

The highest and lowest points are also used to determine the height of the level. If the level is too tall, then each of the points must be scaled down vertically, so that no jumps are too high for the player to platform up. This is done through the following calculation:

$$f = \frac{d_m}{h - l}$$

Where:

- f Represents the scale factor by which the level should be adjusted.
- d_m represents the maximum distance (or height) that the level can be vertically. In this artefact, the value used is 100.
- h And l represent the highest and lowest points of the level respectively.

Now that the finalised level pathway has been calculated, the end point of the level can be placed at the final point of the Spline. This is similar to the method used in Spelunky seen in Figure 1, where the 2D map with rooms going downwards from the top is replaced with a 3D map with points going outwards from the world origin.

3.3.3 Level Geometry Generation

As the pathway of the level has been calculated, the level geometry from the spline can be extruded. This is done using the SplineExtrude component of Unity's Spline extension. Extruding the spline currently calculated would generate one long platform – and as such the Spline must be segmented into smaller sections to act as platforms that the player can jump between.

The SplineExtrude component allows the program to generate a specific range of the spline. The following algorithm is used to extrude the level geometry:

1. Randomly generate a number for how many segments the level will be divided up into
2. Calculate the percentage of the level each of these segments takes up as 100 / number of segments
3. Loop through each segment:
    a. Create a new SplineExtrude component
    b. Set the container to use the spline calculated in 3.3.2
    c. If the segment generated would go over 100% of the spline, or is the last segment:
        i. Set the range of the new SplineExtrude to the end of the level
    d. Else:
        i. Generate a random value within a range of 3 to 12
        ii. Set the start point of the SplineExtrude to be the percentage of the current segment of the spline
        iii. Set the end point of the SplineExtrude to be the percentage of the next segment of the spline, subtracting the random value

3.3.4 Dynamic Challenge Generation

There are two passes for obstacle generation in the program. The first generates dynamic obstacles, as established in the case study. The dynamic challenges chosen to generate were moving platforms that move between the segments of the level. Moving platforms were chosen as they act as a solution to gaps that are two large for the character to jump between, decreasing the likeliness of generating a level which is incompletable. They were also chosen as the track between them clearly indicates to the player which direction to go in the level, streamlining the process. Due to the scope of this project, moving platforms were the type of dynamic challenge included – further development time could allow for more types to be included, such as falling or bouncy platforms.

These moving platforms are generated to go exactly between the gap from two segments of the spline, so the player always has a route to the next segment of the level. These platforms use the same points calculated in the previous section and are simple cuboids. A line is rendered between these two points to act as a visual track that the platform is moving along, to help the player correctly gauge the distance and not overshoot or undershoot the platform and fall off.

A second pass is done to place static obstacles – such as the spikes and the floating cubes that the player can jump on. This is done using the SplineInstantiate component, as part of the Unity Spline extension. These follow the pathway of the spline, being placed at random intervals. The floating cubes are placed specifically so the player can use them as additional platforms, adding some choice in how the player wants to approach each jump. This also adds an additional fallback in case the gap between two segments of the spline is too big – increasing the amount of platforming required and making it much less likely to generate a level that is incompletable. The spike balls are generated to provide an obstacle for the player to avoid, offering some sense of challenge.

3.4 Character Controller

When designing the character controller for the game, two elements were prioritised:

1) Having a character with consistent behaviour to effectively answer any platforming challenge
2) Having a wide enough moveset that allows platforming challenges to be tackled in multiple ways

In order to meet these two requirements, a bespoke character controller was developed using Unity's Character Controller component as a base.

This was done to avoid issues with physics-based variance – ensuring that the character has a consistent jump height and movement speed. Doing so gives the player the tools to reliably navigate the levels without having to worry about the physics system.

The program was intended to be designed, in concept, by building the character controller first, and then building the levels around the skillset and abilities of the character. In execution, it became more reasonable to focus on the procedural generation to generate sensible levels, and then add functionality to the character which meets the level specifications.

3.4.1 Character Controller – High Jump

The character was given a high jump move, with significantly more vertical motion at the expense of horizontal motion, in an attempt to counteract level geometry with extremely steep verticality. Whilst the level generation algorithm could have been altered to prevent geometry such as this – it proved more beneficial to keep the steep terrain as it added significant variety to actions performed by the player, and it could be seen that variety is a component of an enjoyable level.

Figure 7: A screenshot showcasing steep verticality that the player may wish to overcome with a high jump.

3.4.2 Character Controller – Stomp

The character was given a Stomp move primary to act as an immediate and responsive way for the player to ground themselves. This move was implemented to act as a counterbalance between having both significant distance and control in the air. In order to accommodate for varying level design, the character was given a “floaty” jump, where the player has a large influence over their direction in the air. The gravity used for jumping is quite low, giving the player the ability to bounce in very wide, high arcs. The Stomp allows the player to immediately end a jump immediately, quickly move towards the ground and avoid overshooting any terrain. This move cancels forward momentum from the character – so matter which direction the player is travelling in; they will always move directly down from their position.

Specifically, it was implemented to counter-act moving platforms – which were seen as a necessary inclusion in order add dynamic variety to levels, and bridge generated gaps that are too wide for the player to normally jump between.

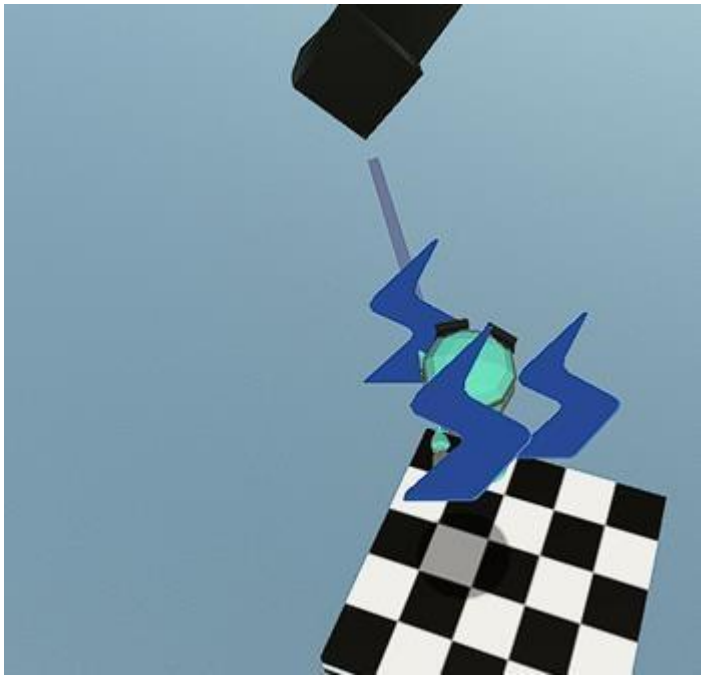


Figure 8: A screenshot showcasing the player using the Stomp to land on a moving platform.

3.4.3 Character Controller – Dive

The character was given a Dive move, allowing them to surge forward at the cost of locking their forward momentum for a few seconds. This essentially acts as a horizontal equivalent to the Stomp move, however it is slightly less restrictive as it is harder to predict the exact forward trajectory compared to downwards trajectory.

This move was added as it synergises with the High Jump move. Whilst the High Jump has very limited horizontal movement, a skilled player could use the Dive at the apex of this move to obtain both good vertical and horizontal movement. In addition to this, it also acts as a further solution to large gaps generated by level generator.

Adding a form of movement variety that allows players to complete levels faster pairs well with the endless nature of the procedurally generated levels. Players will naturally want to make it through these levels faster to see how many levels they can beat without losing.

Figure 9: A screenshot showcasing the player using the Dive move.

3.4.4 Character Controller – Shadow

The character was given an Imposter Shadow that is cast irrespective of Unity's lighting engine and is always visible directly upon the platform that the player will land on.

This shadow is a 2D sprite attached to the player object. Each frame, a raycast is performed down from the character's position. If the raycast hits the ground, the shadow sprite is set to the hit point, translated slightly above as to not clip inside the terrain. The up vector of the sprite is then rotated to the normal of the terrain – so that the sprite will always match the angle of the terrain below. The size and transparency of the shadow is adjusted based on the distance from the hit point, so that the shadow will appear more faded the higher up in the air that the player is.

This was done to help players be more accurate with their jumping, and to provide further accuracy to the Stomp move, so players could securely land upon moving platforms.

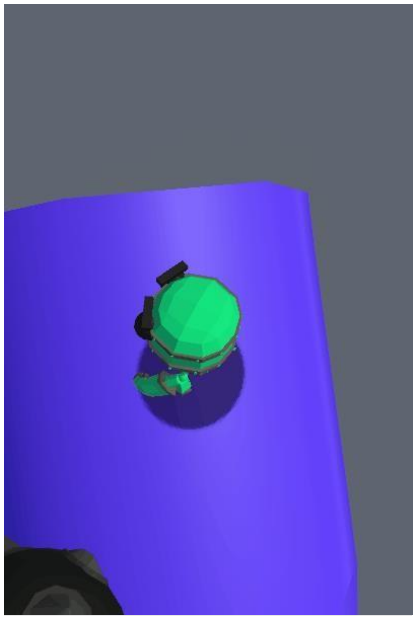


Figure 10: A screenshot showcasing the player's shadow.

3.5 User Testing

In order to analyse the effectiveness of the game at answering the research question, the software was user tested. Users were asked to play a tutorial, in order to make sure they understand the controls, before being asked to play three levels:

1) A level that was human-created – inspired by a level from Super Mario 64. The level recreated was "Bowser in the Dark World", chosen as its linear nature resembles the style of levels analysed in the case study.

Figure 11: Comparison of the level layout from Super Mario 64 (above) with the recreation used for testing (below)

2) A level that was procedurally generated from a seed – to act as a control variable, so all testers would play the same generated level.
3) A level that was procedurally generated completely at random - so that each player would have a level completely unique to them.

Following this, testers were asked to rate the levels out of 5 for how enjoyable they were. Testers were not told which levels were procedurally generated, and which ones were human-crafted, in an attempt to remove subjective, personal bias.

Additionally, players are asked if the character controller had a negative impact on their enjoyment of the game. This is to cleanly separate bias of the level generation to the character controller and more clearly gather distinct results on each.

These testing results will then be used to evaluate how successful the artefact is at meeting the research question. The minimum viable product would be to generate functional, completable 3D platforming levels. By getting testers to rank both human-made and algorithmically generated levels in a series of different metrics, a clear comparison can be made as to how enjoyable the levels are.

Chapter 4 Results

A factual presentation of the results from the questionnaire is shown below.

4.1 Survey Data

The survey consisted of 5 questions:

Question	Response Type
I found the first level (following the tutorial) enjoyable. How much do you agree with this statement?	Linear Scale 1 – 5 of how much the participant agreed
I found the second level (following the tutorial) enjoyable. How much do you agree with this statement?	Linear Scale 1 – 5 of how much the participant agreed
I found the third level (following the tutorial) enjoyable. How much do you agree with this statement?	Linear Scale 1 – 5 of how much the participant agreed
Did you find that the controlling the character was difficult?	Yes - the character controller got in the way of enjoying the game **OR** No - the character controller was not an issue
Is there any specific features you feel like would add to the gameplay? (And/or any bonus comments)	Open text comment

Table 1: Survey Questions

In order to analyse the effectiveness of the game at answering the research question, the software was user tested. Users were asked to play three levels:

I found the first level (following the tutorial) enjoyable. How much do you agree with this statement?

12 responses

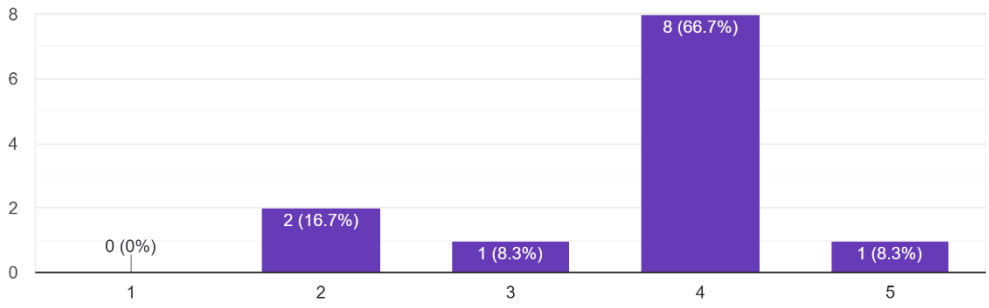


Table 2: Question 1 Responses

I found the second level (following the tutorial) enjoyable. How much do you agree with this statement?

12 responses

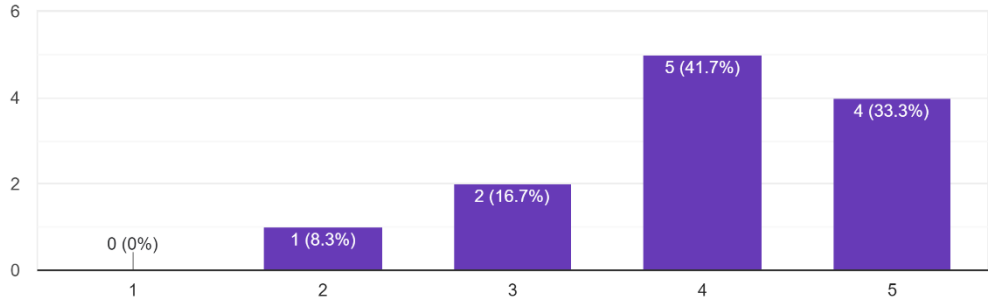


Table 3: Question 2 Responses

I found the third level (following the tutorial) enjoyable. How much do you agree with this statement?

12 responses

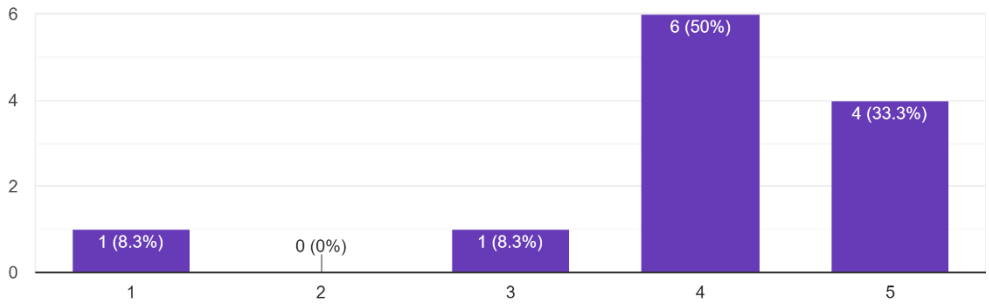


Table 4: Question 3 Responses

The average responses for these questions can be shown as such:

Level	Average Response	Standard Deviation	Variation
Level 1 – Human-created	3.666666667	0.88762536	0.56363636
Level 2 – Generated from Seed	4	0.95346259	0.90909091
Level 3 – Generated completely randomly	4	1.1281521	1.2727273

Table 5: Average of Responses from Questions 1-3

Testers rated the levels very equally in terms of enjoyment – only ranking the human-created level as slightly less enjoyable. The randomly completely randomly generated level had the most variation in responses, whilst the human-created level had the most consistent responses from testers.

Testers were asked to rate the difficulty of controlling the character:

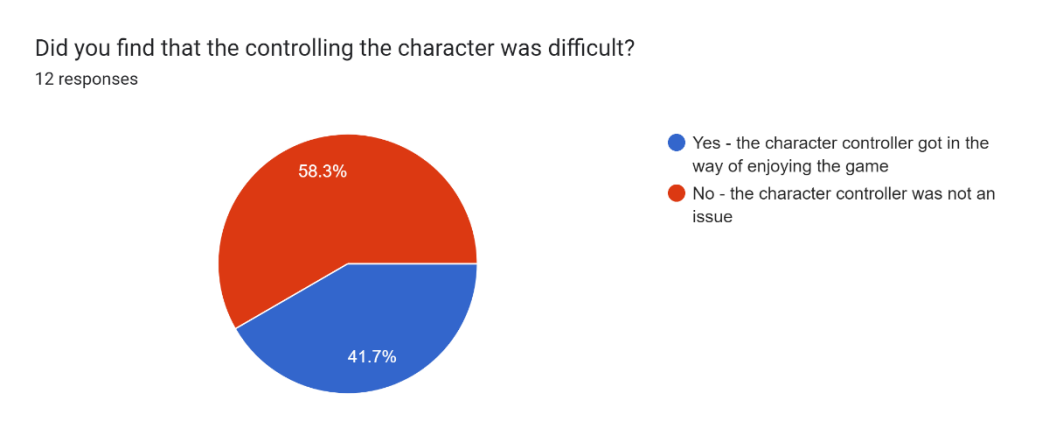


Table 6: Question 4 Responses

58.3% of testers stated that the controls did not impede gameplay, whilst 41.7% of testers found that the controls were an issue. The implications of these results are discussed further in Section 5.2.2.2.

9 of the survey respondents left open text comments:

Tester No.	Open Text Comment

1	Perhaps a marker which communicates to the player where exactly they will land after jumping. Other than that very cool!
2	Character controls were mostly solid, but the air dash felt more restricting than I think it was intended
6	I thought that the gameplay was very good and intuitive. It was very easy to understand the controls and how to play the game.
7	Stronger gravity, checkpoints
8	Inverted camera controls please…
10	Smoother camera when falling off of ledges
11	Slightly more varied levels would be good, to encourage the use of the different abilities more… it might feel a bit batter if [the controls] were slightly less slidy in the air.

Table 7: Relevant Question 5 Responses

Table 7 only includes results relevant to the discussion in Section 5.2.2.3. These comments cover a wide range of topics – such as improving the character controls, camera, and level generation. 75% of testers left optional responses, showing a high level of engagement with the artefact. A full list of responses, including averages for each tester's enjoyment of the levels, can be found in the Appendix.

Chapter 5 Discussion

This section is dedicated to critically discussing and evaluating the strengths of the project and how effectively it was able to answer the research question, using the data gathered from playtesting, and reflect on the techniques and processes used.

5.1 Initial Playtesting

The game was tested at multiple intervals, with final testing being performed to obtain the priorly mentioned results.

Initial testing was performed in order to gauge user feedback regarding issues with the player controller and level generation. This initial testing led to further work on the character controller, leading to the decision to change away from a Rigidbody-based character. This was directly in response to testers finding the original control scheme difficult to work with. Whilst Rigidbody physics could be suitable for some platformers – in a case where level design is the key focus of the project, having bespoke character physics provides significantly more consistency in how the character moves. The inconsistency in character movement served as an obstacle in developing procedural level design that was consistently completable. Switching to a bespoke character controller solved this issue, as the more consistent physics could be easily altered and adjusted to meet the specifications of generated levels.

Another feature implemented in response to tester feedback was a tutorial level. This was due to players finding it difficult to understand the project without having the controls explained. In a game without procedurally designed levels – an explicit “tutorial” level with instructions is not always necessary. Some games, such as Super Mario 3D World (Nintendo, 2013), instead use specifically designed levels or open areas that intuitively teach the controls under the guise of a standard level. This approach would not work for this project, as the premise of random levels has far too much variance in generating the player a “safe” first level to navigate. As such, a simple, premade level had to be designed.

The tutorial level explains how to use the character’s moveset – so that the player can learn how to use the character without any stakes. Additionally, this allows the players to more accurately assess the character controller separately from the level design when it came to gathering data for the final testing.

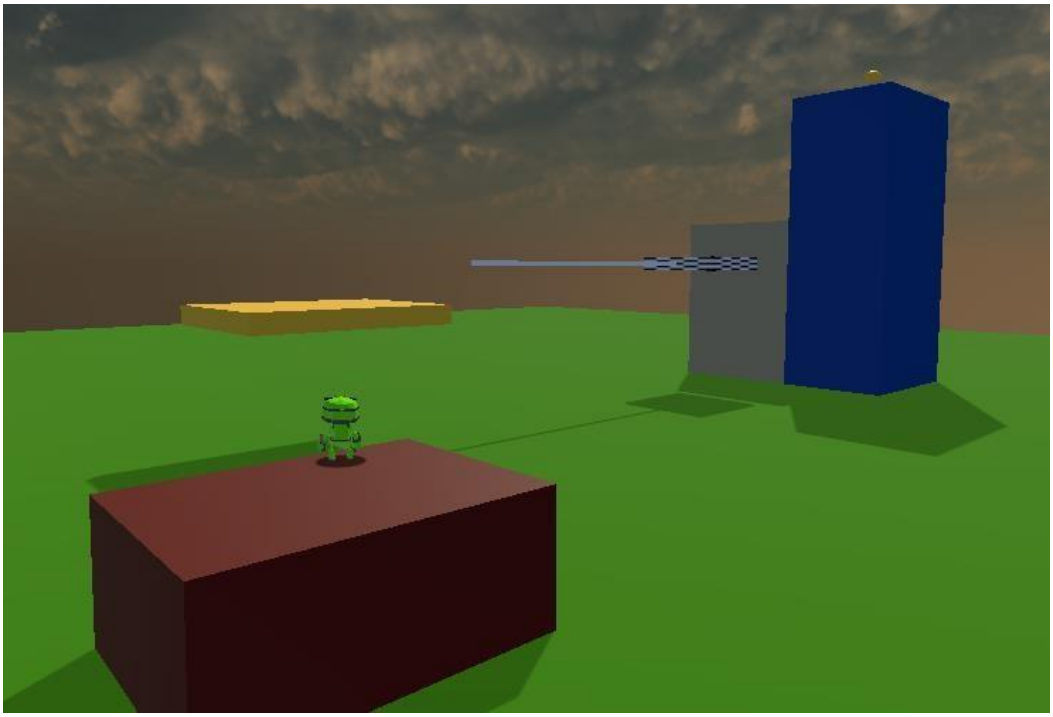


Figure 12: A screenshot showcasing the tutorial level

A significant amount was done to iteratively polish the project during this stage. For example, giving the character an imposter shadow that projects directly downwards was in direct response to analysing playtest gameplay – where players would miss their jumps or not use the Stomp move to land safely.

Adding a dynamic jump height that adjusts to how long the player holds down the button was also in response to playtesting. Testers found it unnatural that lightly tapping the Jump button caused the character to immediately jump at their maximum jump height. This was originally done by increasing the player's velocity smoothly as they held down the button – but this proved difficult to predict the jump arc in actual gameplay. As such, the method was changed to have the player's jump height increase in distinct stages as they held down the button.

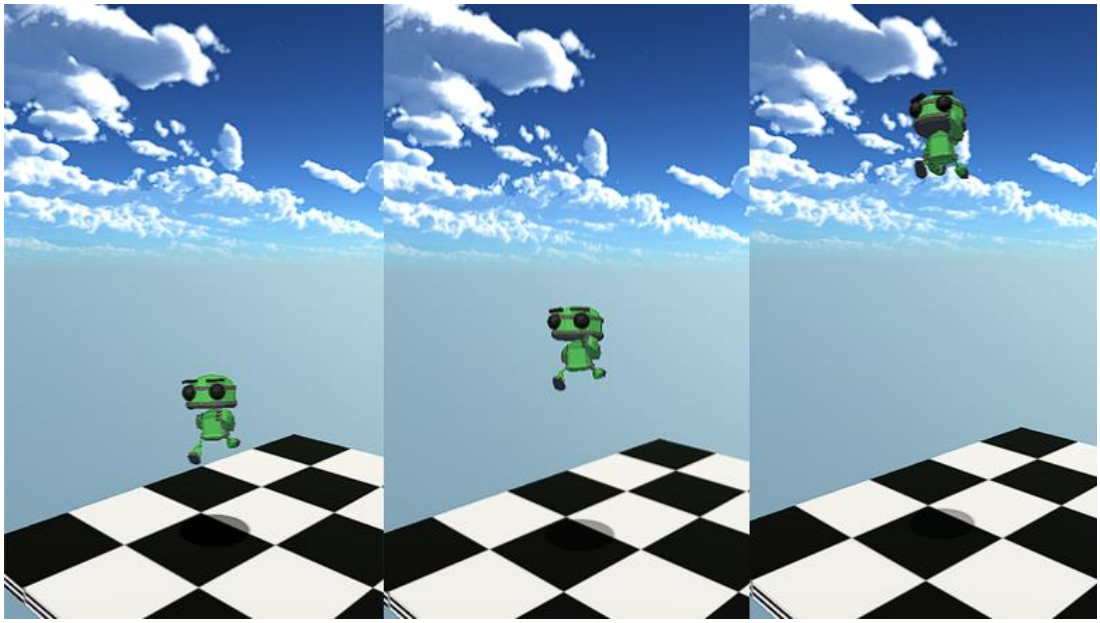


Figure 13: A comparison showing the three tiers of jump height. This shows the jump height on button tap (left), if held for 0.1 seconds (centre), and if held for greater than 0.2 seconds (right).

Another aspect focused on in order to polish the game was increased animation clarity when jumping. Originally, when the character jumped, they would run through the jumping animation before returning to their original walking animation. This led some testers to being confused as to when exactly they were grounded and when they were airborne, so the animation was changed to pause in the jumping animation whilst in the air.

These changes were done both to improve the final artefact qualitatively, but also to remove any conflicting or annoying aspects which may get in the way of the results of the final testing.

5.2 Project Findings

Following the initial playtesting, final testing was done to gather finalised results and to gather opinions on the quality of the game as a whole. The outline of this final testing process was shown in Section 3.5, with the results being analysed in the following sections.

5.2.1 Hypothesis

Looking over the project, the following predictions regarding player opinions were made:

The game created for this project does generate structurally sound, completable and functional levels. Whilst the level designs are unique each time – the structure of each level generated does not vary much with each new level generated.

As the research question is to determine how procedural generation methods can be used to generate enjoyable levels, there become difficulties answering this question to a satisfying degree due to the scope of the project. If the structure of the level is the same every time, it is difficult to parse whether the project generates “enjoyable levels”, rather than just one enjoyable level.

Judging whether a level is enjoyable is based on a wide variety of factors, and meeting all of them is significantly outside the scope and timescale of this project. Factors which were not given much focus during the development for this artefact include:

- Level aesthetics and theming – backgrounds, colour schemes, terrain design, music, etc.
- Variety in level obstacles – enemies, hazards, etc.
- Variety in level formations – having levels which follow more unique patterns. The current project almost exclusively generates levels that are very linear and horizontal.

These factors would increase the distinctness of a generated level – and when looking at video games overall, distinctness and enjoyability are often very closely linked. Good level design can only go so far – and getting objective research on enjoyable aspects of video games would require a far wider breadth of testing than what is available to this study.

5.2.2 Analysis of Results

5.2.2.1 Enjoyability of Levels

In analysis of the results – we can see that all three of the levels were rated very equally. Testers were slightly more negative on the human-created level, there are likely a few reasons for this:

1) This was the first level that testers played, so they were likely to become more comfortable with the controls and gameplay the more levels that they played.
2) As the level design for this level was inspired by Super Mario 64 (1996), it is not as tailored to the control scheme as the procedurally generated levels.

As seen in Table 5, the level generated randomly at runtime had the most variation in enjoyability. It received the most 4 and 5 ratings, but was also the only level to receive a rating of 1 from a participant. This makes sense, as the randomness of the level could lead to variance in quality. The fact that the mean average enjoyability of the random level was the same as the level generated by the seed allows us to infer that the quality of generated levels is consistently enjoyable.

Similarly, Table 5 shows us that the human-created level had the least variance in enjoyability. This makes sense as levels created by hand are intended to be tailored to produce a specific level of enjoyment that does not vary. The enjoyability of procedurally generated levels is less predictable.

One reason for specifically analysing a random level for each tester was in order to test the completability of randomly generated levels. All of the 12 participants were able to complete their random level – and only one rated it negatively. Whilst this is a small sample size – and confidently proving that all generated levels are possible would be a near impossible task – the results do at least indicate that truly impossible levels generated by the game are very hard to come by.

5.2.2.2 Character Controller

Table 6 shows that 58.3% of users did not struggle with controlling the character, whilst the remaining 41.7% did. This shows that significantly more work must be done to improve the character controller – as this is a very large portion of players struggling when there should not be any issues with controlling the game at all. This also lessens the accuracy of the results for questions 1 to 3 – as the testers would have difficulty judging the quality of level design if the controls were getting in the way.

On average, testers that found the character controller difficult to control rated the levels a 3.467 for enjoyability, whereas those who did not have difficulty with the character controller rated the levels a 4.19 for enjoyability. This is a small but not insignificant difference of 14.4%, and whilst the sample size is far too small to come to a significant conclusion, it can be inferred that character controls do have an impact on the enjoyability of level design from a player's perspective.

Both averages lean towards the positive ratings for enjoyability – but it can be said that enjoyable "levels" are not inherently generatable without enjoyable gameplay to go along with it. This is an area in which the research question falters – as it is a fairly one-dimensional question in a subject needing significantly more nuance. Ultimately, no level can be well designed in a game that is not fun to control, and more flowing unpredictable nature of procedural generation makes creating a character controller that is both significantly polished enough to be fun to control and robust enough to meet all of the random elements generated by the software is significantly out of scope for a project of this size.

In addition, the game can only generate levels that follow a specific, linear format without too much variation. The more aspects are added to the level generation algorithm, the more robust the character controller will need to be. This creates an exponential issue of development if one were

to use these procedurally generated mechanics to create a full game that is both polished and has a breadth of level types that it can generate.

At the start of the project, developing a polished character controller was not a significant priority, as the procedural generation of the levels seemed to be far more significant to answering the research question. In retrospect, this led to major oversights throughout the development process. In the case study, a huge focus was put on level design methodologies, whilst research into different character controllers was seen as an afterthought. Whilst developing the project, much more work was done in iterating upon the procedural generation in order to create enjoyable levels, as this was seen as the key component in answering the research question. Whilst that is still somewhat true, it is far more accurate to say that both a polished character controller and high-quality procedural generation are need in order to generate enjoyable levels.

This process has led to the revelation that the two aspects are intrinsically linked. A character controller is more than just a tool to explore good level design – the quality of a character controller directly impacts how the audience perceives the level design. If the project were to be restarted – putting more emphasis on developing a complete, enjoyable feeling character controller before gathering results would be a major priority.

5.2.2.3 Open Text Discussion

Several of the open text comments provide useful areas to consider for further expansion of the project, as well as analysing its shortcomings.

Responses 2, 7, 8 and 10 refer specifically to the character controller and the camera. Comments 7 and 11 from the open text comments both provide specific feedback regarding the air control and gravity, as discussed in Section 3.4. These players found the character less fun to play as due to the “floaty” jump, which in turn makes the game (and level design) less enjoyable. To fix this issue, finetuning would need to be done to both the character controller and the procedural generation – to both make the jump gravity feel more natural and comfortable for players, as well as adjusting the level generation to fit with the new specifications of the player controller.

Responses 8 and 10 both refer to the camera, with one tester finding difficulty with the lack of inverted camera controls and rated all three levels negatively due to this. The other tester had issue with the camera when falling off ledges. The latter issue could most likely be solved by implementing two changes:

1) When the player falls off a ledge, they fall extremely quickly. This is due to the gravity being cumulatively applied when on the ground. When the player jumps their gravity is reset, to have the character

fall in a natural jump arc. A solution to the “fast falling” would be to reset the player’s gravity when they fall off a ledge as well, so that they can react to the fall with their normal jump speed and potentially recover.

2) The camera makes use of Unity’s Cinemachine extension, which simply follows the object passed to it, and attempts to frame the character in the centre as best as possible. The values used for this camera are mostly the provided defaults – and whilst they are functional, they lack nuance for different edge cases. Instead of following the player exactly, the camera could follow a unique path based on the player’s location, and as such better frame the scene to showcase upcoming jumps or the sides of platforms.

Response 1 states that adding “a marker which communicates to the player where exactly they will land after jumping” would be a useful feature. As the shadow directly below the player was implemented exactly for this reason before testing, it can be inferred that the shadow is not useful enough in conveying this information. This can likely be fixed by increasing the size of the shadow, darkening the colour, or potentially even adding a visible line down from the player’s position to the ground.

5.2.3 Testing Improvements

After analysing the gathered results, there are a few clear ways in which results could have been improved.

Testers ended up having more difficulty with the first level, and whilst this was likely due to the fact that it was not created explicitly for this project, there is very likely bias caused by the order in which the levels were played, making the objective data from the surveys less trustworthy. A solution to this would be presenting the levels in a random order for each tester. As the survey presented the levels indeterminately as the 1st, 2nd and 3rd levels, having random levels for the participants to play would cause the results to not line up with a static survey. A potential solution to this could be putting the survey in the software itself, asking the player to rate each level and then storing the data with the corresponding level type internally, or writing that data to a file. Had the three levels been presented in a random order, this would have resulted in more balanced results, and it would have allowed for more objective analysis of the tester’s opinions.

Another way to improve the survey would be to add further detail and additional questions. Specifically, a question with the following structure would have been particularly useful:

Question	Possible Answers
Which of the 3 levels played was your favourite?	Level 1
	Level 2
	Level 3
	There was no discernible difference in quality between the levels.

Table 8: Potential Additional Question

With the current data, it is difficult to answer which type of level was enjoyed the most, as the results all fall around the same average. Adding a question specifically asking testers to decide between levels could either lead to clearer results about whether the procedurally generated levels were more enjoyable, or conversely lead to clearer evidence that the level generation methods are too similar to accurately discern enjoyability.

When playing three levels of very similar structure, it is possible that users become numb to the specific qualities of each level that make it enjoyable. Whilst it can be hypothesised that testers could not pick a favourite level as the levels all share the same structure and general level design qualities, no concrete link can be made between these statements with the current data.

The survey could have additionally asked tester's regarding their knowledge of 3D platformers, and their enjoyment with the genre. The survey was open to all, not just platformer fans, and as such it is possible that the personal opinions of the genre could have influenced testers. It would have been useful to determine if there was a potential link between whether users enjoyed the genre as a whole and whether they had difficulties with the character controller. As the project is mostly tailored to those who already enjoy platformers, determining whether they had difficulties with the controls would be useful information for further development and refinement of the gameplay.

Another way the survey could be altered to receive more detailed feedback would be asking more specific questions regarding the criteria for enjoyability. Whilst research was done into what makes a platformer level "enjoyable" in the Case Study, overall, it is a hugely subjective topic that ultimately comes down to the player's personal tastes.

If the "enjoyable" qualities a level had been broken down into more detail, potentially by asking questions regarding the level layout, obstacle placement, or challenge, this could have led to clearer results regarding what the algorithm's strengths and weaknesses are. Whilst the data

shows that the algorithm generates “enjoyable” levels, the scope of the survey is so small, and the topic is so subjective that it makes it difficult to discern much objectively about user opinions regarding the levels.

5.3 Critical Evaluation

5.3.1 Response to Proposal

The project was proposed with the aim to “develop a prototype system for use in a 3D platformer which procedurally generates level design”. The artefact created successfully meets these requirements.

After initially reviewing examples of literature on the subject, it was determined that using a rule-based system for generation would be most proficient, and initial prototypes were created using this idea. These initial prototypes were functional – but level design felt very artificial. Pathways would be generated exclusively through straight lines, and they would often loop around in ways that trivialised other aspects of the level design.

Further development led to a system which makes use of Perlin worms for the main generation algorithm, working in tandem with rule-based generation to create fully fledged level design. This new system became the finalised one for this artefact and was successful due to the capability of Perlin noise for producing complex yet realistic values for procedural generation. The Perlin noise system was also significantly faster than the original, purely rules-based system.

One aspect of the Literature Review that became especially poignant further in development was the observation that an enjoyable character controller would transitively make the level design more enjoyable – heavily indicated by the gathered results. Further development has proven that breaking game design down into specific components of level design and controls is a somewhat futile task – as players will parse all these elements collectively. Whilst the generation algorithm is the most important aspect of this project – it can also be seen that further work on the character controller would increase the enjoyability of the levels.

5.3.2 Improving the Solution

The solution would be significantly improved by generating levels that are more distinct in layout. Most levels generated follow the format of stretching forward linearly, without too much vertical variation. Even within the format of linear, spline-based levels, more could be done to add distinct layouts – such as creating levels that follow a spiral pattern, levels that stretch upward more than forward, or levels that coil backwards. This could be done by adapting the Perlin Worms, or potentially by working

with a more customisable, rule-based form of generation – as was explored in the Literature Review.

A further extension of this idea would be to include levels with new formats. One such example could be generating a level with a flat plane intersecting the main body of the spline, to give a main piece of ground for the player to anchor themselves to. This would significantly change the challenge of the level, rather than building the level around the challenge just being to stay on the platforms.

Additionally, levels with branching paths could be an area for further exploration. To use the existing structure, these could be generated by spawning in two Perlin Worms, and sampling both to generate distinct splines. A further step would be to calculate the areas where the splines intersect (or get close to intersecting) and alter the terrain generation appropriately – potentially removing segments of the second spline, or even transforming the points of the second spline to appear more connected to the original pathway generated. Both of these ideas still make use of the spline-based generation – an even further step could be taken to add levels generated in completely different ways. These could potentially make use of a grid-based structure, similar to Spelunky but extrapolated into 3D, as a potential example.

Most aspects of the generation process could be expanded indefinitely, simply due to the nature of the procedural generation algorithms. An example of this would be adding more Dynamic Challenges that could be placed in the level. In the current solution, moving platforms are almost exclusively placed between the gaps in the spline – adding more options for obstacles that can be generated in these gaps would add variety.

Similarly, the project currently instantiates some singular, static obstacles along the pathway, being blocks and spike balls. A simple way to add variation to the level generation would to be to add more options for what static obstacles could be generated – such as bumpers that could bounce the player around, or cannons that could fire spike balls in a set direction.

The next natural evolution in terms of obstacles would be to add enemies. In platformers that lack a traditional combat system, enemies are essentially just obstacles with some form of intelligence – either chasing after the player or following their own internal path. These elements, however, would have to be significantly tested to make sure they interact correctly with the generated geometry in a way that still feels natural.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In conclusion, it's very difficult to draw an objective answer to the research question. The artefact created is able to generate functional level design, which most testers found enjoyable; however it still leaves a large amount of room for both expansion and questioning. The scope of an "enjoyable level for a 3D platformer" is incredibly vast and subjective – and without surveying a huge percentile of participants and developing for a far larger period of time, there is no way that a program could generate a level that is enjoyable to everyone.

In addition, some scrutiny can be pointed to the idea of whether the program can develop distinct enjoyable levels. Whilst each level created has a unique layout, the structure of each level falls into very similar territory, which could consider the level generation moderately redundant. With the existing scope, increasing the variety in level generation would lead to significantly unfocused levels – creating levels with both linear focus and variety would simply require much more development time. The program can generate multiple enjoyable levels; but at this stage in development, each level provides a very similar facet of enjoyment to the last.

6.2 Implications

This implies that developing a program to generate enjoyable levels for a 3D platformer is certainly possible, but there is a significant amount of further research and development needed. In addition, the project also implies that whilst some clear points can be made - enjoyment is far too subjective of a quality to directly measure.

Another major implication the results is the direct correlation between satisfying controls and enjoyable levels. When playing a game – the boundaries between different facets of design is blurred. Good controls are necessary to communicate good design work – and the enjoyability of level design can only be truly assessed through putting it into practice and gameplay. Similarly, a game with great controls needs good game design to take advantage of it – otherwise the character controller will not leave an impact. There is a symbiotic relationship between theoretical design and the execution of the controls that makes isolating either part of the process futile.

6.3 Further Work

This subject matter has a huge variety of untapped potential, and working on this project has very clearly explained why – in order to create

uniquely varied levels, a huge amount of development time must be allocated, almost negating the workload alleviation that procedural generation provides.

However, the concept of a procedurally generated 3D platformer is definitely something worth continuing to build upon. The levels generated by this project show the groundwork for what could be something far grander, and generate levels beyond the scope of human-designed ones. Procedurally generated levels act as an excellent framework to allow the player to entire this free-flowing, continuous state of gameplay. Due to the lack of variation in the generated levels, the artefact more so creates so creates this feeling of one, infinitely expanding, endless level – an effect not capturable through standard level design. The usage of noise leads levels generated to fall within a similar enough frequency, creating this endless effect, similar to that seen in isotropic design.

Further work could be done to adapt level generation to use different types of procedural generation. Perlin worms were used for this project, however different methods for PCG, such as Wave Function Collapse or Genetic Algorithms, could be used instead to potentially be more effective. Using different methods for PCG could lead to completely different gameplay experiences for players, resulting in levels that communicate very different atmospheres.

Additionally, this project only covered one type of level generation – that of generating linear levels – with a singular start and end point. There is significant amounts of untapped potential with the exploration of the generation of levels with more variance in structure. Open-ended levels with multiple objectives offer a completely different form of gameplay within the same genre, attempting to generate a level of this style whilst maintaining the coherence of human-designed levels is a worthwhile topic to explore.

Even within the scope of linear levels – there is a lot that can be done in terms of generation. An initial example would be generating levels with branching paths, as could result in more player freedom to tackle the level, increasing enjoyability. Generated levels in this project take the form of standard linear levels, but linear levels in platformers can fall in a wide variety of formats, including levels built around racing an AI opponent, levels built around combat with enemies, or levels built around specific movement mechanics such as sliding. This project only scratches the surface of the possibilities.

List of References

- Gears for Breakfast (2017), *A Hat in Time* [Video game].
- Baghdadi, W., Eddin, F., Al-Omari, R., Alhalawani, Z., Shaker, M. and Shaker, N., 2015. A Procedural Method for Automatic Generation of Spelunky Levels. *Applications of Evolutionary Computation*, pp.305317.
- Ginting, A.K. *et al.* (2019) 'Application of the perlin noise algorithm as a track generator in the endless runner genre game', *Journal of Physics: Conference Series*, 1255(1), p. 012064. doi:10.1088/1742-6596/1255/1/012064.
- Togelius, J., Yannakakis, G., Stanley, K. and Browne, C., 2010. Search-Based Procedural Content Generation. *Applications of Evolutionary Computation*, pp.141-150.
- Cheng, D., Han, H. and Fei, G., 2020. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm. *Lecture Notes in Computer Science*, pp.37-50.
- Mourato, F., dos Santos, M. and Birra, F., 2011. Automatic level generation for platform videogames using genetic algorithms. *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology - ACE '11*,.
- Rare (1998), *Banjo-Kazooie* [Video game]. Microsoft.
- Perlin, K., 2002. 'Better acting in computer games: The use of procedural methods', *Computers & Graphics*, 26(1), pp. 3–11. doi:10.1016/s0097-8493(01)00173-x.
- Benard, S., 2017. *Building the Level Design of a procedurally generated Metroidvania: a hybrid approach.*. [online] Game Developer. Available at: <https://www.gamedeveloper.com/design/building-the-level-designof-a-procedurally-generated-metroidvania-a-hybrid-approach-> [Accessed 16 October 2022].
- Pwnee Studios (2013), *Cloudberry Kingdom* [Video game].
- Naughty Dog (1996) *Crash Bandicoot* [Video game]. Sony Interactive Entertainment.
- Motion Twin et al. (2018), *Dead Cells* [Video game].
- Supergiant (2020), *Hades* [Video game]. Supergiant Games.
- Supergiantgames.com. 2020. *Hades FAQ | Supergiant Games*. [online] Available at:
- <https://www.supergiantgames.com/blog/hades-faq> [Accessed 13 October 2022].
- Mojang Studios (2011), *Minecraft* [Video game]. Microsoft.
- Himite, B., 2021. *Replicating Minecraft world generation in Python*, *Medium*. Available at: https://towardsdatascience.com/replicating-minecraft-world-generation-in-python-1b491bc9b9a4 [Accessed: 16 May 2023].
- Hopoo Games (2020), *Risk of Rain 2* [Video game].
- Glenn Wichman et al. (1980), *Rogue* [Video game].
- Sonic Team (1991), *Sonic the Hedgehog* [Video game]. SEGA.
- Derek Yu et al. (2008), *Spelunky* [Video game].
- Kazemi, D., 2010. *Spelunky Generator Lessons*. [online] Tinysubversions.com. Available at:

<http://tinysubversions.com/spelunkyGen/> [Accessed 15 October 2022].
- Nintendo EAD (2007), *Super Mario Galaxy* [Video game]. Nintendo.
- Nintendo (2017), *Super Mario Odyssey* [Video game]. Nintendo.
- Nintendo (1996), *Super Mario 64* [Video game]. Nintendo.
- Nintendo (2013), *Super Mario 3D World* [Video game]. Nintendo.
- Heaton, R., 2018. *The Wavefunction Collapse Algorithm explained very clearly*. [online] Robert Heaton. Available at: <https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/> [Accessed 17 October 2022].
- Zucconi, A. (2022) *The World Generation of Minecraft*, *Alan Zucconi*. Available at: https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/ [Accessed: 15 May 2023].

Bibliography

- Pharr, M. and Fernando, R., 2006. ‘Chapter 26. Implementing Improved Perlin Noise’, in *GPU gems 2: Programming techniques for high-performance graphics and general-purpose computation*. Upper Saddle River: Addison-Wesley.
- Pedersen, C, Togelius, J & Yannakakis, GN 2009, Modeling player experience in Super Mario Bros. in *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games.*, 5286482, CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games, pp. 132-139, CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 9/7/09. https://doi.org/10.1109/CIG.2009.5286482
- Perlin, K., 2001. *Noise and Turbulence*, *Ken’s Academy Award*. Available at: https://web.archive.org/web/20180501200154/http://mrl.nyu.edu/~perlin/doc/oscar.html#noise [Accessed: 15 May 2023].

Appendices

Appendix A – Raw Questionnaire Results

Test No.	Level 1 Enjoyability	Level 2 Enjoyability	Level 3 Enjoyability	Average Rating
1	4	4	4	4
2	4	4	3	3.67
3	4	3	5	4
4	3	5	4	4
5	4	4	5	4.33
6	4	5	5	4.67
7	4	5	4	4.33
8	2	2	1	1.67
9	4	4	4	4
10	4	4	4	4
11	5	5	5	5
12	2	3	4	3

Test No.	Did you find that the controlling the character was difficult?
1	Yes - the character controller got in the way of enjoying the game
2	No - the character controller was not an issue
3	No - the character controller was not an issue
4	No - the character controller was not an issue
5	Yes - the character controller got in the way of enjoying the game
6	No - the character controller was not an issue
7	Yes - the character controller got in the way of enjoying the game
8	Yes - the character controller got in the way of enjoying the game
9	No - the character controller was not an issue
10	No - the character controller was not an issue
11	No - the character controller was not an issue
12	Yes - the character controller got in the way of enjoying the game

Test No.	Is there any specific features you feel like would add to the gameplay? (And/or any bonus comments)
1	Perhaps a marker which communicates to the player where exactly they will land after jumping. Other than that very cool!
2	Character controls were mostly solid, but the air dash felt more restricting than I think it was intended
3	extra characters, more levels, powerups of some kind?
4	
5	
6	I thought that the gameplay was very good and intuitive. It was very easy to understand the controls and how to play the game.

7	Stronger gravity, checkpoints
8	Inverted camera controls please, Im one of the freaks
9	
10	Smoother camera when falling off of ledges
11	Slightly more varied levels would be good, to encourage the use of the different abilities more. Also although the controls didn't get in the way of my enjoyment with the game, it might feel a bit batter if they were slightly less slidy in the air.
12	a princess at the end that I can kiss on the cheek it wouldmake me happy

Appendix B – Data Collection Consent Statements

- I understand the contents of the participant information sheet and consent form.
- I have been given the opportunity to ask questions about the research and have had them answeredsatisfactorily.
- I understand that my participation is entirely voluntary and that I can withdraw from the research (parts ofthe project or the entire project) at any time without penalty and without having to provide an explanation.
- I understand who has access to my data and how it will be handled at all stages of the research project.
- I consent to take part in this study conducted by Ewan Fisher who intends to use my data for further research examining the use of procedural techniques in the creation of levels for a 3D platformer.

Appendix C – GDPR Research Data Management Sign Off Form

Abertay University

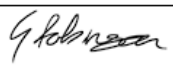
GDPR Research Data Management
Data Sign Off Form

For undergraduate or postgraduate student projects supervised by an Abertay staff member.

This form MUST be included in the student's thesis/dissertation. Note that failure to do this will mean that the student's project cannot be assessed/examined.

Part 1: Supervisors to Complete

By signing this form, you are confirming that you have checked and verified your student's data according to the criteria stated below (e.g., raw data, completed questionnaires, superlab/Eprime output, transcriptions etc.)

Student Name:	Ewan Fisher		
Student Number:	1900176		
Lead Supervisor Name:	Gareth Robinson		
Lead Supervisor Signature			
Project title:	Evaluation of Procedural Generation to Create 3D Platforming Levels		
Study route:	PhD ☐	MbR ☐	MPhil ☐
	Undergraduate ☐	PhD by Publication ☐	

Part 2: Student to Complete

	Initial here to confirm 'Yes'
I confirm that I have handed over all manual records from my research project (e.g., consent forms, transcripts) to my supervisor for archiving/storage	EJF
I confirm that I have handed over all digital records from my research project (e.g., recordings, data files) to my supervisor for archiving/storage	EJF
I confirm that I no longer hold any digital records from my research project on any device other than the university network and the only data that I may retain is a copy of an anonymised data file(s) from my research	EJF
I understand that, for undergraduate projects, my supervisor may delete manual/digital records of data if there is no foreseeable use for that data (with the exception of consent forms, which should be retained for 10 years)	EJF

Student signature : Ewan Fisher

Date: 15/05/2023