Evaluating the Effect of Procedural Level Generation on Movement-Based Gameplay Mechanics

Harry Findlay

BSc (Hon) Computer Game Applications Development, 2025

School of Design and Informatics

Abertay University

Contents

[Table of Figures 4](#_Toc194608537)

[Table of Tables 4](#_Toc194608538)

[Acknowledgements 5](#_Toc194608539)

[Abstract 5](#_Toc194608540)

[Context: 5](#_Toc194608541)

[Aim: 5](#_Toc194608542)

[Method: 5](#_Toc194608543)

[Results: 5](#_Toc194608544)

[Conclusion: 6](#_Toc194608545)

[Abbreviations, Symbols and Notation 6](#_Toc194608546)

[1. Introduction 7](#_Toc194608547)

[1.1 What is Procedural Content Generation 7](#_Toc194608548)

[1.2 Procedural Generation in Movement-Based Video Games 7](#_Toc194608549)

[1.3 Aim 9](#_Toc194608550)

[1.4 Research Question 9](#_Toc194608551)

[1.5 Research Question Critical Analysis 9](#_Toc194608552)

[1.6 Objectives 9](#_Toc194608553)

[2.0 Literature Review 10](#_Toc194608554)

[2.1 Procedural Generation Techniques 10](#_Toc194608555)

[2.1.1 Wave Function Collapse and Noise Techniques 10](#_Toc194608556)

[2.1.2 Realtime Procedural Terrain Generation 10](#_Toc194608557)

[2.1.3 Time-Space Wave Function Collapse 11](#_Toc194608558)

[2.2.1 In-Depth Wall Running Mechanic 12](#_Toc194608559)

[2.2.2 Momentum-based Gameplay in FPS Games 12](#_Toc194608560)

[2.2.3 Third MB-GM source 13](#_Toc194608561)

[3.0 Methodology 13](#_Toc194608562)

[3.1 Overview 13](#_Toc194608563)

[3.2 Application 13](#_Toc194608564)

[3.2.1 Application Design 13](#_Toc194608565)

[3.3 Node and Model Design 14](#_Toc194608566)

[3.3.1 Node Uses and Overall WFC Generation Result 14](#_Toc194608567)

[3.3.2 Propagation (neighbouring tiles generation) 14](#_Toc194608568)

[3.4 MB-GM design 15](#_Toc194608569)

[3.4.1 Wallrunning and sliding 15](#_Toc194608570)

[3.4.2 Objective 15](#_Toc194608571)

[4.0 Results 15](#_Toc194608572)

[4.1 Questionnaire Data 15](#_Toc194608573)

[4.2 Playtest Data 15](#_Toc194608574)

[4.2.1 Player playtest data 15](#_Toc194608575)

[4.2.2 Player generation data 15](#_Toc194608576)

[5.0 Discussion 15](#_Toc194608577)

[5.1 Playtest session 15](#_Toc194608578)

[5.1.1 Questionnaire Data Analysis 15](#_Toc194608579)

[(may split into generation and player subsections again) 15](#_Toc194608580)

[5.2 Project findings 15](#_Toc194608581)

[5.2.1 Project summary 15](#_Toc194608582)

[5.2.2 Research Question 15](#_Toc194608583)

[5.3 Critical Evaluation 16](#_Toc194608584)

[5.3.1 Development Evaluation 16](#_Toc194608585)

[5.3.2 Playtest evaluation 16](#_Toc194608586)

[5.0 Conclusion 16](#_Toc194608587)

[5.1 Main conclusion 16](#_Toc194608588)

[5.2 Implications 16](#_Toc194608589)

# Table of Figures

# Table of Tables

# Acknowledgements

I would like to thank my Wife, Dana and my kids, Roman and August for their continuous support throughout my time at university.

I would also like to thank my advisor, Dr. Naman Merchant. Naman has been so flexible yet available and helpful along every step of the way and his advice has been greatly helpful.

# Abstract

## Context:

Procedural content generation (PCG) is widely used throughout the games industry, covering many genres, the type of content generated and the uses of content that is generated. However, many genres of games seem to avoid the use of PCG as, in many cases, many different genres are better suited to hand-crafted worlds. The goal of this study is to investigate the level-building side of PCG and how this relates to movement-based gameplay mechanics such as wall-running and sliding.

## Aim:

To investigate, compare and evaluate the differences between static-made game worlds and procedurally generated game worlds. The focus will be on the how movement-based and movement-enhancing gameplay mechanics relate and adapt to the game-space that they are in. Should developers and companies be more open to the idea and use of procedurally generated levels and worlds in movement-based video games.

## Method:

An algorithm will be developed (or tool used) to create a very simple procedurally generated level. All tools can be found via the epic store or unreal engine forums/documentation. There will be a focus on wall-like structures with a secondary focus on building-style structures, being more box-shaped. This environment would then be tested against a static environment and compared in certain areas of interest to the research. I then aim to have a group of individuals of no specific set of skills to test PLG against a static level and fill out a questionnaire.

## Results:

I expect that PLG would be a fun addition to MB-VG, however, would be too unstable in its generation to properly compliment the gameplay mechanics. I hypothesize that PLG would work for sub-genres of the MB-VG such as movement-based shooters (Titanfall 2, Respawn Entertainment, (2016)) but be less effective in core movement-based games such as Mirrors Edge (DICE,

(2008)).

## Conclusion:

The conclusion is determining whether procedural generation is an appropriate and effective alternative to level/map creation when using movement-based gameplay mechanics. This paper will be investigating Wave Function Collapse specifically, and whether this is a valid alternative for developers looking to save time on level design, time and development costs.

# Abbreviations, Symbols and Notation

Procedurally Content Generation – PCG

Procedurally Generated Level(s) – PGL

Procedural Level Generation - PLG

Movement-Based Gameplay Mechanics – MB-GM

Movement-Based Video Games– MB-VG

Wave Function Collapse - WFC

# Introduction

## What is Procedural Content Generation

Procedural Content Generation (PCG) is the process of algorithmically creating content at run-time using a set of pre-defined rules and processes created by a developer.

PCG has been a programming technique since the 1980’s in, not just video games, but in many areas of computer programming to create content at run-time. Generally, PCG follows rules or guidelines when generating content as to ensure that the content fits with the profile of the overall application or video game. Some strong examples of early uses of PCG in video games include Beneath Apple Manor” (The Software Factory, (1978)) and “Rogue”, (originally developed by Michael Toy and Glenn Wichman ((1980)).

PCG was originally developed to produce semi-random content for video games or applications in a fast and low-cost manner, at runtime. PCG was also used in other ways, some uses being: ways to bypass system restrictions and as a way to compress file sizes on older, lower memory computers.

However, modern uses of PCG are very different when compared to the historic uses. In video games, modern uses of PCG is generally aimed at enhancing the video game’s content, replayability and general purpose, rather than the earlier uses of combatting system restrictions and file size issues that plagued older systems and computers. Another example of modern used of PCG in video games is procedural level generation (PLG), to which is where this research paper will be focused. Levels, structures, terrain and other object-based topics can be semi-randomly generated at run-time to allow for replayability and to offer the video game with a wider arsenal of content to be used. Other, none-related areas of PCG include: procedurally generated stories, quests, weather patterns, designs and in-game surfaces, and in some cases, even gameplay mechanics.

PCG, however, does come with negatives as well, especially if poorly executed. Some examples may include: PCG stories lacking depth and purpose. Some PCG levels can feel empty and meaningless, many PCG creatures and NPC’s (non-player characters) may appear illogical or too-random to make sense.

## Procedural Generation in Movement-Based Video Games

The development of video games has become one of the worlds largest and most successful industries, and with this, comes bigger and better computers capable of processing much more in a shorter span of time. With these improvements, PCG has been able to really thrive and bloom. This can come in a multitude of different forms ranging from entire worlds being generated to a complete arsenal of weapons, armour, characters or in some cases, entire solar systems or galaxies. Some examples to look at when investigating the expanding capabilities of PCG in video games: Valheim (Coffee Stain Studios) with its early access release in 2021 and its official release in 2023. Valheim is a flawless example of large-scale world generation featuring different biomes, enemies, bosses and more. Another example is No Man’s Sky (Hello Games (2016)), though No Man’s Sky had a shaky release regarding player experience, upon release, No Man’s Sky used PCG to generate not one solar system, but 4.3 trillion. These, however, are video games that do not have an emphasis or primary gameplay loop related to movement-based gameplay mechanics (MB-GM).

Movement-based video games (MB-VG) as a genre in whole tends to neglect the use of procedural level generation (PLG), and this is usually for a multitude of reasons. Titanfall 2 (Respawn Entertainment (2016)) is an example of a successful MB-VG with an entire gameplay loop with strong ties to MB-GM yet has absolutely no PCG content throughout the entire product. The same goes for other successful MB-VG including Call of Duty Black Ops 3 (Treyarch (2015)) and Mirrors Edge (DICE (2008)), again, both extremely successful MB-VG with absolutely no PCG content or levels. To add perspective, Call of Duty Black Ops 3 generated over $550 million in revenue during its first three days of release, and Mirrors Edge generated approximately $125 million in revenue.

The reluctancy to use PCG in MB-VG is very clear, and, as stated prior, this is for some obvious reasons. Research done by Angel, J. (2014) in the book Game Maps: Parkour Vision and Urban Relations. In: Schiller, G., Rubidge, S. shows that making well-flowing maps for parkour or MB-VG requires “parkour vision” as these levels or game worlds need fit well enough together to grant a smooth and enjoyable user experience. This sparked the start of this research paper as the goal here is to develop a prototype PLG algorithm and a series of MB-GM and have human users test this and relay their opinions and experiences. If done correctly, the hypothesis is that PLG will, not only, reduce development time and costs, but can add replayability, fluidity and overall enjoyability to MB-VGs going forward. This includes triple A industry-level products as well as indie-level development projects and video games.

Some video games lead very heavily into PCG as a whole for example, Spore (Maxis (2008)) that uses PCG on multiple scales ranging from creature design and animations, complete civilisations, planetary ecosystems, galactic scale exploration and even the games music. Another example being Borderlands (Gearbox Software (2009)) and its use of PCG to generate weapons with various elemental damage types, parts – that change the stats and behaviour of the weapon, and visuals, weapon sights included. Again, both are examples of PCG in video games however, yet again, both examples are not MB-VG, furthering the evidence of PCG being popular, but overall neglected in MB-VG.

Many MB-VG do use PCG and despite this being on a smaller scale, the MB-VG that do generally use PCG tend to be two dimensional (2D). This is usually as 2D games are generally considered easier to create flowing maps and levels in a way that is entertaining and well-flowing. Some examples including: Prince of Persia (Jordan Mechner (1989)) and Spelunky (Mossmouth (2008)).

## Aim

This project aims to research the implementation of Procedural Level Generation (PLG) in Movement-based video games (MB-VG) and evaluate whether PLG is reliable and suitable enough to be used in future instalments into the MB-VG genre.

## Research Question

To what extent is the user experience of movement-based gameplay mechanics effected by procedural-based level generation and is this this a negative or positive effect.

## Research Question Critical Analysis

PCG is an evolutionary technique and step in game development and is avoided in many situations when it should be embraced and enhanced via skilled developers and trial and error. The consistency of PCG in MB-VG is very important as the play-worlds and levels require a flawless flow and path to keep the MB-GM intact and enjoyable. These traits introduce a wide range of difficulties and challenges to maintain a smooth player experience and keep the generated worlds quality and worth-while.

This raises the question, can PCG be used within MB-VG to develop complete and playable levels/maps, in a smooth and effective manner, whilst able to keep the fluidity and enjoyability of MB-GM.

## Objectives

The primary objectives of this research project include:

1. To study the uses of procedural level generation in movement-based video games and investigate how movement-based gameplay mechanics, such as wall running, will respond to the non-manually sculpted game worlds.
2. To study how the use of movement-based gameplay mechanics can vary in different or procedurally generated environments.
3. Investigate the development and uses of level-based procedural generation tools and algorithms inside of Unity Engine.

# 2.0 Literature Review

In order for this project to be successful and accurate, external sources were investigated and analysed. This research included Wave Function Collapse (WFC) as well as other methods of PCG, all of which were related to PLG. The following sub sections review these sources and their relevance to the project.

## 2.1 Procedural Generation Techniques

## 2.1.1 Wave Function Collapse and Noise Techniques

Büyükşar, O., Yıldız, D. and Demirci, S. (2024) Enhancing wave function collapse algorithm for procedural map generation problem, Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi. Available at: <https://dergipark.org.tr/en/pub/ngumuh/issue/86158/1361413> (Accessed: 13 October 2024)

This source started by introducing and explaining a range of PCG techniques and discussed their strengths and weaknesses. Such PCG techniques that were discussed included; Particle Swarm Optimization (PSO), to which it was explained that PSO was a strong technique due to its fine-grained control offering strong results. Digital Elevation Map (DEM) was also discussed, explaining that this uses a 2D grid system paired with elevation values in order to achieve efficient and effective 2D PLG results. Erosion Based Simulation was expanded upon, explaining that this is a viable approach for generating terrain that resembles actual landscapes, however, struggles to stitch together neighbouring tiles, which can result in unnatural aesthetics. Noise techniques, in general, were discussed, explaining that, despite being a valid and efficient choice in some cases, noise generated outcomes are very efficient at generating terrain, however, lack depth when producing features and can create irregular patterns. WFC is then introduced and explained to be a very reliable approach to PLG but can be high maintenance due to requiring pre-made assets/models and can be expensive due to the overall resources needed. The paper then proceeds to analyse and explore their research and findings regarding a hybrid approach where WFC and noise are used to create a two-in-one system, where the noise generates the terrain and the WFC generates the textures and assets. This hybrid approach was generally considered a success, producing diverse and interesting maps. Overall a very strong source due to the un-biased and effective approach of discussing the strengths and weaknesses of various PCG techniques and their uses.

## 2.1.2 Realtime Procedural Terrain Generation

Olsen, J. (2004) ‘Realtime Procedural Terrain Generation’, Department of Mathematics and Computer Science (IMADA) University of Southern Denmark.

The aim of this literature is to investigate and evaluate various types of erosion-based procedural generation techniques in computer games. With the increase of general computer processing power, erosion-based techniques are a very solid and fast technique it used, giving near run-time results when emphasising speed over quality. Two types of erosion algorithm are used within this research: thermal erosion and hydraulic erosion. These were first described by Ken Musgrave et al (1989) and has been described as a foundation to which various improvements have been suggested and made. An overview, thermal erosion simulates the breaking of material, and how said material would slide down a slope and rest at the bottom. Hydraulic erosion is the simulation of the effects in which flowing water has to terrain and dissolving materials, usually transforming the position of the material and leaving it elsewhere. These erosion types were also paired with a different type of PCG algorithm, which yielded some very interesting and powerful results. This secondary PCG technique is Voronoi Diagrams, and this algorithm is particular focuses on the procedural generation of textures described by Steven Worley. Overall, this approach and research produced sturdy and robust results and is an interesting approach to level-generation.

## 2.1.3 Time-Space Wave Function Collapse

Facey, K., and Cooper, S. (2024) ‘Toward Space-Time WaveFunctionCollapse for Level and Solution Generation’, Khoury College of Computer Science.

This source introduces the process and goals of wave function collapse. It is explained that wave function collapse is a method of PCG which is generally easy and fast to implement as well as being usually low maintenance and setup if done correctly. It is discussed, however, that WFC can be slow when in-process and require a designer to sit and implement the constraints, which act as ‘rules’ for the generation to follow when producing generated content. Additionally, Facey and Cooper’s approach add an additional feature to the traditional WFC, adding time (T) as a factor when generating maps. Normally, WFC uses either a 2D or 3D grid (usually consisting of height, width and length, if 3D), splitting each section of the grid into nodes, however the approach used within this source expands this, turning the nodes into ‘space-time blocks’. The point of adding time to the, generally considered, stable algorithm, is because instead of their algorithm only focusing on, only the level – adding time adds an additional focus: the steps to solve the generation. These generated maps were then tested against a series of gameplay mechanics: Maze, Field and Sokoban. It was found that Sokoban was extremely reliable, generating successful maps 70% vs. 41%, and accurate when used as a testing environment, however, took over 100 times longer to generate a single level – averaging at 51 minutes, 10 seconds. Comparing this to Field that took only 27 seconds, however, was only able to generate a successful map (52% - 64% of the time). Finally, Maze, when compared to Field, it was less likely to return a successful map and was also found to take slightly longer to do so as well, despite the Maze map being less than half the size of Field’s map. Overall, it was found that the generation, as a whole, was very successful. This was due to the fact that the generation was easy, fast, produced content visually similar to the data passed in and, lastly, the levels were completable. It was found, however, that additional game data was required to be passed in as global constraints which, in turn, increased the complexity and time of the setup of this PCG technique. This did prove the WFC to be a reliable algorithm.

## 2.2.1 In-Depth Wall Running Mechanic

Hardman, C. (2024). Advanced 3D Movement. In: Game Programming with Unity and C#. Apress, Berkeley, CA. <https://doi.org/10.1007/978-1-4842-9720-9_38>

This source is a very strong source when exploring the background and mechanical side of strong wall running mechanics – which translates to other MB-GM. Throughout pages 363 to 375, the chapter starts off with a general guide to player movement mechanics in a three-dimensional space, with a general explanation of WASD movement, jumping and wall running (or “push off” mechanics). Momentum is then introduced and explored, explaining that many MB-GM work with a powerful emphasis on momentum as a baseline for fluidity. This, however, is accompanied with many complexities and difficulties as well. Additionally, it is explained that the movement, in a three-dimensional environment, needs to be local to the player – unlike in many two-dimensional games, and this is especially important for the process of the wall running and general MB-GM as the player needs to be able to easily guide themselves through obstacles as well as keep the fluidity of the gameplay. The source then teaches the reader about Vector 3 coordinates, and how they are important to the player transform within the context, using this as an introduction to vectors in general, and then advancing on to explaining magnitude. The reason behind why the vector magnitude is important is because this is the process of converting a vector into a direction, and the normalised direction is required for smooth and reliable MB-GM. Additionally, the source talks about how the behaviour of the MB-GM change depending on whether the user is on-ground or in-air. When airborne, the player direction is then locked to a global state, disabling the player’s ability to turn based on camera (look) direction, which is then re-enabled when grounded. The chapter then C# Unity script and a briefing of what the methods called within the script do. This, overall, is a very detailed and effective approach to wall running, and had a strong influence in the mechanics used within this research.

## 2.2.2 Momentum-based Gameplay in FPS Games

Zhu, C., and Zhang, Y. ‘A First-Person Game Designed To Educate And Aid The Player Movement Implementation’, Beckman High School, University of California. <https://csitcp.org/paper/13/132csit03.pdf>

Within this source, there was a heavy emphasis on MB-GM via a momentum-based movement system. Within this research piece, Unity was utilised to craft a three-dimensional first-person shooter (FPS) game. This game made use of a first-person perspective as, like many other games, first person perspective allows for immersion and, in some cases, a better relationship with certain gameplay mechanics (such as MB-GM). This is the same as in the research paper you are reading now. Though, this game did not contain any advanced mechanics such as wall running or dashing, it did contain a slide mechanic which allowed the player to maintain their momentum – pairing this with the ability to shoot and a series of enemies which attack the player. Both the player and enemies have health points and so can be ‘killed’ if enough damage is done to the target. It was noted within this source, however, that although the MB-GM were polished and well-functioning, many of the other mechanics (such as shooting) were unpolished and missing some aspects such as visuals or audio queues. Another example being the fact enemies only had melee-style damage outpluts. This negatively affected some of the feedback gathered within this study, however, it was generally stated that the MB-GM were well received and ‘welcome feature based on the optional feedback’. Some challenges were stated though out the development of this study’s game, some examples including: the idea of the setting and design of the game, how the health system should be implements and how it should function, and how the shooting to function as a whole – to which a raycast approach was elected. The source goes into much detail regarding many aspects of the game, such as the fire rate of the gun and enemy spawning. After two experiments: the first being the testing of the movement within the game, and the second experiment being aimed at the overall game experience as a whole. It was found, as mentioned, that the movement was very well received and welcomed however, the lacking and less polished areas, such as shooting, affected the results in a negative manner.

## 2.2.3 Third MB-GM source

# 3.0 Methodology

## 3.1 Overview

In order to investigate the research question fully, it was mandatory to create a prototype application that contained a static, man-made level and had the ability to generate a multitude of stable and reliable environments. The primary purpose of the application was to gather user-driven data in relation so the developed MB-GM can be utilised and compared within a static and generated environment. This application made use of Unity 3D and its C# scripting in order to supply the users with MB-GM and a series of grid-based generated levels. The user then controlled a playable character (PC) with access to all MB-GM and were tasked with navigating the level(s) with the aim of finding one of two hidden objectives. One objective was a simple objective to give less-experienced users a means to complete the level, and the second contained a platform-based objective that was raised and required the user to make use of the more advanced MB-GM to reach and trigger this objective. The overall aim is to gather real-player statistics which will be used to investigate and evaluate the use of PGL in movement-based video games.

## 3.2 Application

### 3.2.1 Application Design

The PCG within the prototype application was designed the traditional use of WFC. This means that the generation is calculated via a grid that runs along the X and Z axis and places nodes with attached models depending on the neighbouring and pre-existing nodes (propagation) (\_insert source here\_). Each grid tile contains a node, as previously mentioned, and each node contains a series of data within them. The data includes: a node name, a 3D model which represents the in-world asset (such as a wall, corner, tree or empty space), and a series of rules which drive the WFC, only allowing certain nodes/models to be places next to other, specific nodes. Once all grid slots have been assigned a node, the level grid will ‘collapse’ and produce a level.

## 3.3 Node and Model Design

### 3.3.1 Node Uses and Overall WFC Generation

The node is arguably the most important component of the WFC. This is due to the fact that the nodes are slotted into the grid when the world is generating and is the sole container for all data out with the main generating algorithm. Though, without said nodes, the main algorithm is rendered useless.

As mentioned in the overview, the node contains a multitude of different data. The first field contained within the scriptable object that is a node, is the node name. The node name is simply an identifier for the node, allowing the client/user to identify the node being used. The second piece of data contained within the node is the prefab game-object. This prefab is used as the physical body of the node, containing all in-world assets such as the ground, walls and extras (such as trees) which, by extension, contains all colliders and any interactive-based properties which the user may need in order to play the prototype. Lastly, the neighbouring relationships which act as the sole rules that the generation follows. The node contains relationships for neighbours north, east, south and west of the current node in question, and are labelled in-script as up, down, left and right.

The main algorithm starts off by setting up the fields and data required, especially the grid size which are exposed to the game engine as public fields. From this point, the Collapse() method is called – this is the main functionality behind the generation as a whole. The algorithm starts by iterating through the entire grid, cell by cell, and records the cells that still need to be assigned nodes and collapsed. Additionally, a list of potential nodes for each un-collapsed cell is initialised which is one of the main drivers of the algorithm’s propagation. From here, the iteration of the propagation and collapsing initiates, starting by ensuring the cell being checked is valid and within range of the grid before adding the cells neighbours to a list for later generation. The neighbour each node is then initialised in a recursive fashion, and each neighbour’s propagation is then pruned. Pruning is the action of removing the possibility of generating an incompatible neighbour node – driven by the rules within each node.

### 3.3.2 Propagation

Propagation is a fundamental aspect of the WFC algorithm. Propagation is what drives the generation of an already generated node’s neighbouring nodes (offsets). The propagation of a constantly updating possibility of what a neighbouring node could be. For example, if we have nodes X, Y and Z where X and Z as well as Y and Z are compatible – this means that the algorithm will actively neglect to generate nodes X and Y beside one and other, as per stated in the node rules. As each node is generated, the act of pruning occurs to keep the propagation up-to-date and reliable. To summarise, propagation is the act of updating the neighbouring possibilities of generated nodes, and pruning is the active removal of incompatible nodes.

### 3.3.3 Prefabs and Models

The main component of the WFC from the player’s perspective is the models to which are attached to the nodes. The model is the visible and interactive property of the working nodes and is what the player will have direct contact with. Within this prototype, there are a series of different models used, and these models are formatted as prefabs. Prefabs are a reusable and preconfigured game object template that is saved as an asset to be used within a Unity project. Prefabs can take on many forms ranging from enemy templates, player templates, system manager templates or, in the case of this project, objectives and in-world scenery including walls, floors and extras such as trees. The prototype’s prefabs take form of various world pieces, usually in the form of a floor and a type of wall. The wall types include straight walls (horizontal and vertical), parallel walls to encourage wall running and corner walls. All wall types include flipped variants as well, this way the algorithm was made easier, excluding a rotating algorithm with the addition of some extra node rules.

## 3.4 MB-GM Design

### 3.4.1 Wall Running and Sliding

Another main focus of this research is the MB-GM, as they will be closely paired with the PLG to research the relationship between them. The first of the MB-GM is the wall running mechanic, arguably the strongest of the gameplay mechanics and the mechanic with the strongest relationship the playable environment. The wall running mechanic has a couple of strongly defining features and provides the player with an alternative means of navigating the environment – and this goes for all games and applications that contains this mechanic, this prototype included.

The wall running mechanic was built up from Dani’s movement script[[1]](#footnote-1) as this movement considers momentum, interactivity and is a strong basis for building on top of. As the base movement script was split into 2 sections, the camera and the player camera, this allowed for some fine tuning and manipulation when adding additional features. For example, the wall running mechanic makes use of the camera being segregated from the main players body (being a child object) and allowed for camera tilting when wall mounted.

Starting with the basic movement script, there are many layers of depth to the scripting. Firstly, the script casts a series of checks to investigate whether the movement should behave a certain way, or some cases, not at all. These checks being: can the player jump and is the player currently standing on the ground layer. Behind the Movement() method, the player is applied a force downwards to add a more responsive gravity to the movement before calculating the magnitude of the player and setting each magnitude for the X and Y axis. This magnitude is then utilised alongside the player’s x and y position to calculate the friction of the player’s movement, this being used to help the player feel realistic when traveling fast as well as being well-flowing. Once the movement has been applied, the ‘is player grounded’, ‘is player jumping’ and ‘is player crouching’ are checked to ensure the movement doesn’t need alteration before clamping the player’s max speed and then applying the forces to move the player accordingly.

Player jump is a simple implementation, simply checking if the player is standing on a surface and is able to jump before adding an upwards force to the player’s rigid body and resetting the jump cooldown. Where the jump gets complicated is actually an addition to the script – where wall running functionality is implemented. The wall running implementation within the jump method checks, firstly, to ensure the player is not already wall running, and if it is found that they are not, the player is attached to the wall via a conditional check (checking for the direction of the wall to the players local position). Once attached, the player gains a forward force to add to the momentum-based mechanic.

The wall running component is where a majority of the functionality for this MB-GM is contained. Within this script, the wall is checked for using a left and right raycast and is attached or detached according to the result of the raycast finding a wall or not. Additionally, the input is read, ensuring that the correct inputs are inserted in order to connect the player to the wall. Lastly, the two methods responsible for starting and stopping the wall running which either ensures the player doesn’t exceed the clamped max speed, a forward force is applied, and the player is attached to the wall via a directional force towards to the wall – or the player dismounts the wall and normal gravity is reapplied to the player.

The player also has access to a slide mechanic which can be used to ‘bunny hop’ and if done correctly, can grant speed and maintain momentum.

### 3.4.2 Objective

The objective is the user’s main objective within the level and plays a crucial part in the user experience of each level. The objective gives the user a goal for when they are within the level, allowing them to navigate and use the MB-GM with a goal, avoiding having an aimless environment.

There are two main objective types: an on-ground objective and a parkour-accessible objective.

The on-ground objective is for less experienced users who may struggle to use the MB-GM effectively for whatever reason. This was added to ensure every user, despite their level of experience and exposure to video games, has a means of completing the level. The parkour-accessible objective was added as a means for more experienced player with prior experience of gaming. It was also added to encourage players of all levels of experience to utilise the MB-GM in order to complete each level. Without this additional objective type, users would have little-to-no incentive to use the MB-GM, which would directly oppose the research goal.

After the objective’s trigger is activated through direct contact with the user, the user is moved to a win-screen, with options to replay or access the main menu of the application. All levels generated and complete will be tracked, separately and this data is persistent across scenes. The level tracking is automatically increased by the trigger activation of one of the objectives, increasing the level complete tally before loading the win-screen. The reason behind why levels generated and levels complete are tracked separately is due to the possibility of a level being incompletable which, in turn, would prevent the user from ever reaching either objective.

# 4.0 Results

## 4.1 Questionnaire Data

## 4.2 Playtest Data

### 4.2.1 Player playtest data

### 4.2.2 Player generation data

## 5.0 Discussion

### 5.1 Playtest session

### 5.1.1 Questionnaire Data Analysis

### (may split into generation and player subsections again)

## 5.2 Project findings

### 5.2.1 Project summary

### 5.2.2 Research Question

## 5.3 Critical Evaluation

### 5.3.1 Development Evaluation

### 5.3.2 Playtest evaluation

# 5.0 Conclusion

## 5.1 Main conclusion

## 5.2 Implications

1. D, Devy. (2019) FPS ‘Movement Rigidbody’. Available at: https://github.com/DaniDevy/FPS\_Movement\_Rigidbody [↑](#footnote-ref-1)