Master Thesis

Procedural Level Generation for 2D Auto-Runner Games Based on Musical Features

L. Vanparijs

Master Thesis DKE 03-22

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. M. Stephenson
Dr. C. Browne

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

March 11, 2022

Acknowledgements

This thesis was written under the supervision of Dr. M. Stephenson, to whom I owe a special thanks as without his guidance and patience this work could not have been finished. The code that was written for this thesis can be found on GitHub through the following link: https://github.com/lvanparijs/MasterThesis. Last but not least, I would like to thank my family and friends for their trust and support during the whole process.

Abstract

This master thesis explores a novel method for procedurally generating levels for 2D Auto-runner games based on musical features. Designing 2D Platformer levels is a costly and time consuming aspect of the game design process, especially when trying to create a level that feels synchronous to a piece of music. By using procedural generation to unburden the game designers, more time could be spend on other aspects of the game.

The method proposed in this thesis consists of a combination of music feature extraction methods, generative grammars and evolutionary algorithms, to create a level generator for the videogame The Impossible Game. The rules and level structure of the game are linked to the set of musical features in order to allow the music to influence certain aspects of the level. With this link established, a geometry generation grammar is used to generate many possible candidate levels. Each candidate level is rated by a set of level critics. This scoring system is utilised in tandem with a genetic algorithm to evolve the levels towards a more desirable state.

A survey is performed to obtain feedback about several aspects of the generated levels such as fun, song similarity and difficulty. These aspects were argued to be of key importance in music based 2D Platformer levels. The results of this survey indicated that overall the levels were too difficult for the users, as the completion rate was very low. Furthermore, users rated the aspects related to song similarity relatively high. This indicates a general satisfaction from the participants towards the song matching of a level. Moreover, the proposed emptiness critic resulted in the highest correlation to the user ratings. Showing a particular liking towards that critic. Conversely, the proposed line critic had the strongest negative correlation with user ratings. Meaning when the level closely matches the pitch of a song, it becomes harder to satisfy the other critics.

Contents

Chapter 1

Introduction

The following chapter contains a brief introduction to the field of Procedural Content Generation and the motivation behind using such a method. Secondly, a brief overview of the role of music in games is given, followed by an introduction to the problem domain. Furthermore a summary of the research is provided with the problem statement and research questions included. Finally this chapter concludes with an outline of the related work and an outline of the entire system that will be discussed in this thesis.

1.1 Procedural Content Generation

Procedural Content Generation(PCG) refers to the algorithmic creation of game content with limited of indirect user input[1]. This is in contrast with the manual approach that is most commonly used in the video game industry. It is important to examine the meaning of game content in this context, as it will aid in understanding the boundaries of the definition of PCG. Shaker et al. [2] observe that relevant game content includes: items, quests, levels, maps, textures, music, vehicles and characters. Furthermore, there also exist some important components which are not considered content such as Non Playable Character Artificial Intelligence(NPC-AI) and the game engine. This distinction is made due to the fact that within the field of Artificial Intelligence, there is much more research done on applying AI methods to NPC behaviour than there is on PCG. The tightening of this definition is mainly used to set PCG apart from the more common methods where AI is used to play a game. However, it must be noted that the field of PCG itself is based on various AI methods, thus leaving the definition of what is and is not PCG slightly fuzzy.

The types of games PCG is applied to can come in several forms, video games, board games, card games and puzzles are all examples of these. More importantly than the type of game, PCG takes into account the possibilities, constraints and design of each game in order to generate suitable content.

Now that the definition of game content has been given, it is important to

consider the procedural generation part of PCG. In essence, procedural generation refers to the fact that an algorithm or method creates something. Meaning a computer can run a PCG method, optionally with limited human involvement, and output something. The next section will go into greater depth about the benefits and drawbacks of the application of PCG in certain situations.

1.1.1 Motivation

Typically content in video games is human designed, since this often offers a more detailed approach to designing a game. PCG attempts to alleviate the need for a human designer to work on the design of a game. This approach offers several benefits.

Firstly, using PCG to create aspects of a game can greatly reduce the time needed to create such content. For example, if there is an open world game that needs a large forest somewhere. A human designer would have to painstakingly design the varieties of trees and place them in a natural pattern that mimics a forest. A PCG method could generate both the varieties of trees as well as their location in the forest, thus saving a lot of time. The time saved on these tasks is also reflected in the cost of producing a game. There is potential to save on hiring human designers, this makes it possible for smaller teams with lower budgets to produce games of comparable size as larger teams that do not use PCG.

Moreover, the use of PCG can greatly enlargen the scope of a game. Coming back to the forest example, if a PCG method can generate a forest, it can do so many times over thus yielding much larger worlds human designers alone could never create. This becomes very apparent when looking at games like Minecraft[1] and Binding of Isaac[2], where the scope of the game is virtually infinite and non repetitive.

Another benefit of using PCG is that certain content can be tailored to a specific users needs. By combining player modelling and PCG, the user experience can be modified to fit the play style of the player. For example, the difficulty of enemies can be determined by profiling the skill of the player. If the player is a beginner the enemies will be easy, as they gain more skills, the enemies will get progressively harder. Generating content in this manner can lead to a more enjoyable experience for each distinct user as the learning curve is tailored to the users abilities.

Finally, another benefit of using PCG is that such methods can mimic creative design. Algorithmic approaches can come up with radically different solutions to problems compared to humans. It can provide a valid but surprising new solution to a content generation problem, leading to possibly more interesting game play. The next section will provide a brief overview of the role music plays in video games.

[1] https://www.minecraft.net/
[2] https://bindingofisaac.com/

1.2 Music in Video Games

Once an afterthought in terms of game design, nowadays music in video games has developed into an industry of its own. It is not uncommon to find video game soundtracks being released on audio platforms as a standalone album. This shows the evolution in importance of music in games.

The first use of music in a video game dates all the way back 1972 with the release of Pong. Machine storage was very limited in those days so full songs could not be used at the time. However simple sound effects were used to immerse the player more into the game, providing some auditory feedback to what was happening on the screen. As processing power and memory increased over the years, so did the length and quality of the music used in game. In the 8-bit and 16-bit era of games, music pieces could be played back with limited quality. Leading to often repetitive sound schemes that are easily remembered by the user. A famous example of this is the Super Mario Bros.[1] game from 1985, the theme song of which has become a widely known tune. However music at that time was more often used to enhance the game experience, instead of influencing it.

In the modern era of games, sound and music design has become integral to the overall experience in a game, to the point that nowadays it is not uncommon to hear fully fleshed out music pieces with a lot of detail. The use of music has also changed over time. Games have started to mould the user experience around the music, leading to new types of games such as Dance Dance Revolution[2] where the user has to move synchronous with the music. This opens up a whole new area of game design where music can be used to determine the game play, instead of the other way around.

[1] https://mario.nintendo.com/history/

[2] https://www.konami.com/games/asia/en/products/ddr$_a$/

1.3 Concepts

The game used to structure this thesis around is called The Impossible Game. The game and its rules will be explained in Chapter 2, to limit the size of the introduction. Instead a short introduction to the genre of game will be given below, followed by a brief introduction to music feature extraction and a list of useful terms.

1.3.1 2D Auto-runner Games

The genre of game used in this thesis is known as a 2D Auto-runner game. Auto-runner games are a subset of "platform 'n' run" games, racing games and skill games which is characterised by the player automatically following along some route while avoiding obstacles and possibly collecting items. Usually, the player will not be able to return to a past location. Some examples of this type of game include Jetpack Joyride [1], Temple Run [2] and Subway Surfer [3]. This type of game has gained popularity alongside the rise of the smartphone and tablet computer where games with simple controls are favoured. This is reflected in the number of downloads of the aforementioned games. For example, Subway Surfer was the most downloaded mobile game of the 2010-2020 decade with over 1.5 billion downloads[4].

1.3.2 Music Feature Extraction

Extracting features from a song requires the use of Music Information Retrieval (MIR) methods in the time and frequency domain [3]. MIR is concerned with the extraction and inference of meaningful features from music, indexing of music using these features, and the development of different search and retrieval schemes, as defined by Downie [4]. MIR methods are available to extract features of many kinds, ranging from rhythm based features to harmonic or melodic features. In this thesis low level features such as beats and notes will be extracted as well as high level features like the genre of a song. A deeper insight into this field will be provided in Chapter 3.

[1] https://www.halfbrick.com/games/jetpack-joyride
[2] https://www.imangistudios.com/games.html
[3] https://subwaysurfers.com/
[4] https://www.businessinsider.com/most-downloaded-games-of-decade-subway-surfers-to-fruit-ninja-2019-12?international=truer=USIR=T

1.3.3 Useful Terms

There are some useful terms related to music and games that are important to know when reading the rest of the thesis. A short list of them is provided below:

Music Terms

- *Rhythm* : The patterned, recurring alternations of contrasting elements of sound.
- *Pitch* : The musical quality that allows for distinction between "higher" and "lower" sounds
- *Note* : A sound of definite pitch
- *Genre* : A category of musical composition, marked by a distinctive style, form or content.
- *Timbre* : Tone-colour of a sound. Allows for distinction between instruments, even if they play the same notes.

Other Terms

- *Level Geometry* : The collection of all geometric shapes making up the structure of a game level
- *Feasible* : Capable of being brought about
- *Novel* : Something new
- *Fun* : A source of enjoyment

1.4 Research Overview

The main components of this research consists of using previously established PCG techniques to make a level generator that creates feasible levels for The Impossible Game. The level generator will use several musical features to influence the generation process. The findings from building this level generator will be reported on in Chapter 4. To guide the research process, a problem statement and research questions were formulated.

1.4.1 Problem Statement & Research Questions

From the problem domain and the aforementioned techniques the following problem statement arises.

- *How can musical features be mapped to a level generator for a 2D Auto-runner game to produce a feasible, novel and fun level?*

The following research questions have been formulated to address the problem statement.

1. What set of musical features can feasibly be mapped to facets of a level?
2. What aspects of a 2D Auto-runner level can be influenced by these musical features to provide an adequate level of difficulty, engagement and novelty?
3. How well do users feel the levels match the music used to generate them?
4. Can the resulting level generator be abstracted to a more general model for music based 2D Platformers?

The first question will be answered after investigating the possible musical features and deciding on the tangibility of each when integrating into a game. The second question will be answered by reviewing the games rules and physics constraints while considering which of the possible musical features can map to each aspect of a level. The third question will be answered by conducting a user survey to obtain feedback on the connection between the level and music. The final question will be answered once the level generator is fully explained.

1.5 Related Work

In this section previously researched topics similar to this thesis are described. Firstly, an approach to rhythm based level generation is shown. Afterwards, a research topic containing music based PCG for games is discussed.

1.5.1 Rhythm-Based Level Generation for 2D Platformers

Existing methods for procedural level generation are mostly focused on terrain generation and fitting together large, human designed chunks. While these methods can lead to satisfactory results, they are prone to repetition and still rely on the human designer for a large part. A novel rhythm-based method is proposed by Smith et al. [5], where the rhythm refers to that which the player feels with his hands while playing. The main novelty presented in this approach is the explicit separation of rhythm and geometry. Whereas previous approaches have opted for a pattern-building approach [6] in which the rhythm is implicitly chosen based on the geometry. Comparatively, the novel method yields a larger variety of possible levels and allows for a connection with music to exist as the rhythm forms the basis of a level.

1.5.2 Music-Based PCG for Games

PCG has been used as a tool for content creation in video games since the 1980's. It allows simpler creation of levels and other game content while creating more variety for the players of a game. Generally PCG methods are used with an interface where the creation parameters for a level need to be chosen by a human. In "Music-Based Procedural Content Generation for Games" [7], a new design paradigm is proposed where the entire level is created only by using existing game content and parameters without the need of human intervention. The choice of these parameters for generation are defined by the music. In order for music to define the parameters, different musical features are extracted using Musical Information Retrieval(MIR) tools. These features can then be mapped in various different ways to the game parameters. Many different kinds of levels and game play can be generated this way by simply changing to new pieces of music. However, it must be noted that this paper only uses musical features to indirectly influence aspects of the level. Whereas this thesis will link some features directly to the level design, rather than only use them as parameters.

1.6 System Overview

The system that will be described in this thesis is quite large and consists of many different components. To provide the reader with enough information a visual aid is provided in Figure 1.1. It shows all the components used to generate a final level, divided up under different categories. The main modules are the feature extraction(Chapter 3), the Level Generator(Chapter 4) and the Genetic Algorithm(Section 4.6).

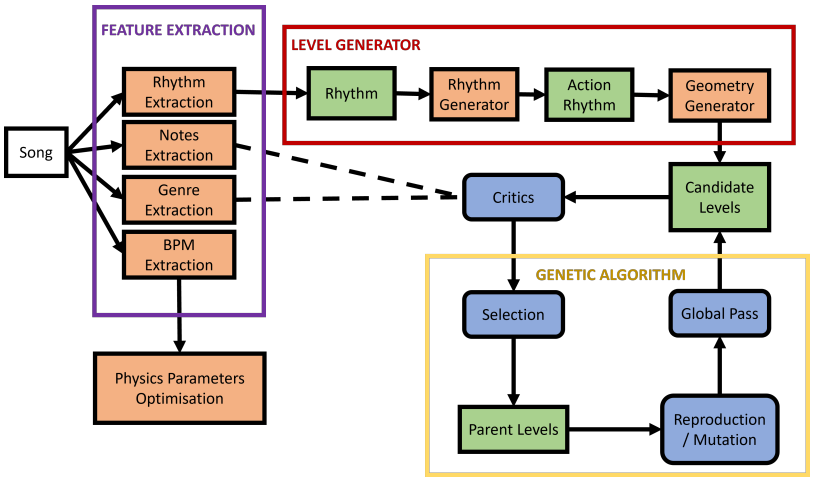


Figure 1.1: An overview of the overall system and its components. Generated entities are denoted with green, constraints are denoted in blue and other processes are shown in orange.

Chapter 2

The Impossible Game

This chapter contains information about the game that was chosen as subject for this thesis. Firstly, a brief introduction to the game is given. Subsequently, certain aspects such as obstacles and rules are described to familiarise the reader with the environment of the game. Following these sections, the characteristics which make the game stand out are defined. Ultimately a brief overview of the clone of The Impossible Game is given showing off the user interface.

2.1 Original Game

The Impossible Game is one of the best selling games on the Xbox Live Indie Game Store made in 2009 by FlukeGames[1]. It is a minimalist platformer with only one jump button the player can use. Jumping over a series of spikes, pits and blocks to get to the finish may sound easy, but as its name implies, this game could possibly be one of the hardest games ever. The technical genre of The Impossible Game is a 2D Auto-runner combined with a rhythm game. The levels tend to be closely integrated with the music, giving the player a feeling of being synchronised with the music.

There are some qualities to The Impossible Game that are harder to quantify. Firstly, this game gained popularity, as the name suggests, due to its apparent difficulty to play. Every level requires the players full attention to the level and the rhythm of the music. The actions the player performs in the game are often closely linked to the music playing in the background. This gives off a feeling of synchronicity between the game and the music.

The intensity of the music used plays a big role in the overall difficulty of the levels. As the player progresses through levels the rhythm used for each level gets progressively faster. Furthermore, the game is also characterised by a very localised level of information. Meaning the player does not have much knowledge of the obstacles coming ahead, at most 3 obstacles are visible ahead of the player at once. The localised information can be overcome once the player

[1] https://impossible.game/1/

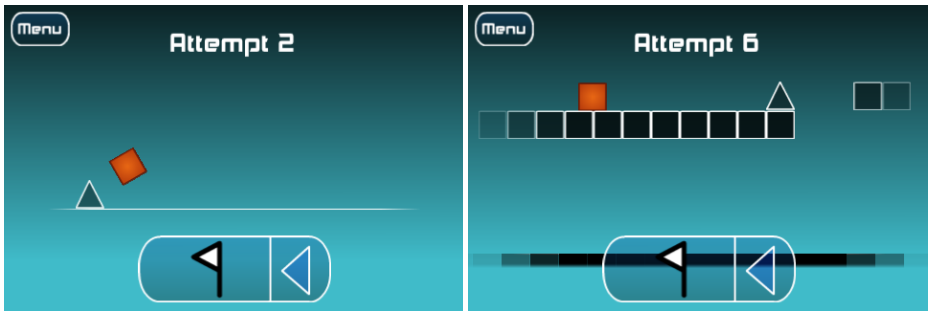
is familiarised with the level and the music, making the game more predictable and less difficult.

2.1.1 Environment

The Impossible Game has a minimalist style, making the most of the few components available. Together these components form the basis of every level and describe the overall environment of the game. This section will list and explain all the components that create the environment. The different components can generally be divided into 3 types: deadly obstacles, neutral obstacles and the player.

Deadly Obstacles

The first type of obstacle is of the deadly kind. These are obstacles that, when touched by the player, immediately result in a game over. These obstacles are the spike and the lava, examples of these can be seen in Figure 2.1(b), where the spikes are represented by the dark grey triangles with white outline and the lava is the black rectangle without outline. These obstacles can be avoided by using the jump button at the right time.



(a) Sample game play of the player jumping over a spike

(b) Sample game play of the player sliding on a platform

Figure 2.1: Sample of game play situations in The Impossible Game

Neutral Obstacles

The neutral obstacles also consist of 2 components, the platform and the block. The platform is a constant in every level and is represented as the white line horizontally spanning the width of the screen. This essentially serves as the ground and limits the player from going lower. It is not dangerous to the player as it is not possible for the player to collide with it in any way to cause a game over. Other pieces can be placed on top of the platform. Both the lava and spike components are allowed on top of the platform as well as the block component. Represented by a black square with white outline, the block component is the

only component that can both be used to run/jump on top of, and cause a game over when the player runs/jumps into it from the side.

Player

The third type of component, the player, is represented as a orange square with a black edge enclosing it. The player moves to the right at a constant speed and is also able to jump, making it the only dynamic component in the game.

Starting Setup

The starting setup for this game is the same for every level. The player starts on the left side of the screen and is static until the game commences. To the right, the player will possibly see the first obstacles lined up for the level. A small silence will be replaced by the music beginning to play and the player moving to the right, the game has started. The player has only one action at their disposal, a dedicated button can be pressed to perform a jumping action. The basic starting position can be observed in Figure 2.2.

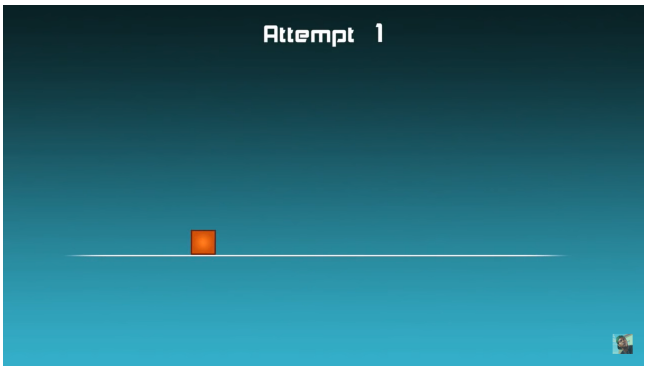


Figure 2.2: The basic starting position of the game

2.1.2 Rules

While a detailed list of official rules is not provided by FlukeGames, an attempt will be made in this section to do just that. By using and observing the game client a set of rules can be inferred. The scope of the game is limited enough to determine these rules from mere observation. However, it must be noted that these rules could become outdated information in case that the rules are changed or new content is added that requires additional rules. As of the publishing of this thesis, these rules are still accurate.

The main objectives for the player of the game are threefold:

- *Make it to the end of the level*
- *Avoid the dangerous obstacles on the way*
- *Use as few attempts as possible*

The rules underlining the game are as follows:

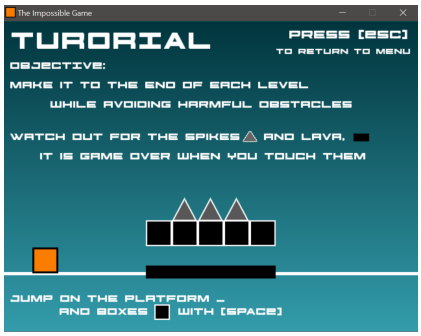
- *The player moves at a constant speed to the right, no returning is allowed*
- *The player is allowed to jump at any time if and only if:*
  - *The player is touching the platform*
  - *The player is touching the top of a block whilst not touching a deadly obstacle at the same time*
- *Every time there is a game over, the attempt counter is incremented*
- *The game is over when one or more of these losing conditions are met:*
  - *The player runs/jumps into the side of a block*
  - *The player runs/jumps into the side of a spike*
  - *The player jumps on top of a spike*
  - *The player jumps on top of a lava component*
- *The level is completed once the win condition is met:*
  - *The player reaches the finish line*

2.2 Clone

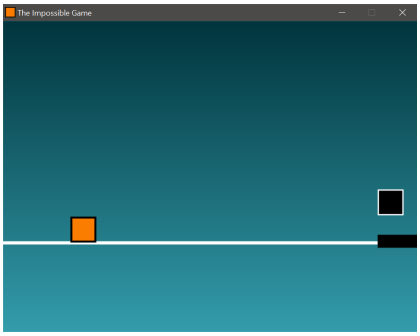
The Impossible Game is not open source, meaning that the original game engine could not be used to build a level generator for. This necessitates the creation of a clone for this game. There are few clones available online, and each of them is less than desirable. Therefore a clone was created from scratch to mimic the original game.

The clone consists of 3 different scenes. Firstly the main menu, as shown in Figure 2.3a. The main menu consist of 3 options that can be selected by using the UP and DOWN arrow keys and pressing ENTER. The currently selected option is highlighted in white.

(a) The main menu, the player can select a level, the tutorial or quit the game.



(b) The tutorial level, the player gets the chance to read the rules and familiarise themselves with the mechanics and obstacles of the game.



(c) The basic starting position of the game.

Figure 2.3: The different scenes in the clone

If the user selects the "tutorial" option, the tutorial level depicted in Figure 2.3b pops up. Here the player can practice navigating past the various obstacles available in the game, as well as read up on the rules they need to follow.

Alternatively, if the user wants to select a level to play they must navigate to the level option and use the LEFT and RIGHT arrow keys to select the specific level. This leads the user to the game screen as depicted in Figure 2.3c. A audible countdown is heard after which the music starts and the game has commenced. The player can decide to jump by pressing the SPACE key. Finally, if the user selects the "quit" option on the main menu the window will close. The clone of the game was written entirely in Python 3.8 by making use of the PyGame[1] library.

2.2.1 Physics Constraints

The physics system consists of several key variables. The gravity, x and y position of the character, the jump velocity and the players horizontal velocity. The original The Impossible Game utilised songs of similar BPM, leading to the physics system and level pieces being unchanged between levels yet still leading to a synchronous experience by the player. For example, the jump distance between two pieces is always 3 empty blocks of space/obstacle to jump over. However for this thesis different musical genres with often very different BPM were used. Since the speed of the player is dependent on the BPM of the seed song, the physics system had to be adapted to suit these different speeds. In order to be able to use the same level pieces for songs of different BPM compared to The Impossible Game, the gravity should adapt to the speed of the player to ensure that the players jump distance remains the same. This was done by calculating the players gravity by slowly incrementing it and simulating the players jump distance. This method results in the player being able to jump the same distance regardless of the song being used to generate the level.

[1] https://www.pygame.org/

Chapter 3

Music Feature Extraction

This chapter will provide an overview of Music Information Retrieval(MIR) and the types of features that can be extracted. Furthermore, the processes underlying the extraction of the wanted features are explained in Sections 3.2.1, 3.2.2, 3.2.3. At the end, a brief overview of the tools used for feature extraction is given.

3.1 Music Information Retrieval

Research into Music Feature Extraction also known as Music Information Retrieval (MIR) started in the 1990s [7]. In the early days, research was heavily focused on symbolic representations of music such as the MIDI format. Over time with the increase of computing power, MIR has adapted its focus around extraction and inference of meaningful features, indexing of music and search retrieval schemes [3]. From these subfields, extraction of meaningful features will be the central focus of this research.

The features used in this thesis are distinguishable via their abstraction level. The abstraction level covers the range of low-level signal features such as pitch or loudness to semantic descriptors of high abstraction level such as genre or mood [3]. These features can be used to describe a song's characteristics, searching for similar music or classify the music based on genre.

Low-Level Features

Low-level features are extracted within the time and frequency domains representations of a song.

Features derived in this way from an audio signal are quite meaningless on their own, as they describe very time sensitive information about the song. However these low-level features can be used to infer higher level features. The main lowest level features consist of pitch, loudness, timbre and spectral moments as described by Wessel [8].

With these lowest level features, several other low-level features can be inferred. For example, in order to compute the rhythm or instrument class of a song, the loudness and timbre features can be used. Furthermore features such as melody, harmony and chords can be extracted by using the pitch feature. The melody and the chords can be used to describe aspects of a song like the musical key it is played in.

There are several rhythmic features that can be computed with timbre and loudness. Mainly the timing, tempo and grouping of rhythms are extracted this way. An example of this rhythm extraction can be observed in Figure 3.1, where the beat positions, loudness and beats per minute (BPM) can be calculated. The top image represents the initial musical signal in the time frequency domain. The middle image shows the estimated onsets of beats based on the loudness and timbre features. The bottom image shows where only the main beats have been preserved as the beat positions, thus an estimate of the BPM can be made.

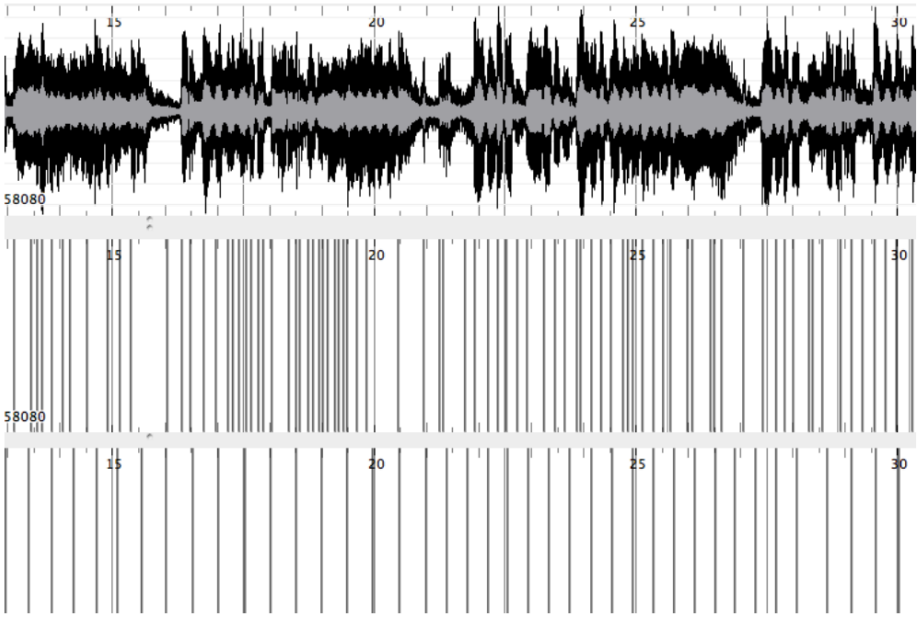


Figure 3.1: Rhythm seen as periodicity of onsets. Example of an input signal (top), estimated onsets (middle), and estimated beat positions (bottom). [3]

High-Level Features

All the aforementioned low-level features can be used to infer higher level features by using classification or auto tagging methods [7]. Research efforts on music classification have been mainly concerned with classifying music with respect to instruments [9], genre [10], mood [11] and culture [12]. These underlying technologies work to a certain extent, but show a "glass-ceiling" effect [3], with the state-of-the-art algorithms yielding around 80% accuracy. Lippens et al. [13] and Seyerlehner [14] have demonstrated a cause of this effect by showing that human agreement on music belonging to a certain genre reaches between 75% and 80%. Meaning that the training set used to train the classifier will differ depending on the expert labelling the songs.

3.2 Feature selection

In order to select a set of musical features to be extracted several aspects should be considered. The main consideration is how a musical feature could be linked to features in a video game.

The most straightforward feature to select is the rhythm. Following the definition of Desain and Windsor [15], rhythm is related to the architectural organization of musical events along time and incorporates regularity and differentiation. Timing actions on a musical rhythm has been a staple of music based video games, this is also very prevalent in The Impossible Game. The aim of this feature is to have the player experience a sense or rhythm and synchronicity with the music.

The second selected feature was the notes of the song. This was chosen instead of chords or key of the music since the notes span a shorter time and describe the music in higher detail. The idea is to transfer some of this detail into the final level. One can imagine a song building up to a chorus, the notes will often climb higher towards a climax. If at the same time the player starts climbing higher it adds to the immersion experienced by the player. This can add to the feeling of anticipation of the player about what is to come as well.

Finally, the genre of the song was chosen to be another feature. A connection between genre and a game is less concrete compared to notes or rhythm. But the idea is to have the genre determine a more global aspect of the level generation. In some games genre has been used to alter the visuals of the level giving of a different mood. Intuitively, more intense music like rock or hip hop should have different types of levels compared to classical music. An explanation of how each of these features is extracted can be found in next sections.

3.3 Tools for feature extraction

The are a wide array of frameworks, libraries and tools that perform Musical Feature Extraction. Some of the popular examples include Essentia[1](C++ Library), Spotify WEB API[2](cross platform). libROSA[3](Python library), jMIR[4](Java library) and Sonic Visualiser[5](Visual application). Most of these tools have similar audio analysis capabilities, due to the same algorithms being used in them. The focus in this study is the libROSA library, because the project was written in Python, and Spotify WEB API to classify the genre of each song.

3.3.1 libROSA

LibROSA[1] is an open-source Python library that can be used for audio analysis and MIR. The library has a focus on algorithms that analyse an audio signal, with a specific focus on music. It provides the building blocks to build a MIR system. More specifically it includes all the low-level features that are desired for this research, it can extract rhythmic features as well as tonal features. It is however lacking in the classification department and can not estimate the genre of an audio file.

3.3.2 Rhythm Extraction

The rhythm features extracted for this research are twofold: the Beats-per-Minute(BPM) and the tracked beats of the song. Firstly, the tracked beats are obtained by using the corresponding function in libROSA named *beat_track*. This is a dynamic programming beat tracker which extracts the individual beats of a song. The beats are detected in three stages:

1. Measure onset strength
2. Estimate tempo from onset correlation
3. Pick peaks in onset strength approximately consistent with estimated tempo.

Initially the strength of onsets is measured. An onset refers to the beginning of a musical sound, often a musical note or a beat of a rhythmic instrument. Onsets can be detected by a variety of features, such as changes in detected pitch and increase in spectral energy(a measure of how quickly the power spectrum of the signal changes). There are many more, however for the sake of brevity these will be omitted in this thesis. By correlating the onsets returned in step 1, the tempo can be estimated. This estimate is subsequently used to approximate

[1]https://essentia.upf.edu/
[2]https://developer.spotify.com/documentation/web-api/
[3]https://librosa.org/
[4]http://jmir.sourceforge.net/index_jAudio.html
[5]https://sonicvisualiser.org/
[1]https://librosa.org/

which peaks in the onset strength are consistent with the tempo. The most likely onsets are selected to form the tracked beat.

It is important to note that not every single rhythmic element of the song is extracted this way. Only the main rhythm of the song. Figure 3.2 illustrates this concept in more detail. A more detailed explanation of this method can be found in [16].

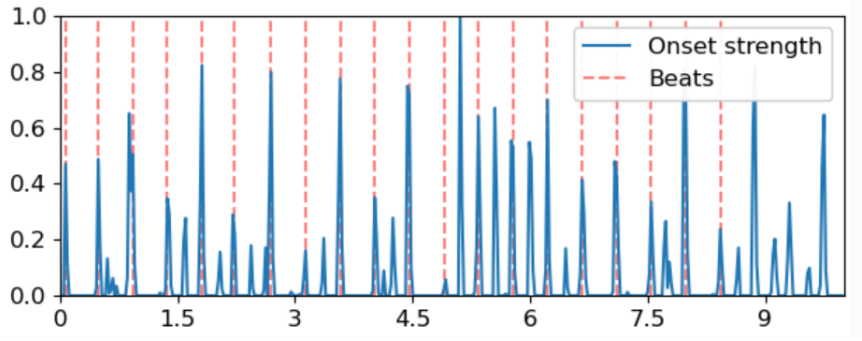


Figure 3.2: An example of the returned beats by the beat tracker. Not every rhythmic element is included as shown by the onsets.

The extraction of the BPM of the song starts the same way as the beat tracker. Firstly, the strength of the onsets is measured which yields the magnitudes of the onsets as shown in the top half of Figure 3.3. Via a local onset auto-correlation method as described in [17], a tempogram is constructed as shown at the bottom of Figure 3.3. The tempogram is a feature matrix that uses local estimates to construct a more global image with respect to the song. The prevalence of a certain tempo at a moment in time is displayed by the purple to orange gradient, where orange denotes a higher prevalence of the tempo at that moment.

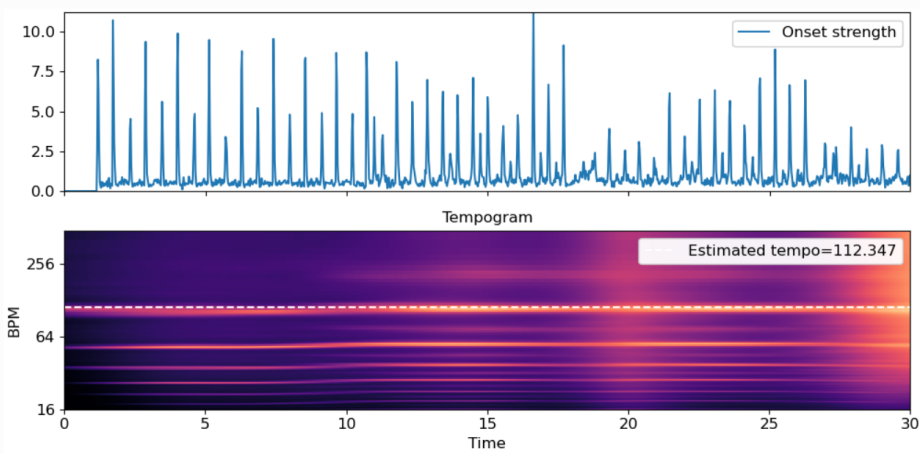


Figure 3.3: Onset detection (top) and the tempogram (bottom). The tempogram is a feature matrix which indicates the prevalence of certain tempi at each moment in time.

3.3.3 Notes Extraction

In order to use the notes to base the levels on, a timeline of notes should be extracted. Each note should take up an interval of time, these intervals and notes should guide the height of the generated game level. In order to achieve this a fundamental frequency (F0) estimation is performed along the time dimension. The fundamental frequency is estimated using the Fourier transform, which yields a frequency in Hz. These frequencies can then be converted to a height map, with the difference in frequency serving as a height difference between level pieces.

The algorithm used for this purpose is the probabilistic YIN (pYIN) algorithm. pYIN [18] is a modified version of the YIN algorithm [19] used for fundamental frequency estimation. In the first step of pYIN, F0 candidates and their probabilities are calculated using the YIN algorithm. Secondly, a Viterbi decoding [18] is used to estimate the most likely fundamental frequency sequence.

As can be seen from Figure 3.4, the extracted frequencies form a sort of height line on their own. This is part of what is used to score the height line of the levels.

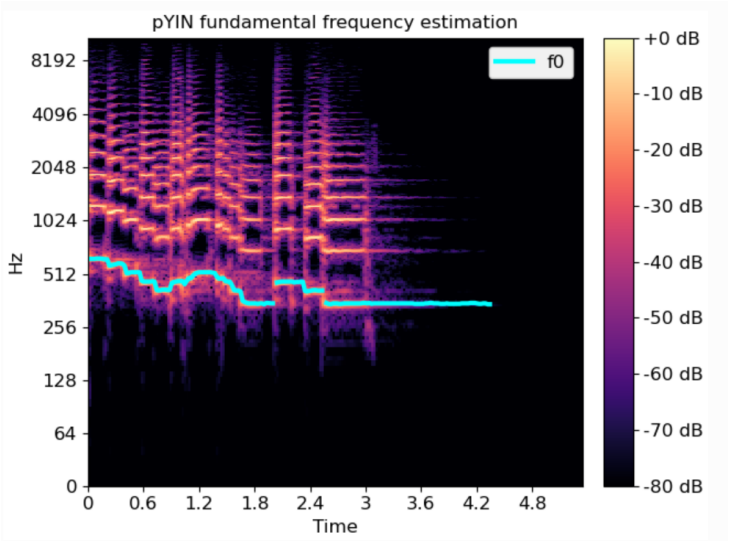


Figure 3.4: Fundamental frequency(F0) estimation using the pYIN algorithm

3.3.4 Genre Extraction

Genre classification is done via the Spotify Web API, the exact method is unknown as Spotify keeps their methods private.

3.3.5 Spotify WEB API

The Spotify WEB API[2] provides users access to data from Spotify's library, such as playlists, songs, podcasts and more. Every single song in the Spotify library has been examined through a series of audio analysis and classification, from which the results were made available to users of the API. Although this tool can not be used for every single song in existence, the scope of the library was large enough to have plenty of choice with regards to the music. Since all analyses has been done before the songs are added into the library, accessing information about the songs is very fast. Unfortunately however, using the web API requires an internet connection and a premium Spotify account, which make it less accessible for the average person. However this was not an issue in this research.

[2]https://developer.spotify.com/documentation/web-api/

Chapter 4

Level Generator

This chapter will cover the topic of the level generator created for generating levels for The Impossible Game. It contains an overview of the system as a whole, followed by a explanation of the rhythm generator used to generate the level components on locations that will fit the musical rhythm. Afterwards an overview of everything involved in generating the level geometries based on the rhythm is given. This consists of the level pieces used to construct a level, the grammar by which a level is generated and the algorithm with which a level is generated. Subsequently, the critics with which the levels receive a score are defined. Finally, the Genetic Algorithm used to evolve the levels is given.

4.1 System Overview

To be able to generate levels from a piece of music a three step plan is used.

Firstly, as previously explored, the musical features are extracted from the given music. These features are passed on as input to the level generator, where the rhythm is used to generate candidate levels. The other features are used to partially critique the level and give each candidate level a score. These steps will be further explored in their respective sections.

Secondly, the input rhythm is used to generate level geometries for candidate levels. A level geometry is the collection of all obstacles(platform, boxes, spikes, lava) of a single level. These geometries are critiqued by using musical features such as notes or genre. The level geometries are generated via a two-tiered grammar based approach. They are subdivided into different sections, with each section containing a level piece, also called a rhythm group [5]. Generating level geometries is done in two stages. The first stage of the process takes the musical rhythm of a song and generates a set of actions that fit the musical rhythm. For the sake of clarity this set of actions will be referred to as action rhythm. The second stage of this process takes this action rhythm and uses a geometry generation grammar to generate a level geometry. From this grammar many different levels are generated that form a candidate set from which the

final level will be selected. This process can be observed in Figure 4.1.

Once a set of candidates have been generated and critiqued, the fittest candidates are selected and used to create a new generation of candidates. This is done by using a Genetic Algorithm(GA). Over generations the levels are gradually improved to fit the desired criteria.

This approach includes both a generation and a testing step and is thus a generate-and-test approach to PCG. In fact, the level generator is a special case of generate-and-test, known as search-based PCG (SBPCG), as defined by Togelius et al. [20]. This narrower definition requires a test function that does not simply accept or reject levels, but scores them on a continuum. This type of function is called a fitness function. The specifics of the fitness function for this application will be given in its own section. Furthermore, SBPCG requires newly generated candidate levels to be contingent on the fitness value assigned to previously generated candidates. This is ensured by using a form a elitism in the selection process of the GA.

Since there is no general proof of convergence of GAs [21], there is no guaranteed completion time or solution quality that can be expected. Therefore it was determined that this SBPCG approach was unsuitable for generating the levels online, and is better suited to offline generation. This is further supported by the fact that the analysis of musical features requires context based on the entirety of the song. For example, when rhythm is extracted, likelihood estimations are done on singular beats based on the global context of all beats to estimate the most likely rhythm sequence.

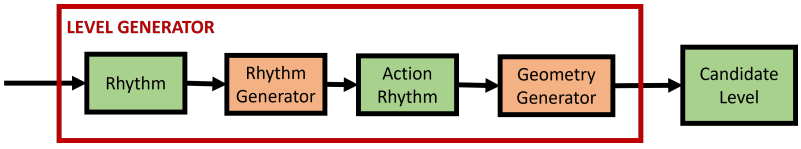


Figure 4.1: An overview of the level generator and its components. Generated entities are denoted with green and processes are shown in orange.

4.2 Rhythm Generator

Rhythm groups are non-overlapping, small sections of a level that describe a sense of rhythm the player needs to follow. To output these rhythm groups, the rhythm extracted from the song as described in Section 3.3.2, is combined with one of two possible actions: JUMP or NO_JUMP. To introduce variety, these actions are assigned with equal probability to each extracted beat from the song. Once this action rhythm has been generated it is used to generate all the geometries of the geometry generator. An example of an action rhythm is given in Table 4.1, here the first 4 beats of Figure 3.2 are used to create this example action rhythm.

Time (s)	0	0.45	0.95	1.45
Action	JUMP	JUMP	NO_JUMP	JUMP

Table 4.1: A simple example of an action rhythm based on Figure 3.2, the action is supposed to be performed at the time given.

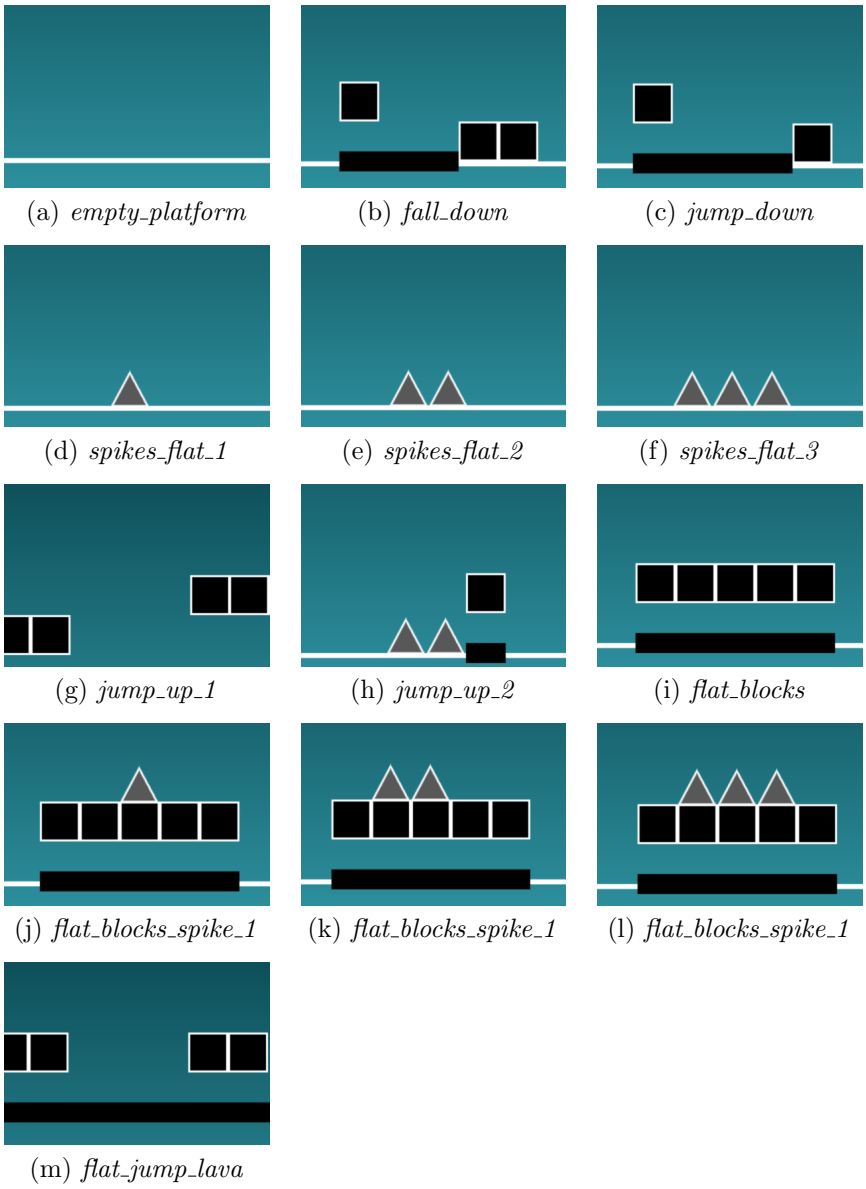
4.3 Geometry Generator

The Geometry Generator's responsibility is to take the action rhythm from the Rhythm Generator and use it to generate many level geometries. This can be done by using the action rhythm and combining it with a geometry generation grammar. Generative Grammars are most often used in theoretical linguistics, it aims to provide a set of rules that precisely predict which combinations of words are able to make grammatically correct sentences. In the realm of PCG however, a generative grammar can be used to define the rules of creating a level. The words in linguistics are parallel to level pieces. The rules of the game determine whether combinations of these level pieces are allowed, which is parallel to being grammatically correct. The purpose is to generate a wide array of interpretations for the same action rhythm. These interpretations vary due to the different possibilities for each action as dictated by the generation grammar. For example, a single JUMP action can result in jumping to a higher level, or jumping over an obstacle but maintaining the same height after the jump. This can be observed by considering Figure 4.2 (g) and (h), these can be generated equally likely when the player needs to jump to a higher level, thus resulting in a differing level geometries. The generation of these geometries is stochastic in nature as multiple level pieces can be used for interchangeably at the same place in a level. The stochastic generation process allows for the exploration of different possible level geometries for the same song.

4.3.1 Level Pieces

From The Impossible Game a set of 13 distinct level pieces could be identified. These pieces were named and used to define the geometry generation grammar. Determining which piece belonged where in the grammar was done by considering the height difference between the start and end of the piece. If the end height was higher the pieces could only fit with a JUMP_UP action, whereas if the heights are the same it can fit a JUMP_FLAT action or a NO_JUMP_FLAT if there is no obstacle.

The first level piece is the most basic one. An empty platform(Figure 4.2 (a)) without any blocks, spikes or lava on top of it. This piece is useful to add to the beginning of a level, giving the player some time to prepare for the obstacles. It can also be used to give the player some respite for sections in a song that are calmer and do not require any obstacles.



(a) *empty_platform* (b) *fall_down* (c) *jump_down*

(d) *spikes_flat_1* (e) *spikes_flat_2* (f) *spikes_flat_3*

(g) *jump_up_1* (h) *jump_up_2* (i) *flat_blocks*

(j) *flat_blocks_spike_1* (k) *flat_blocks_spike_1* (l) *flat_blocks_spike_1*

(m) *flat_jump_lava*

Figure 4.2: All the level pieces visually represented with their respective names.

The next level pieces are a set of 3 variations on the same obstacle. In The Impossible Game a varying amount of spikes is used, the higher the amount of spikes, the more precision it requires to successfully jump over them. Figure 4.2 (d), (e) and (f) show the three different variations on spikes on the platform.

The following level piece can be used when no action is performed by the

player, resulting in the player sliding over the gap from the leftmost block to the next block. The piece is depicted in Figure 4.2 (b).

In contrast to the previous piece, the *jump_down* piece is used when the player performs a jump action. However both result in a lower player height at the end of the piece, thus both pieces can be used when the notes in the song get lower between pieces. The geometry of this piece is shown in Figure 4.2 (c).

The next two pieces are the only two pieces used to jump to a higher height. They can be used when the player jumps. Most commonly they are used when the notes go higher between pieces. Examples of these pieces can be found in Figure 4.2 (g), where the jump height is only one block, and Figure 4.2 (h) which has a height difference of 2 and requires more precision than the 1 block jump. In Figure 4.2 (g), the leftmost half block does not belong to this level piece.

Similar to the *empty_platform* level piece, the *flat_blocks* piece also serves as filler piece when the players action rhythm dictates that no action should be used. This piece's geometry can be seen in Figure 4.2 (i).

A more dangerous version of *flat_blocks*, these next 3 pieces are variations with different numbers of spikes. Similar to the *spikes_flat_(1,2,3)*, the more spikes in a variation, the more precise the player needs to be to cross the level piece. All variations of this piece and their names can be seen in Figures 4.2 (j), (k) and (l).

The final level piece from the 13 is the so called *flat_jump_lava*, similar to the pieces in Figures 4.2 (j), (k) and (l)it requires a jump to cross the piece. Performing no action will lead the player to fall down into the lava. The geometry of this piece can be seen in Figure 4.2 (m).

4.3.2 Geometry Generation Grammar

Now that all pieces have been defined and named, they can be integrated in the Geometry Generator. To define which pieces fit the players action, a Geometry Generation Grammar is used. These grammars have been used previously to generate levels, missions and quests [22] and is thus well suited for this task. For this particular grammar the common Backus-Naur Form (BNF) is used.

This BNF grammar can generate basic structures of a level, however parameterising this to move towards some desired level structures is not possible. That is why this BNF grammar generation is combined with an evolutionary algorithm [23], where certain features of the levels can be parameterised and optimised for. The following is the proposed BNF grammar for the level generator.

GEOMETRY GENERATION GRAMMAR

JUMP_UP $\rightarrow <jump\_up\_1> \mid <jump\_up\_2>$
JUMP_FLAT $\rightarrow <spikes\_flat\_1> \mid <spikes\_flat\_2>$
$\mid <spikes\_flat\_3> \mid <flat\_blocks\_spike\_1>$
$\mid <flat\_blocks\_spike\_2> \mid <flat\_blocks\_spike\_3>$
$\mid <flat\_jump\_lava>$
JUMP_DOWN $\rightarrow <jump\_down>$
NO_JUMP_UP $\rightarrow$ **NO_JUMP_FLAT**
NO_JUMP_FLAT $\rightarrow <flat\_blocks> \mid <empty\_platform>$
NO_JUMP_DOWN $\rightarrow <fall\_down>$

Recall the action rhythm has at each beat time an action for the player to perform. Either JUMP or NO_JUMP. Before the generation process a random direction is assigned to each action, resulting in 6 options: JUMP_UP, JUMP_FLAT, JUMP_DOWN, NO_JUMP_UP, NO_JUMP_FLAT, NO_JUMP_DOWN. Each action has one or more possible level pieces that fit with the action as shown in the Geometry Generation Grammar. To give an example, the action rhythm from Table 4.1 is adapted to reflect the direction of movement that was assignment before generating the geometry. The result of this can be observed in Figure 4.2.

Time (s)	0	0.45	0.95	1.45
Action	JUMP_UP	JUMP_FLAT	NO_JUMP_DOWN	JUMP_FLAT

Table 4.2: A simple example of an action rhythm with directions based on Figure 4.1, the action is supposed to be performed at the time given.

4.3.3 Geometry Generation Algorithm

The Geometry Generation Grammar from the previous section was used as a guideline to define the Geometry Generation Algorithm. By generating the level pieces with equal probability, the most variation amongst interpretations is achieved.

Algorithm 1 Geometry Generation Algorithm

```
1: Initialise list of level pieces and their positions
2: Put en empty piece as first in the list
3: for Every possible location for a level piece do
4:     if The player action is JUMP then
5:         Choose direction UP, FLAT or DOWN with equal probability
6:         Pick level piece based on the direction and the geometry grammar
7:         if JUMP_UP then
8:             Choose jump_up_1 or jump_up_2 with equal probability
9:         end if
10:        if JUMP_FLAT then
11:            Choose spikes_flat_(1,2,3) or flat_blocks_spike_(1,2,3) or
    flat_jump_lava with equal probability
12:        end if
13:        if JUMP_DOWN then
14:            Choose jump_down
15:        end if
16:    else (The player action is NO_JUMP)
17:        Choose direction UP, FLAT or DOWN with equal probability
18:        (NOTE: UP and FLAT have the same result for NO_JUMP)
19:        Pick level piece based on the direction and the geometry grammar
20:        if NO_JUMP_UP or NO_JUMP_FLAT then
21:            Choose flat_blocks or empty_platform depending on the end height
    of the previous piece.
22:        end if
23:        if NO_JUMP_DOWN then
24:            Choose fall_down
25:        end if
26:    end if
27: end for
```

The algorithm results in a sequence of level pieces being generated, with no pieces overlapping each other. This implies that each level piece is a necessary component for the player to be able to complete the level. The levels resulting from this algorithms alone are rhythmically synchronous with the beat of the music. However, they may not be closely matched to the pitch or genre of the song. An example of these could be a few consecutive *jump_up_1* pieces whereas the pitch of the song tends to lower, or a very calm song that contains a high percentage of spikes when you would expect the level to be more calm.

To illustrate the function of the geometry generation process, consider the action rhythm with direction obtained in Table 4.2. The example below shows a possible outcome of such a action rhythm being applied to the geometry generation algorithm.

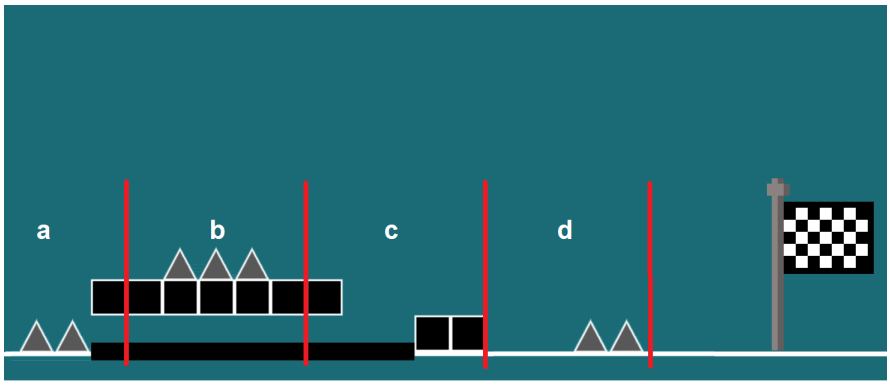


Figure 4.3: An example of a level generated by the action rhythm given in Table 4.2. The names of the corresponding level pieces are as follows: a) *jump_up_2*, b) *flat_blocks_spike_3*, c) *fall_down* and d) *spikes_flat_2*

4.4 Critics

Using a grammar as presented in this thesis commonly leads to over-generation [24]. Over-generation happens when the constraints specified in the grammar are not tight enough, leading to results that are often undesirable. This is caused by the level design space being too large. Such a grammar is good at generating levels based on local constraints. These constraints are based on the action by the player(JUMP, NO_JUMP) and the direction(UP, FLAT, DOWN). However, this grammar does not take into account global constraints. These constraints come in the form of the musical features and other desired level features. For example, the height of the notes of the seed music track is a desired path to follow for the level generator, as the aim is to produce a level that is synchronous with the music track. Following this track can not be done with a grammar, as every music track has a different notes line.

In this section these constraints will be explored in the form of level critics. Smith et al.[25] introduces the notion of critics as follows. Given a generated level, a critic can perform tests over the entire level to determine how well a given global constraint is met. Each critic analyses the data of a level and returns a score between 0-1 based on its specific requirements. This score is a partial measure of what makes a level desirable. Combined, these critics achieve a more well rounded evaluation of a level, which should yield an overall more desirable level. The used critics are a combination of adapting the proposed critics from [25] and [24], and proposing new critics based on 3 key desirable aspects that should be included in the final levels. These components consist of:

- Song Similarity: A measure encapsulating the level synchronicity with the seed music track.
- Difficulty: A measure of the level of skill needed to complete a level, based on the type and frequency of obstacles.
- Fun: Hard to define exactly, but variation in the level structure is definitely a part of it, as well as variation in obstacles.

Because the definition of "fun" is rather fuzzy, it is left up to the player/survey taker to determine whether it is a fun level or not. Furthermore, some abbreviations for critics will be used in this chapter. A brief summary of these can be found below.

- LC : Line Critic (score)
- CFC : Component Frequency Critic (score)
- EC : Emptiness Critic (score)
- VC : Variety Critic (score)
- JC : Jump Critic (score)

4.4.1 Song Similarity

Line Critic

One of the musical features given as input to the level generator is the height line of the notes, which is extracted from a song as described in Section 3.3.3. This line serves as a guideline for the level that is being generated. More specifically, the cumulative squared distance of the height line of the level to the height line of the song is divided by the theoretical maximum cumulative squared distance. The concept behind this critic was adapted from the proposed line distance critic in [25] to work with musical input. Figures 4.4 and 4.5 show respectively good and bad examples of the distance between a notes lines and a height line of a level. The distance shown is then divided by the theoretical maximum distance between these two lines. This ratio is then subtracted from 1 in order to make the critic score 0 if the distance between the two lines is equal to the theoretical maximum, and 1 if the lines match exactly. While it is rare to generate a level that matches a notes line exactly, this critic should allow the level to evolve towards being similar in feeling to the song. ie. If the song has a section with higher notes, the level will generally follow suit.

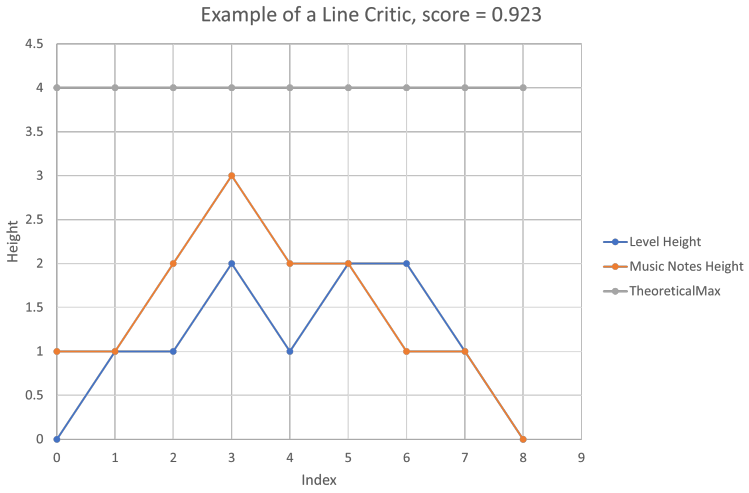


Figure 4.4: An example of a level height which would result in a good line critic score.

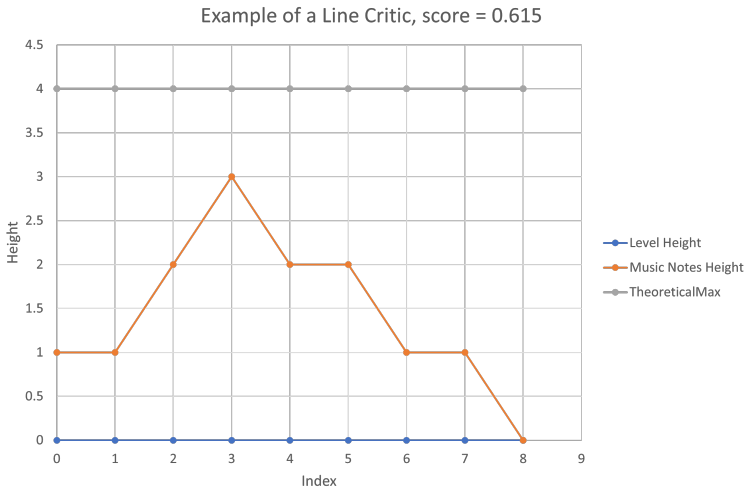


Figure 4.5: An example of a level height which would result in a bad line critic score.

$$LC = 1 - (squaredSumDistance/squaredMaxDistance)$$

4.4.2 Difficulty

Component Frequency Critic

The component frequency critic is a critic that is based on the proposed component frequency critic in [24]. It calculates the ratio of dangerous spikes relative to the amount of harmless boxes. This ratio is then subtracted from the desired ratio which is specific to each genre of music. Table 4.3 shows the desired ratios for each genre.

Classical	Jazz	Electronic	Rock	Rap
0.05	0.15	0.25	0.35	0.55

Table 4.3: A table showing the desired spike ratio for each genre of music.

$$CFC = 1 - (SpikeRatio - DesiredSpikeRatio)^2$$

Using 1 minus the squared distance ensures that the level will evolve towards the desired ratio, as well as get punished more the further it is away from the ideal ratio. The desired ratio for each genre of music was determined by the intensity of the music genre. Since classical music is often the calmest from the set, it was given the lowest desired ratio of spikes. Jazz and Electronic are often

wilder than classical music, and thus get a higher ratio. This logic also applies to Rap and Rock, which are set at a highest ratios. The main idea behind using the spike ratio is that intenser music will yield more dangerous levels, potentially suiting the music better.

Emptiness Critic

The emptiness critic was put in place to discourage the level generator from generating completely empty, or mostly empty levels. This is undesirable since this would lead to a lack of difficulty and engagement for the player. The score for this critic is calculated by the ratio of level pieces that are not the *empty_platform* piece, over the total amount of level pieces.

$$EC = NumNonEmptyPieces/TotalNumPieces$$

4.4.3 Fun

Jump Critic

The next critic is the Jump Critic. This critic was designed to counteract monotonous level with very few jumps. This was especially prevalent in calmer songs while testing. The jump critic scores a level based on the ratio of jumps over the total amount of actions in a level

$$JC = NumJumpActions/TotalNumActions$$

Variety Critic

The variety critic was implemented with the aim to encourage the generated level to have a lot of height variation in it. This was done by the logic that a more varied level should be more fun to play for the user. The score is calculated by computing the variance of the level height line and the variance of the theoretical maximum height variation in a level. The theoretical maximum is a level that has alternating *jump_up_2* pieces and *fall_down* piece, this results in a constantly alternating height between 0 and 2. This critic attempts to encourage the evolution of levels to move towards less monotonous levels. The more variation in level height, the higher the score. Figures 4.6 and 4.7 show a good and bad example of the variety critic score.

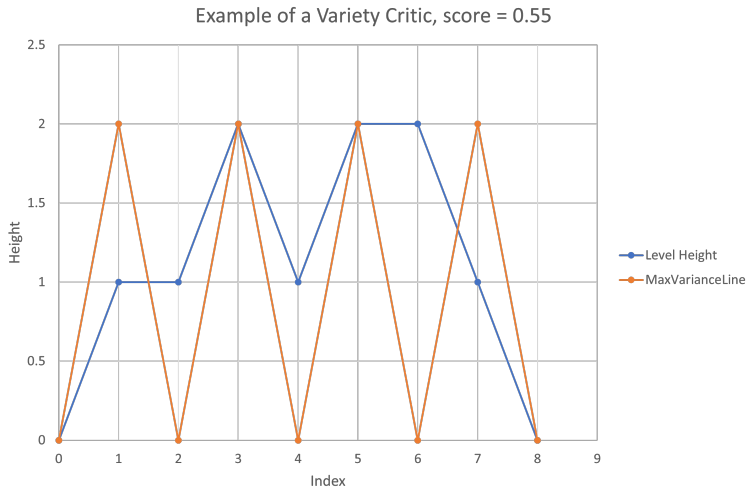


Figure 4.6: A small example of what score the variety critic would give.

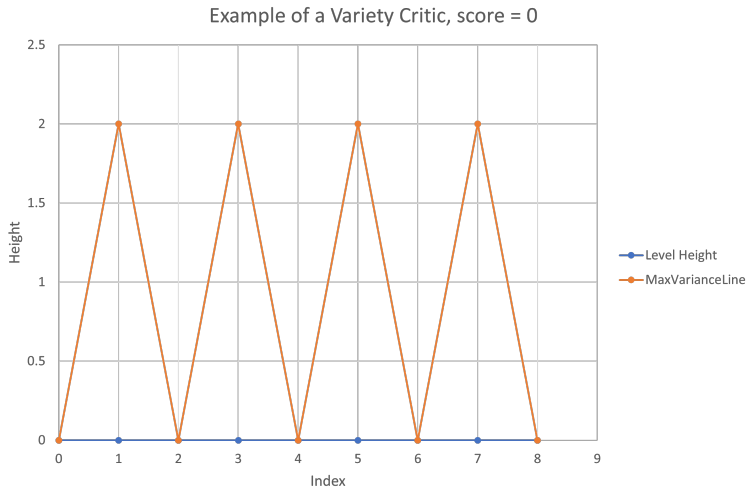


Figure 4.7: A small example of what score the variety critic would give.

$$VC = Var(HeightLine)/Var(MaxVarianceLine)$$

4.4.4 Combining Critics

All the aforementioned critics scores are added up and serve as the overall fitness score of a level. It was determined to not use a weighted sum since there was no justification for weighing one critic over the other. Feedback on the critics and their effectiveness are determined via the user survey as described in Section 5.2.1. Based on the feedback certain critics could be weighed more or less to generate even more enjoyable levels.

Fitness Function

The fitness function is simply a linear sum of the critics. No weights were given to the critics since determining these is not possible without some user feedback on what types of level are more desirable. Only then can the weights be fine-tuned a bit.

$$Fitness = LC + CFC + EC + JC + VC$$

From the three classes of fitness function for PCG purposes as proposed by Togelius et al. [21], this fitness function is of the *direct fitness function* type. This is characterised by extracting features from the generated levels and using these to map directly to the fitness function. As opposed to a simulation-based or player interaction based function.

4.5 Global Pass

A global pass can be used to enforce some global constraints on a level, as well as decorate the levels with collectable items if so desired. In The Impossible Game, no collectable items are available and thus the global pass will only be utilised to enforce the playability of a level. This means that the whole level is checked such that it conforms to the constraints of the level generator. More specifically this consists of making sure the level pieces are linked together and there are no height differences which are impossible to jump for the player. The Geometry Generator always generates feasible levels. However, since the aim is to evolve the candidate levels given by the Geometry Generator, which requires reproduction operators such as mutation and crossover, a global pass can be used to ensure that mutated and reproduced individual levels still maintain the constraints as required by the game.

4.5.1 Smoothing Algorithm

The global pass comes in the form of a smoothing algorithm. The algorithm checks the level piece by piece, comparing the heights of the pieces to ensure it is traversable by the player. When a faulty connection is discovered, the algorithm simply replaces the pieces after the faulty connection with new pieces that are linked up with the previous ones. To illustrate this idea more clearly, observe

Figure 4.8 which shows the effect this algorithm has on a sample level. The pseudo-code for this algorithms is as follows.

Algorithm 2 Smoothing Algorithm

```
Input: Level
for Every level piece do
    Calculate height difference between current and previous level piece.
    if Absolute value of height difference > 0 then
        if Height difference is negative then
            Set current piece as new piece of height previous piece - 1
        else
            Set current piece as new piece of height previous piece + 1
        end if
    end if
end for
```

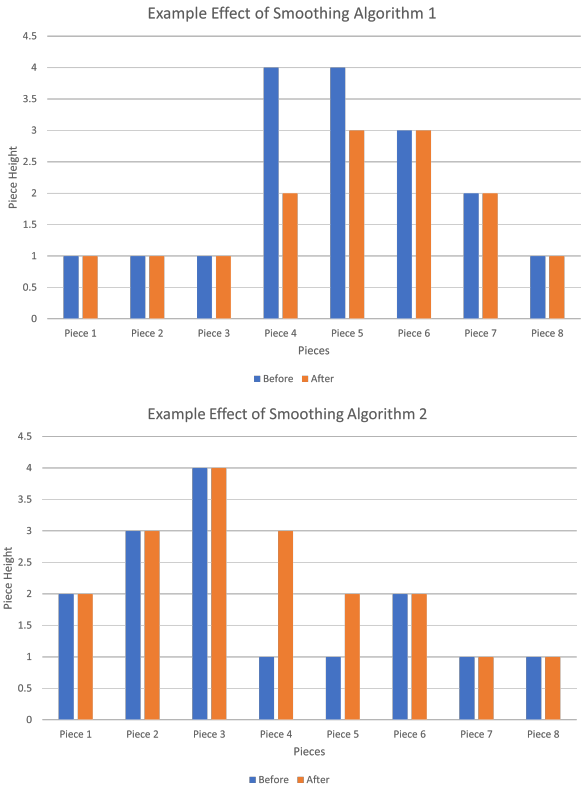


Figure 4.8: Two examples of the effects of the smoothing algorithms.

This smoothing algorithm was used in combination with the genetic algorithm, which will be described in the next section.

4.6 Genetic Algorithm

As mentioned before, the geometry generator by itself can generate a wide variety of levels. Many of which are undesirable with regards to the music track. Therefore an evolutionary method is applied to optimise towards certain criteria. More specifically, a Genetic Algorithm (GA) is used for this purpose since there is a large amount of variables that determine what makes a level desirable. Using simpler methods based on local search techniques such as Hill Climbing or Tabu Search are insufficient to solve towards a global optimum in this vast design space. This section will present the specifics of the used GA. In particular, Section 4.6.1 will lay out the details on the genetic representation for the GA. Subsequently, Section 4.6.2 briefly describes the manner of reproduction and mutation used to determine the next generation of candidate levels. Together, these sections form the different parts of the GA.

4.6.1 Genetic Representation

In order to use a GA to evolve levels, a genetic representation was formulated to make it possible to reproduce and mutate them. The choice was made to use level pieces as the genes, which would together form the chromosomes of the individuals or levels. Since levels based on the same song have the same amount of genes, performing crossover between these becomes easy. Each gene stores a level piece with the complete information about the height and position of the piece. An example of this representation is shown in Figure 4.9.

Figure 4.9: An example of the genetic representation of a level.

The chosen genetic representation has a direct encoding between the genotype and phenotype of each level, as per the definition in [21]. This means that the size of the phenotype is linearly proportional to the genotype, since each gene in the genotype can be represented exactly in the level(phenotype). This is in contrast with an indirect encoding where there is no linear mapping between genotype and phenotype [26]. The direct encoding was mainly used because the aim was to have the generated level match closely to the line of notes extracted from the song. The calculation of the distance between the desired notes line and the level height line is dependent on the position and height of each level piece. Indirectly encoding this height and type of the piece would lead to at least a loss of positional information. Another important consideration

for the representation problem is the dimensionality. The problem respresentation should have the right dimensionality to avoid the "curse of dimensionality" [27]. Essentially, this means that the dimensionality of the genotype should be limited to avoid an exponential explosion of the search space. For this application, this was achieved by limiting the song length to approximately 30 seconds. The limiting of the song length was initially done to decrease the difficulty and time spent for the participants of the user survey, but also has the side effect of limiting the dimensionality.

4.6.2 Reproduction & Mutation

The reproduction process in this GA is mainly concerned with 3 different steps: selection, reproduction and mutation. Firstly, for the selection process an elitism strategy is applied that directly transfers the highest scoring individuals to the next generation without needing to mutate or reproduce. This is done in order to avoid the high scoring level from mutating and reproducing which could possibly cause the level to degrade in quality.

The rest of the individuals are selected as parents to produce children through crossover. The crossover strategy chosen was the one point crossover. Multipoint crossover was avoided because there would possibly be more than one disconnected spot between the crossed over individuals. The one point crossover reduces this to maximum one disconnection, which is easier for the smoothing algorithm to handle correctly. This crossover method is visualised in Figure 4.10. Afterwards there is still a possibility for the new child to mutate.

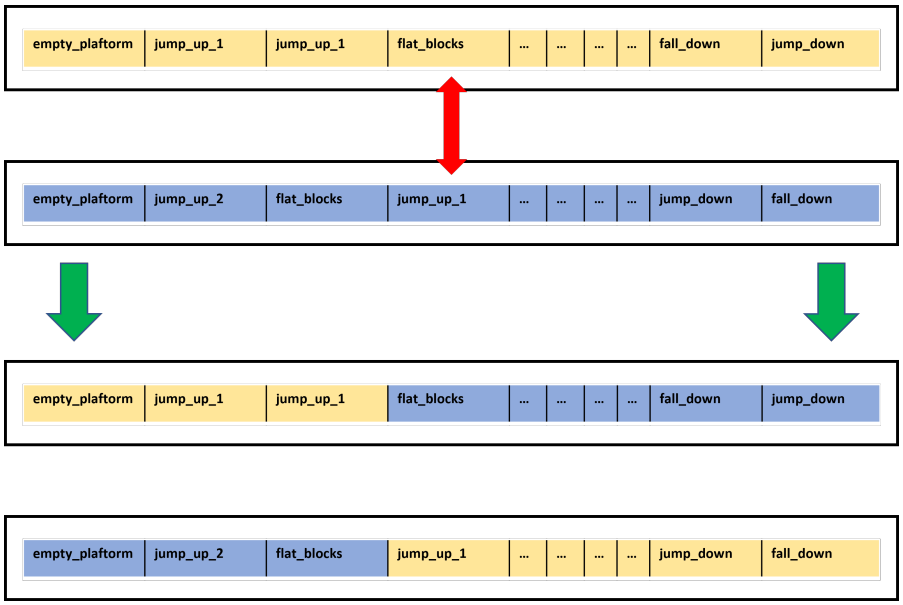


Figure 4.10: An example reproduction in the GA. The red arrow depicts the crossover point, the bottom two individuals are the results of this crossover

The mutation works by randomly selecting a gene after reproduction has happened. This gene is replaced by a randomly chosen level piece. Resulting in a new possibly non-feasible level. Mutation introduces a random element to the new generation that may not have existed before. This can result in new types of levels being evolved with level segments that otherwise would not have been evolved. Figure 4.11 shows this process.

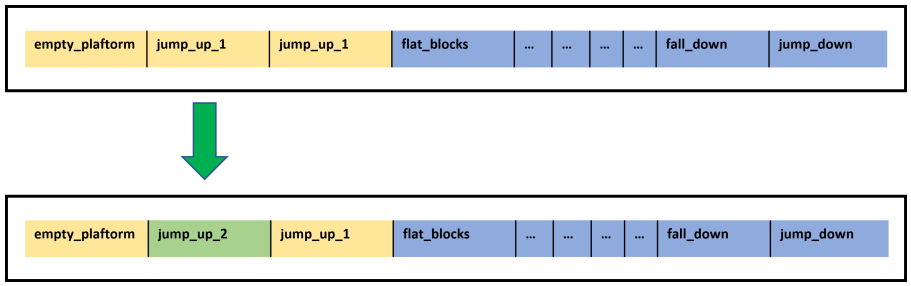


Figure 4.11: An example of mutation in the GA. The green arrow depicts the gene to be mutated, the bottom individual is the results of this mutation with the new green gene included

Chapter 5

Experiments & Results

This chapter covers the experimentation that was done with the level generator. It starts off by explaining the chosen parameters related to the Genetic Algorithm and how they were obtained. Furthermore, this chapter covers the user survey that was performed to gain some feedback about the generated levels. It explains the reasoning behind the survey as well as the received results.

5.1 Level Evolution

5.1.1 Parameters

The parameters used for the evolutionary process were determined by trial and error testing whilst considering the hardware limitations of the laptop being used. The hardware limitations mainly concerned the population size of each generation. It was therefore determined that a population size of 250 levels was sufficient to provide ample variation between levels yet limit the processing time and use of memory. Similarly, with the interest of processing time in mind, the number of generations used to evolve the final level was limited to 50. Moreover, evolving levels beyond 50 generations generally yielded no higher fitness value. As can be seen from Figure 5.1, the fitness value plateaus before the 20th generation. Thus 50 appears to be a safe choice. More generations have been tried, but did not result in a remarkable difference in fitness value. In the selection a form a elitism was used where the 20 individuals with the highest fitness values get selected directly to enter the next generation without mutation or crossover happening. This ensures that the highest valued individual is never lost but genetic diversity is still maintained in the population.

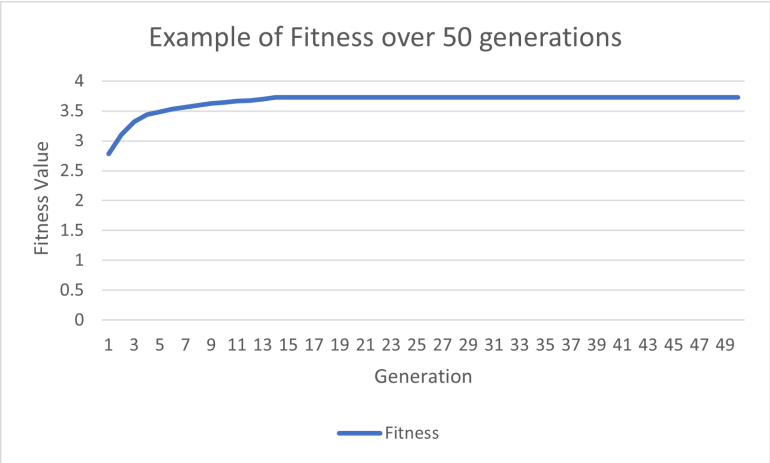


Figure 5.1: Shows the development of the fitness over 50 generations.

The chance of mutation was determined by setting it to a range of values and determining the optimal value overall. Two approaches were tested, the first approach was a simple genetic algorithm with a constant mutation rate. Three different values were tested for this approach, 0.5, 0.3 and 0.1. A second approach spans the range of the mutation rates tested for the first approach. It starts at 0.5 and gradually decreases over the generations towards 0.1. The large mutation rate is initially used to overcome local maxima and ensures sufficient genetic variability in the first few generations [28]. The results of the tests on mutation rate can be viewed in Figures 5.2, 5.3, 5.4 and 5.5.

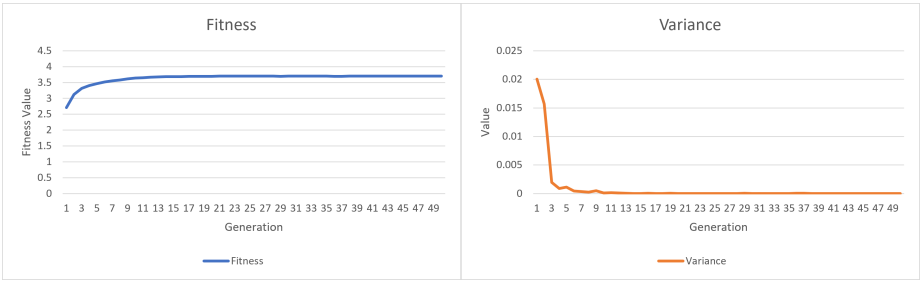


Figure 5.2: Shows the development of the fitness and variance values over the generations. Here a constant mutation rate of 0.1 is used.

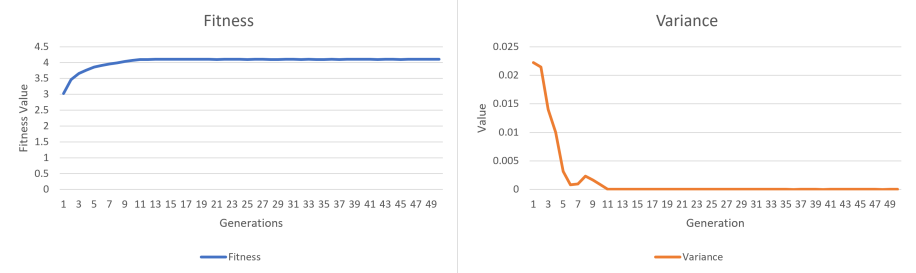


Figure 5.3: Shows the development of the fitness and variance values over the generations. Here a constant mutation rate of 0.3 is used.

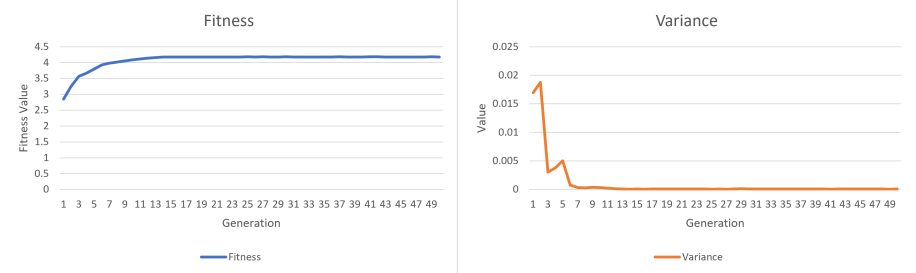


Figure 5.4: Shows the development of the fitness and variance values over the generations. Here a constant mutation rate of 0.5 is used.

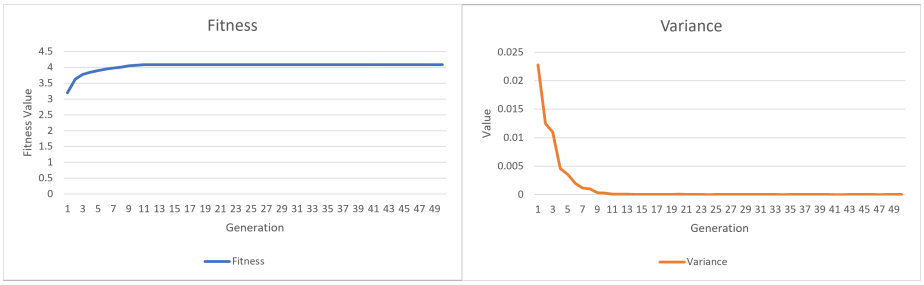


Figure 5.5: Shows the development of the fitness and variance values over the generations. Here a adaptive mutation rate ranging from 0.5 to 0.1 is used.

From the figures above it is clear that the different mutation rates can reach similar average fitness values, however the variance curves do display some difference. The lower the mutation rate the faster the genetic diversity diminishes.

Ideally, the variance displays a more gradual curve than Figure 5.2. Comparing all the options it is clear that the adaptive mutation rate performs better in this regards than the other constant mutation rates. Therefore the adaptive mutation rate was used in the rest of the experiments.

5.1.2 Training

The evolution throughout the generations was performed on a personal laptop computer. This laptop has a Intel(R) Core(TM) i5-7Y54 CPU @ 1.20GHz processor with access to 8GB or RAM.

Each level was generated from a different song. Since 5 genres were considered in this thesis, each genre gets 2 levels in the final generated set. Each music file used as a basis for the level generator was shortened to approximately 30 second clips. This was done in order to greatly reduce processing time of the level generator and reduce the time needed for users to learn and complete the levels. The songs used to generate the levels are the following:

- *Level 0 [Rap]* : Luv (Sic) Part 3 - Nujabes Feat. Shing02
- *Level 1 [Classical]* : Voices of Spring Op. 410 - Johann Strauss
- *Level 2 [Rock]*: Johnny B. Goode - Chuck Berry
- *Level 3 [Jazz]*: Bird's Lament - Moondog
- *Level 4 [Electronic]*: Honda Civic - M1C4H
- *Level 5 [Rap]*: Feather - Nujabes Feat. Cise Starr & Akin
- *Level 6 [Electronic]*: Slow - Yu-Utsu
- *Level 7 [Rock]*: My Generation - The Who
- *Level 8 [Jazz]*: Breeze - Jiro Inagaki Soul Media
- *Level 9 [Classical]*: Nocturne No.20 in C Sharp - Chopin

The selection of these songs was party done by using some criteria, and partly done by taking a random sample of songs from a genre and picking one according to those criteria. Because there are millions of songs on Spotify a random sample of 10 songs of each genre was taken. For each song of every genre, the tempo in BPM was extracted. A selection was done eliminating songs that had a BPM of 200 or higher. In practice this rules out the extremely fast songs that would result in a very fast moving player, which should help for players who have never played such a game. Furthermore, songs were listened to and judged based on the amount of vocals exhibited in each. Songs without vocals were generally preferred as this should lead to more successful feature extraction from the music. However some songs were still allowed with vocals as long as there was enough instrumentals in between. The songs with the most vocals are the rap songs, since vocals are an integral part of this music genre.

5.2 Validating Levels

5.2.1 User Survey

Validating whether the fitness function truly yields a desirable level is not a straightforward task. There is no direct numeric validation for this as the feeling of synchronicity is subjective depending on the current user. Therefore a user survey was performed to obtain some feedback. The aim of this survey is to gain an understanding of the relation between the proposed critics and the notions of fun, difficulty and song similarity. By considering these notions and the related critics, aspects of the user experience can be quantified as well as critics adapted to improve the user experience. According to the parameters and method explained in the previous section, 10 levels were generated.

The levels were presented in random order to the users in the survey. The survey contains the same list of questions for every level in the set. The differences in answers between the levels serves as a metric for comparing the aspects of each level, as well as the individual critics.

The questions contained in the user survey range from binary questions to questions which were answered by a rating from 1 to 5. The total set consists of 10 questions, a full list of the questions can be viewed in Figure 5.6. Questions such as "Did you try out the level?" serve as a filter whether or not to include the users rating in the set of questions to be analysed. For example, if a user does not try out a level, the answers to the remaining questions are not included in the analyses since the user does not have adequate information to judge the level. The question "Did you complete this level?" serves as a metric for completion percentage of each level. This can give an impression of how difficult each level is. The rest of the survey consists of qualitative questions where the user has to give a score between 1 and 5 regarding the different level characteristics. These type of questions will be used in the experiments and will be referred to as question 1 through 6 in order of appearance. All the aforementioned questions are mandatory, not giving an answer to one is not allowed. to avoid any confusion, the numbered list of questions can be found below:

1. How fun would you rate this level?
2. Was there enough variation in the level to keep you engaged?
3. Do you think that the level was closely synchronised with the rhythm of the song?
4. Do you think that the level was closely synchronised with the pitch of the notes?
5. Do you think that the number of spikes used in the level match the mood of the song?
6. Do you think the difficulty of the level matched the genre of the song?

After this set of required questions, the user is able to leave a longer form answer about any other subjects concerning the game. The questions were posed in such a way that they could relate to the notions of fun, song similarity and difficulty. The survey was shared with students of the DKE faculty at Maastricht University, some personal friends and other willing participants recruited via social media. The survey was anonymous so no personal information about the participants has been gathered.

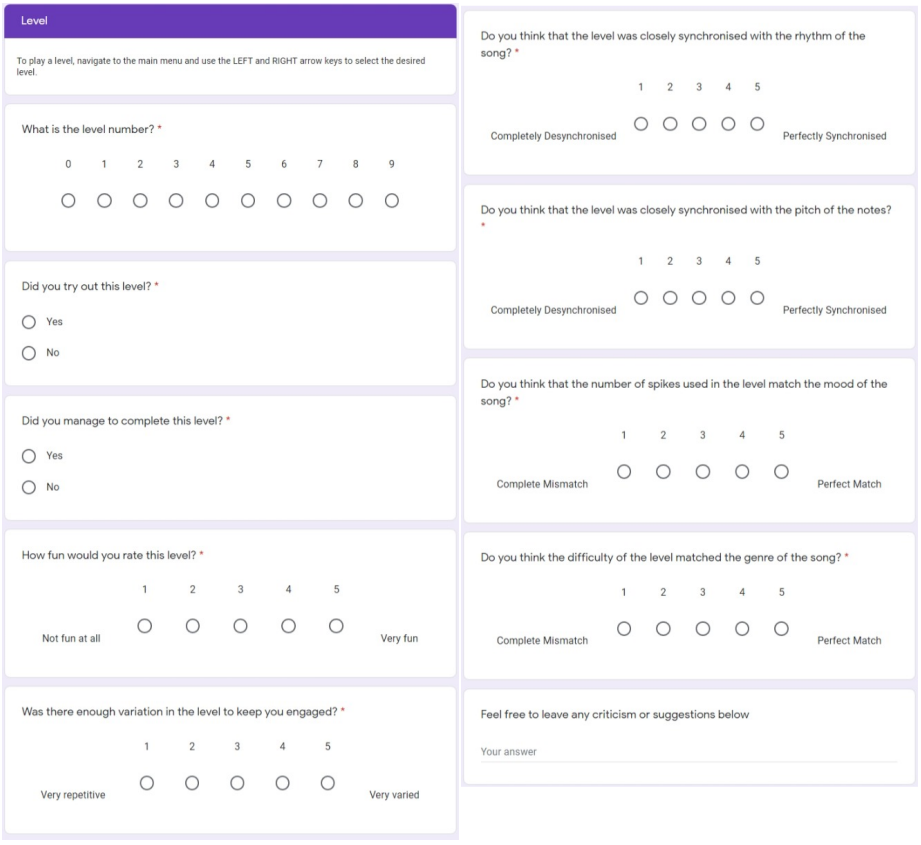


Figure 5.6: Shows an example of the questions a user had to answer for each generated level

5.2.2 Results

In total 19 people responded to the survey, of which 15 respondents tried out every level. After receiving the results the data was treated in order to provide deeper insight into the answers. Firstly, the completion rate of each level was calculated in order to find out the relative difficulty between levels. Secondly, an overview containing the average scores for each level per question was calculated. Moreover, a correlation matrix relating all questions to each other was established. This matrix contains the Pearson correlation coefficients between two respective questions. Similarly, a correlation matrix between all the critics scores was also calculated as well as an overview of the critic scores for each level. Finally, a correlation matrix between each question and each critic was made, giving insight into the connection between the posed questions and the proposed critics. In the next results several abbreviations will be used to denote critics and categories of critics. The abbreviations for critics will be the same as the ones used in Section 4.4. The abbreviations for the categories of critics are provided in a short list below.

- *SIM* : Song similarity (Line Critic)
- *DIF* : Difficulty (Component Frequency Critic, Emptiness Critic)
- *FUN* : Fun (Jump Critic, Variety Critic)

Completion Rate

The completion rate for each level was calculated from the user survey. This was done in order to obtain a relative difficulty measure of the levels, while also potentially seeing differences between the level difficulty for each genre. Essentially it aims to test the hypothesis of "Level X is more difficult than level Y" for all level pairs, and "Genre X is more difficult than genre Y" for all genre pairs. This also shows the relationship between this experiment and the second research question, which inquires about which musical features can be used to influence the level difficulty among other aspects.

Table 5.1 shows the rate of completion for every level included in the user survey. From the table it can be seen that there is quite some difference between the completion rate of some levels. The highest completion rate is found in level 1 where half of the participants managed to complete it, followed by level 0 and 9 with 33.33% and 31.25% completion respectively. The lowest completion rate is found in level 3 with level 8 not far behind. Interestingly enough, the two levels with the highest completion rate are of the same genre: Classical. Conversely, two of the three levels with the lowest completion rate are also of the same genre: Jazz. Overall, no completion rate exceeds 50% showing the relative difficulty of the generated levels as experienced by the participants.

Level	0	1	2	3	4	5	6	7	8	9
Completed	33.33%	50%	22.22%	5.55%	18.75%	12.5%	12.5%	12.5%	6.25%	31.25%

Table 5.1: Table showing the completion rate(%) for each level in the user survey.

User Scores

The users were asked 6 quantitative questions pertaining to the different aspects of the levels. These questions are related to various level aspects including fun, song similarity and difficulty, which is the main focus of the second and third research questions. These questions aim to give some feedback about which aspects of the generated levels are relatively enjoyable and which are not, enabling the tweaking of the used level critics to obtain more desirable levels. The average scores given by all the users per question are compiled in Table 5.2. The bottom two rows contain the averages and variance given for each question. Additionally, the average and variance for each level is presented as the last two columns in this Table.

Question	1	2	3	4	5	6	μ	σ^2
Level								
0	3.222	3.722	4.167	3.944	3.833	4.111	3.833	1.449
1	3.722	4	4.167	4.056	4	4.111	4.009	0.999
2	3.333	3.833	4	3.889	3.778	3.889	3.787	1.403
3	3.111	3.778	3.722	3.889	3.778	4	3.713	1.44
4	3.5	3.722	4.333	4.111	4.111	4.056	3.972	1.336
5	3.056	3.611	3.944	3.944	3.889	3.944	3.732	1.413
6	3.889	3.889	3.944	4	3.833	3.944	3.833	1.523
7	3.556	4.056	4.278	4.167	4.111	4.111	4.046	1.166
8	3.5	3.944	4	3.944	3.944	3.833	3.861	1.466
9	3.611	3.944	3.889	4.111	4.056	4	3.935	1.538
μ	3.4	3.85	4.044	4.006	3.933	4		
σ^2	1.012	1.212	1.417	1.379	1.515	1.453		

Table 5.2: Averages obtained for each of the quantitative questions per level, also includes the overall average ratings(μ) and variance(σ^2).

Firstly, the highest average level score given to any level goes to level 7 with a score of 4.046. Closely followed by level 1 with a score of 4.009. Both these levels are the only levels to achieve a score above 4 on average. Conversely the lowest scoring levels are levels 3 and 5, receiving scores of 3.713 and 3.731 respectively.

In addition to Table 5.2, a correlation matrix was calculated. This matrix contains the correlation coefficients between each pair of questions, potentially showing a correlation between certain pairs. The resulting correlation matrix can be viewed in Figure 5.3.

Question	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*
Q1	1					
Q2	0.800924	1				
Q3	0.596127	0.751127	1			
Q4	0.631515	0.786875	0.878777	1		
Q5	0.544933	0.68106	0.871349	0.850288	1	
Q6	0.55286	0.686311	0.825537	0.801061	0.858625	1

Table 5.3: Pearson correlation coefficients between each combination of questions. The colours indicate a range of values with red representing a relatively weak correlation and green representing a relatively strong positive correlation.

Overall, this correlation matrix contains only positive correlations, the majority of which are very strong. The weakest correlations exist between question 1 and every other question, with the exception of question 2. On the other hand, the strongest correlations can be found with question 3. Both question 4 and 5 are strongly correlated to question 3.

Critics

To show the influence of the critics on the generated levels, an overview containing the critic scores for each level was calculated. Differences between critic values can show a bias of the level generator towards a certain critic. Understanding these differences can help in refining the fitness function to obtain more desirable levels. This experiment relates to the first research question and helps to determine which musical features are suitable for this type of level generator. Results of these critic scores can be observed in Table 5.4.

Type	SIM	DIF	DIF	FUN	FUN	
Critic	LC	CFC	EC	JC	VC	TOTAL
Level						
0	0.878	1.0	0.978	0.326	0.264	3.446
1	0.801	0.96	0.967	0.367	0.311	3.406
2	0.856	0.999	0.976	0.5	0.213	3.544
3	0.789	0.949	0.712	0.407	0.499	3.356
4	0.798	1.0	0.978	0.4	0.320	3.496
5	0.736	0.999	0.939	0.489	0.479	3.642
6	0.769	0.947	0.982	0.427	0.422	3.547
7	0.280	0.997	0.98	0.44	0.308	3.005
8	0.549	0.994	0.757	0.473	0.156	2.929
9	0.403	0.932	0.95	0.467	0.672	3.424
μ	0.686	0.977	0.922	0.429	0.364	
σ^2	0.038	0.001	0.009	0.003	0.021	

Table 5.4: Individual critic scores for each level used in the user survey. The average(μ) and variance(σ^2) of the critic scores are shown in the bottom two rows.

The total scores of each level in Table 5.4 indicate that level 5 received the highest score of all the levels with 3.642. This level received relatively high scores for each critic, having the highest score for the jump critic (JC) and third highest score for the component frequency critic (CFC) and variety critic (VC). On the lower end, level 8 received the lowest score of 2.929, being the only level to be rated under 3. Notably, the component frequency critic receives an almost perfect score in every level whereas the line critic has the most difference between its highest and lowest rated levels.

Critics	*LC*	*CFC*	*EC*	*JC*	*VC*
LC	1				
CFC	0.135333	1			
EC	0.036408	0.184537	1		
JC	-0.37872	0.060634	-0.12698	1	
VC	-0.23791	-0.71964	-0.0131	0.128041	1

Table 5.5: Pearson correlation coefficients between each combination of critics. The colours indicate a range of values with red representing a negative correlation and green representing a positive correlation.

To get a better understanding of the relationship between different critics, a correlation matrix between all pairs of critics was created and is shown in Table 5.5. This was done in order to understand the connection between critic pairs. The aim is to test whether any of the critics are correlated or if they are independent. The overall trend in this Table is the fairly weak correlations between critic pairs. An outlier can be seen in the correlation between the variety critic and component frequency critic, where there exists a very strong negative correlation, meaning when the value of one increases the other decreases. There are some other slight negative correlations between the line critic and the variety critic and jump critic. However due to the limited magnitude these correlations are of weak to moderate strength.

Correlation between Critics and User Scores

The final correlation to inspect is the correlation between the critics scores and the average score per question. This can provide insight into the relation of a critic to the experience of the user. There are several hypothesis being tested here, one for each critic and user scores pair. For example, the hypothesis between the line critic and question 4 would be: "The line critic forces the level to be synchronous to the song with regards to the pitch.". The correlation between these critic and user score pairs are used to relate to the second research question. Which is concerned with determining which musical features can influence the generated level to yield an adequate level of difficulty, engagement and novelty. Connecting the user scores and the critic scores is vital to understand how well the critics reflect the users opinions. The resulting 6x5 matrix is shown in Table 5.6. Overall, two of the critics stand out. Firstly, the emptiness critic is positively correlated with every question asked. The positive correlation is most pronounced between the emptiness critic and question 3. Moreover, for each question apart from question 2, there is a moderate to strong correlation.

On the contrary, the strongest negative correlation overall is held by the line critic. Apart from question 3 and 6, the line critic has a moderate to strong negative correlation with every question. However, the strongest negative correlation between any critic and question pair is achieved by the jump critic and question 6. Furthermore, the jump critic also has a moderate negative correlation with question 3.

The variety critic, on the other hand, does not have any strong correlations apart from one strong negative correlation between it and question 3.

Finally there is also the component frequency critic. Apart from the strong positive correlation between this critic and question 5, there are only weak positive or negative correlations. Implying that this critic has no influence on the level aspects asked in questions 1,2,4,5 and 6.

			Critics		
Question	*LC*	*CFC*	*EC*	*JC*	*VC*
Q1	-0.45192	-0.23396	0.299279	-0.08621	-0.1232
Q2	-0.63111	-0.33814	0.031081	0.050406	-0.12296
Q3	-0.08311	0.585863	0.602887	-0.37795	-0.52508
Q4	-0.63647	-0.11742	0.488966	-0.12312	0.21395
Q5	-0.63796	0.057904	0.312235	-0.03042	0.086265
Q6	-0.04566	-0.02769	0.42951	-0.75663	0.118985

Table 5.6: Pearson correlation coefficients between each combination of critic and question. The colours indicate a range of values with red representing a negative correlation and green representing a positive correlation.

Chapter 6

Discussion

This chapter provides a discussion about the results obtained in the previous chapter. It will cover the results of the completion rate, user scores, critics and the correlation between the latter two. The data will be used to formulate some interpretations and to what extent the critics represent the user's desires. Finally, some limitations of this research which may impact the results are brought to light.

Completion Rate

In Table 5.1 the completion rates of every level were compiled. The highest completion rate belongs to level 1, whereas the lowest completion rate belongs to level 3. It was observed that the two levels with the highest completion rate were from the same genre: Classical. Similarly, two of the three lowest rated levels also belonged to the same genre: Jazz. This alludes to the fact that there is a tangible difference in difficulties between the genres. At first glance this is not particularly surprising, since the component frequency critic influences the amount of spikes based on the genre of the song, generally resulting in levels with more spikes being more difficult to navigate successfully. However considering that the component frequency critic sets the desired spike ratio of Jazz and Classical to similar rates of 0.15 and 0.05, this is a remarkable result.

The expectation was that, the higher the spike ratio, the more difficult a level would become. Since both Jazz and Classical are on the lowest end of the desired spike ratios, it would be expected that they would end up at similar difficulty levels on the lower end of the scale. However, the exact opposite has happened. The Classical music yielded the least difficult levels, whereas Jazz resulted in a much higher difficulty of the related levels, overtaking the other genres which have a higher desired spike ratio. This could be in part caused by the tempo of the songs. Since Jazz tends to have a much faster rhythm compared to Classical music, the speed the player will follow in the generated levels will also be higher. The higher speed is a probable cause for the difference in experienced difficulty by the users.

User Score

In Table 5.2 the average scores for each question per level were compiled alongside Table 5.3 which compared the correlation between pairs of questions. One of the findings in Table 5.3 is that question 1 contains some of the least strong correlations to the other questions. Overall, the questions are all positively correlated. The difference can be found in the degree of that positive correlation. Relative to all the other questions, the only very strong correlation with question 1 is question 2. Since question 1 asks the user feedback on the fun of the level and question 2 is concerned with the variation in the level, it can be inferred that fun and variation are correlated to each other. This is an expected outcome as a level without much variation would not be enjoyable to play to most humans. This implies that the notion of fun as presented previously in this thesis consists partly of a variation component.

Furthermore, questions 3,4,5 and 6 all have strong correlation between each other. Recall that these questions were formulated around the matching of a level with some particular aspect of the song. Questions 3 and 4 in particular ask about matching the rhythm and pitch of the song, these are less abstract concepts than matching the mood or genre in question 5 and 6 respectively.

The relationship between all these questions can be viewed as a measure of song similarity. Matching the rhythm and pitch of the notes gives the user a direct feeling of song similarity as the user experiences jumping on the rhythm and following the pitch of the song. Together, the rhythm and notes create a song with a genre and mood. Since genre and mood are a result of the combination of notes and rhythm, it follows that matching a songs mood or genre would be correlated to matching the rhythm or pitch of a song.

As these 4 questions are all strongly correlated to each other, it follows that as a group they determine a big part of the overall user score. This results in these 4 questions all being strongly correlated to the overall score. This correlation is less pronounced in question 1 and 2, which as shown before are correlated to each other and form a different notion of fun.

Critics

Table 5.4 showed the individual critic scores per level and their total scores. As well as the average and variance for each critic. Combined with Table 5.5 these provide insight into the relationship between critics. In Table 5.4 it can be observed that the line critic has high variance between the levels relative to the other critics. A reason for this can be that the notes line extracted from the songs are difficult to satisfy. For example, the notes line can contain two subsequent notes that have a large difference in pitch, resulting in a notes line that has height differences larger than the geometry generator can handle. So when dealing with many note lines it is possible that some are generally flatter with less extreme differences in height than others. This results in the flatter lines being easier to match and thus higher line critic scores being achieved. Furthermore the table shows that both the component frequency critic and the

emptiness critic have little variation and always scores relatively high. The emptiness critic likely scores relatively high because of the interchangeability between an empty piece and a platform with spikes on it. Changing a empty piece to the same spike does not change the height line or the rhythm of the level, it simply increases the number of filled pieces in the level, thus enabling the generator to add extra pieces to increase the emptiness critic value while not causing any other critic to drastically change. For the component frequency critic on the other hand, this is likely due to the fact that the squared difference between the actual spike ratio and desired spike ratio is used. Resulting in the level generator being heavily discouraged to stray away from the desired spike ratio. This causes the component frequency critic to have a greater pull towards having an ideal score of 1, relative to the other critics.

More insight can be gathered from Table 5.5. As mentioned before, there are mostly weak correlations between the critics. The exception to the rule is the correlation between the variety critic and the component frequency critic. They share a strong negative correlation of -0.71964. The negative correlation means that if the variety in level height increases the ratio of spikes decreases, and vice versa. Once again, since the spike pieces are mostly flat pieces without height difference between the beginning and end, they counteract the height variation in the level.

Correlation between Critics and User Scores

Table 5.6 shows the correlations between each critic and each question. Overall the emptiness critic has the most positive correlations with the questions from the user survey. However the strongest of these correlations are to questions 3 and 4. Since questions 3 and 4 pertain to the song similarity of the level with the rhythm and notes of the song, it seems like the less empty a level the more likely it is to be similar to the song. This does make sense considering that, with more empty pieces and a lower emptiness critic score, it will be harder to match pitch and rhythm. This is likely due to the fact that empty pieces will result in the user not jumping and being less synchronised to the rhythm. Similarly, for the pitch of a song, using empty pieces makes the level have less height variation and is thus less likely to be able to match notes that are higher on the tone ladder. Overall the emptiness critic is positively correlated to the questions, showing the importance of having non empty levels.

Furthermore, the line critic is generally negatively correlated to each question. The strongest negative correlations are found with questions 2,4 and 5. The negative correlation with question 2 indicates that closely following the note line of a song comes at the cost of more variation in the level. Similarly, questions 4 and 5 are related to pitch similarity of the song, and use of spikes in the song respectively. This is a surprising result considering question 4 basically asks the user to rate the similarity between the level and the pitch of the notes. It is possible that some critics counteracted the effect of the line critic since the notes line can not be sufficiently matched to convey that feeling to the user.

Finally, the jump critic is mostly weakly correlated to the questions, with the

outlier being question 6. Between the jump critic and question 6 there exists a strong negative correlation indicating that the more often the player needs to jump the less difficult a level becomes. This is likely due to two aspects of the critics. Firstly, the emptiness critic forces the level to contain pieces. This likely results the user needing to jump over those pieces. However, once the frequency of jumping is increased to a very high level, the variation in actions the user needs to take decreases. For example, consider a level that has 10 rhythmic beats in it, and each beat is matched by a jump. The feeling for the user would be very monotonous and every jump would be at the same interval, therefore decreasing the difficulty and variation in the level. To counteract this a desired jump ratio could be introduced similar to the desired spike ratio in the component frequency critic.

Limitations

Additionally, there are some limitations to consider about this thesis. It was argued that fun, song similarity and difficulty were good measures of a levels quality. However, it is possible that there are other facets that can contribute but were not considered in this thesis. This could result in the user survey not reflecting the actual quality of a level.

Furthermore, it is possible that the questions posed to the users do not perfectly reflect the notions of fun, song similarity or difficulty. Each of these notions have no precise definition by which the levels can be rated, therefore the subjectivity of the user comes to the forefront. This subjectivity could be exaggerated by the low response rate to the user survey, as single individuals have more influence over the overall results compared to a having a much larger group of respondents than 19.

The levels provided for the users to try were overall very difficult, it was found that users had a very low completion rate for the levels with very few exceptions. The lack of completion is a problem because the users ratings are likely affected by the lack of knowledge of the whole level. This can lead to false conclusion being drawn about the song similarity or fun of the level as well. When a level is too difficult, usually it is not very fun to play. Additionally, another difficulty metric could be introduced. Counting the number of attempts a user did on a level can give information about levels with very low completion rate. For example, if there are 2 levels with identical completion rates but the average attempts on one level is higher than the other. It follows that the level with more attempts is likely the more difficult one.

Moreover, the fitness function from the genetic algorithm as well as the critics that make it up are both experimental in nature. The critics aim to reflect certain desired qualities of a level such as it being non-empty or having variety across the level. However, it is probable that there are other critics that could be used to deliver an even more desirable result. The fitness function is a sum of all the critics together. However, it could be beneficial to give each critic a certain weight that would reflect its importance in a level. A good candidate for a smaller weighting is the component frequency critic, which always returned

a high value with very little variation.

Finally, it may have been difficult for some users to differentiate between certain questions on the user survey. As it is not guaranteed that the participants have any musical or video game knowledge at all. Therefore it may be useful to limit the survey group around a certain type of person whom has knowledge of games and music, to more accurately reflect the true nature of the levels.

Chapter 7

Conclusion

This chapter is concerned with reviewing and discussing the problem statement and research questions as stated in Chapter 1. To reiterate, the problems statement is as follows: "How can musical features be mapped to a level generator for a 2D Auto-runner game to produce a feasible, novel and fun level?". The following research questions were formulated to address the problem statement:

- What set of musical features can feasibly be mapped to facets of a level?
- What aspects of a 2D Auto-runner level can be influenced by these musical features to provide an adequate level of difficulty, engagement and novelty?
- How well do users feel the levels match the music used to generate them?
- Can the resulting level generator be abstracted to a more general model for music based 2D Platformers?

What set of musical features can feasibly be mapped to facets of a level? Chapter 3 provides answers to the first research question. The chosen musical features were rhythm, notes and genre. Rhythm and notes were chosen due to their direct connection with a level. Whereas genre was chosen to influence the amount of spikes the player could encounter in a level.

What aspects of a 2D Auto-runner level can be influenced by these musical features to provide an adequate level of difficulty, engagement and novelty? The musical features chosen were connected to several facets of a level. The rhythm was used to time the jumps of the player, integrating the auditory experience into the level. This was done by placing the level pieces at a position corresponding to the rhythm of the song. The notes were used to construct a height line, representing the desired trajectory of the level. Finally the genre was integrated to influence the ratio of spikes in a level. Depending on the genre of the music more or less spikes would be placed. However, statistically verifying the adequacy of these feature integrations has proven difficult.

Overall, it has not be proven that these features yield an adequate level of difficulty, engagement and novelty. Possibly due to the small sample size of survey takers.

How well do users feel the levels match the music used to generate them? The answer to this research question was obtained by processing the user survey data as shown in Chapter 5. Question 3 and 4 from the user survey give the clearest answer, as this research question is related to the levels matching the song used to generate them. The level matching the rhythm of the song was rated at 4.044 out of 5 on average over all the levels. Furthermore, the level matching the pitch of the song was rated by the users as 4.006 out of 5. From this, it follows that the users were relatively satisfied with the levels matching the music. Especially compared to the other aspects of the level that were questioned in the user survey.

Can the resulting level generator be abstracted to a more general model for music based 2D Platformers? It is possible to abstract this system to a more general form. Chapter 1 shows the system as a whole. Using this template, should give enough guidance to create a similar system for another game. However for each game, game specific obstacles and physics need to be added, as well as game specific dynamics such as double jump, dash or collectables like coins. For this the current system would have to be extended to include such facets.

7.1 Further Research

This section contains suggestions for possible future work that could be attempted based on this research. These suggestions are divided into a few categories: changes to the current research for improved results, integration of new musical features, adaptive difficulty for an improved user experience.

7.1.1 Changes to Current Research

Due to the fact that in many levels the statistical significance was not demonstrated, a larger sample size of test subjects should be found to participate in the user survey. Generally the more participants, the more power the resulting statistics carry. It is likely that, if there is a statistical significance between fun, song similarity and difficulty, with a larger sample size these differences would be found. This would simply require more time to perform the user survey until the desired sample size is obtained. For this thesis, only 19 participants were gathered. Whereas for the sake of statistical significance, a sample size closer to, or over 50 would be desirable.

Another possible change to the current form of the system could be the set of critics used to generate a level. The current set has been partially compiled by trial and error testing and common sense criteria. However it is likely that

there exists other more descriptive critics that yield an overall more desirable level. Furthermore, the combination of these critics into a fitness function could also be changed. Different weights could be applied to every critic in the set, yielding varying sets of generated levels. These sets could then be used in a large scale user survey to obtain some feedback on the different weightings of the critics.

7.1.2 Integration of Other Musical Features

To expand the effect music has on the generated game levels, more musical features could be mapped to aspects of the game. There is a wide array of musical features that could possibly be used, some examples from [7] include: melody, chords, loudness, mood or harmony. For each of these features a new way of mapping it to a level feature needs to be devised. Furthermore each mapping of a feature needs to be verified by performing another user survey.

7.1.3 Adaptive Difficulty

From the user survey it was inferred that the majority of the users thought the levels were too difficult for them to play. This is in part due to the nature of The Impossible Game, which is meant to be a very difficult game. However, this resulted in little variation in the difficulty scores obtained from the users. Furthermore, this resulted in most of the levels having a very low completion rate, which leads to incomplete knowledge of these levels being reflected in the user survey by the participants. In order to solve this problem a form of adaptive difficulty could be implemented. This could change the level based on the skill of the current user that is playing. Such a method is proposed in [29], where a Rank-based Interactive Evolution (RIE) strategy is introduced. This method creates computational models based on the users play style/preferences. These models are in turn used in the fitness function of the evolutionary algorithm in order to optimise towards user specific needs. These user preference model are built via ranking-based preference learning. This method has been shown to outperform other state-of-the-art evolution approaches. However, this will require the participation of many players of different skill levels in the development process of the level generator.

7.1.4 Generalising to Other 2D Platformers

The scope of this thesis focuses on level generation with the aid of music. However, 2D Platformer games that are not based on music far outnumber the music based games. Thus the question naturally arises whether this system can be generalised to fit other 2D Platformer games. Doing this comes down to replacing the musical features by an alternative which can generate levels of similar enjoyment.

In particular, the concepts of rhythm, pitch and genre to base the level generation on can be replaced to work for non music based games. For example,

the musical rhythm could be replaced by sampling jump times from a Poisson distribution, leading to generated level with a varied jump rhythm without directly matching any song.

Moreover the notes line could be replaced by a randomly generated guide line. This could be achieved by generating points along a level and creating a Bézier spline from them. Although many different methods for this height generation could be explored such as a Markov Chain based approach presented in [30].

Depending on the game, the component frequency critic could be replaced by any other desired difference between levels. One example of this could be coin frequency. As many 2D Platformers tend to have collectable coins spread around their levels, a critic encouraging higher coin amounts in more difficult level could be formulated. This could add in an additional incentive for the player to improve their skills as they will be able to collect more coins if they do.

The critics that are not related to music can be used in other games, these are the jump critic, emptiness critic and variety critic. These critics simply encourage the levels to contain more jumps, avoid empty levels and add variety to avoid flat monotonous levels. These are generally desirable attributes of 2D Platformers that can be adapted to yield interesting results across many different 2D Platformers. For example, one can set the emptiness rate of the level to yield the desired results depending on what the level calls for.

Bibliography

[1] J. Togelius, E. Kastbjerg, D. Schedl, and G. Yannakakis, "What is procedural content generation? mario on the borderline," in *2nd International Workshop on Procedural Content Generation in Games, PCGames 2011 - Co-located with the 6th International Conference on the Foundations of Digital Games*, ser. ACM International Conference Proceeding Series, 2011, ISBN: 9781450308724. DOI: 10.1145/2000919.2000922.

[2] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[3] M. Schedl, E. Gómez, and J. Urbano, "Music information retrieval: Recent developments and applications," *Foundations and Trends in Information Retrieval*, vol. 8, pp. 127–261, Jan. 2014. DOI: 10.1561/1500000042.

[4] J. Downie, "Music information retrieval," *Annual Review of Information Science and Technology*, vol. 37, pp. 295–340, Jan. 2003.

[5] G. Smith, M. Cha, and J. Whitehead, "A framework for analysis of 2d platformer levels," in *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*, ser. Sandbox '08, Los Angeles, California: Association for Computing Machinery, 2008, pp. 75–80, ISBN: 9781605581736. DOI: 10.1145/1401843.1401858. [Online]. Available: https://doi.org/10.1145/1401843.1401858.

[6] K. Compton and M. Mateas, "Procedural level design for platform games," in *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2006*, ser. Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2006, 2006, pp. 109–111, ISBN: 9781577352808.

[7] P. M. R. V. de Castro, "Music-based procedural content generation for games," Jul. 2017.

[8] D. L. Wessel, "Timbre space as a musical control structure," *Computer music journal*, vol. 3(2), pp. 45–52, 1979.

[9] P. Herrera, A. Klapuri, and M. Davy, "Automatic classification of pitched musical instrument sounds," Jan. 2006, pp. 163–200, ISBN: 978-0-387-30667-4. DOI: 10.1007/0-387-32845-9_6.

[10] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002. DOI: 10.1109/TSA.2002.800560.

[11] C. Laurier, O. Meyers, J. Serrà, M. Blech, P. Herrera, and X. Serra, "Indexing music by mood: Design and integration of an automatic content-based annotator," *Multimedia Tools and Applications*, vol. 48, pp. 161–184, 2009.

[12] E. Gómez, M. Haro, and P. Herrera, "Music and geography: Content description of musical audio from different parts of the world," Jan. 2009, pp. 753–758.

[13] S. Lippens, J.-P. Martens, and T. De Mulder, "A comparison of human and automatic musical genre classification," vol. 4, Jun. 2004, pp. iv–233, ISBN: 0-7803-8484-9. DOI: 10.1109/ICASSP.2004.1326806.

[14] K. Seyerlehner, P. Knees, D. Schnitzer, and G. Widmer, "Browsing music recommendation networks," in *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, Jan. 2009, pp. 129–134.

[15] P. Desain and L. Windsor, *Rhythm perception and production*. Swets & Zeitlinger Publishers, 2000.

[16] D. P. Ellis, "Beat tracking by dynamic programming," *Journal of New Music Research 36.1*, pp. 51–60, 2007.

[17] P. Grosche, M. Müller, and F. Kurth, "Cyclic tempogram - a mid-level tempo representation for music signals," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.

[18] M. Mauch and S. Dixon., "Pyin: A fundamental frequency estimator using probabilistic threshold distributions," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[19] A. De Cheveigné and H. Kawahara., "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America 111.4*, pp. 1917–1930, 2002.

[20] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, vol. 3, no. 3, pp. 172–186, 2011. DOI: 10.1109/TCIAIG.2011.2148116.

[21] ——, "Search based procedural content generation," in *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, vol. 6024, Springer LNCS, 2010, pp. 141–150.

[22] J. Dormans, "Adventures in level design: Generating missions and spaces for action adventure games," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10, Monterey, California: Association for Computing Machinery, 2010, ISBN: 9781450300230. DOI: 10.1145/1814256.1814257. [Online]. Available: https://doi.org/10.1145/1814256.1814257.

[23] J. Togelius, T. Justinussen, and A. Hartzen, "Compositional procedural content generation," 3rd Workshop on Procedural Content Generation in Games, PCG 2012, Organized in Conjunction with the Foundations of Digital Games Conference, FDG 2012 ; Conference date: 29-05-2012 Through 01-06-2012, 2012, pp. 66–69.

[24] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, "Launchpad: A rhythm-based level generator for 2-d platformers," *Computational Intelligence and AI in Games (CIG), IEEE Transactions on*, vol. 3, pp. 1–16, Apr. 2011. DOI: 10.1109/TCIAIG.2010.2095855.

[25] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2d platformers," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG '09, Orlando, Florida: Association for Computing Machinery, 2009, pp. 175–182, ISBN: 9781605584379. DOI: 10.1145/1536513.1536548. [Online]. Available: https://doi.org/10.1145/1536513.1536548.

[26] K. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial life*, vol. 9, pp. 93–130, Feb. 2003. DOI: 10.1162/106454603322221487.

[27] N. Altman and M. Krzywinski, "The curse(s) of dimensionality," *Nature Methods*, vol. 15, May 2018. DOI: 10.1038/s41592-018-0019-x.

[28] T. Bäck, "Self–adaptation in genetic algorithms," Oct. 1994.

[29] A. Liapis, H. Martinez, J. Togelius, and G. Yannakakis, "Adaptive game level creation through rank-based interactive evolution," in *2013 IEEE Conference on Computational Intelligence in Games, CIG 2013*, 2013, ISBN: 9781467353113. DOI: 10.1109/CIG.2013.6633651.

[30] G. M. Costa, "Procedural terrain generator for platform games using markov chain," 2018.