Analysis and application of rhythm in the design of 2D platformer levels

University of Oulu
Information Processing Science
Master's Thesis
Joona Luodonpää
2023

Abstract

The video game industry has grown quickly from its humble beginnings to one of the largest entertainment industries in the world. Fuelled by the continuous advancements in technology, the quality and quantity of content in AAA video games continues to rise along with customer expectations. But with the ever-higher ambitions, the development budgets and durations rise with them, making the cycle unsustainable on the long run.

Procedural content generation is a technique that has the potential of helping break the cycle. The automatic generation of game content, such as levels, could help game developers reach the desired quantity of content with a fraction of the time and money required. However, commercial applications of procedural content generation so far have been largely limited in scope and lacking in quality, with the more successful cases being found in smaller budget indie games.

In this study, the possibility to use the idea of rhythm in guiding procedural level generation towards better quality was studied. Using a design science research approach, the gameplay rhythm of original Super Mario Bros. levels was extracted and used to build a rhythm-based procedural 2D platformer level generator. The nature of the generated levels was investigated by computational metrics, and the quality of them was evaluated by a series of playtests.

It was found that the existing platformer levels included an extractable rhythm. The rhythm-based level generator that was built upon the found rhythm data produced levels that were closely on par with the original levels, indicating that rhythm has potential applications in informing how a procedural content generator could create more meaningful and higher quality content. Finally, this experimental approach in incorporating music theory to procedural content generation opens up many interesting new avenues for future research.

Keywords
video games, procedural content generation, PCG, platform game, platformer, level design, rhythm

Supervisor
PhD, university lecturer Mikko Rajanen

Contents

Abbreviations

2D	Two-dimensional
3D	Three-dimensional
AAA	Triple A, a classification of high budget video game production
AI	Artificial intelligence
FPS	First-person shooter
GMA	Gestalt Music Analysis
NES	Nintendo Entertainment System
PCG	Procedural content generation

1. Introduction

Video games have risen over the years from their humble beginnings to become a massive industry worth almost $180 billion by 2020. It has even surpassed the movie industry's 2019 revenue of $100 billion, earning video games their place among the big entertainment industries. The constant growth has been fuelled by, for example, the multitude of ways to play games. From PCs and consoles to mobile devices and further to streaming, gaming has never been as accessible as it is now. (Witkowski, 2020). With advancements in technology, game developers continuously aim to reach new heights with bigger and better experiences and higher fidelity graphics, while players' expectations grow at the same pace. But this evolution does not come free as video games' budgets continue to rise and development times become longer and longer. As an example, Grand Theft Auto V, a highly anticipated and successful game by Rockstar Games, was developed by over 1000 people as a joint effort across multiple game studios with an estimated budget of $265 million (MCV Editors, 2013; Villapaz, 2013). Another highly anticipated game, Cyberpunk 2077 by CD Projekt Red, had its development team grow to over 500 people with its initial budget estimated to be around $330 million. The game was at least in partial development for eight years and even with all the effort put into it, it had a disastrous launch with the game being so broken that Sony, in a highly unusual move for an AAA game, had to remove it from the PlayStation store. (Andreadis, 2019; Lyons, 2021; MacDonald, 2020). Although the aforementioned games are some of the bigger examples that there are, it does highlight the huge amount of money and effort required to bring the biggest video game experiences to life. But just growing the development team and the budget is not enough to create a successful game, and spending many years on a game's development with no certainty of success is understandably not sustainable for a game studio.

Empowering game developers with better methods and tools could enable their vision to be reached quicker and cheaper, and is thus a more and more important avenue of research. This is where procedural content generation (PCG) comes in. Simply put, procedural content generation refers to the method of generating game content by algorithms as opposed to manual creation. The exact nature of the generated game content can be basically anything from the game's levels and non-player characters' artificial intelligence to entire game systems. (Togelius, Yannakakis, Stanley, & Browne, 2011). How much PCG is utilized, and how successfully, varies highly but at least a small amount of procedural content can likely be found from a majority of modern games. A highly successful application of PCG is the critically acclaimed game Minecraft by Mojang. The game is highly centred around procedural content generation with its whole world being created pseudorandomly on the fly, making each new world feel fresh to play. By 2020, the game had surpassed 200 million sales. (*Minecraft*, 2011; Warren, 2020). Although procedural content generation may sound lucrative, it is not without its challenges. For example, the game No Man's Sky by Hello Games also revolves around procedural world generation, but it released to generally negative response with a lot of negative feedback being directed towards the repetitive nature of the procedural worlds (*No Man's Sky*, 2016). PCG may not be a silver bullet to completely solve the issue of ever-increasing development time and cost, but it is potentially a powerful tool in alleviating the situation. Its efficient application in commercial projects warrants further studies, however. As there are risks associated, PCG hasn't been too widely used in AAA games. The aforementioned Minecraft and No Man's Sky both come from indie studios, which usually have more freedom and desire to experiment with new technologies. For

larger budget games, more robust studies could help in the adoption and understanding of how PCG can be used consistently and effectively in game development.

This study is implemented using a design science research process as defined by Peffers et al. (2007). The objective is to investigate if and how the idea of rhythm could be used to guide the automatic generation of 2D platformer levels. A technique based on music theory is used to analyse the rhythm of an existing game's levels, and the results of that are used to implement a procedural platformer level generator. The target game for the analysis and generation of levels is the classic platformer Super Mario Bros. (1985) by Nintendo due to its simplicity. The ways to interact with the game are limited to only some core gameplay mechanics of the genre, namely: walking, running, jumping, and crouching. The following research questions are used to guide the research:

> **RQ1**. How can the rhythm of player input in 2D platformer levels be analysed?
>
> **RQ2**. What kind of results can be achieved by creating new 2D platformer levels based on the rhythm of existing levels?

Existing literature is used to answer the first research question. Different aspects relating to platformer gameplay are associated with elements of music and rhythm. The resulting connections are then analysed with methods based in music analysis to produce coherent patterns of rhythm from the original Super Mario Bros. levels.

For the second research question, a rhythm-based procedural platformer level generator is built by feeding it the extracted rhythm information of the original Super Mario Bros. levels. The level generator's output is investigated through a combination of computational evaluation and user evaluation. Evaluation based on computational metrics is used to gain insight into the nature of the rhythm-based level generator and what kind of levels it produces. Playtest sessions are arranged to investigate the quality of the generated levels by comparing the original Super Mario Bros. levels to the generated levels. A handful of levels are played in each session, with each played level rated according to multiple aspects.

The rest of this thesis is structured as follows. Chapter 2 introduces platform games, the related core concepts, and the Super Mario Bros. game used in the study. Chapter 3 examines how rhythm can be found and used in the context of digital interaction, and especially in games. Chapter 4 details procedural content generation in general and in 2D platformers. Chapter 5 provides an overview of the study's research methods, and Chapter 6 explains the implementation of the study in more detail while also presenting the findings. Chapter 7 discusses the implications of the findings in the context of existing research, and the concluding Chapter 8 provides a summary of the results, the limitations of the study, and recommendations for future research.

2. Platform Games

Platform games, which are also referred to as platformers, is a genre of video games that can be defined simply as games where a player controls a character on screen by running, jumping, and climbing between platforms with the goal of reaching a specific destination. Due to the genre's age, platform games started as two-dimensional (2D) games, but three-dimensional (3D) platform games have become at least as popular nowadays. (Hosch, n.d.; "Platform Game," n.d.). Platformer gameplay mechanics are also highly prevalent in games where the main genre is something else (Helgeson, 2011).

2.1 History

The platformer genre is one of the oldest, and one of the most popular, video game genres with the first platform games being released in the early 1980s. The game that started the platform genre is a debated topic among historians and gamers alike, but the top contenders are the 1980 video game Space Panic and the 1981 Nintendo game Donkey Kong. The reason for the contention is that the former does not include the ability to jump as opposed to the latter, and jumping is considered a defining feature of the genre by many. (Hosch, n.d.; Klappenbach, 2021).

Regardless of which game came first, the first platformers were games played on a single static screen, meaning that the camera does not move. What was displayed on the screen was the whole play area. In these early games, the player jumps and climbs towards their objective, and when they reach that objective, the controlled character is transported to another screen with a possibly new area and objective. The natural evolution from these single screen platformers were the later scrolling platformers where the screen would show only a portion of the whole playable area. The camera would move horizontally or vertically around the area to reveal more of it in response to player character movement. The basic idea still remained the same: reach the objective while collecting items and avoiding enemies, and move through different areas to harder challenges. (Klappenbach, 2021). As an example: the genre classics, Super Mario Bros. and Sonic the Hedgehog, both fall into this scrolling platformer category (*Sonic the Hedgehog*, 1991; *Super Mario Bros.*, 1985).

With hardware growing more powerful, 3D games were made a reality during the 1990s. And so did platformers, too, make the jump to the third dimension with games like Super Mario Bros 64, Banjo-Kazooie, and Crash Bandicoot being some of the highly successful pioneers in translating the classically 2D genre into 3D. At the same time, more mature game genres, such as first-person shooters (FPS) and real-time strategy games started to rise and eclipse the platformer genre. With a wider variety of genres and more complex themes, the popularity of platform games started to decline over the years. (Braun, 2018). Nowadays, platformers have largely been relegated to an inspiration or a piece of a bigger whole for games of other genres, as is the case in many blockbuster games like Uncharted and Assassin's Creed. Games with platforming as the main focus have become quite rare. (Helgeson, 2011). Though they never regained the same kind of dominating popularity they once had, highly successful pure platform games do still keep getting developed and released from time to time. One of the largest AAA companies still producing platformers on the regular is Nintendo. Through their games like Super Mario Galaxy and Super Mario Odyssey, they show their mastery over the genre time and time again. The genre is more popular, though, with indie companies. Games such as Celeste, Shovel Knight,

and Super Meat Boy are proof that there is still a wide demand and also room for evolution in the 2D platformer genre. (Braun, 2018).

2.2 Gameplay

Platformer is a genre of games where movement in different manners is the main focus. The basic shared gameplay mechanic of platformers revolves around jumping and running across platforms to complete an objective. (Klappenbach, 2021). The vast majority of the time, the gameplay happens from the third-person perspective, meaning the player exists outside the game world and is looking at the player character. Contrasted to this, there are also first-person platformers like Mirror's Edge where the game world is viewed through the eyes of the player character. (Easton, 2019).

Platform games are usually divided into levels, which is a term referring to the area where the gameplay happens. Levels can have their own goals and themes, though often different levels share characteristics with each other. (Schell, 2014). Completing the level's main objective, which usually revolves around reaching a specific place in the level, transports the player to the next level with new challenges. Sub-objectives also often exist, either implicit or explicit. It is, for example, customary to have collectable items like coins scattered around the levels to entice the players to explore outside the main path, too. The reward for collecting such items may simply be an accumulation to the score meter that describes how well the player fared through the level, but it can also be something more. Another common type of collectable item, the power-up, is also commonplace in all kinds of platform games. When collected, the power-ups can grant the player character new gameplay abilities either temporarily or permanently. (Easton, 2019; Klappenbach, 2021; Schell, 2014).

In addition to collectable things, platformer levels also include things the player will want to avoid. Some of these include enemy characters, and environmental obstacles like spikes and fatal pitfalls. Enemies can usually be avoided completely, and sometimes they can be wiped out with specific gameplay moves or by using power-ups. (Klappenbach, 2021; Schell, 2014). Coming into contact with enemies or environmental obstacles will ultimately cause the player to lose a life. When the player dies, they will come back to life and be teleported to an earlier place in the level. This concept of dying and rebirthing, which is also called respawning in the context of video games, is central in platform games, and the exact nature of it varies from game to game. Some games or obstacles may cause the player character to die instantly, while others may only reduce the player's health. The usage of health points in video games enables the player to be able to make some mistakes, making the game more forgiving. If the health points end up reaching zero, the player character will die. Upon death, the player character is then usually teleported to an earlier checkpoint, the beginning of the level, or even the beginning of the game in some cases. (Melcer & Cuerdo, 2020).

Platformer games have seen a large variety of different movement related mechanics throughout the years. Some of these are widely used in the games of the genre, for example: rolling, dashing, double jumping, and wall jumping. There are also unique game-specific movement mechanics that can fundamentally change how the platforming is approached. The Super Mario game series is especially known for having unique mechanics in various releases. These include the planets and gravity changes in Super Mario Galaxy, and the ability to use Mario's hat as a throwable, temporary platform from which to jump further in Super Mario Odyssey. (Stewart, 2013).

There are also various subgenres to the platformer genre. Cinematic platformers like Limbo tend to be more grounded in reality and have a more realistic and slower movement. In puzzle-platformers, too, the movement mechanics often take a backseat in favour of various unique puzzle-based mechanics. (Vole, 2014). For example, in the indie puzzle-platformer Braid, the player has the ability to rewind time in various ways (*Braid*, 2008), and in Fez the player moves in a two-dimensional space, but the world is actually three-dimensional and can be rotated to unveil and play on the other sides of that 3D world (*Fez*, 2012).

2.3 2D platformer levels

Platformer games are often divided into multiple different spaces, called levels, where the player controls their character. Each level has its own objective that usually involves reaching a specific place. (Schell, 2014). After the completion of a level, the player is usually presented with another one. The levels may be connected to each other, for example, thematically as is the case in the 1991 game Sonic the Hedgehog. In Sonic, the levels are divided into what the game calls "zones", and each zone is further divided into acts like in a play. Each act corresponds to one level, and each act within a zone employs the same visual theme. (Schell, 2014; *Sonic the Hedgehog*, 1991).

The design of platformer levels is dependent on various different factors. For platform games, things like its subgenre, whether the game utilizes side-scrolling or static single screen levels, and how the level geometry is implemented technologically can all fundamentally change how the levels are designed. Figure 1 demonstrates the different look and feel that platformers can have due to differences in the aforementioned choices.

Figure 1. Left: Celeste, a pure platformer with tile-based single screen levels (*Celeste*, 2018). Right: Braid, a puzzle-platformer with smooth side-scrolling levels (*Braid*, 2008).

Single screen levels, like the name implies, are played with a static camera. The player sees the whole of the level at all times. Scrolling levels, on the other hand, show only a portion of the game level at a time. As the player moves, the camera follows. The camera may move in the horizontal direction, which is referred to as side-scrolling, or in more rare cases it can be vertical. (Klappenbach, 2021). In Figure 1, examples of both static and side-scrolling levels are shown. In Celeste, levels are usually presented with a static camera, whereas Braid utilizes side-scrolling levels where the camera can move in both horizontal and vertical directions (*Braid*, 2008; *Celeste*, 2018).

Two-dimensional platform games have utilized many different approaches for the technical implementation of the level geometry like the ground where the player can walk. One approach is the tile-based one. In this approach, a texture atlas which is also commonly referred to as the tileset, is created to contain the levels' building blocks. This tileset is a collection of images laid out in a grid of same sized cells. These images, called

tiles, can then be used to build the levels. (Monteiro, 2012; Wolf, 2012). An example of a tileset can be seen in Figure 2.

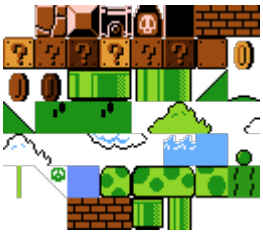

Figure 2. The tileset distributed with the Mario AI Framework that it uses to visualize the Super Mario Bros. levels (Khalifa, 2019).

Each tile in a tileset is the same size in pixels. In Figure 2, each tile is 16 pixels in both width and height. Those tiles are laid out on the screen next to each other to form a tile map, the graphical representation of the level. Each tile can, and usually does, appear in the level multiple times. For example, in order to create a fence, a tile image of a small section of the fence is repeated in a row for the desired length. While the tile-based approach may be easier to implement compared to other methods, it also imposes some restrictions to the design of the levels. Tile maps don't work very well with curves since the collision information, which defines where the player character can move, is often directly defined by the used tiles. For example: in Figure 2 the first tile represents the sky, through which the player character is able to move. The second tile in the first row on the other hand is the ground with which the player character would collide. The player can move on top of the ground tile, but they cannot pass through it. (Wolf, 2012). It is possible to implement curve-based collision on tile-based levels, too, but it requires more work and as such, it is an approach that's much less common. (Monteiro, 2012). A modern example of tile-based levels can be seen in Figure 1 with Celeste.

For a smoother look and movement, mask images or a vectorial approach can be considered. For the mask image approach, an image containing the graphical representation of the whole level is first created. Then another version of that image with simpler colours, the mask image, is used for collisions information. The mask image can be, for example, simply black-and-white where each black pixel in the image is considered to be impassable. In the vectorial approach lines or polygons determine the boundary for the collisions. This is something used in Braid, resulting in the smooth level geometry as shown in Figure 1. These kinds of approaches offer much more artistic freedom to represent unique shapes that are not always possible in the tile-based approach. (Monteiro, 2012).

In addition to the ground geometry, platform games, as the genre name implies, also have platforms on top of which the player can jump. There are many different types of platforms and other building blocks that can be seen frequently in different games. Some examples of platforms can be seen in the leftmost screenshot in Figure 3 where the player character is able to jump on top the platform comprised of the brick and question mark blocks. Platforms can have special behaviour or rules. For example, some platforms allow the player jump through them from below, but are impassable when the player is on top of them. Moving platforms, like the orange beam in the rightmost screenshot of Figure 3, are constantly on the move, and in the gridlike tile-based levels they are implemented outside the tile system for more smooth movement. Some other platforms have the ability to modify player movement; for example, a conveyor belt platform that is constantly moving player in one direction when making contact with it. Even more types of commonly used platforms are: platforms that are visible and able to be jumped on only

periodically, and platforms that disappear when standing on them for too long. (iD Tech, 2012). Stairs, ladders, and other climbable objects like the fence in the Braid screenshot of Figure 1, are other widely present building blocks used in platformer levels (Monteiro, 2012). While platforms and many other level components help the player traverse the level, obstacles on the other hand are there to hinder player movement and provide challenge. Obstacles can be environmental, like gaps in the ground or spikes that must be jumped over, or they can also be hostile non-player characters. These hostile entities, usually referred to as enemies, have their own specific behaviours and rules that determine how they move and how they can be defeated. (Bright, 2014).

2.4 Super Mario Bros. for NES

Super Mario Bros. is a 2D side-scrolling platform game developed and released by Nintendo in 1985 for the Nintendo Entertainment System (NES). The game revolves around two Italian plumber brothers Mario and Luigi who are on a mission to rescue Princess Toadstool who was kidnapped by the game's main antagonist King Bowser. The game is widely considered as one of the gaming classics with over 40 million copies sold. On top of that, it was largely responsible in reviving the North American gaming industry following its crash in 1983. Even nowadays, Mario is one of Nintendo's most valuable characters and intellectual properties. New entries in the game series continue to be released every now and then with the focus having switched mostly from two-dimensional to three-dimensional platforming. (Goh, 2016; "Super Mario Bros," n.d.).

The levels in the NES game are divided into eight sets of four levels each. Each set of four levels is referred to as a "world", meaning for example World 2-1 refers to the first level in the second world; or the fifth level of the game overall. The levels are played with a side-scrolling view where the camera follows the player character. The main objective of most levels is to simply reach the end that is indicated by a flagpole the player jumps on. The last level of each world culminates in a short single-screen boss battle against King Bowser as demonstrated in the rightmost screenshot of Figure 3. The secondary optional objective for the player is to accumulate as many points as they can. Most actions, like defeating an enemy and acquiring the optional collectables like coins, reward the player with more points. A major point boost can be achieved by jumping higher on the end-of-level flagpole; a feat that can be achieved by utilizing the blocks and platforms that are placed before said flagpole. (*Super Mario Bros.*, 1985).

There is a small variety of different themes in the available levels. The vast majority of the levels are similar to the leftmost level in Figure 3. A couple of the game's levels are underwater levels where the movement is temporarily fundamentally changed from simple running and jumping to freely swimming across the screen. Some levels are completely or partially underground. The boss battle levels are similar to the regular levels for the most part with some unique environmental obstacles and the end-of-level boss battle section added in. The levels are tile-based with no curves in collision detection, meaning each block is either completely passable or it completely blocks movement through it. A lot of different types of platforms make up the levels, and often they are arranged in a way to provide branching paths for the player to take as demonstrated in Figure 3. Additional elements in the levels include moving platforms, trampolines, and more. (*Super Mario Bros.*, 1985).

Figure 3. Screen captures of the different types of Super Mario Bros. level types. From left to right: a traditional platformer level, an underwater level, and a boss battle section. (*Super Mario Bros.*, 1985).

The NES Super Mario Bros. is a pure platformer in that its main focus is on movement. The movement mechanics are relatively simple: the main things the player can do are walk, run, crouch, and jump. In the underwater level the behaviour of the mechanics changes quite a bit, but the player is still in essence just moving left and right (walking and running), and moving up and down (jumping). Jumps have a variable height depending on how long the jump button is held down. In other words: tapping the jump button will perform a small jump, and holding the button down will cause the jump go higher, up to a predetermined maximum height. Jumping while running enables the player character to cross larger distances. (Goh, 2016; *Super Mario Bros.*, 1985).

A wide variety of different enemies and environmental obstacles exist in the game to challenge the player. Enemies have different movement patterns and different rules for defeating them. For example: the first enemy seen in the leftmost screenshot of Figure 3 can be defeated by simply jumping on top of it. Coming into contact with enemies or environmental hazards, like pitfalls or spinning bars of fire, will cause the player character to die. The player has a set number of lives, one of which is spent upon each death. If the player still had lives left on death, they will respawn to the beginning of the current level. But should the player run out of lives, they will have to start again all the way back from the beginning of the game. Additional lives can be acquired by finding specific power-ups, or by gathering one hundred of the collectable coins. Another power-up, the mushroom, on first pickup grows the Mario character from small to big state. The big state acts as a buffer against hazards: while it is active, the player can come into contact with some hazards and enemies once, and instead of death they'll only be returned back to the small state. Some hazards like pitfalls will cause death no matter the power-up state. While in the big state, mushroom power-ups are transformed into fire flowers. The fire flowers give the player the ability to shoot fireballs for a short while, which in turns allows the player to defeat enemies from a distance with more safety. (Goh, 2016; *Super Mario Bros.*, 1985).

3. Rhythm and Games

From living beings' biological rhythm to various types of human expression like music, poetry, and play, rhythm is something that is felt and experienced all the time whether consciously or not. There's even rhythm in this text. Some sentences are short. Others are long, causing a different rhythmic feeling that can be adjusted with punctuation; all of which can carry its own implicit message to the reader. (Costello, 2018a).

Rhythm is in essence a pattern of some specific event's occurrences in relation to time. In music, rhythm happens in the context of a song's tempo. Tempo is the speed of the song that is measured in beats per minute. The regularly occurring beats are the pulse of the song, and it's that repeating beat which listeners often feel and replicate by, for example, tapping their foot. The events that form the rhythm are the actual notes that ring out from an instrument, and they may happen on the beat or outside it. (Costello, 2018a; *Rhythm*, n.d.).

While the notes' placements in relation to each other in the context of the song's tempo are what form the rhythm, notes also have a pitch associated with them. Pitch refers to the frequency of the played note that defines how high or low the listener experiences the sound to be. Playing the same note on a piano two times a second, and alternating between two different notes at the same speed of two notes per second, both produce the same rhythm but a different experience to the listener. Continuing the same example, the pianist could also alternate between a quick tap and holding down the piano key at the same tempo. The duration between each note stays the same, meaning the rhythm stays the same, but the experienced sound is different as every other note is heard for only a brief moment, and the others are heard for a longer time. (Costello, 2018a; *Rhythm*, n.d.)

Rhythm has powerful expressive capabilities as different rhythms can induce various types of emotional responses. Some rhythms may be perceived as soothing, while others may cause joy, anxiety, or even sadness. It can also keep the person who experiences the rhythm engaged, or cause a loss of interest depending on how it's used. In the context of performing a rhythm, it can be an invaluable tool in enabling creativity. (Costello, 2018a).

3.1 Rhythm in games

Costello (2018a) in her book examines rhythm in the context of digital interactions with the main focus being playful applications like video games. The author interviews eighteen experts of varying fields like dance, film, art, and interaction design. For each interviewee, rhythm plays an important role in their work, and it is their unique perspectives the author strived to understand and apply to digital interactions. The result is a book full of insight and guidelines that could potentially help improve human-computer interaction, be it video games or more traditional applications. (Costello, 2018a).

Rhythm is something that is experienced, but also something that is produced. It is ever-present in human-computer interaction, and perhaps even more so in games than in regular software. In games, depending on the interaction device, the players may click, hold, and gesture to control their avatar in the virtual world. Specific events happen as a response, and the chain of these interactions and their responses results in a rhythm. This all happens in the context of the game world and its rules, meaning different games could allow different rhythms to be performed. On the other hand, there is an implicit rhythm

in the gameplay itself to which the player can attune themself. As an example of this in another context, think of an audience clapping. Each individual person produces their own rhythm when they start. However, as they listen to others clapping, they usually attune to the rhythm they feel from other people clapping, ultimately causing the audience to synchronize their rhythm. (Costello, 2018a).

Games have music, and the rhythm of that music can be an important tool for improving player experience and game feel. It can create the feeling of tension and difficulty, or on the other hand soothe the player in calmer gameplay situations. It can even unconsciously direct the player how they need to interact with the game as players may find themselves, for example, jumping synchronized to the beats in the music. Achieving such an experience requires an intentional design, however. The player can keep jumping synchronized to the music only if the gameplay rules and the level design allow them to do so. A mismatch between the experienced inherent rhythm of the game and the rhythm required to interact with the game to proceed can cause a highly distracting negative dissonance. A fast and frantic background music may not be suitable to a simple, visually peaceful, platformer level with little obstacles as there is no need to interact with the game in any way close to the inherent rhythm caused by the music. (Costello, 2018a, 2018b).

Games as interactive experiences are also full of rhythmic experiences outside the music in both active and passive sense. When the player is navigating through the menus, there is an implicit rhythm since the menu item selection can be changed only in specific minimum intervals. When the player is controlling their avatar and constantly jumping, there is an implicit rhythm they may end up following in the key presses because the rules of the game, specifically gravity, control how long the player stays in air and thus how often they come back to ground to be able to jump again. The cycle of death and respawn also provides an abstract kind of rhythm, and its intensity rises as the player feels more pressure the closer they are to losing their last life. When the player is exploring the world, there is rhythm in how often they come across different points of interest. A possible day and night cycle may also provide a steady overarching rhythm like it does in Minecraft where the enemies are more active during the nights. There is a designed rhythm in how often the player unlocks new skills and abilities. There is even rhythm in the story, in how its events play out and how the intensity of each story event varies. Some of these examples happen at a very low level where the player is directly interacting and causing the rhythm, while others are more abstract, high-level rhythms. Viewing the different kinds of game systems and components through the lens of rhythm can help design with more intent, and to reach planned results in a more consistent manner. (Costello, 2018a, 2018b).

While macrolevel rhythms like the pacing of the story's plot points may be more straightforward, the rhythm at the microlevel is more heavily affect by both the interaction and the response. The story simply happens, it can seldom be affected directly. The rhythm of controlling the player character, however, is the result of both the player input and the game's response to that input. Costello (2018b) explores this by comparing the player interaction when chopping trees in the games Minecraft and Don't Starve. The key finding by Costello (2018b) is that the game's response in Minecraft is more closely tied to the player's interaction. In Minecraft, the player has to hold down the mouse button for longer periods of time when chopping the tree. If the player lets go of the mouse button, the character immediately stops their action. In Don't Starve, on the other hand, instead of holding down the mouse button, the player just clicks it and waits for the action to end. In other words, the game's response continues for some period of time after the player has ended the interaction. (Costello, 2018b).

The movement in Don't Starve also has a longer response to interaction. When the player clicks somewhere in the game world, the player character starts automatically navigating to that point. During this, the player simply waits for the character to reach its destination. In Minecraft, the player character is controlled more directly with keyboard keys: when the correct key is pressed down, the character starts moving in the corresponding direction, and continues to do so only while the key is pressed down. The prolonged response in Don't Starve causes the player to feel like they were only triggering rhythms instead of performing them. In Minecraft the player feels they have more control due to the faster interaction-response cycle, and thus it can result in a more rhythmically expressive gameplay. It is to be noted, however, that having rhythmically expressive gameplay is not necessary in every game, but it can be useful in enhancing specific type of gameplay. Different behaviours suit different games, and it's important for the game designer to understand the consequences of their decisions. (Costello, 2018b).

3.2 Rhythm intensity

When aware of the different types of rhythm that exist, a designer needs to then understand what are the events that happen in the rhythm, and how to evaluate their intensity. In the rhythm of a movie's storytelling, the events are the scenes which are constructed from many components like the acting, writing, and camerawork. Each scene has a different intended emotional impact that each of the components work towards. The scenes alter in intensity from minor to major events at varying frequencies. It is then the filmmaker's job to chart out a suitable rhythmic structure of these varyingly impactful scenes. Having an entertainment experience consist of only high intensity events, the audience will quickly tire from sensory overload. On the other hand, constant low intensity events can have the same effect of tiring out the audience, but in this case, it would be due to boredom. As such, it is important to have the intensity and the frequency of the events be more dynamic. (Costello, 2018a). One way to chart the intensities of an experience's events is with interest curves (Schell, 2014), an example of which can be seen in Figure 4.

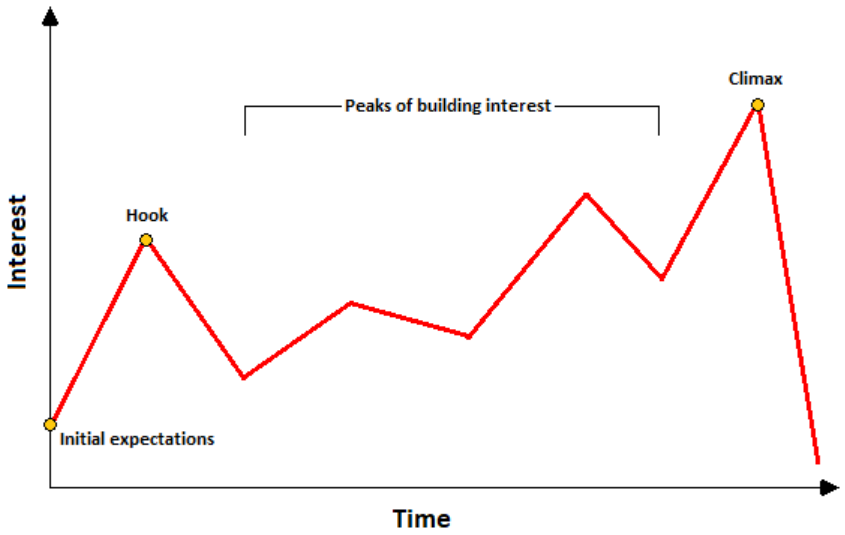

Figure 4. An example of a good interest curve based on the description by Schell (2014).

Schell (2014), in his widely referenced book about game design, likens all entertainment experiences to a sequence of moments. Each moment has a level of impact on the person who experiences it. Designing the experience around these moments and arranging the moments in a suitable order is the key to an enjoyable and engaging experience. Schell

(2014) suggests an example of a good sequence of moments, an example visualization of which is charted in Figure 4. At the start of an entertainment experience, the audience already has some level of interest due to their initial expectations. Very early in the beginning a moment with high level of interest happens. This is referred to as the "hook", and it's used to grab the attention of the audience by having something highly exciting happen. The resulting boost in interest helps keep the audience engaged through the middle that is filled with moments of varying levels of interest. During this time in the example, the interest gradually builds up to the climax, a highly impactful moment. In the end it all comes to a close by winding down and wrapping up the experience. (Schell, 2014).

The interest curve shown in Figure 4 is just one example, though it is useful in conveying the main points of a good experience. The experience should be dynamic in its intensity, and it should gradually build up interest before finishing up in a climactic event. Other types of curves can be effective, too, but this particular example can be observed to be widely used in all kind of entertainment whether on purpose or not. Similar curves can be seen, for example, in the widely used three-act structure of Hollywood movies, and in the structure of popular songs. It is to be noted, however, that there can be multiple layered levels of interest curves in entertainment experiences. In the context of games, the top-level could be the overall story of the game. Going one level deeper, each level could have its own interest curve comprising where different types of challenges, for example puzzles or platforming sections, could cause different levels of interest. Finally, each challenge in itself could be designed around an interest curve with the challenge being presented interestingly, and continuing with gradual and dynamic build up to a big finish. This kind of perspective into the design of entertainment experiences can be helpful not only when charting the expected interest in the beginning, but also in confirming later in the development process that the implemented moments actually have the designed impact at correct points in time. (Schell, 2014).

When it comes to designing the interest curves, the hard part can be to define what interest is exactly. What interests different people is inherently a subjective thing, meaning we cannot really measure it or objectively design it. Instead, we can define the interest levels in a relative manner; using the designer's experiences, empathy, and imagination to compare different events. Three example factors of interest that can be used as a guideline are inherent interest, poetry of presentation, and projection. (Schell, 2014).

Inherent interest refers to the fact that some things are inherently more interesting than others. For example, it's easy to see that a person juggling chainsaws is inherently more interesting to see than a person juggling couple of regular juggling balls. From this example we can understand that risk is usually seen as more interesting than safety. Other examples could be: something being fancy can be seen as more interesting than plain things, and unusual things are usually inherently more interesting than ordinary things. An important thing to note is that the events may build upon others. This is especially true in story arcs where the more boring events, like in the fairy tale Goldilocks eating the three bears' porridge etc., are required in the build up to the more interesting events that happen when the bears notice someone has been to their home. (Schell, 2014).

The poetry of presentation factor on the other hand refers to the aesthetics, which could be anything from writing and drawings to acting and music. For example, in movies the scenes rarely rely only on the inherent interest of the story being told through acting; instead, the beautiful scenery, camerawork, and music all together form the total interest of the scene. (Schell, 2014).

Lastly, the projection factor of interest is something more abstract that deals with things like whether the experience causes the person to get lost in it, to imagine things, or to empathize with a character of a story. For example, when watching a person juggling chainsaws, there is some level of projection as the watcher may imagine what would happen should the juggler catch the wrong end of the chainsaw. In the game of Tetris, projection can be very high as the player is free to make all the decisions, and the success depends wholly on those decisions. (Schell, 2014).

Interest curve and the idea of controlling the level of interest throughout the experience is highly related to pacing, which is a much more widely known concept when it comes to games. Pacing, too, can be used to refer to multiple different aspects of a game, like the story, the amount of action, and the difficulty. In FPS games, one aspect of difficulty pacing could be how many and how often enemies appear. (Bleszinski, 2000; Schell, 2014). In platformer levels, the notion of rhythm is often used to guide the level design (Lindley, 2002) which is apparent in the pacing of the obstacles that need to be avoided, meaning how they are placed in rhythmically varying densities (Bleszinski, 2000).

Difficulty can also be measured by the number of deaths players on average have on each level, which in turn is directly tied to many things like the aforementioned enemy count, pacing, and number of obstacles. Schell (2014) brings up the average death count from the critically acclaimed game Half-Life 2 Episode 1. In each difficulty level, the average number of deaths follows a curve relatively similar in shape to Figure 4. Notably, the same type of gradual and dynamic build up towards the end is present. The main difference is in the beginning of the curve. Whereas the interest curve in Figure 4 has a peak at the beginning, the death count curves do not include such a peak. Reason for it is that an early peak in difficulty could actively discourage the player from playing. (Schell, 2014). Keeping the difficulty at a suitably challenging level is a major positive contributor to the desirable flow experience (Nakamura & Csikszentmihalyi, 2002) which helps the player stay engaged through a sense of accomplishment after overcoming a challenging level (Nicollet, 2004).

3.3 Gestalt Music Analysis

Pagnutti (2016) a way of utilizing music analysis in platformer level design by adapting the idea of beats and rhythm into platformer gameplay. In this context, beat would correspond to, for example, a singular player interaction like jumping or moving. As the musical analysis of rhythm focuses on investigating what effects different variations have on the listener, so can different patterns of gameplay interactions elicit various reactions and feelings from the player. (Pagnutti, 2016).

A crucial mismatch between beat-based music theory and beats in platformer levels, however, is the fact that gameplay is not directly connected to time. Music has a tempo that grounds the beat occurrences into specific speed. For most games, the player can choose to interact with the game at any speed they desire, and they can usually even pause the game. To solve this, instead of analysing a continuous section like music with its specific tempo, Pagnutti (2016) suggests that platformer levels can be divided into smaller discrete sections where the player has to perform a specific set of interactions, a specific gameplay gestalt, to advance. (Pagnutti, 2016). Gestalt refers to a pattern of interaction where the single interactions are “so unified as a whole that it cannot be described merely as a sum of its parts”. In games, this pattern of interaction is something that the player performs in order to progress forward. (Lindley, 2002). In the case of platform games, a single gameplay gestalt could be a set of very specific consecutive jumps the player has

to make in order to progress through a certain type of a section in the level. (Pagnutti, 2016). A gameplay gestalt consists of perceptual, cognitive, and motor operations, and those specific parts may be able to be measured, helping analysis of different gameplays through the lens of gameplay gestalts (Lindley, 2002). Gestalt analysis is an established field, having been introduced in the 1930s. The novel approach Pagnutti (2016) presents for platformer level design analysis and design is based on Gestalt Music Analysis (GMA). (Pagnutti, 2016).

In GMA, the music is divided into smaller sections, internally consistent patterns called the gestalts. A song consists of multiple layers of gestalts, with the song itself being one gestalt that can be divided into smaller gestalts, which in turn can be subdivided multiple times. At the boundary of two sequential gestalts there is some type of change that causes the two sections to be separated into their own gestalts. It is, however, not an exact science how to define these gestalt patterns. The example given by Pagnutti (2016) for applying GMA in music is to calculate for each note a distance value from the previous note. (Pagnutti, 2016). A sample visualization of this can be seen in Figure 5.

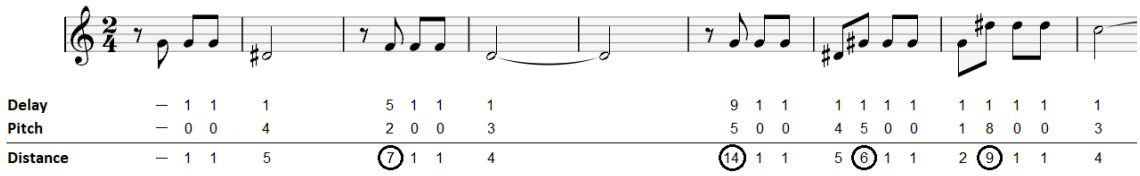

Figure 5. Example of GMA applied to the beginning of Beethoven's 5th symphony using the approach defined by Pagnutti (2016).

Pagnutti (2016) calculates the note's distance to the previous by the difference in delay and pitch. In Figure 5 the delay value of a note is in practice the length of the previous note measured in eighth notes. For example, in the figure the delay from the fourth to fifth note is five, meaning the duration of five eighth notes. The pitch value is the difference in the specific note played with zero meaning the same note is repeated. The delay and pitch values are simply added together to form the full distance value to the previous note. A new gestalt is considered to begin when a distance value is higher than both the previous and next distances. (Pagnutti, 2016). In Figure 5 the beginning of new gestalt is denoted by the numbered distance value, meaning this example can be divided into five separate gestalts.

Each note's duration and pitch interval can be thought to be the state of that note. Pagnutti (2016) points out that a game's player character, too, has a specific state at all times. In the case of platformers, the player character may be standing still, jumping, moving, or even jumping while moving. Each of these options can be considered a distinct state. Based on the player's input, the state lasts for a certain period of time before changing to another. Pagnutti (2016) utilizes this viewpoint to analyse previously recorded gameplay traces from an existing dataset called Platformer Experience dataset. The dataset contains real players' gameplay data from Infinite Mario Bros, a slightly modified clone of Super Mario Bros, that shows the player character's states and their durations. For the purpose of GMA distance calculation, five parameters were chosen to act as the state of the player character. (Pagnutti, 2016). The parameters can be seen in the following list.

- Movement direction (left, right, none): changing state from moving to not moving, or vice versa, was assigned distance value of 1. Changing direction from left directly to right, or vice versa, results in distance of 2.
- Power-up state: there are multiple different power-ups. Some power-up state changes have a distance of 1, and some have a distance of 2.
- Ground state (crouching, running, none): changing state between running or crouching, and none state has a distance of 1, and changing state directly between running and crouching has distance of 2
- Airborne state (jumping, none): moving between jumping and being grounded has a distance score of 1
- Time: difference of the time when the compared states started

Using all of the five parameters, the total distances are calculated with Euclidean distance. Time was discovered to have a much larger effect on the result than the other parameters. This resulted in awkward gestalt boundaries, so weighting was applied to the parameters with time being adjusted the most by a factor of 0.01. Direction, ground, airborne, and power-up states were given weighting values of 0.75, 0.25, 0.5, and 1.0, respectively. The weighting improved the results, though it was noted there were still some imperfect gestalt groupings remaining. Additional issue found in the Platformer Experience Dataset was that it lacks the context between the player input and the resulting state change. The model doesn't know, for example, why the player jumped, only that they did. Without the level geometry data, the resulting gestalts do not differentiate between changes in the level and changes in player's goals. This in turn may make gestalt analysis less effective, though it can still be useful for revealing interaction patterns. (Pagnutti, 2016).

It is to be noted that the gestalts may evolve or change as the player makes progress through the game. By playing the game, the player learns the rules set by the game. Through new understanding and interpretation of those rules, the performed gestalts may change even if the context where the gestalt is performed is the same as previously. In addition, if the gameplay rules change, for example, due to the game introducing new gameplay elements like movement abilities or enemy behaviours, so then will the gestalts change. (Pagnutti, 2016).

4. Procedural Content Generation

Procedural content generation (PCG) can be defined, in short, as the process of automatic content generation with the usage of algorithms. Such algorithms have the responsibility, in addition to the actual generation process, to also distinguish and select the results that would be entertaining to the players. There is no strict exclusion of what can be procedurally generated. It can be, for example, anything from game levels to dialogue and music. As such there are a lot of different approaches that can be taken with some being more effective for different purposes. (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013; Togelius et al., 2011).

As a concrete example of PCG, a technique called displacement mapping can be used to create 3D landscapes from simple 2D images. A grayscale image, called heightmap, is created manually or programmatically using a noise algorithm, an example of which can be seen in Figure 6. The image can then be transformed into a 3D terrain algorithmically by treating each pixel of the heightmap image as a point in 3D space. More specifically: each pixel's colour corresponds to the height of the 3D point at that position with black being the lowest point and white being the highest point. An example result of the application of the aforementioned heightmap is shown on the right side of Figure 6. This process from automatic heightmap image creation to its application in 3D is a prime example of the potential that procedural content generation can have. (Shaker, Togelius, & Nelson, 2016).

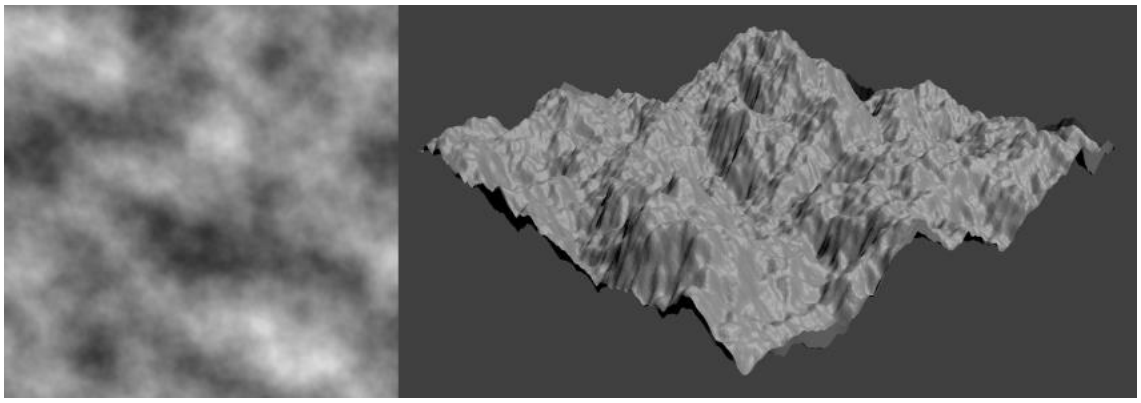

Figure 6. Left: heightmap created in GIMP using the Perlin Noise algorithm.
Right: one possible result of the heightmap visualization in 3D using Blender.

It is important to note, however, that although the content generation process may be automatic, it doesn't exclude game designer input. Depending on the type of content and the situation where it is used, it may be desirable to allow various degrees of customization for the user. (Hendrikx et al., 2013). For example, the resulting 3D representation of the heightmap in Figure 6 can look different based on potential input parameters. A commonly used one in displacement mapping would be the strength of displacement, meaning how much different colours actually affect the 3D points' positions. (Shaker, Togelius, et al., 2016).

4.1 Taxonomy

The research field of procedural content generation is quite fragmented, and it can be hard to compare results between different studies (Togelius, Champandard, et al., 2013). It is

also not uncommon for different studies to draw inspiration from a wide variety of different disciplines. For example, some of the categories proposed in the taxonomy of common PCG methods by Hendrikx et al. (2013) include generative grammar methods, which are rooted in linguistics theory, and genetic algorithms, which are inspired by the process of natural selection. (Hendrikx et al., 2013). Even quantum mechanics has served as an inspiration, for example, in a tile-based image and level generator based on the concept of wave function collapse (Gumin, 2016).

In one of the most widely referenced papers in the field, Togelius et al. (2011) propose a different type of taxonomy for procedural content generators. They surveyed a wide variety of published studies that dealt with automatic game content generation. Based on their findings, their proposed taxonomy is presented as a series of distinctions of various aspects of procedural generation. (Togelius et al., 2011).

The first distinction offered has to do with when the content generation is performed. Here, online refers to generation of content during gameplay, and offline refers to content generation during game development. These two approaches have highly differing requirements. Online generation requires the algorithm to perform fast with predictable outcome quality, while offline generation doesn't strictly have such requirements since a designer can confirm the results before they are included in the game. On the other hand, offline generation could potentially result in more interesting results due to its lesser focus on predictable outcome. (Togelius et al., 2011).

The second distinction has to do with the necessity of the content. Necessary content is required for advancing in the game while optional content is content that can be completely avoided or ignored. For necessary content there are additional requirements imposed, similar to online generation. For example: necessary content has to be correct to ensure the player can complete the game. (Togelius et al., 2011).

The third distinction deals with the degree of control over the algorithm. On one end, the algorithm's input can be just one number, referred to as the seed value, that is used to initialize the random number generator. On the other end, the algorithm's input may be comprised of a vast number of different parameters. For example, a dungeon generator's input might include parameters for room count, degree of branching for corridors, and so on. (Togelius et al., 2011).

The fourth distinction proposed by Togelius et al. (2011) is the level randomness. Deterministic algorithms output the same result always given a specific set of inputs. Stochastic algorithms' results, on the other hand, may vary wildly even if using the same set of inputs between different executions of the algorithm. Seed values can be used to enable creation of deterministic content. (Togelius et al., 2011). This is the case in Minecraft, which generates its world based on a single seed value. The benefit of this is that the seed values are easy to share between players, allowing different people to experience the same game worlds. (*Minecraft*, 2011).

The last distinction defines PCG algorithms as either constructive or generate-and-test. A constructive algorithm generates its content once but it ensures its quality at the same time. A generate-and-test type of algorithm on the other hand would produce candidates and test them according some criteria like completability. The failing results are discarded, and the algorithm continues until a result that fulfils the criteria is found. (Togelius et al., 2011).

4.2 Benefits

Procedural content generation can be a highly lucrative tool for game developers for a multitude of reasons. First, it can help keep memory consumption low. (Togelius et al., 2011). For example, the game Elite by Acornsoft managed to generate a large universe for the player to explore as early as 1984 with only tens of kilobytes of runtime memory used (*Elite*, 1984) The displacement mapping technique from Figure 6, on the other hand, can be used to reduce required disk storage as the resulting 3D object file would take more disk space than the 2D image file used to create it (Shaker, Togelius, et al., 2016).

Second major motivation for using procedural content generation is cost cutting. SpeedTree is an example of a commercial procedural content generation software. It is used to create large areas of vegetation algorithmically based on various input parameters. It is widely used in different industries from AAA games to Hollywood movie productions. While player expectations regarding quantity and quality of content continues to rise with the development of more and more powerful hardware, procedural content generation techniques have the potential to help in meeting the expectations while keeping costs manageable. (Togelius et al., 2011).

Third motivation for PCG as detailed by Togelius et al. (2011) is innovation. PCG is a powerful tool that has allowed the creation of completely new types of games and game experiences (Togelius et al., 2011). Minecraft is a prime example of this with its gameplay revolving around procedural generation. The whole world is procedurally generated, and the world is completely different each time the player decides to create a new one. This ability to start a different, fresh, experience is a major helpful factor in making the game as highly replayable as it is. (*Minecraft*, 2011). However, higher or even infinite replayability is just one of the more obvious examples of new kind of player experiences that PCG methods enable (Smith, 2014). Dynamically adapting the game to the player is another promising topic being researched (Shaker, Yannakakis, & Togelius, 2011). There have been attempts to automatically adjust the generated content's difficulty based on the player's skill, and even attempts to tailor the content to individual player preferences (Shaker et al., 2011; Yannakakis & Togelius, 2011). Some approaches have also included the player themselves in the decision-making process by letting them decide the direction the generator takes, shifting the focus from traditionally explicit narrative and goal to a more open-ended exploration experience (Smith, Gan, Othenin-Girard, & Whitehead, 2011). This is just the beginning, however, and as time goes on we can expect more and more innovation (Smith, 2014).

The final important motivation presented by Togelius et al. (2011) for the usage of PCG is that it can augment the game designer's imagination. Designers draw inspiration for their work from various sources. Similarly, the algorithmically created content can serve as an inspiration for new ideas in all kinds of game content. (Togelius et al., 2011).

4.3 Challenges

One of the main challenges in procedural content generation is that it is easy for the result to look generic and feel repetitive. It is naturally easier for designers to create, for example, levels with interesting details, levels that feel unique from one another, and levels that elicit specific emotional reactions. Even if a procedural generator achieves such results, there are further requirements like the levels needing to form a cohesive world and experience, which again can be easier for a human designer to control. Another common requirement is that the style between the generated levels should remain the

same. The style of the game can set it apart from others and make it more memorable and iconic to the players. Representing this kind of unique style consistently in procedural content generation is a challenge that requires extensive research and experimentation. (Togelius, Champandard, et al., 2013). Uninteresting and repetitive generated content was one of the biggest areas of criticism of the highly anticipated and heavily procedural game No Man's Sky during its launch (*No Man's Sky*, 2016). Modelling player experience and having that data influence the procedural generator is one promising approach to creating more meaningful content that could have the feeling of hand-created content (Shaker et al., 2011; Yannakakis & Togelius, 2011).

Procedural content generators are usually crafted with only a single type of game content as the objective. The more specific the generators' objectives are, the less reusable they are. Utilization of PCG methods especially in high budget AAA games has been relatively low. SpeedTree is one of the few more successfully used PCG tools but even it only focuses on the single, non-essential, content that is vegetation. Creation of more general and reusable content generators could help lessen the related risks and increase adoption of PCG techniques, and is thus one of the bigger researched challenges related to procedural content generation. (Togelius, Champandard, et al., 2013).

Increasing the influence designers can exert over the generated content is yet another challenge that can be important to solve. Togelius et al. (2013) argue that it is seldom enough to have a generator that just creates content with little to no input from the designer. Instead, it can be important for a level generator, for example, to take into account the desired difficulty to enforce good flow and enjoyability. (Togelius, Champandard, et al., 2013). On the other hand, there have been suggestions for future research to investigate the possibility of generating content with minimal input with the generator actually making design decisions similar to how actual game designers would. This kind of AI-intensive deep process should be able to rationalize its design decisions, and could potentially create completely new types of games and tools. Though this kind of approach is still in its early stages, it has the potential to offer interesting results even in the early stages. (Smith, 2014).

4.4 Generation of 2D platformer levels

In the field of procedural content generation, 2D platform games are one of the more researched genres. Especially regarding 2D platformer levels, numerous different approaches have been utilized in research. Below, some of these papers' approaches are accounted to demonstrate the diversity in the algorithms that can be used to generated platformer levels. So far there hasn't been, and likely never will be, a one-size-fits-all approach as games even within the same genre may vary wildly in many different aspects. Thus, it becomes important to acknowledge and ponder the different approaches' strengths and weaknesses to find content generation methods that support the made design choices. (Hendrikx et al., 2013; Togelius et al., 2011).

Compton and Mateas (2006) introduce a level generator for a custom 2D platformer engine where the focus is on representation of repetition, rhythm, and connectivity. They take a more methodical approach for level generation when compared to some other game genres' generators as they point out that platformer levels require more attention to their completability. This means that the more random nature of, for example, rogue-like level generation may be out of the question for platformers. Instead, the authors take a pattern-based approach. The patterns are created by combining components, such as platforms and little hills, and the patterns that fit with each other are further combined to form the

level itself. The authors hypothesize that repeating and shuffling even just a handful simpler patterns in the level can produce long and interesting levels, and thus helps reach one of the procedural content generation benefits of lessening the workload. However, the approach presented in the paper is partly theoretical as only the pattern building is implemented. (Compton & Mateas, 2006).

Smith, Treanor, Whitehead, and Mateas (2009) present a level generator for 2D platformers that creates levels based on consecutive rhythm groups. Their generator is a two-tier grammar-based approach where first the rhythmic representation is generated. The generated rhythm comprises of player actions, such as moving and jumping, and the timing of when said actions start and end. Given the generated rhythm, the generator in the second tier creates the actual level geometry that fits the rhythm. Multiple different geometry interpretations can fit the same rhythm. (Smith, Treanor, Whitehead, & Mateas, 2009).

The rhythm groups presented by Smith et al. (2009) roughly correspond to the idea of patterns utilized by Compton and Mateas (2006). However, where the first keeps its rhythm and level geometry generation separate, the latter paper has the rhythm implicitly contained in the geometry patterns themselves. The benefits for the two-tier approach by Smith et al. (2009) compared to the approach by Compton and Mateas (2006) include added variety in generated levels and also, on the other hand, ability to ensure that the intended rhythm exists in the level regardless of the geometric representation. In addition, the level generator by Smith et al. (2009) has the benefit of having more control over the generated levels as their approach is meant to support human designer's work instead of replacing it. (Compton & Mateas, 2006; Smith et al., 2009).

Mawhorter and Mateas (2010) share an approach to Super Mario Bros level generator using their Occupancy-Regulated Extension algorithm. The algorithm uses premade chunks of levels as input. One chunk is selected as the starting point for the generation. A random matching chunk is selected with a multi-step filtering logic. Finally, the chunks are integrated in the level. The chunks are chained until a level of desired length is created, with final post-processing step filling any gaps in level geometry. The algorithm works without knowledge of the mechanics of the game, which the authors state to potentially improve the generator's usability in various different games. (Mawhorter & Mateas, 2010).

A common issue in procedural content generators is the possibly generic and repetitive end result (Togelius, Champandard, et al., 2013). Incorporating player experience into the generator is one approach designed to make the results more interesting and varied. Pedersen, Togelius, and Yannakakis (2010) present promising results in their study where the level generator relatively successfully creates Mario levels that induce certain desired player experiences in the players. (Pedersen, Togelius, & Yannakakis, 2010). Related topic to this is dynamic difficulty adjustment based on player skill. As different players have different skill levels, dynamic adjustment of the game's difficulty can help in creating similar experiences for all kinds of players. (Jennings-Teats, Smith, & Wardrip-Fruin, 2010).

Kerssemakers, Tuxen, Togelius, and Yannakakis (2012), take procedural level generation to a new level. In their study, a procedural generator for the level generator itself was attempted. (Kerssemakers, Tuxen, Togelius, & Yannakakis, 2012). Their approach belongs to the genetic algorithms in the Hendrikx et al. (2013) taxonomy. The genetic algorithms, as the name suggests, take inspiration from the process of natural selection. First, a selection of candidate content is generated. Then, a so-called fitness function is

defined and used to evaluate the candidates' quality. Those deemed the most fit for the content generator's purpose are then mutated or evolved to produce new generations of content candidates until a defined stopping point. (Togelius et al., 2011). The goal for Kerssemakers et al. (2012) is to create a level generator for any game that has its level represented as a two-dimensional matrix. This goal turned out to be too ambitious, and focus was narrowed to only 2D platformer level generators. The resulting levels created by the level generators were not really commercial release quality, though the approach may have potential. Future research would be necessary to prove the suitability for more varied gameplay outside the Mario levels they created as the differences in gameplay, for example movement abilities the player character has, fundamentally affect the design of the level generator. It is also to be investigated whether this approach has any benefit over the traditional more straightforward approaches where the level generator is created directly and tailored to the needs of the game's design. (Kerssemakers et al., 2012).

4.5 Evaluation of 2D platformer levels

There exists a plethora of approaches to generating procedural content, with new approaches being tested constantly. In a sense, it is easy to generate content procedurally, but it is harder to have the generated content be valuable. One of the main challenges for procedural generation is to consistently create interesting, creative, and original content that has a purpose. (Shaker, Smith, & Yannakakis, 2016; Togelius, Champandard, et al., 2013).

Procedural content generators need to be evaluated to confirm that they reach the goals set for them. However, this is easier said than done. First of all, the generator has constraints that require it to tailor its results for different situations. For example, the game may have puzzle mechanics that the level generator needs to incorporate. Satisfying these constraints already adds value to the generator. However, within these constraints the generated content still needs to be of good quality, too. For platformer levels, subjective things like player preference and skill, or the unpredictability of player behaviour, are hard to translate into data that the algorithm can use to steer its behaviour. Unsurprisingly, the creativity of procedural content generators is a subject area that hasn't received much attention. (Shaker, Smith, et al., 2016).

Generally speaking, there exists two main approaches to evaluating procedural content generators: computational evaluation and user evaluation (Shaker, Smith, et al., 2016). Computational evaluation is performed by coming up with a way to quantify an aspect of the generated content, and performing necessary calculations automatically on the content. User studies can be used for evaluation purposes, but they have also been used for data collection to inform or teach procedural content generators on how to create content of good quality. (Mariño, Reis, & Lelis, 2015). Naturally, incorporating both approaches in the evaluation process can provide more comprehensive understanding of the procedural generator's nature and quality (Shaker, Smith, et al., 2016).

4.5.1 Computational evaluation

Computational evaluation can be especially useful in procedural content generation as the content generators are usually capable of producing thousands, millions, and even more, of unique results. To get a proper overview of the generator's qualities it is necessary to have a computational evaluation method that can handle the large amount of content. (Shaker, Smith, et al., 2016).

Expressive range is the most used computational evaluation method used for procedural 2D platformer level generators. As the name indicates, the expressive range of a generator refers to the style and variety of content it can produce, including any biases towards specific type of content. The expressive range is represented by calculated metrics. The data for the metrics is collected by running the content generator a large enough number of times to produce a representative sample. For each generated level, an algorithm is used to calculate that level's value for each metric. Various different metrics have been proposed to represent different aspects of a generator's expressive range. The original two metrics introduced at the same time as the concept of expressive range are linearity and leniency. (Smith & Whitehead, 2010).

Linearity metric can be thought to measure the height profile of the level. Levels with little height difference have a high linearity value, and others with more height variance have lower linearity. The metric is measured by performing linear regression on the generated levels with the positions of each platform as the data points. With the line defined, the level is scored by summing up the absolute distance of each platform position to the expected value on the line, and finally dividing by the total number of points. The result is finally normalized to the range [0, 1]. (Smith & Whitehead, 2010).

Leniency metric represents the forgiveness of the level to potential player mistakes. Different aspects of the level are assigned scores. For example: gaps and enemies have a score of +1, while jumps with no gaps have a score of -1. The value of the metric, also normalized to the [0, 1] range, provides an intuitive sense of how lenient the level is to the player based on the amount and danger of different obstacles. The authors specifically mention that the leniency metric doesn't directly represent the challenge of the level as there are other factors, like the relative positioning of the level obstacles, that the metric doesn't take into account. Thus, the authors opted for the term "leniency". (Smith & Whitehead, 2010).

Horn et al. (2014) implement the linearity and leniency metrics in the Mario AI Framework along with four other metrics. The authors compare various different 2D platformer level generators by calculating the metrics for each generator. The other metrics include the density, pattern density, pattern variation, and compression distance. The density metric measures the number of platforms there are in the level on top of which the player can stand. The pattern density metric measures the occurrences of patterns from the original Super Mario Bros. levels, and the pattern variation metric measures the unique occurrences those patterns. The compression distance metric works differently in that instead of being a value for a single level, it is calculated on a pair of levels to measure how different they are. (Horn, Dahlskog, Shaker, Smith, & Togelius, 2014).

The metrics alone provide some insight into the variety of levels generated, but it is important to also examine the relationship between different metrics. For example, if a generator has high linearity and leniency, the metrics alone do not provide information on whether the highly linear levels also have high leniency, or if the levels are usually highly linear or highly lenient but never both at the same time. Figure 7 demonstrates a 2D histogram, which is an effective way to visualize the expressive range and the relationship between different metrics. (Horn et al., 2014).

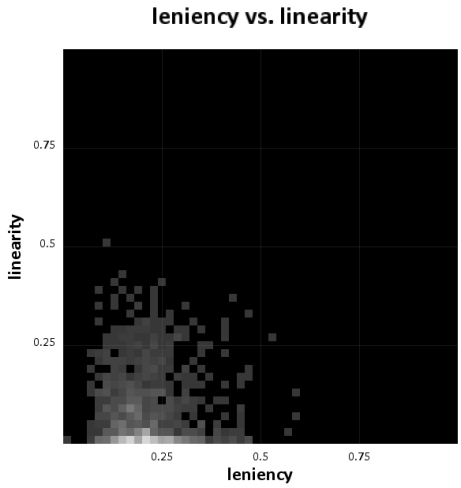

Figure 7. An example visualization of expressive range based on the leniency and linearity metrics.

In Figure 7, each dot represents a portion of the results, and the whiter the point, the more levels fell into those specific values. In this example by Horn et al. (2014), a total of 1000 levels' metrics were calculated. A point is completely white in their 2D histogram when 40 or more of those levels have the same values for the two compared metrics. The shape of the 2D histogram shows off the generative space of the generator, and provides a good visual way to compare different generators. Horn et al. (2014) have publicly released a package that contains the source code they used for both calculation and visualization of these metrics[1]. The package also contains all the text representations of the levels from their evaluations. (Horn et al., 2014).

The metrics leniency, linearity, density, and compression distance have been empirically tested to observe their validity and correlation to player-perceived quality of the visual aesthetics, enjoyment, and challenge of generated levels. It is noted that none of the metrics correlate with enjoyment as the players evaluated it, although this was expected as none of the metrics were specifically designed for measuring enjoyment. The quality of the visual aesthetics, distinguishing good results from bad, is also something the metrics are not able to capture. What the metrics can measure, however, is the diversity, which is also more what the metrics were designed to capture originally. Difficulty is also noted to not be measured well with current metrics as it is very hard to measure difficulty computationally due to the sheer number of related factors. Regardless of the potentially limited applicability, it is still argued that the computational metrics have a place in the implementation and evaluation of procedural content generators. The metrics provide a cheap and fast way to achieve statistical significance by calculating the metrics for a large number of generated levels. In addition, they provide insight that user studies would struggle to achieve. For example, verification of the diversity the generator can output would be impractical to achieve via user studies due to the large number of levels that would be needed to be played by the users. The common metrics can also be used to compare the nature of different generators, even though they don't help compare the quality between the generators. As computational evaluation in its current form is nowhere near enough to stand alone, at least when evaluating 2D platformer level generators, it is important to perform further evaluation with a more suitable method like

[1] http://sokath.com/fdg2014_pcg_evaluation/

user studies to gain insight into more important components of quality such as enjoyment, visual aesthetics, and challenge. (Mariño et al., 2015).

4.5.2 User evaluation

User studies can provide a lot of value for confirming the quality of a content generator. The users play through varying numbers of levels and offer their opinion based on specific questions designed to shed light on specific aspects of the generated content. With enough players, it is also possible to gather enough data to feed into a machine learning algorithm to inform it what players consider good or bad, and thus theoretically produce desired quality of content. Players can assess the quality of the tested content by answering questionnaires that are designed based on the type of content being evaluated. A common example in the evaluation of 2D platformer level generators is to assess the levels' novelty, playability, challenge, visual aesthetics, and fun, on a numerical scale. (Shaker, Smith, et al., 2016).

The user studies performed to evaluate procedural 2D platformer level generators have largely been ad hoc in nature. The authors are interested in specific aspects of the generator, and come up with related questions or aspects of the generator to evaluate, and then try and tailor the questionnaire for this intended purpose. For example: Horn et al. (2014) focus on enjoyment, visual aesthetics, and difficulty, while Dahlskog and Togelius (2013) are interested in fun, difficulty, and the similarity between the played levels. Togelius et al. (2013), on the other hand, only ask users to select the level they prefer from the ones played. (Dahlskog & Togelius, 2013; Horn et al., 2014; Togelius, Shaker, Karakovskiy, & Yannakakis, 2013). One important detail present in multiple papers is the usage of a practice level. Before evaluating any levels, the players are asked to play a practice level to get familiar with the game and its controls. This helps provide a common starting point for the evaluated levels; otherwise the first played level could be misevaluated due to time spent learning. (Mariño et al., 2015; Togelius, Shaker, et al., 2013).

Attempts at more standardized questionnaires for evaluating user experience in video games have also been made, though none of them have been deemed reliable enough to be adapted into wide usage. Two of the more commonly referenced ones are the Game Experience Questionnaire and The Player Experience of Need Satisfaction. Both include many different constructs that are evaluated based on a set number of questions, for example flow, immersion, and challenge. A standardized questionnaire would help especially when comparing results between different studies, however, attempts at empirical validation have found too many issues with them. (Johnson, Gardner, & Perry, 2018; Law, Brühlmann, & Mekler, 2018).

5. Research Method

The study is implemented using a design science research process as defined by Peffers et al. (2007). An artifact, a rhythm-based level generator for 2D platformer levels, is developed and evaluated as part of the process.

5.1 Design science research

Peffers et al. (2007) introduce a six-step process to guide in the implementation of design science research (DSR) studies. The authors base their suggested process on existing literature in an effort to arrive at a more generally accepted process of performing design science research in the discipline of information systems. The resulting six steps are listed below. (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007).

1. Problem identification and motivation
2. Objectives for a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

The first step requires to define the problem and demonstrate its importance (Peffers et al., 2007). In this study, the identified problem is the rising costs of game development. As required by the second step of the process (Peffers et al., 2007), an objective is set to target this problem. A possible solution is identified in procedural content generation as it could be used to automatically generate game content with little effort from the game developers. The scope is narrowed to only include the levels of 2D platformers due to their relative simplicity. The narrowed focus is also hoped to result in a potentially more meaningful result due to less variables involved. Because of the limitations in the current PCG solutions especially regarding the quality of the generated content, the objective is set to gain more insight into how the quality could be improved. It is theorized that the feeling of rhythm when playing games, especially 2D platformers, can be highly influential in the perceived quality of platformer levels, so the study will investigate how to use rhythm to influence platformer level generation.

For step three of the DSR process by Peffers et al. (2007), the current literature is first reviewed. Couple studies exist that utilize the idea of rhythm in influencing 2D platformer level generators, however, these studies only use rhythm as the basis for the technical side. Little consideration is given to what kind of rhythm could be good quality, and instead the game designer is given the responsibility to experiment. (Compton & Mateas, 2006; Smith et al., 2009). The artifact designed for this study takes the original levels of the widely acclaimed 2D platform game Super Mario Bros. as the base comparison of levels with potentially good rhythm. The levels, and the rhythm of interaction required to complete them, are analysed with methods based in music theory in order to extract a rhythm representation. This information is then used to develop a procedural level generator for Super Mario Bros. that replicates the rhythms of the original levels.

Since the study objective deals with the quality of platformer levels, the artifact's ability to solve the problem is demonstrated through a series of playtests. In other words, it is examined whether the implemented idea of rhythm was successful in generating enjoyable levels. The results of these playtests are then used to properly evaluate the

quality of the generated content as per step 5 of the DSR process. In addition, computational evaluation is performed to examine the generative space of the artifact. This gives insight into what kind, and how extensively differing levels the generator can produce. Comparisons are also made to other approaches. Finally, for step 6 of the process, the results of this study will be published and openly available via Jultika, the open access repository by the University of Oulu.

5.2 Mario AI Framework

The analysis of the existing Super Mario Bros. levels and implementation of the level generator are made with the help of the open-source Mario AI Framework by Ahmed Khalifa. The framework is based on the work done by Togelius & Karakovskiy from 2009, which in turn is based on the open-source Super Mario Bros. clone called Infinite Mario Bros by Markus Persson, the creator of Minecraft. The original version of the framework by Togelius & Karakovskiy was created for the purpose of benchmarking AI agents that complete procedurally generated Super Mario Bros. levels. (Khalifa, 2019). After that, the framework has seen a lot of use in studies regarding both AI agents and procedural level generation (Togelius, Shaker, et al., 2013). The framework by Ahmed Khalifa provides an updated version of the original implemented in Java. It contains the original features, such as support for AI agents and level generators, both of which there are also multiple examples included. The framework also includes many original Super Mario Bros. levels already implemented. (Khalifa, 2019).

The rhythm of the levels is analysed with an AI agent. The decision to analyse the levels using an AI agent was made based on the fact that real player behaviour can be inconsistent and suboptimal. Players have the freedom to choose in what way they approach the levels. For example, they may decide to sometimes complete the secondary objective, which in Super Mario Bros. would be gathering coins and defeating enemies, while sometimes deciding not to do it. Artificial intelligence, on the other hand, can be much more consistent in its decisions, making it more suitable for the study. All of the AI agents available in the framework are tested for their suitability to the task. In order to find a baseline rhythm of interactions required to complete the existing levels, the AI needs to be able complete the levels consistently and optimally, meaning it uses as little input as possible.

5.3 Rhythm extraction

The Mario AI Framework is extended to allow data gathering while the AI agent is playing through the levels. This includes, but is not limited to, all the input and their timing. Each unique input causes a new state for the player character. For example, jumping causes a new state that reflects the fact that the character is airborne. All levels have an optimal minimum set of states the player character can go through in a specific order, and with specific intervals and durations in order to complete the level. In this study, methods based on music theory are used to represent and analyse this set of states to find a list of rhythm patterns from existing levels.

The player character states are likened to musical notes in the context of Gestalt Music Analysis (GMA). The same way that notes have a state comprising of their duration and pitch, player character states are essentially combinations of the input events that have taken place to reach the state. (Pagnutti, 2016). The AI agent gathers these input events, and the resulting states are then programmatically compiled based on them. Similar to

how notes are chained in music to form longer sections, unified musical ideas, so is GMA used to extract sections of unified gameplay patterns (Pagnutti, 2016). And similar to how rhythm in music is formed from the notes' intervals and durations, so do these extracted patterns have rhythms associated with them.

The approach used by Pagnutti (2016) with GMA is modified slightly for the purposes of this study. The power-up state component is omitted because it is argued that it isn't directly related to the input events. As we are interested in the rhythm of the input, only the remaining states of movement direction, ground state, airborne state, and duration are considered to be related. Another difference to Pagnutti (2016) is that in this study AI agent is used to play the levels instead of analysing real player gameplay. In the original application of GMA in platformer levels, it wasn't possible to determine whether the player input was caused by changes in the level or in player goal, making the extracted gestalts imperfect at times. (Pagnutti, 2016). A suitable AI agent, on the other hand, would keep the same goal throughout, and thus addresses this issue.

Finally, not all of the levels from Super Mario Bros. are suitable for the purposes of this study. Some level archetypes have unique and even completely different gameplay mechanics used in them. It is deemed necessary to only include the most commonly found type of level in the rhythm extraction because it is out of the scope of this study to examine the differences the unique mechanics would cause in relation to the extracted rhythm patterns.

5.4 Level generation

A procedural level generator is created in Mario AI Framework using the extracted rhythm patterns as the first input parameter. The rhythm patterns are in essence a collection of slices of the original levels where each of them has a unified rhythm-based gameplay idea. In other words, each pattern is a collection of obstacles and enemies that requires a specific minimum set of inputs to complete. Enough patterns are chained together to form a level with the length of a comparable original Super Mario Bros level.

The second main input for the level generator is the comparison level, meaning an original level from Super Mario Bros. The rhythm patterns represent the lowest level of rhythm. The second level of rhythm is then comprised of these patterns. Rhythm patterns of differently varying intensities need to be chained together to form a level-wide rhythm. For this purpose, the concept of interest curves is incorporated to the level generator. Interest curve is a tool that can be used to design and gauge the level of interest in an entertainment event, the playthrough of the platformer level in this case. Each section of the curve has a level of interest associated with it. (Schell, 2014). It is argued that the amount of input required to progress in a platformer game is directly linked to the amount of interest, with higher required input count being inherently more interesting. The intensity of each extracted pattern, meaning the amount of input required to complete them, is calculated. Since each rhythm pattern is a coherent unified idea, it is theorized that the exact pattern used doesn't matter as much as the variation between them, meaning the resulting interest curve. The level generator takes the calculated intensities of the original level's patterns and randomly picks new patterns in the same order with intensity values close to the original. In other words, the generator uses the original level's interest curve to generate new levels with similar rhythm.

5.5 Evaluation

The nature of the levels the generator creates is investigated by computational metrics-based evaluation. Two metrics, linearity and leniency, from existing literature are computed, and the resulting data is used to construct 2D histograms to visualize the generator's expressive range. The expressive range is compared to the original levels and two other example studies' generators.

The quality of the level generator is evaluated via playtests. The participants play through a practice level to get familiar with the game and the controls, enabling more balanced evaluation. Then, the participants play through three pairs of two levels and answer a short survey at the end of each pair to numerically evaluate both levels. Each pair of levels consists of one original Super Mario Bros. level and a procedurally generated level that uses the aforementioned original level as the basis for the interest curve. The order of the pairs, and the order within each pair, is randomized.

6. Level Generator Implementation and Findings

In this chapter, the process of the study is detailed. First, the different AI agents of Mario AI Framework were evaluated to find the one most suitable for the task. Then, using the selected AI agent, the original Super Mario Bros. levels were automatically played while gathering information, such as all the input decisions the AI made. Utilizing Gestalt Music Analysis (Pagnutti, 2016), the playthroughs were analysed in order to split each level into rhythm sections where each section is a logically coherent set of obstacles and enemies that requires a specific rhythm pattern of input actions to complete. A rhythm-based level generator was built to use the extracted rhythm patterns. Finally, the resulting level generator was evaluated using both computational metrics prevalent in existing literature (Horn et al., 2014; Smith & Whitehead, 2010), and user evaluation via playtests and surveys.

6.1 AI agent selection

This study focused on the base rhythm of the platformer level, or in other words: the minimum required inputs to complete the level. Since there are endless ways the player character could chain inputs together to traverse through the levels, the decision was made to use AI instead of real players to gather the input data. AI is highly likely more consistent and optimized in decision making than a person would be even if that person was told to try and complete the level optimally. Any unnecessary input that is not strictly required for level completion could have resulted in the Gestalt Music Analysis being less effective (Pagnutti, 2016).

The Mario AI Framework contains 11 different AI agents. Each of them has a different decision-making logic for moving the character through the level geometry while avoiding the obstacles and enemies. (Khalifa, 2019). A single AI agent was selected for the rhythm analysis as it was out of scope for this study to compare multiple different agents' results. The criteria for the selection were as follows:

- *Completion*: The agent reaches the end of the level because failing to complete the level would skew the data by omitting the rhythm of a portion of the level
- *Consistency*: The agent makes same decisions each test run, and thus also is able to complete the level each time it is tested
- *Efficiency*: The agent makes very little or no unnecessary inputs at all in order to find a valid baseline rhythm

Each AI agent was tested for the selection criteria. The tested sample level is the very first level of the first world of Super Mario Bros. A total of five test runs were completed for each agent. The consistency and efficiency of the AI agents was approximated by watching them try to complete the levels in real time. The results of this can be seen in Table 1.

Table 1. Results of the AI Agent suitability tests.

Name	Completion	Consistent	Efficient	Notes
andySloane	NO	–	–	Got stuck in a place where variable height jump is required.
doNothing	NO	–	–	Does nothing as indicated by its name.
glennHartmann	YES*	YES*	YES*	Does not seem to account for enemies.
human	–	–	–	Not an actual AI agent but instead used for manual play as indicated by its name.
michal	NO	–	–	Does not seem to account for enemies. Does not jump over pipes.
random	NO	–	–	Movement is random as indicated by its name.
robinBaumgarten	YES	YES	YES	Seems optimal in its inputs. Works well with other levels, too.
sergeyKarakovskiy	NO	–	–	Jumps constantly even when not needed.
sergeyPolikarpov	YES*	NO	NO	Seemingly the most inconsistent with each test run ending differently. Completed only once from five attempts.
spencerSchumann	YES	YES	NO	A lot of unnecessary jumping.
trondEllingsen	YES*	YES*	NO	Jumps over stair-like constructs with multiple small jumps instead of utilizing the variable height jump.

* Only if enemies are removed from the level.

The results were interesting in that most of the AI agents performed quite poorly even on the very first level of the game. As only one agent fulfilled the criteria with no special conditions, further evaluation was deemed unnecessary. The *robinBaumgarten* agent was chosen for the rhythm analysis. Enemies or no, this agent performed consistently and optimally even when testing multiple other levels.

The robinBaumgarten AI is based on A* (a-star) path searching algorithm. Whenever it's time for this AI to decide a new plan of action, it will first create list of all possible actions the player character is able to do at the given time. As an example, it checks if the player can jump currently, which in turn is only possible if the player character is currently on the ground and there is no block above them that is blocking the jump. The AI has knowledge of the state of the whole level and its enemies, which it uses to evaluate each of the possible actions to pick the most efficient one. (Khalifa, 2019).

6.2 Existing level analysis

The selected AI agent robinBaumgarten was then used to automatically play through a set of levels from the original Super Mario Bros game. Some extensions to the Mario AI Framework were required to be made to allow gathering the input decisions made by the AI. The gathered data was analysed, resulting in each level being split into logically distinct sections, also referred to as rhythm patterns here. In order to help the level generator be able to chain different rhythm patterns together in a way that makes sense, the second level of rhythm was also analysed with the pattern intensity values.

6.2.1 Recording player character state

Rhythm in music is formed from the relationship between sequential musical notes, or more specifically: the intervals between them (Costello, 2018a). Notes being the building blocks of music, the comparison was made to player input, which is in a sense the building block of platformer gameplay. The next step was to understand how to put together the notes, or player input, in order to form the rhythm. An existing approach of using Gestalt Music Analysis was found from the literature. In music, gestalt refers to a unified entirety, something that makes more sense as a whole. For example, the singular notes are meaningless alone but together they can form unified musical ideas, gestalts. In the GMA method, the corresponding concept to musical note is the state of the player character. The state comprises of player movement direction, whether they're walking or running, whether they're grounded or airborne, the currently active power-up, and the duration of the state. Since this approach was already deemed promising in an earlier study, the same approach was taken into use here. (Pagnutti, 2016). The currently active power-up component, however, was dropped because focus in this study is the rhythm of the gameplay, meaning the inputs required to complete the levels. All components except the power-up component are the direct result of player input, making this single component unsuitable in this context. With the remaining components, the state is built from direct player interaction, and is thus argued to be suitable for analysing the rhythm of the levels as discussed here.

The AI agents in the Mario AI Framework update 30 times a second. Each update is referred to as a "tick", and each tick the AI analyses its situation and makes a new input decision, if required. The framework was extended to record the state of the player character at the end of each tick. For most of the time, the robinBaumgarten agent held the states for naturally long periods of times, meaning it seemed relatively comparable to a highly skilled human player. However, from time to time it made state changes that lasted only one or couple ticks. These unnaturally short states also made the results unnatural, resulting in patterns that were only a couple tiles wide and thus had no meaningful rhythm in them. As the goal was to mimic optimal human player decision making, the decision was made to remove states where duration was less than 6 ticks. This number was reached by experimenting with different values; it was the lowest minimum tick duration that resulted in consistently logically coherent patterns. The removed states were merged to their subsequent state by moving the latter state's beginning time to the beginning of the removed state in order to not leave any gaps in the analysed playthrough data.

With the AI agent capable of recording state information of its playthrough, it was time to let it complete all the Super Mario Bros levels. However, not all of the game's 32 levels were found suitable for the purposes of this study. As seen in Figure 3, there are three main archetypes of levels: the traditional, the underwater, and the boss levels. The game's two underwater levels contained fundamentally different gameplay since in those the player character swims instead of runs. Thus, these levels were excluded. The eight boss levels also contain unique gameplay mechanics, most notably the boss battles at the end of each of them. The boss levels were also completely excluded to keep the scope of the level generator manageable. Finally, the traditional levels, which form the bulk of the game, were all tested. It quickly became apparent that even the robinBaumgarten AI agent cannot complete some of these levels. Some levels weren't originally included in the Mario AI Framework at all, and instead it was necessary to create them manually. These levels contained some mechanics that were not actually implemented at all in the framework, for example, moving platforms and the trampoline, which was a major reason they were not completable. Some other levels had to be excluded because they had

sections where there was no ground, only platforms, and that also caused the AI to not to be able to complete it. In the end, a total of 15 of the game's 32 levels were included.

6.2.2 Rhythm patterns

Next, the AI agent played through all the selected levels and recorded its states throughout. Gestalt Music Analysis as detailed by Pagnutti (2016) was used to divide each level into sections, or rhythm patterns in this case. Each component's weighting values that were used in GMA had to be adjusted when comparing to the original study's values. This was likely because the referenced study used human player data. Real players approach playing the levels in a different manner when compared to an AI. The values were adjusted away from the original ones in an experimental manner by examining the resulting rhythm patterns, and determining if they made sense as individual units. The adjusted weighting values are listed below with the original values used by Pagnutti (2016) found in parentheses:

- Movement direction: 0.25 (original 0.75)
- Power-up: not included in this study
- Ground state: 0.25 (original 0.25)
- Airborne state: 0.5 (original 0.5)
- Time: 0.1 (original 0.01)

The biggest difference in the weighting factors was the duration of the state. Pagnutti (2016) doesn't explicitly state the unit of measurement for time, though it is likely seconds or milliseconds. In this study, however, the duration was measured in ticks. The reason for choosing ticks as the unit was that the Mario AI Framework was prone to small lag spikes from time to time due to some AI decision making update cycles taking more time than allocated to finish. Since in Mario AI Framework each second consists of 30 ticks, it means that each tick, or update cycle, the AI has a maximum 1/30 seconds to make its decision[2]. The real amount of time is even shorter, since a portion of the update cycle goes into other things like drawing the graphics. Sometimes, for some reason, the AI would take a bit more time than expected to execute, and this would delay everything in the game for that amount of time. As the game runs on a single thread, nothing else is updated while the AI is making its decision. In other words, it has no major impact on the gameplay itself since, for example, enemies wait patiently for their turn to be updated. In other words, same decisions are made sequentially but just a little delayed. However, it would have had an impact on the extracted rhythm patterns if the duration of the states was measured in real-life seconds. So, ticks were used to measure the duration of each state due to their consistency.

Following the method outlined by Pagnutti (2016), Gestalt Music Analysis was applied to the recorded states with the aforementioned weighting values, resulting in a total of 135 extracted rhythm patterns from the 15 levels that were selected. Examples of extracted patterns can be seen in Figure 8.

---

[2] If an update cycle completes in shorter time than allocated, the game simply waits until the full time of 1/30 of a second has passed before initiating the next update cycle.

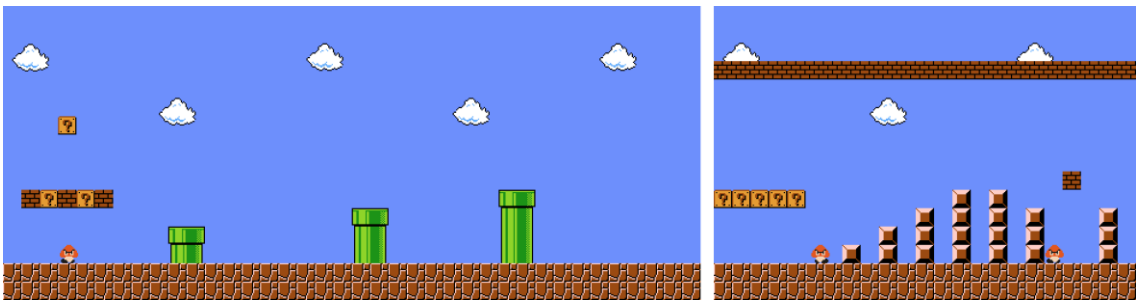

Figure 8. The first extracted patterns of World 1 levels 1 and 2, respectively.

However, during the pilot test of the level generator's user evaluation, it was noticed that the Super Mario Bros levels, even the original ones, are quite challenging for the average player. In order to keep the playtests at a more suitable difficulty level, the patterns from the final two worlds, World 7 and World 8, were dropped. This decision was made because it is theorized the difficulty of the levels rises with each level, and because it was especially with some patterns from the aforementioned worlds that were causing the issues.

The final pattern count was then 97, which is ultimately extracted from a total of 11 levels as a result of the exclusions. The average pattern count in a level was 8,82 with a standard deviation of 0,83. The shortest pattern was 7 tiles in length, while the longest pattern was 63 tiles. The average pattern length in tiles was 22,18 and the standard deviation from average length was 10,1 tiles. A total of 63,92% of the selected patterns were found to be within one standard deviation from the average, and 95,88 % were within two standard deviations.

6.2.3 Rhythm intensity

At this point the final component required for the level generator implementation was the ability to meaningfully chain together the rhythm patterns. In other words, the second level of rhythm was still necessary for the output levels to be comparable to the original Super Mario Bros. level. The concept of interest curve was picked for this purpose due to the similarities in its description to rhythm, especially regarding pacing and intensity. (Costello, 2018a; Schell, 2014).

The interest is comprised of three separate components: inherent interest, poetry of presentation, and projection. The poetry of presentation component focuses on aesthetics, which in this study would be the graphics of the game. (Schell, 2014). The different levels in Super Mario Bros. have a couple of different graphical styles, some of which are demonstrated in Figure 3. In order to keep the scope more focused, each level in the playtests, whether original or generated, was made sure to look thematically the same by using the same tileset from Figure 2. Thus, the effect of the poetry of presentation component was ruled out.

The projection component is an abstract one that is related to the person imagining things, emphasizing with characters, and so on (Schell, 2014). For the purposes of this study, only specific types of levels were selected for the rhythm extraction. Most notably, the underwater and boss battle levels were omitted. The underwater and boss battle levels would have potentially enabled the players to imagine different kinds of things like being in the same situation underwater, holding breath, or dodging the hammers the boss throws at the player. However, with their omittance the context of playing the remaining selected

levels is relatively the same, meaning the players project themselves similarly regardless of the played level in the playtests. As such, the effect of the projection component was also ruled out.

The remaining component, inherent interest, was the only factor left. The focus of this study was in the rhythm of gameplay interaction, and it was theorized that the higher the amount of required input to progress through a section of level, the more inherently interesting it is to the player. As such, inherent interest was measured by the amount of interaction required to complete the level. As the last remaining component, it makes up the whole of the interest curves here.

An interest curve was then graphed for each of the original Super Mario Bros levels that the AI agent completed. This was done by calculating the running average of unique input events. Each plotted point is the average number of input events from the previous 15 ticks. Then, the different levels' graphs were compared to one another. Interestingly, the very first level of the game, World 1-1, had very similar curve as what Schell (2014) proposed as an optimal curve for interest. Some other levels also had similar graphs, while others had very differing ones. The choice was made to take three original Super Mario Bros. levels to be the reference levels for the playtests in order to keep the playtest duration suitable for the players. In other words, these three levels were used by the level generator to create new levels that mimic those same levels' original rhythms. Three levels with differing interest curves were chosen: World 1-1, World 3-2, World 5-2. Their interest curve graphs can be seen in Figure 9.

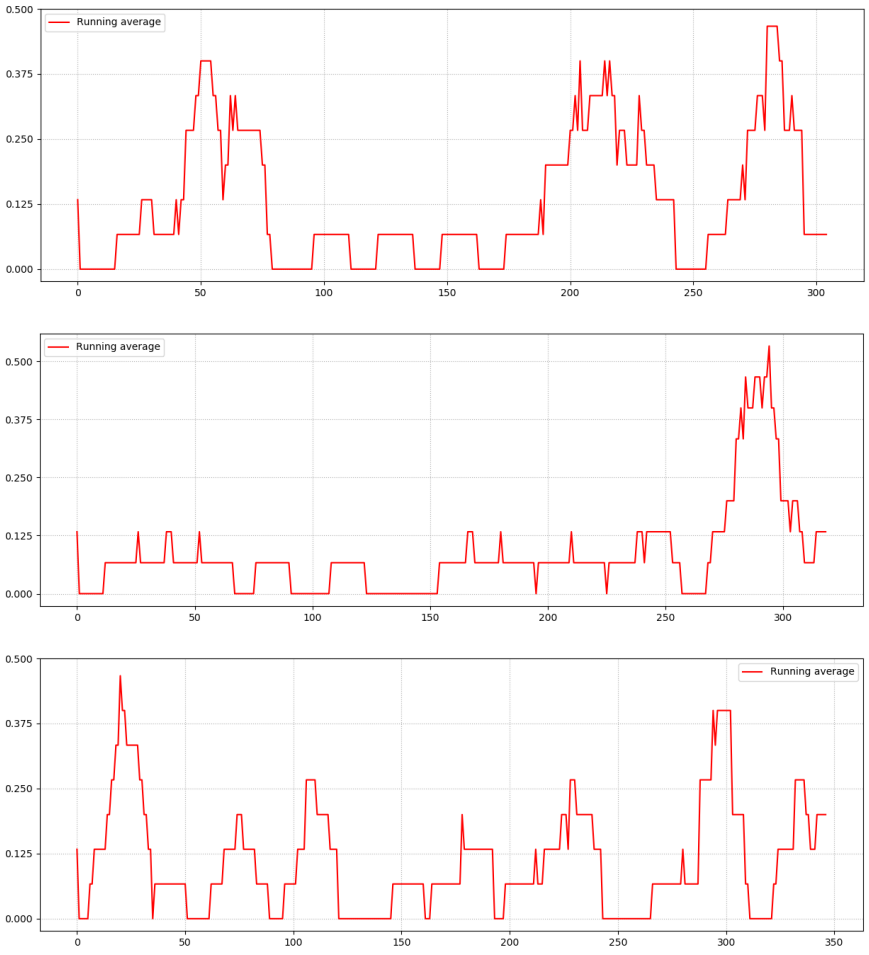

Figure 9. Reference levels' interest curve graphs. From top to bottom: World 1-1, World 3-2, World 5-2.

As can be seen in Figure 9, the first level World 1-1 is quite similar to the example of a good interest curve as seen in Figure 4. The level starts with high interest, continues with smaller intensity, and finishes with the two build-ups. The three selected levels' graphs can be seen to be very dissimilar to one another.

Finally, it was necessary to implement a way to map different rhythm patterns to the interest curve of a level for the level generator to be able to simulate the same interest curve in the generated levels. For this, each pattern was assigned an intensity value in a similar manner to the interest curve in Figure 9 by counting the number of input actions needed to complete it. An example breakdown of the first level's pattern intensities can be seen in Figure 10. Each bar represents one pattern's intensity. The graphs are on the x-axis in the order that they appear in the original level.

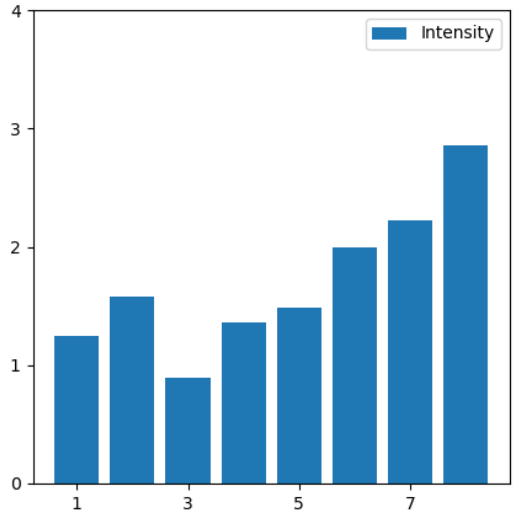

Figure 10. The input intensities of all extracted World 1-1 rhythm patterns.

For each new input event that happens the pattern, the intensity of the pattern was raised by one. For example: a pattern where the AI agent moved right, left, then right again, and finally jumped, would have resulted in an intensity value of 4. In order to make the intensity of the patterns comparable to each other across different levels, this resulting value was adjusted by dividing it by the tile length of the pattern. In other words, the intensity of each pattern was measured with the unit of input actions per tile.

6.3 Level generator implementation

Using the categorization of Togelius et al. (2011), the implemented level generator was an online, deterministic, constructive generator that creates necessary content with some degree of control. The levels were created "online" by default, meaning during gameplay. Support for offline generation was also added for later use in playtests by adding the ability to save the generated levels to disk. These levels are necessary content as there would be no platform game without the level. The generator was also constructive, meaning the levels are created once without any evaluation and re-generation mechanic.

The algorithm was controlled by three inputs: the list of patterns, the reference level, and a seed number. The list of patterns were the 97 extracted slices of the original levels as described in an earlier section. The reference level parameter refers to one of the original Super Mario Bros. levels, and more importantly to the calculated intensities of its patterns as demonstrated in Figure 10. The level generator worked by first checking of how many

patterns the reference level consisted. The same number of patterns were picked for the generated level. For each pattern that needed to be selected, the calculated intensity of the pattern in the same position of the reference level was checked. Then, a pattern was picked randomly from all the patterns that had an intensity that was within 10% of the intensity of the corresponding pattern in the reference level. The variance was allowed in an effort to allow for more unique combinations while still closely following the reference level's interest curve. The last input, the seed number, was used to initialize the Java random number generator. This resulted in the deterministic nature of level generation: with the same list of patterns, same reference level, and the same seed, the result was exactly the same at least when the generator was run on the same computer with the same Java version. Different Java versions could potentially have differences in the implementation of the Random class, meaning the generator may not be deterministic between different computers.

As a final adjustment, some empty space was added to the start of each level. The need for this was noticed during the pilot test: some patterns were not initially suitable for being the first pattern of the level. For example, when some patterns appeared as the first one, the player character would start immediately next to an enemy, and die before having time to react. To solve this, the empty space was added to the beginning of each level to allow some room for the player to analyse the start of the level.

6.4 Computational evaluation

The types of levels the generator can produce was examined with computational metrics. The leniency and linearity metrics for the level generator were calculated and visualized using the scripts provided by Horn et al. (2014). The metrics were calculated separately for each reference level that was to be used for the playtests because varying the generator's reference level, or list of patterns, fundamentally changes the types of levels it can generate. It is to be noted, however, that the seed value parameter doesn't change the nature of the generator but instead just allows it to generate different options of the same nature. For comparison, metrics were also calculated for when there is no reference level used at all. In this case, any and all patterns were allowed by the level generator in each position, effectively making the interest curve of the generated level random. When using the reference level, same pattern count was used that the reference level had. In the case of no reference level, the pattern count was a randomized value from within two standard deviations of the average pattern count because most (90,91%) of the levels fell within that range. In practice this led to a range from 7,16 to 10,49 with the result being rounded to the nearest integer value.

A total of 1000 levels were generated for each variant of the level generator, which is the same number of levels as Horn et al. (2014) used for their metrics calculations. The variants refer to the different reference level inputs: World 1-1, 3-2, and 5-2, as well as the generator with no reference level as outlined above. Each level was created using the same set of extracted patterns as presented in Chapter 6.2.2. All levels for a variant of the level generator were created using unique seed values to ensure unique levels within the other parameters. A summary of the leniency and linearity metrics when using different reference levels can be seen in Table 2.

Table 2. Overview of the leniency and linearity metrics when using different reference levels. The value is the average of all 1000 levels on a scale from 0 to 1 with the standard deviation in parentheses.

Reference Level	Leniency	Linearity
None	0.14 (0.07)	0.06 (0.07)
World 1-1	0.21 (0.08)	0.09 (0.09)
World 3-2	0.14 (0.05)	0.06 (0.07)
World 5-2	0.21 (0.09)	0.07 (0.08)

Expressive range can be used to visualize the relationship between metrics in an effort to gain insight on the type of content the generator produces (Smith & Whitehead, 2010). Here it was visualized by mapping the leniency and linearity metrics of all individual levels in a 2D heatmap. The expressive range is visualized for all the reference level variations in Figure 11.

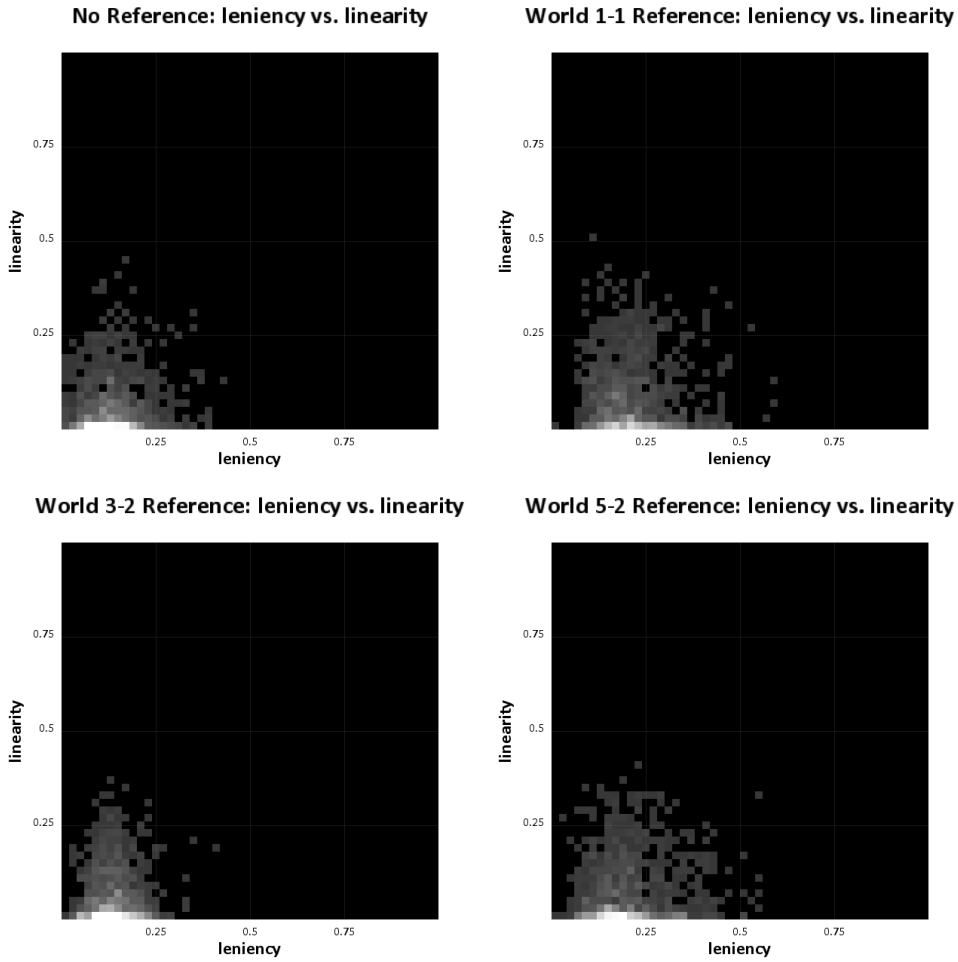

Figure 11. Heatmaps visualizing the expressive range based on leniency (x-axis) and linearity (y-axis) for four different reference level parameter variants of the generator.

The whiter the area, the more of the generated 1000 levels' metrics were included within the dot. The dot is fully white when 40 or more levels had the corresponding metrics values. (Horn et al., 2014). As seen in Figure 11, the expressive range by leniency versus linearity metrics was focused in the same lower-left quadrant in all cases, though some variance could be found. Figure 12 contains the same metrics visualized for the original Super Mario Bros. levels as well as two other generators from other studies. The original

levels are visualized a little differently since there are only a limited number of them. Instead of 40 levels, only 5 levels are required for a dot in their heatmap to be fully white. (Horn et al., 2014).

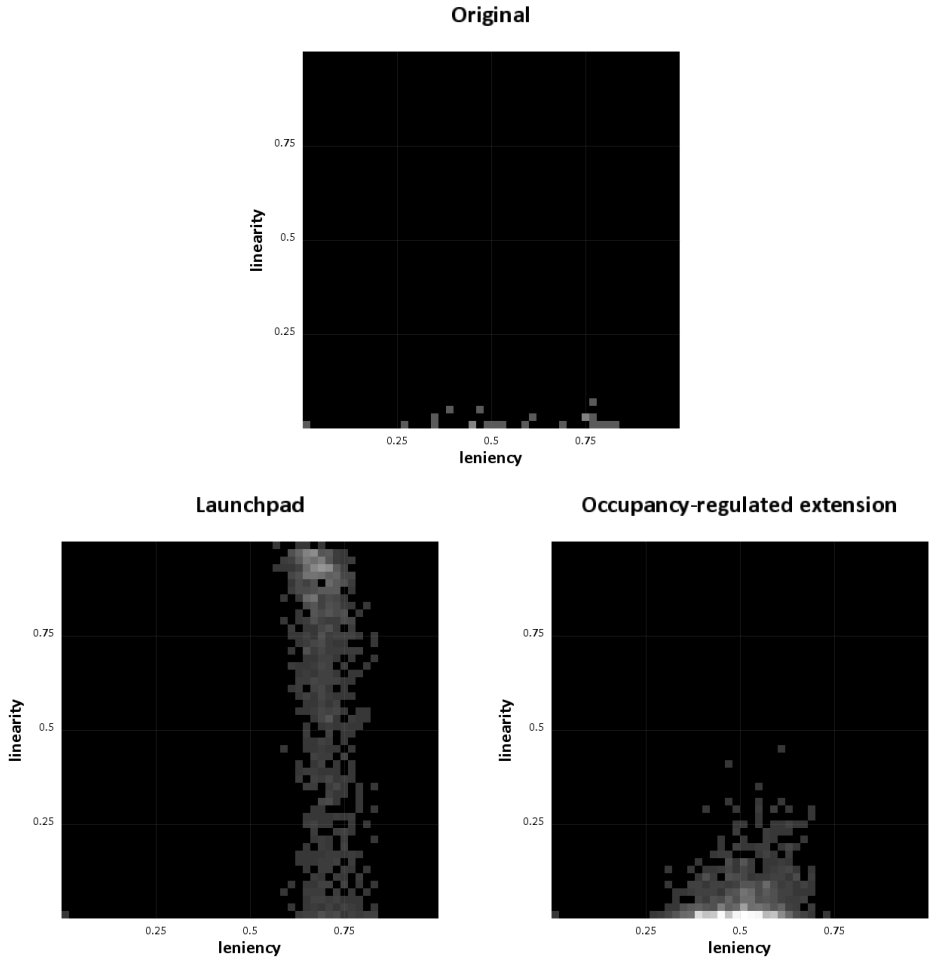

Figure 12. Leniency vs. linearity heatmaps for the original levels at the top, Launchpad platformer level generator (Smith et al., 2009) at bottom left, and level generator that uses occupancy-regulated extension (Mawhorter & Mateas, 2010) at bottom right. Recreated using data provided by Horn et al. (2014).

The linearity metric in Figure 11 was spread relatively evenly in the range from 0 to 0.25 in each case, with a small emphasis closer to 0 and with some cases going over 0.25. This means that the study's generator produced relatively non-linear levels, meaning there was a relatively good number of optional platforms and branching paths. The linearity of the original levels in Figure 12 can be seen to be very close to zero. This would mean that this study's generator produced more somewhat more linear levels than what the original levels were, even though the original levels were the building blocks of the new levels. However, it is to be noted that there is a very limited number of original levels, so the comparison may not be too accurate. Comparing to the other two generators, this study's approach is quite similar in linearity to the occupancy-regular extension approach (Mawhorter & Mateas, 2010). The Launchpad generator (Smith et al., 2009), however, had a much wider range for linearity, though the emphasis was more towards higher values meaning Launchpad's levels were more linear.

The leniency metric provides an intuitive sense of the amount of danger based on the obstacles present in the level. Notably, obstacles and enemies that can result in player death tend to cause the levels to be rated less lenient. (Horn et al., 2014; Smith &

Whitehead, 2010). In the case of Horn et al. (2014) whose scripts were used to generated the above visualizations, the lower the leniency value, the more danger the levels are estimated to provide to the players. Looking at Figure 11, the leniency for this study's generator with World 3-2 as the reference level is mostly in the range from around 0 to 0.25, and for World 1-1 and World 5-2 the leniency is spread from around 0 to around 0.5. For the case of no reference level, the leniency range is between the other cases. These values mean that the generated levels can provide relatively high amounts of danger to the player. Most notably, World 3-2 has the smallest leniency range with values that are spread around 0.15, meaning World 3-2 could potentially pose more danger than the other reference levels. The low leniency for this level likely comes from the fact that the start of this level's interest curve as seen in Figure 9 is very flat. For a long period of the level, there is likely low amount of choice in patterns, and those patterns may be some of the less lenient ones.

It is also interesting to contrast the differences in leniency in Figure 11 to Figure 9, in which we can see that the AI agent needed much less input actions to complete World 3-2. Since the levels based on World 3-2 are less lenient and require less input, it could mean that failing those required inputs, the player character is much more likely to die when compared to the other two levels.

Comparing this study's leniency to the original levels' leniency we can see that this study produced less lenient levels. Although, again it is to be noted that due to the low sample size of original levels this comparison may be inaccurate. The Launchpad generator (Smith et al., 2009) produced quite evenly and tightly distributed levels in terms of leniency in the range from 0.6 to 0.8, meaning those levels may be considered much more lenient, meaning less dangerous, than this study's generator's levels. The occupancy-regulated extension generator (Mawhorter & Mateas, 2010) produced levels with a wider range of leniency values, from around 0.3 to around 0.7. This could mean these levels, too, may be considered less dangerous. However, the amount of variance in the possible values is closer to this study's generator.

6.5 User evaluation

The quality of the level generator was evaluated with playtests. Before any proper playtest sessions were held, however, one pilot playtest session was performed. It was quickly noticed that even the original Super Mario Bros. levels are relatively hard for the average player. As a result, the patterns from World 7 and World 8 were excluded from the level generator in an effort to keep the challenge of the generated levels manageable. This restricted the generative space of the generator since the number of patterns was reduced as reported in Chapter 6.2.2.

A total of 6 players participated in the tests with each playtest session taking around 20-30 minutes to complete. The recruited players were fellow students of Information Processing Science from University of Oulu, and were known to have experience playing video games. The participants were between ages 25 and 31.

In the playtest sessions, each player played through a total of six levels. These levels were organized in pairs with each pair containing one original Super Mario Bros. level and one generated level which used the original level as the reference level. The original, reference levels were World 1-1, World 3-2, World 5-2, so, for example, the pair of levels containing the original level World 1-1 also contained a generated level that used World 1-1 as the reference.

The playtest sessions started with a practice level. The players were instructed on how to control the player character, which was done using a PlayStation 4 controller since the original Super Mario Bros. game is a console game, meaning it was also played with a controller. The instructions also included insight into the jump button, which could be held down to cause the character to jump higher up to a specific point, and the run button that could be held down when jumping to be able to cross over longer distances. The players were also instructed on the different enemy types, and how to defeat them. Players were free to play the practice level for as long as required to become comfortable with the controls. The practice level was a generated level, with the same generated level used for all players.

The pairs of levels were always played in a random order. The players were instructed that the objective is simply the completion of the level. In addition, the order of the two levels in each pair was also randomized. The seed values for the generated levels were between 1 and 100 million, and were generated beforehand using an online random number generator. At the end of each pair of levels the players were asked to rate the two levels' fun, challenge, novelty and pacing on a scale from 1 (least) to 5 (most). Evaluation instructions for novelty were to rate how interesting was the level. Instructions for pacing were to rate how interesting was the pacing of the obstacles and enemies. Fun and challenge were not separately given evaluation instructions since they are easier to interpret. In addition, the players were asked to rate how similar it felt to play the two levels on the same scale. Lastly, in order to help with the rating, and in an effort to keep levels evaluations comparable to each other, screenshots of both of the played levels were also provided as a reminder. An example can be seen in Figure 13.

Figure 13. Example overview screenshot of a full level the playtester could use as a reminder. The players were able to freely zoom in the picture to see it in better detail.

The average and standard deviation of each level's user evaluations can be seen in Table 3. The values are calculated based on each tester's evaluation of the specific level. The "Original" row contains the calculated values of each tester's evaluation when the original Super Mario Bros. level was played. The "Generated" row then contains similar values, but for the generated levels that used the original level as a reference for the pattern selection.

Table 3. Overview of the user evaluation of each variant level generator. The first value is the average between the participants, with the standard deviation in parenthesis.

Reference Level	Variant	Fun	Challenge	Novelty	Pacing	Similarity
All three	Original	3.444 (0.784)	3.222 (1.003)	3.333 (0.84)	3.167 (0.786)	2.667 (0.97)
	Generated	3.333 (0.84)	3.222 (0.878)	3.111 (0.676)	3.0 (0.84)	
World 1-1	Original	3.5 (0.548)	2.5 (0.548)	3.333 (0.816)	3.5 (0.837)	2.5 (0.837)
	Generated	3.167 (0.752)	3.5 (0.837)	3.333 (0.516)	2.833 (0.752)	
World 3-2	Original	3.167 (1.17)	3.333 (1.211)	3.167 (0.983)	2.833 (0.753)	2.833 (0.983)
	Generated	3.333 (0.816)	3.333 (0.816)	3.0 (0.632)	3.0 (0.632)	
World 5-2	Original	3.667 (0.516)	3.833 (0.752)	3.5 (0.837)	3.167 (0.753)	2.667 (1.211)
	Generated	3.5 (1.049)	2.833 (0.983)	3.0 (0.894)	3.167 (1.169)	

Looking at the first row that contains the combined evaluations from all three levels, it can be seen that the original levels had on average a little bit higher fun, novelty, and pacing. The differences are, however, quite small. Challenge was relatively the same with the original levels spread a bit more widely. This result gave initial indication that having two levels with the same interest curve could in fact produce similar levels when comparing the level of fun, challenge, novelty, and pacing. It is also encouraging that the similarity that the players felt was on average quite low, with the average being 2.667. Even though the two levels that used the same interest curve were played in a row, the players felt they were a little dissimilar. In other words, this could mean that the same interest curve could be utilized to produce multiple levels that would still feel different to play.

Comparing the individual levels' results, it can be seen that there are couple cases where there is a bit bigger difference in the original versus generated level averages. The biggest differences were in the evaluation of challenge in World 1-1 and World 5-2. The pacing of World 1-1 and novelty of World 5-2 had the second and third biggest differences. It is to be noted, however, that the sample count for the individual levels is smaller, meaning individual evaluations have a higher impact on the average than when compared to the aggregated results in the first results row of Table 3. This could mean, for example, that some players got unlucky with the level generator picking some of the harder patterns.

Even if two extracted patterns had the same intensity, meaning the amount of input required to complete it, they could still have different challenge associated with them. For example, one pattern could be void of any enemies or other danger to the player, while the other could be full of them. Anecdotally, levels in a game can be assumed to get harder the further the player plays. This also seemed to be the case in study's original Super Mario Bros levels. The original World 1-1, the very first level of the game, had the lowest challenge on average. The second was World 3-2, the tenth level of the game, and the highest challenge average was found in World 5-2, which is the 18th level of the game.

The order of the evaluated challenge in the generated levels, however, was reverse. The generated level using World 1-1 as a reference was deemed the hardest, while the one using World 5-2 was felt to be the easiest of the generated levels. This was likely the result of the level generator not considering the challenge of the patterns when picking them. As such, the generator wouldn't likely be suitable for generating a lot levels for a platform game in its current state since good pacing of difficulty would be important for making the gameplay experience enjoyable and engaging (Bleszinski, 2000; Lindley, 2002). While the level generator did consider the pacing of a single-level in the form of the patterns' intensity, in a commercial setting it would be required to go one step higher to analyse the challenge of all the generated levels. With this higher-level analysis, the generated levels could potentially be ordered in a suitably challenging order.

7. Discussion

This study had the goal of investigating the applicability of using rhythm in helping procedural level generators create higher quality content. This was approached with two research questions that are restated below.

> **RQ1**. How can the rhythm of player input in 2D platformer levels be analysed?
>
> **RQ2**. What kind of results can be achieved by creating new 2D platformer levels based on the rhythm of existing levels?

The research questions are answered and discussed further below based on the findings of the study.

7.1 Research question 1

For the first research question, existing literature was reviewed. Few existing studies that use rhythm explicitly in platformer level generation were found. Those studies focused more on using the rhythm as a technique and a way for a designer to control the generated levels. As such, the rhythms used on the level generators were created based on intuitive understanding of what could be good. (Compton & Mateas, 2006; Smith et al., 2009). In order to gain some insight into what rhythm could be good, this study focused on analysing the rhythm of an existing game. For this purpose, an approach by Pagnutti (2016) that uses a method based on music theory, was selected.

There can be multiple levels of rhythm in digital interactions (Costello, 2018a), and so in this study the rhythm of existing platformer levels was also analysed on multiple levels. The lowest level of rhythm was found by utilizing a slightly adjusted version of the GMA method (Pagnutti, 2016). This produced a representation of the level's baseline rhythm as multiple sets of player character states, also referred to as rhythm patterns in this study. As the states are the direct result of player input, it is argued that these states are suitable to represent the rhythm closest to the player, the rhythm they feel while playing similar to how the rhythm of notes in music can be felt.

After the pattern extraction, the rhythm analysis was expanded to the next level. With the parallels between rhythm and interest curve regarding pacing and intensity (Costello, 2018a; Schell, 2014), the interest curves were introduced as the tool to analyse and chart out this second level of rhythm in the existing levels. Here, the rhythm is formed from the rhythm patterns instead of player character states. In order to keep the focus still on the player interaction, each pattern was assigned an intensity value based on the amount of input needed to progress through that pattern.

The similarities between the sample of a good interest curve in Figure 4 and the extracted interest curve of the Super Mario Bros levels in Figure 9 and Figure 10 imply that the taken approach was successful in mapping the rhythm of the existing levels. This, combined with the user evaluation finding the original and generated levels closely similar in various surveyed aspects, would mean that it is possible to use rhythm with intent in designing platformer levels. Further research into this topic could help procedural level generators be more informed of what makes a level good, and thus achieve output of enough quality for a commercial game.

7.2 Research question 2

The second research question builds upon the results of the first one. The successfully extracted rhythm patterns and the interest curves they make up, were used to create a procedural platformer level generator. The resulting generator was then inspected via computational and user evaluation. The main finding was that the generator was able to create levels of similar quality compared to the original levels. The implication of this is that the platformer levels do seem to have a rhythm associated with them, which implies that the approach taken for the first research question is valid. With more research into this, it could be very helpful in increasing the quality of procedurally generated platformer levels by purposefully creating levels with specific rhythms.

Due to the limited correlation to quality aspects like enjoyment (Mariño et al., 2015), the computational metrics were only used to inspect the type of levels the generator outputs. Three different interest curves were tested with the same list of rhythm patterns, and the resulting expressive range visualizations in Figure 11 were highly similar in each case. This seems to implicate that the other main input parameter, the list of rhythm patterns, is much more influential in controlling the types of levels that are created.

The generated levels were generally not very lenient, meaning they posed potentially a lot of danger to the player. Compared to the original levels and some other approaches to procedural platformer levels, this study's generator produced on average much less lenient levels. This could potentially indicate more unforgiving and thus potentially more challenging levels which could be undesirable in a commercial game since unsuitable level of challenge could actively hinder player engagement and enjoyment (Nakamura & Csikszentmihalyi, 2002; Nicollet, 2004). The results do not seem to give an indication into why the original levels have much higher leniency than the levels created from the sections of those original levels.

The level World 3-2, whose interest curve was the most different from the other two as seen in Figure 9, had the most difference visible in the expressive range in that its range in leniency was more suppressed. But even in this relatively drastic difference in the interest curve, the majority of the generated levels focused on the same area of the heatmap as seen by the accumulation of the whiter colours of Figure 11. While the used rhythm patterns seem to control the nature of generated levels on a higher level, the interest curve seems to allow some finetuning. Although this may be a result of the small sample size of tested levels, too, or it could also be a result of the used rhythm patterns. The World 3-2 curve has a long section of little input, so it could be that the specific extracted rhythm patterns that fit the low input count were actually challenging sections of the original game. At this point it is hard to determine if there is any meaningful connection between the amount of input and leniency of the level.

The results of the playtests indicate that the generator was creating levels of similar quality in different aspects when compared to the original levels. Especially the fun factor was very similar on average as seen in Table 3. Fun was, however, on average a little lower in the generated levels in all cases. This could implicate that the current approach may be lacking the kind of final polish that a human designer can more easily apply. The biggest difference in the playtest evaluations between original and generated levels was in the level of challenge. The challenge of generated levels was in the reverse order when compared to the originals. The implication of this is that the amount of input that was used to calculate the intensity of the rhythm patterns cannot be used to determine their challenge. Instead, additional analysis would be required to enable the generator consider the challenge.

8. Conclusion

In this chapter the results of the study are summarized and the identified limitations are presented. Finally, further research topics are suggested to continue and extend on the topic.

8.1 Results

This study set out to gain more insight into how procedural level generators could be improved in terms of the quality of the content. It was theorized that rhythm may be a key component in making platformer levels enjoyable, and so the research questions were aimed at analysing the existence of rhythm in the gameplay of platformers. It was found that music theory can be utilized in finding the rhythm of the platformer levels, and that rhythm actually exists on multiple levels. A rhythm-based procedural level generator was implemented and evaluated. The generated levels were assessed to be around similar quality as corresponding original levels in a couple surveyed aspects. This implies that rhythm could indeed be used to guide procedural level generation.

The findings reinforce the fact that PCG research benefits from the utilization of existing literature from other disciplines; music theory in this case. In addition, the existing theory found in some PCG studies that rhythm could be useful in procedural platformer level generation is also reinforced. The next step would be to gain better understanding of exactly what kind of effects different gameplay interaction rhythms have, and how they could be used. Following this, both evaluation and generation of platformer levels could potentially be approached through rhythm and methods from music theory. The results are also helpful for game developers. Good level design is arguably highly important in making a commercial game successful. However, it is hard to know what it is that makes a level feel good to play. As such, it is recommended for game developers to try and approach level design from the point of view of rhythm; better pacing could potentially be achieved by considering not only the immediate rhythm of interaction, but also the rhythm of adjacent obstacles, and the rhythm of the levels as a whole.

8.2 Limitations

The playtests were limited in both the number of participants and demographic focus. As such, the results should only be treated as initial findings to build upon, and more playtests would be warranted to further validate the results. The computational evaluation is also something that could have been expanded to include a wider variety of metrics to analyse the nature of the generated content more comprehensively.

The applicability of the demonstrated approach outside games similar to Super Mario Bros is likely to be limited. This is because the scope was kept deliberately narrow due to limited resources and the experimental nature of the study. Super Mario Bros includes a relatively limited amount of gameplay mechanics, so features like double jump and their potential effects on the feeling of rhythm are not included in the analysis of rhythm. The rhythm pattern extraction and usage in the level generator worked well within the tile-based levels of Super Mario Bros, but games with other graphical representation style may need a different or adjusted approach. Also, the levels in Super Mario Bros. are linear side-scrolling levels, so this level generation approach wouldn't work in single-screen platformers like Celeste. The rhythm extraction focused on the baseline rhythm of the

levels, meaning the minimum input required to complete them. This likely wouldn't be enough for commercial usage as games have various secondary objectives, like coin collection in Super Mario Bros., that can affect the way players approach playing through the levels.

8.3 Future research

While larger scale studies would be necessary to validate the findings, the results of this study already have the potential to inspire interesting avenues for future research. Following further validation, the application of more music-based theories and methods could be very interesting to investigate as the large amount of existing music theory has great potential if applicable. This, combined with studying the effects of different types of rhythms by, for example, monitoring the players' physiological responses, could have great potential in helping understand what exactly it is that makes a platformer level good.

Closer look at the lowest level of rhythm is another topic worth considering. The results of this study do not include information on what makes a rhythm good at this lowest level. Instead, the extracted rhythm patterns were snippets of the original Super Mario Bros levels' rhythms. It is theorized that these original levels have a good rhythm as the game is so widely acclaimed, but it is currently not known what makes a specific input rhythm feel good.

Lastly, the effects of rhythm should also be investigated in a wider context. Any content from levels to story pacing, and any genre from platformers to FPS, could potentially benefit from a better understanding of how to apply rhythm to help make more informed decisions not only within procedural generation but also in handcrafted content.

References

Andreadis, K. (2019, October 17). Cyberpunk 2077—CD Projekt Red on Pushing Graphics Tech Forward and Building Night City. *AusGamers.Com*. Retrieved from http://www.ausgamers.com/features/read/3621721

Bleszinski, C. (2000). The Art and Science of Level Design. Retrieved February 18, 2022, from https://web.archive.org/web/20100129011119/http://www.cliffyb.com/art-sci-ld.html

Braid. (2008). Number None.

Braun, E. (2018, July 11). The Rise and Fall and Rise of The Platformer. Retrieved December 17, 2021, from Culture of Gaming website: https://cultureofgaming.com/the-rise-and-fall-and-rise-of-the-platformer/

Bright, G. (2014, April 22). Build a Bad Guy Workshop—Designing enemies for retro games. Retrieved February 12, 2022, from Game Developer website: https://www.gamedeveloper.com/design/build-a-bad-guy-workshop---designing-enemies-for-retro-games

Celeste. (2018). Extremely OK Games.

Compton, K., & Mateas, M. (2006). Procedural level design for platform games. *Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 109–111. Marina del Rey, California: AAAI Press.

Costello, B. M. (2018a). *Rhythm, Play and Interaction Design*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-67850-4

Costello, B. M. (2018b). The Rhythm of Game Interactions: Player Experience and Rhythm in Minecraft and Don't Starve. *Games and Culture*, *13*(8), 807–824. https://doi.org/10.1177/1555412016646668

Dahlskog, S., & Togelius, J. (2013). *Patterns as Objectives for Level Generation*. Presented at the Design Patterns in Games (DPG), Chania, Crete, Greece (2013). ACM Digital Library. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-16753

Easton, D. (2019, August 22). What Is A Platform Game? Everything You Need To Know. Retrieved January 20, 2022, from Music Gateway website: https://www.musicgateway.com/blog/gaming-industry/games-business/what-is-a-platform-game

Elite. (1984). Acornsoft.

Fez. (2012). Polytron Corporation.

Goh, R. (2016). *Short Game Analysis of Super Mario*. Retrieved from https://www.academia.edu/30307394/Short_Game_Analysis_of_Super_Mario

Gumin, M. (2016). *Wave Function Collapse*. Retrieved from https://github.com/mxgmn/WaveFunctionCollapse

Helgeson, M. (2011, July 14). Why The Platformer Still Matters. Retrieved December 17, 2021, from Game Informer website: https://www.gameinformer.com/b/features/archive/2011/07/14/why-the-platformer-still-matters.aspx

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, *9*(1), 1:1-1:22. https://doi.org/10.1145/2422956.2422957

Horn, B., Dahlskog, S., Shaker, N., Smith, G., & Togelius, J. (2014). *A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework*. 1–8. Society for the Advancement of the Science of Digital Games. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-16786

Hosch, W. L. (n.d.). Electronic platform game. In *Encyclopædia Britannica*. Retrieved from https://www.britannica.com/topic/electronic-platform-game

iD Tech. (2012, October 22). What is a Platform Game? | 10 Design Types & Video Game Examples. Retrieved February 2, 2022, from ID Tech website: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games

Jennings-Teats, M., Smith, G., & Wardrip-Fruin, N. (2010). Polymorph: Dynamic difficulty adjustment through level generation. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1–4. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1814256.1814267

Johnson, D., Gardner, M. J., & Perry, R. (2018). Validation of two game experience scales: The Player Experience of Need Satisfaction (PENS) and Game Experience Questionnaire (GEQ). *International Journal of Human-Computer Studies*, *118*, 38–46. https://doi.org/10.1016/j.ijhcs.2018.05.003

Kerssemakers, M., Tuxen, J., Togelius, J., & Yannakakis, G. N. (2012). A procedural procedural level generator generator. *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 335–341. https://doi.org/10.1109/CIG.2012.6374174

Khalifa, A. (2019). *Mario AI Framework*. Retrieved from https://github.com/amidos2006/Mario-AI-Framework

Klappenbach, M. (2021). What is a Platform Game? Retrieved October 29, 2021, from Lifewire website: https://www.lifewire.com/what-is-a-platform-game-812371

Law, E. L.-C., Brühlmann, F., & Mekler, E. D. (2018). Systematic Review and Validation of the Game Experience Questionnaire (GEQ)—Implications for Citation and Reporting Practice. *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, 257–270. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3242671.3242683

Lindley, C. (2002). The Gameplay Gestalt, Narrative, and Interactive Storytelling. *CGDC Conf.*

Lyons, K. (2021, January 16). Cyberpunk 2077 full development reportedly didn't start until 2016. Retrieved September 18, 2021, from The Verge website:

https://www.theverge.com/2021/1/16/22234452/cyberpunk-2077-development-2016-pc-console-projekt-red

MacDonald, K. (2020, December 18). Cyberpunk 2077: Sony pulls game from PlayStation store after complaints. *The Guardian*. Retrieved from https://www.theguardian.com/games/2020/dec/18/cyberpunk-2077-sony-pulls-game-from-playstation-store-after-complaints

Mariño, J. R. H., Reis, W. M. P., & Lelis, L. H. S. (2015). An Empirical Evaluation of Evaluation Metrics of Procedurally Generated Mario Levels. *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*. Presented at the Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference. Retrieved from https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/view/11537

Mawhorter, P., & Mateas, M. (2010). Procedural level generation using occupancy-regulated extension. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 351–358. https://doi.org/10.1109/ITW.2010.5593333

MCV Editors. (2013, October 4). Inside Rockstar North – Part 2: The Studio. *MCV*. Retrieved from https://www.mcvuk.com/development-news/inside-rockstar-north-part-2-the-studio/

Melcer, E. F., & Cuerdo, M. A. M. (2020). Death and Rebirth in Platformer Games. In *Game User Experience And Player-Centered Design* (pp. 265–293). Springer International Publishing. https://doi.org/10.1007/978-3-030-37643-7_12

Minecraft. (2011). Mojang Studios.

Monteiro, R. (2012, May 20). The guide to implementing 2D platformers | Higher-Order Fun. Retrieved February 2, 2022, from http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/

Nakamura, J., & Csikszentmihalyi, M. (2002). The Concept of Flow. In *Handbook of positive psychology* (pp. 89–105). Oxford University Press.

Nicollet, V. (2004). Difficulty in Dexterity-Based Platform Games. Retrieved February 18, 2022, from https://gamedev.net/tutorials/game-design/game-design-and-theory/difficulty-in-dexterity-based-platform-games-r2055

No Man's Sky. (2016). Hello Games.

Pagnutti, J. (2016). What Does Bach Have in Common with World 1-1: Automatic Platformer Gestalt Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *12*(2), 60–64.

Pedersen, C., Togelius, J., & Yannakakis, G. (2010). Modeling Player Experience for Content Creation. *Computational Intelligence and AI in Games, IEEE Transactions On*, *2*, 54–67. https://doi.org/10.1109/TCIAIG.2010.2043950

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Platform Game. (n.d.). In *Oxford Learner's Dictionaries*. Retrieved from https://www.oxfordlearnersdictionaries.com/definition/english/platform-game

Rhythm. (n.d.). Retrieved from https://www.britannica.com/art/rhythm-music

Schell, J. (2014). *The Art of Game Design: A Book of Lenses, Second Edition* (2nd ed.). New York: A K Peters/CRC Press. https://doi.org/10.1201/b17723

Shaker, N., Smith, G., & Yannakakis, G. N. (2016). Evaluating content generators. In N. Shaker, J. Togelius, & M. J. Nelson (Eds.), *Procedural Content Generation in Games* (pp. 215–224). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-42716-4_12

Shaker, N., Togelius, J., & Nelson, M. J. (2016). Fractals, noise and agents with applications to landscapes. In N. Shaker, J. Togelius, & M. J. Nelson (Eds.), *Procedural Content Generation in Games* (pp. 57–72). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-42716-4_4

Shaker, N., Yannakakis, G. N., & Togelius, J. (2011). Feature analysis for modeling game content quality. *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, 126–133. https://doi.org/10.1109/CIG.2011.6031998

Smith, G. (2014). The Future of Procedural Content Generation in Games. *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment; Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*. Retrieved from https://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/view/9078

Smith, G., Gan, E., Othenin-Girard, A., & Whitehead, J. (2011). PCG-based game design: Enabling new play experiences through procedural content generation. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 1–4. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2000919.2000926

Smith, G., Treanor, M., Whitehead, J., & Mateas, M. (2009). Rhythm-based level generation for 2D platformers. *Proceedings of the 4th International Conference on Foundations of Digital Games*, 175–182. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1536513.1536548

Smith, G., & Whitehead, J. (2010). Analyzing the expressive range of a level generator. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1–7. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1814256.1814260

Sonic the Hedgehog. (1991). Sega.

Stewart, S. (2013, November 19). Top Ten: Platforming Game Mechanics. Retrieved January 19, 2022, from Nintendojo website: https://www.nintendojo.com/features/columns/top-ten/top-ten-platforming-game-mechanics

Super Mario Bros. (1985). Nintendo.

Super Mario Bros. (n.d.). In *Encyclopædia Britannica*. Retrieved from https://www.britannica.com/topic/Super-Mario-Bros

Togelius, J., Champandard, A. J., Lanzi, P. L., Mateas, M., Paiva, A., Preuss, M., & Stanley, K. O. (2013). Procedural Content Generation: Goals, Challenges and Actionable Steps. In *Dagstuhl Follow-Ups*: *Vol. 6*. *Artificial and Computational Intelligence in Games* (pp. 61–75). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/DFU.Vol6.12191.61

Togelius, J., Shaker, N., Karakovskiy, S., & Yannakakis, G. N. (2013). The Mario AI Championship 2009-2012. *AI Magazine*, *34*(3), 89–92. https://doi.org/10.1609/aimag.v34i3.2492

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

Villapaz, L. (2013, September 8). "GTA 5" Costs $265 Million To Develop And Market, Making It The Most Expensive Video Game Ever Produced: Report. Retrieved July 28, 2021, from International Business Times website: https://www.ibtimes.com/gta-5-costs-265-million-develop-market-making-it-most-expensive-video-game-ever-produced-report

Vole, E. (2014, August 19). Top 10 2D Cinematic Platformers. Retrieved January 20, 2022, from 1 More Castle website: http://www.1morecastle.com/2014/08/top-10-2d-cinematic-platformers/

Warren, T. (2020, May 18). Minecraft still incredibly popular as sales top 200 million and 126 million play monthly. Retrieved October 3, 2021, from The Verge website: https://www.theverge.com/2020/5/18/21262045/minecraft-sales-monthly-players-statistics-youtube

Witkowski, W. (2020, December 22). Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic. *MarketWatch*. Retrieved from https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990

Wolf, M. J. P. (2012). *Before the Crash: Early Video Game History*. Wayne State University Press.

Yannakakis, G. N., & Togelius, J. (2011). Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing*, *2*(3), 147–161. https://doi.org/10.1109/T-AFFC.2011.6