FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Music-Based Procedural Content Generation for Games

Nuno Filipe Bernardino Oliveira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Eugénio Oliveira (PhD)

Second Supervisor: Pedro Nogueira (MSc)

July 29, 2015

Music-Based Procedural Content Generation for Games

Nuno Filipe Bernardino Oliveira

Mestrado Integrado em Engenharia Informática e Computação

July 29, 2015

Abstract

Thousands of people spend plenty of their time playing videogames or other platforms everyday and the game content plays the most important role in their entertainment. In order to create the same experience in different players with distinguished preferences, it is needed to generate customizable and adaptable content. To help in these demands, there have been recent studies on the advantages of using algorithms to generate content procedurally and automatically, which will lead to a drastically decrease in the memory consumed and will save human resources.

A possible way to perceive the feelings and preferences of the player is through music, as he can relate his game experience to the music surrounding him and the music can embrace the player into a unique environment through emotions and avoiding boredom. It is common to find games that use the music as a supporting tool to enhance the gameplay, but the possibility of using different songs as an input source to create playable content accordingly is something that would improve the player's experience by creating the environment expected as well as using the game to match the player's preferences.

The goal of this dissertation is to develop a game, in a complete procedural way, through music segments, so that it will be possible to distinguish significantly different levels and also allow to create an opinion about the usage of music as a procedural content generation. Through a methodological analysis of a chosen song, this one will be segmented and have each part associated with important music features. For each one of the segments there will be a level with its content generated following the features retrieved, granting diversity and non-predefined levels, mainly to change the way the game can be played.

Resumo

Milhares de pessoas passam diariamente muito tempo a jogar jogos de vídeo, desempenhando o conteúdo do jogo o papel mais importante. De forma a criar a mesma experiência em jogadores diferentes, com preferências distintas, é necessário gerar conteúdo capaz de se modificar e adaptar às preferências individuais. No sentido de concretizar estas exigências, tem havido estudos recentes sobre as vantagens de usar algoritmos para gerar conteúdos de forma procedimental e automática, o que levará a um decréscimo acentuado na memória utilizada e poupará recursos humanos.

Uma forma possível de entender os sentimentos e preferências do jogador é através da música, dado que ele pode relacionar a sua experiência de jogo com a música que o circunda e a música pode envolvê-lo num ambiente único através de emoções, evitando o aborrecimento. É comum encontrar jogos que usem a música como uma ferramenta de apoio para aperfeiçoar a jogabilidade, mas a possibilidade de usar músicas diferentes como uma fonte de informação para criar conteúdo jogável concordante é algo que melhoraria a experiência do jogador, ao criar o ambiente esperado e ao usar o jogo para ir ao encontro das suas preferências.

O objetivo desta dissertação é desenvolver um jogo de forma completamente procedimental, a partir de segmentos de música, para que seja possível diferenciar de forma significativa os diferentes níveis e permitir tirar conclusões referentes à utilização da música como gerador procedimental de conteúdos. Através de uma análise metodológica de uma música escolhida, esta será segmentada e cada parte associada a características musicais fundamentais. Para cada um dos segmentos haverá um nível com os conteúdos gerados de acordo com as características detetadas, garantindo diversidade e níveis não pré-definidos, principalmente para alterar a forma como o jogo pode ser jogado.

Acknowledgements

I would like to thank everyone who contributed to the development of this dissertation and that helped me through it.

I would like to give a special thanks to both of my advisers, Prof. Eugénio Oliveira and Pedro Nogueira, for their constant support throughout this work and, mainly, for proposing this topic, giving me this opportunity and guiding me through it with confidence in my work as well as great enthusiasm, making this thesis a better work due to their supervision.

I would like to thank all my friends that kept by my side through these last months and helped me to overcome any problem I found while working on this thesis.

Finally, I would like to thank my family for all their support. Furthermore, I would like to thank my parents for the continuous worry about my work and for having trusted me on my journey so far.

Nuno Filipe Bernardino Oliveira

"Remember how far you've come, not just how far you have to go.
You are not where you want to be, but neither are you where you used to be."

Rick Warren

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Program Interface
BPM	Beats Per Minute
HSB	Hue-Saturation-Brightness
I/O	Input/Output
MIDI	Musical Instrument Digital Interface
NPC	Non-Playable Character
PCG	Procedural Content Generation
PCG-G	Procedural Content Generation for Games
RMS	Root Mean Square
STFT	Short-Time Fourier Transform

Chapter 1

Introduction

This chapter introduces the context of the thesis. A general introduction to the targeted research area is first given, followed by the motivation and objectives of this work. In the end, an outline document is presented, briefly describing the contents of each chapter in the document.

1.1 Context

Thousands of people spend plenty of their time playing videogames or other platforms everyday and, recently, it has been perceptible a great increase in the population interested in what concerns to games, as well as an increase in the demographic diversity of the players [2]. For them, the game content plays the most important role in their entertainment and therefore, each player's preferences and emotions can differ a lot from one another when regarding to the same game. Thus, in order to create the same experience in different players, it is needed to create different game content matching each player's profile [3].

It has been acceptably assured so far that the quality and the quantity of game content available could answer the demand. However, this exponential growth in the community of players, in addition to the production costs, have been forcing the need of new techniques to create playable content [4]. The procedural content generation for games stand against those problems by aiding in their automatic creation, reducing at the same time their cost. There are many reasons to use these techniques, with the most compelling one being the fact that it can reduce the memory usage by compressing the game content and only expanded when it is needed, saving resources [5].

These methods of game content generation also increase the capacity of games to adapt to each player, making the game less tiring. If the game adapts to the player, both in his preferences and in the game difficulty and game style, it can allow a unique and continuous experience. However, the viability of these methods to generate game content is still little explored, only being applied in racing games, shooters and few more [6].

Nowadays, the music is connected to the player's experience, allowing him to predict an event during the game or even increase the impact of the event in the game like in horror games. Tomáš Dvořák states that some sounds can be annoying to the player [7], since one of the most important roles of game music is to embrace the player in his environment through emotions and avoiding boredom [8].

Music Information Retrieval (MIR) has been the subject of intensive research by an ever-increasing community of academic and industrial research laboratories, archives, and libraries [9]. Although there is already a great advance in this industry, it is still far from perfect and it is in an ongoing adapting to everyday songs. There are a lot of ways to use this information retrieved from music, from personal usage matching each ones preferences to music professionals. Less is the usage of this information to generate new game content able to differ from current games.

1.2 Motivation

Even though there are many games including music and other sound components to create an ambiance around the player by helping in the rhythm of the game or creating a relevant atmosphere of the current state of the game, this is still usually done by creating the music previously and place it where it is needed, as it is done in the cinematography industry. The games that actually use the music as an input source of information to create game content are still rare. Even in games where the music is a main feature, its connection with the game can be static and manually generated.

To use the music as an input to automatically generate new game content in real time or before the start of the game is something new and it is now possible due to the growth and advance in the hardware, allowing the analysis of the music and the generation of the game automatically. These new methods can assure a better and brand new type of game, mainly because it allows the player to change the difficulty, the atmosphere, the mood and the gameplay by simply choosing different songs. More than just changing the game according to what the player feels like playing, it will also make the player curious and would make him rather try new songs just to have a new experience, which will translate into an increase of the diversity of the game.

Furthermore, the fact that the games out there that currently use music to generate new game content are all shooters or racing games, creates a new opportunity to develop a game based on a different style, which can let the player decide the pace that the game takes and let the player think and choose a music according to the kind of level he wants to play. Changing the base game also grants the opportunity to create different game modes that are applied accordingly to the song chosen. Those game modes bring a unique difference in the game industry by using the music to change the content of the game and use that exact content to change the way the game is meant to be played, because it grants a huge diversity that cannot be covered by making different games, one for each game mode.

1.3 Objectives

Based on the current state of the art, it is clear that music-based games rely on creating the content manually and the ones that generate it automatically are far from having a unique type of game, making it seem less about the music and more about the game. Besides the usage of the same game types, they also use the songs the same way, independently on the song chosen.

To study the viability of using music to generate game content procedurally it is needed to develop two main modules. One capable of receiving music and extract all the required features and the second one capable of generating a game based on input parameters. To connect these two modules, it is needed to map the output features of the first module into the input parameters of the second module and that will evaluate the viability and answer to the main goal of this thesis.

Regarding the first module, it is intended to create a game capable of receiving input parameters and generate its content accordingly. It is a Stealth-based game, therefore it is a game that requires the player to play it sneakily, which somehow is also related to music due to the fact that the sound plays a distinctive role in stealth games. Also, the game is playable and changes its level playstyle according to the content created.

In what concerns to the second module, in this thesis it is intended to develop a simple program that allows the user to choose a song and it segments the song into different parts. After having the song divided into different sections based on its main events, each segment is methodically analysed in order to retrieve the most important features of the sound and the ones that can be easily interpreted and used as comparison by the listener.

Finally, in order to connect the two of them, an algorithm is created to better map the music features into the parameters that will generate the game content, being this algorithm independent on the music or the game and it has no knowledge of each other process.

With this, the main goal of this thesis is to study the viability of such mapping and if it is possible to generate game content based on the song chosen. Hence, the main research questions targeted in this thesis are:

A1. Is it possible to generate a stealth game procedurally only based on input parameters?

B1. What are the most important music features to be used in game content generation?

C1. How viable is the usage of music features in the generation of game content?

C2. Is it possible to change the gamestyle based on the content created?

1.4 Outline

This report is organized as follows. The first (current) chapter provides a general introduction to the topics addressed in this thesis. The second chapter presents the literature review, including both

the state-of-the-art and related work in this topic. In the third, forth and fifth chapter, it is stated the approach proposed in this thesis and the described implementation of each module as well as an explanation of the mapping algorithm. In the sixth chapter, it is presented the experimental results obtained after applying these processes and how the method used is important to obtain better results. Finally, the last chapter completes this report with the conclusions on the research done and future work perspectives.

Chapter 2

Literature Review

This chapter presents the results coming out of the performed literature review on the domain of the problem focused by this thesis, providing an analysis on the related work, as well as background knowledge. This will be done by a time-line describing the evolution of the games related to the techniques involving this thesis and later on it will be followed by analysis, found in literature, of these techniques used together with music.

2.1 Procedural Content Generation

Procedural Content Generation (PCG) is often described as the use of algorithms to create content automatically [10, 3]. Togelius et al. [5, 11] and Shaker, Togelius & Nelson [12] define PCG as "the algorithmical creation of game content with limited or indirect user input". In other words, it is a software able to create content with human help or not. In this definition, the term content refers to any component of the game, including sounds, textures, levels, etc. To some of these authors it is better to exclude NPC (Non-Playable Character) as a component, because character behaviour is a different study field, more as Computer Intelligence than as PCG. Nonetheless, Nuno Barreto et al. [13] defends that generation of behaviours can be included in this definition, since it can be considered as adaptative behaviours and this way it will agree with the definition presented by Shaker et al. [12].

For a better understanding of the meaning of PCG, it would be better to go to its origin. In 1987, Crawford introduced the notion of "process intensity" [14], which claims that the software that uses more time for calculations than consulting data is better. Following this idea, all the games benefit from "process intensity", as well as PCG. Usually, the PCG algorithms create specific content from an initial description (initial parameters), which occupy a much lower space than the final result. It is common for this content generation process to be at random, although it might not be always like that[10]. For an improved understanding of the randomness in this context, it is shown by Togelius[11] that the generators include some stochasticity, creating, this way,

strong limitations to the content generated, which will vary randomly between these limitations. Although most of the PCG implementations use it as it was before, it is not needed to include stochasticity in a generator. With a different point of view, the developer of Roguelike, Andrew Doull, claims that the definition of PCG includes randomness [15].

PCG is also connected to different study fields as Computation Aesthetics and Computational Creativity[16], yet the academic studies regarding this topic have only been coming up in the recent years. It started with a PCG workshop happening in 2009 and the first article about this matter written only in 2011, even though there are often articles regarding Artificial Intelligence [17]. It quickly changed in 2011 with the creation of the IEEE CIS Task Force, workshops, a wiki[1], as well as an international competition in PCG[2]. Currently, it is easier to do some research in this topic with the release of the first textbook about PCG[3]. The solutions studied academically usually focus in adapting or controlling algorithms, still, the ones mostly used all over the last years in games are the simplest, fastest and least controllable ones.

As it was already stated previously, the biggest reason to use these methods of generating content is to remove the need of having a designer creating and drawing that same content. The human resources are expensive and slow, but they are more and more required. Currently it is common to find hundreds of people working and developing the same game, for a year or more, which will lead it to be less profitable, ruining the market and the game itself. Even Will Wright, in his talk "The Future of Content" in Game Developers Conference 2005, stated that the company that would be able to replace the game designers by algorithms would take the lead, since the games would be produced faster and cheaper, while still keeping the same quality. It is possible to also look at PCG as a tool that can increase and improve the creativity of the game designers that work in smaller teams. Or, at least, let them be able to take advantage of this technique to avoid more tiring others, as for example, to create background terrains, that will be the starting point to develop the rest[18]. Another great advantage of the usage of PCG is being able to adapt all the generated content to the player's needs and preferences. It would be possible then to create customizable games, which would keep trying to maximize the fun in the player's experience.

Following the stated idea about creativity, PCG can be used as well to keep the designers engaged, since it will allow them to discover a whole new ideas different from what they initially would expect. For this reason, some of these techniques are already used in evolutionary design[12].

2.1.1 Procedural Content Generation for games

As stated in the previous section, the definition of game content is all the aspects that affect the game style but not the character's behaviour or game engine[3]. It is important to always keep in mind that the game content is the main reason that keeps players engaged in the game. So, the

[1] http://pcg.wikidot.com
[2] http://www.marioai.org
[3] http://pcgbook.com

demand of new and more specific contents keeps increasing, while its production by humans, in addition to being expensive, cannot follow it[19].

PCG for games will allow the creation of new content, which distinguishes themselves and at the same time will focus on the player's preferences. This is not an easy task, not only because of the required computer power, but also because it needs to value the generated content to be able to reach better results. For these reasons, it is noticeable why it has not yet been possible to make a game fully procedurally generated [4].

Currently, Procedural Content Generation for Games (PCG-G) is possible to be the most important research in videogames [10], mainly to extend the possibilities of content creation and to increase the complexity and detail in games, as required in recent games, while it becomes harder to organize and manage time in these projects [20]. Even though games like Creature Created of Spore (Electronic Arts 2009) or Sim City (Maxis 1990) may be able to generate a different content in real-time or before the start of the game, they both need human intervention every time. However, it is possible that similar games may end up generating the rest of the content by algorithms [11].

2.1.2 Application Examples

There have been attempts to procedurally generate playable content since a long time. During the year of 1980, the game Rogue [21, 22] appeared, considered by many as the pioneer of the procedural generation [3]. In this game it was possible to generate automatically maps for the player to explore. The fact that you could repeat the game continuously, without an ending, would end up inspiring many other games, known later as the Roguelike games[23]. Years later, some of these versions of the original game would be crucial to some more modern and recent games, like the trilogy of Diablo (Blizzard Entertainment 1996) or Dwarf Fortress (Bay 12 Games 2002), which already include a complex interaction with multiple different objects placed in the game world procedurally. However, in the latter, for each one of the content created, all the creation process is very simple and the worlds generated show little variation [17].

One of the first goals of PCG was to surpass the hardware limitations. Initially, because the computers did not have enough memory, games like Elite (Acornsoft 1984) and Starflight (Eletronic Arts 1986) arose during the eighties, as a result of their way of applying PCG to generate worlds that would be impossible in the computers of that time. Moreover, currently due to the advances of the software, this kind of applications is rare.

In the game Elite, it was possible for the first time to create hundreds of solar systems in few kilobytes of memory available in the hardware, encoding each planet in few numbers. After expanded, every planet would have names, populations and so on. This game was an interesting example of deterministic content generation and due to the lack of memory already stated, it was developed a PCG algorithm capable of generating every information deterministically from a single seed, and all the universe of the game was saved only in a list of seed values, using PCG as a way of compressing data [11].

While in these last games mentioned every piece of content generated was determined before the game start, new researches aim to allow the player to directly influence the content created during play time[6]. Games like the Charbitat [24] and Facade [25] are examples of evolutionary content generation.The former consisting in determining the player's behaviour and transforming it into seed values to be used later on in the generation of new spaces, and the latter creating interactive stories based on the player's decisions along the game.

Yannakakis [26] e Brown et. al. [27] also defend different applications of PCG. They claim that PCG can be used to give to the player a customizable experience by the way he interacts with the game. One of the biggest examples of the usage of this technique is the game Galactic Arms Race [6], which is an online game of spaceships for 32 players. In this game it was implemented a new algorithm cgNEAT, created solely to generate automatically game content based on the preferences of the player in real time. Unlike other evolutionary algorithms, this one selects content based on the behaviour and the way the player plays in that given time, i.e. content as weapons or ships used more frequently are more alike to be generated. Also following the same idea, the weapons of the players are improved based on the player's preferences.

Figure 2.1: Galactic Arms Race

Even though these games use PCG techniques, they are only variations of the game genres already known and not unique genres, whose existence is due to the PCG [17].

The game content procedurally evolutionary is a field that can contribute largely to the videogame industry. Some of the current examples include race tracks [28] and the generation of their rules [29]. To evolve the race tracks, the artificial controllers are transformed comparing their performance to the one from human controllers, on the same track. Later on, the artificial controllers are tested in the new tracks generated and used to value their performance and keep generating new ones, in order to keep being a challenge to the player [28].

In the case of the rules, instead of artificial controllers, they are evolved by a grid system, where the aspects of the game are changed in order to reach better fitness to beat the game [29]. This idea was important to show that it is possible to evolve games like Pac-man [13].

Recent studies have been searching a way to combine the evolution of games with their graphic art, like in the game Avera [30, 31], in which its system evolves interactively puzzle pieces.

Currently, PCG is almost only used in specific fields and almost always during game developing. One of the most known techniques is SpeedTree (Interactive Data Visualization 2013), which generates automatically great number of trees looking alike to populate big spaces in the game world. Other technique also known is Euphoria, a tool that provides adapting animations based on specific mechanisms [32].

2.2 Music in games

Music is an important element in the majority of games. It is used to create an ambience around the game, with all the sound effects being important for the player to have a proper feedback of all his actions and changes in the game atmosphere [4].

2.2.1 Music as main feature

The games that are based on music have been valued along the recent years by attracting people that usually would not care about videogames [20]. Examples of games known by this are Dance Dance Revolution (Konami 1998), Singstar (Sony 2004) and Guitar Hero (Activision 2005), where all the ambience of the game is related to the music chosen. Each one of these games is well-known in this field, because they are able to engage the player in different ways among them, even if the music would be the same. However, all the content used in these games is prepared manually and before the game is played, ending up creating restrictions in the number of songs available to be played.

To address this problem some different games came up, like Rez (Sega 2001). It is a game that already combined the visual effects and the enemy appearance with the composed music. Also Electroplankton (Nintendo 2005) allowed the player to interact with the game in different ways to create sounds and visual effects connected with the music.

More recently, there has been an increase of interest in creating music related to games in mobile platforms. One example is the game Scratch-Off [33], which requires the player to interact with the rhythm of the music. Likewise, games as The Impossible Game (FlukeDuke 2011) and Geometry Dash (RobTop Games 2014), create obstacles according to the music events. Bit Trip Runner (Gaijin Games 2010) already has six different gamestyles, all of them related to the rhythm of music but reacting distinctly. All these games are a fine example that game content can be reactive to the music around, but still they remain limited because all of their content is prepared before and would then only work in some songs.

In addition to these games mentioned, there has been some research relating PCG with music, this means, games that use music as the main input to generate their game content. Yet, it is

always a challenge to create this type of games, firstly because the input used must be extracted automatically from the music given, and secondly because it should create a more complex game that would at least allow the player to distinguish the content generated for each song [20]. To follow the player satisfaction, as shown by Csikszentmihalyi [34], the impact of the extracted features must be visible, but not predictable, so that they can keep the game difficulty balanced as well as its simplicity.

2.2.2 Music as Procedural Content Generation

There are already games that have created an impact in this field, as it is the case of the game BeatTheBeat [20], which is a multi-player game, in mobile platforms, that implements in a completely automatically way the feature extraction of music and, later, generates the game content procedurally. Based on a music list and their features, the game organizes a map in a hexagonal board where each field in the grid contains a music and one of the three mini games available. Each one of these is later played with the correspondent song. This game has also a unique feature that allows the game to learn if the player is able to distinguish between the levels that use the music chosen and the ones which music was chosen randomly. Using this feature will make possible for the game to change the levels accordingly, making them less predictable.

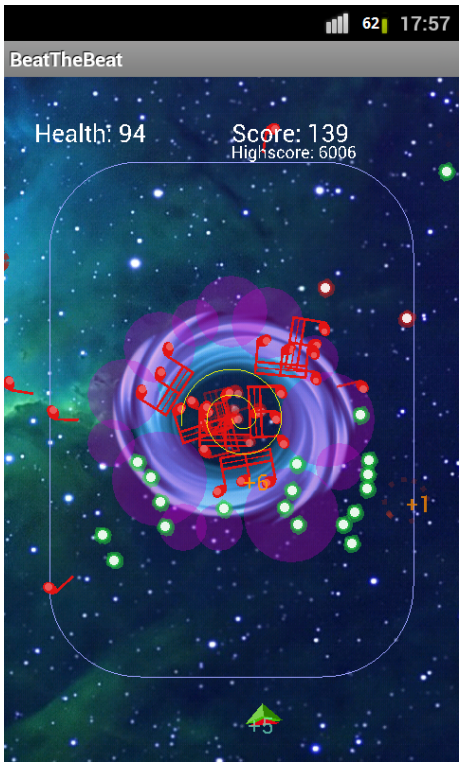

Figure 2.2: BeatTheBeat in one of its game modes

Audiosurf (Steam 2008) is also a well-known game that generates the game ambience according to the music chosen. It is a spaceship racing game that allows the player to choose his own

music, which will lead to new and unique shapes and velocity of the track, as well as their curling. In this game, the main goal is to keep getting points when gathering the blocks of the same color that are showing up during the music. Although it may seem they are generating in real time according to the rhythm or music effects, all the content was already generated previously during the analysis of the music. It already has its follow up, Audiosurf 2, which ends up with little changes in its music related algorithms, mainly focusing on graphic and user interface improvements.

To conclude, another game that also uses music as main input to content creation is Beat Hazard (ValuSoft 2011), where the player controls a spaceship and destroys different enemies which will appear according to the rhythm of the music. Following the music flow, different powers are generated and given to the player to keep the game evolving and connected with the music chosen. This game already has its follow up, Beat Hazard Ultra that uses the same idea of the previous one but mainly improving in its design and features.

Chapter 3

Music-based Stealth Game

This chapter presents the approach proposed in this thesis. The game, developed to evaluate the usage of music as an input source to generate content procedurally, is first presented.

3.1 Stealth Game

In order to evaluate the effectiveness of the proposed approach, the game developed should have enough variations, so that the results could vary significantly according to the parameters. Following this idea, the game required distinct features that would make the game playable as well as diverse input parameters that would change the game content.

As mentioned in the literature (2.1.2), there are only few genres using PCG, and most of them rely on the rhythm of music, creating races or on-hit games, where the player has to click at the same time, or follow some steps according to the music. Some other game genres require the player to shoot on enemies, which also appear following some pattern according to the music. All in all, even though these games change according to the music, they end up not changing the play-style. The variations are often only visual or difficulty wise. Due to these aspects, it was important to develop a game that would be considered in another genre and at the same time could relate to the songs chosen to change play-style, so it was developed a Stealth-based game.

This kind of game is known for challenging the player to avoid alerting enemies altogether. This way, the player must avoid combat, minimize noise done and strike enemies while hidden. It excels in the fact that there are multiple ways to achieve the final goal, with distinct pathways or even play-styles.

In this genre of games, the player usually has to choose between killing or passing by the enemy, use certain objects in his advantage, or control the enemy paths and use the available items the better he can. Also, typically if the player attracts the attention of enemies, he must hide and wait while the enemies thoroughly search. Thus, planning is very important, although some stealth

games allow physical combat skills, there are also games that imply trial-and-error every time the player is spotted.

In order to include stealth gameplay in the game, the Artificial Intelligence (AI) of the enemies must be restricted [35], making them ignore parts of the game world, and only focus on player's actions, like noise made on purpose. Typically, the enemies have a line of sight that can be easily avoided by hiding behind objects or walls, or moving while the enemy faces another direction. It is often for the enemies' movements to be predictable and regular [36], so the player can devise a strategy to overcome and pass the level.

3.2 Game Description

The game developed and used as the purpose of this thesis is, as mentioned in the previous section, a stealth-based game. So, the main goal of the game is to pass all the levels while avoiding being caught. For that, the player can move in eight directions, the four cardinal directions plus the four ordinal directions, and go through the walkable tiles, this includes everything that is not a wall or a box, until he reaches the destination, which is the final spot marked with green lights.

While playing, the player can collect three distinct items and make use of them to help in the level. The game continues as long as the player is not caught, but when that happens, all the level is reset and the player must start again.

Each level is generated based on its input parameters and even when resetting the level, it is generated again, but it keeps the same parameters as before. This avoids non-playable generated levels, which also include the most difficult ones, and also keeps the game from being too predictable.

A whole game is composed with various levels that come from the segmentation of the music. Each segment has its own parameters that will be the source of the generation for the respective level. Every time the player reaches the final spot, he automatically passes to the next level and so on, until every level is completed.

The game was implemented using the java framework Slick2D that allowed easier manipulation of the rendering and update functions by frame. The game is fully 2D, making it simpler to implement but with enough conditions to show the purpose of this thesis. It is a tile-based game, therefore the map is composed by a grid of square tiles. Although the player can move in a diagonal way, all the game is adjusted to fit in the tiles and in each tile is rendered its own sprite. Thus, the game is easily understandable as a tiled game.

To avoid slow performances, most algorithms were optimized, mainly in the rendering functions, still, to avert impairing movements and low frame rate, the camera zoom out is limited to a certain number of tiles, as well as the dimensions of the map.

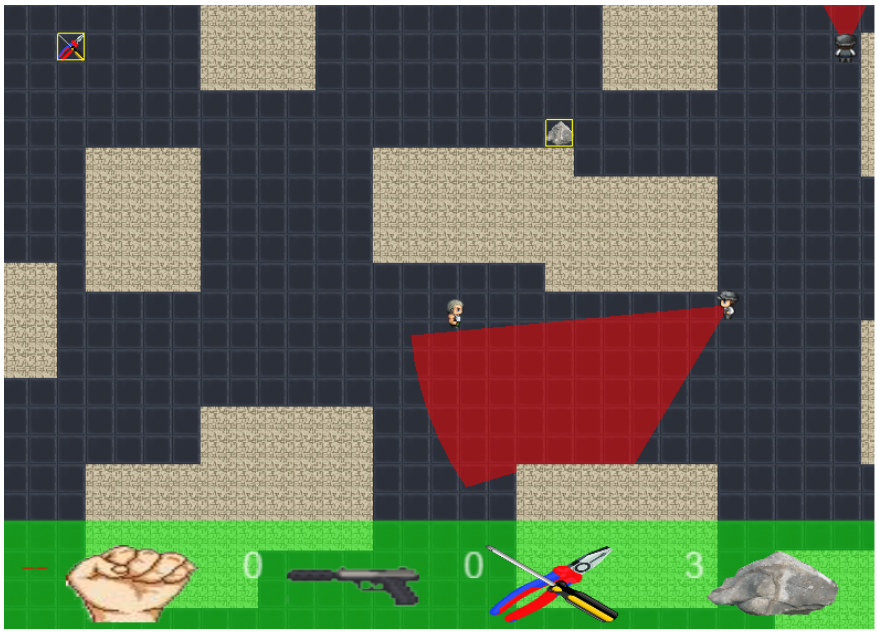

Figure 3.1: Screen capture of the game

3.3 Game Features

The diversity of the existent features in the game is important as they will contribute to be able to distinguish significantly each level. It will be explained what is the role of each non-playable character as well as other elements of the game. This explanation will include their behaviour, their reactions and how they are generated and embedded in the game map.

- **Movement** — The player has three types of movement. The main one, where the player moves faster will make some noise for each step. The second movement is the slow walking. This will prevent the player from making noise with the cost of moving slower than normally. The last movement available is to crouch. Crouching is the slowest type of movement and will also not make any noise.

- **Noise** — The noise is a feature implemented in the game which can aid or harm the player. Every time the player walks, except the special movements, he creates noise in a circle around him, being louder from the center to the outside circumference. The only elements in the game that can interact with the noise are the enemies and if they get inside the noise circle mentioned before. If this happens, the enemy will notice and the loudness of the sound will define if the player is caught or not. Hence, it's important to stop making noise if only a bit was heard by the enemy. Otherwise, it will trigger him, as it will accumulate if the noise continues.

  Another way of making the noise interact with the enemy is when a rock is thrown. It will create a noise in a circle where it lands, the same way it is created when the player walks. This can make the enemy notice, which will make him react to it.

- **Enemy** — The enemies are the guards walking around the map. They only move in the four cardinal directions and they follow a generated path that will not change as long as the level keeps going. Addressing this previous idea, they will go back and forth the same path, and will rotate over them every time they have to change direction.

  Since the enemy is moving, they have a defined orientation, over which is projected a vision field. This vision is not able to overcome walls or boxes and will stop every time it collides with these objects or reaches the end of its sight distance, defined before the game start. The vision has an angle of 30 degrees and also rotates according to the enemy rotations. This vision is implemented using the *Raycast algorithm* in 2D [37].

  Each enemy can trigger two actions with his vision: first, and more important, if he caught the player in sight of him, the game stops and a message appears telling the player that the level will restart. The other action is to trigger alert system, which will be explained later, and happens every time a guard sees a fainted enemy, and will only trigger it one time for fainted body.

  If an enemy hears a rock hitting near him, he will react by changing his path to where the rock was landed. After checking it, he will come back to the place he was and keep following the same path as before. Also, if the enemy faints because of any action of the player, it will remain like that for the rest of that level, without moving or causing any impact on the player's actions.

  The generation process for each individual enemy follows certain rules. First of all, it has already knowledge of the shortest path between the player and the final spot, giving that, the enemy must start in a position that is, at least, their sight vision range distanced from the player starting position and the final position. Next, dependently on the number of enemies, roughly at least a third of the enemies are placed in the shortest path, while the others are randomly placed throughout the map. While choosing their starting position, the process takes into account where the previous enemies have already been placed, so it tries to avoid placing them close to each others. After placing all the enemies in their starting position, it calculates their path, and it should be within a certain range, from 4 to 10 tiles, and if the enemy was already placed in the shortest path, he will be patrolling that same path. For the others, they will have their patrol route calculated randomly, preferentially if they close the maximum paths possible.

- **Camera** — The Cameras are, as the name says, the surveillance cameras placed along the map. They cannot move, so they stay placed in the same spot since the beginning of the level. They can be placed anywhere, as long as it is not inside a wall, but other than that, they can be placed near a wall or an open field.

  Cameras have a vision sight similar to the enemies with an angle of 30^{o} degrees and also with their vision blocked by obstacles. Instead of keeping their vision in the same angle while walking, cameras keep rotating over themselves, doing a full circle. Cameras may

keep rotating and doing circles always in the same direction, or always after completing 360 degrees, they change direction and restart. Whether it is one or the other option, it is related to the rotation speed of the cameras, which is defined previously from the input parameters.

Also similar to the enemies' vision, cameras can trigger the same two actions. They can spot the player and make him restart the level or they can spot a fainted guard and trigger the alert system.

Cameras' generation process resembles the enemies' one. Having knowledge of the shortest path between the player and the final spot, it assigns a third of the cameras to the path. Although their placement remains random, inside the limitations stated before, it also avoids placing cameras next to one another and with vision to where the player begins.

- **Traps** — The traps are an essential element of the game to improve the game play changes. They are composed by a red line that connects two parallel walls, creating a perpendicular angle between the line and the opposite walls. Trap lines can cross between each other and can be in both vertical or horizontal direction.

  The line is invisible to the player unless he gets close enough. Because of that, it is unpredictable to know whether there is a trap near or not, so walking carelessly while there are not enemies around can be punishing. In order to pass the trap without triggering it, the player must crouch and move below the line until he completely reaches the other side of the trap. If the player is not crouched and intersects the trap line, it is signalled as caught and the level will be reset. Traps have no more extra actions and cannot move or signal any enemy in the presence of the player. Although the player must crouch to overpass it, any guard can pass by without triggering the trap.

  The generation process of the traps is in parts similar to the other elements. With the knowledge of the shortest path, a third of the traps are placed there, while the others are randomly placed in the rest of the map in order to cover other paths. A trap is divided in three components, the two ending points and the line that connects them. A first endpoint is generated and placed next to a wall. After calculating the perpendicular line of that wall and if the next wall intercepted by this line is under a given limit of 200 pixels, a second endpoint is placed in the intercepted wall and a line is created between the two endpoints. For each trap created, there is not a limitation if one trap is closed to another, only if they overlap in both endpoints. Even though this can keep the traps all stacked up in one part of the map, it is interesting how traps can be really closed to maintain the game unpredictable.

- **Boxes** — The boxes are a less important element in the game. They occupy two tiles of the map horizontally or vertically and they are designed to block the vision from both enemies and cameras. Since the boxes can start in less useful places, they can be moved to a better location. To move the boxes, the player must walk towards them, which will slow the players' movement, along with the box movement, by half.

In spite of being able to move the boxes through the map, the player must be careful, because if an enemy or camera spots movement from the box, regardless of the player's position, the level will end and reset as if the player was caught. The same applies if the enemy touches the box, or the box interferes with the enemy's path. Since it was not supposed to happen, it will work as if they knew the player was there and his mission of avoid being caught is compromised.

The process of generating boxes is pretty simple compared to the previous ones, since it will place them randomly throughout the map as long as they are in a place with possibilities of being pushed, and not in the middle of the enemies' paths. During the generation, it also takes into account whether the boxes are close to one another, since it would not bring more advantages that way.

- **Inventory** — The player has an inventory with four item slots. Each slot is already assigned to an item that the player can use in order to create an effect in the game. The items are spread all over the map and each one of them has an image identifying itself. The player can collect every item he wants, there is no limit, and can see in the inventory how many resources are left for each item, if the item has a 0, it means the player has no resources of that item and consequently he cannot activate it.

  The first item is the *hands* which have unlimited use. When activated, the player will use it in his direction and will only affect the next tile. It can only be used to knock out the enemies and in order to do that the player must sneak in close to the enemy and activate the *hands* item. If he is close enough and is still not detected by the guard, he will make him faint.

  The second item is the *pistol* which uses bullets. The player starts with no bullets but can get more collecting items with the pistol image. Every time he activates the pistol, he uses it in the direction he is facing. The bullet will cross all the map and stop if it collides with a wall. If before a wall, the bullet collides with an enemy, it will knock the enemy out, resulting in the same effect mentioned while describing the previous item. Since there is no range limitation, the player can make use of the limited vision of the enemy and shoot him before the enemy spots him. The use of the *pistol* will not trigger anyone, since it is a silent gun. Nevertheless, if it knocks out an enemy while being watched, it will trigger the alarm system.

  The third item is the *tools* and are specific for camera usage. The player starts without any tool, but he can collect them. Even with resources available and activating them, it will not work unless the player is facing and close to a camera. If the latter happens, the camera will be deactivated and stop working for the entire of the level. This is the only consequence, since no more elements in the game can notice that the camera is not working, nor can the player use his *tools* to anything else. It will, however, make things easier than shooting an enemy in more patrolled paths, since it will not be detected.

The fourth and last item is the *rocks* already mentioned before. The player starts already with three rocks and can gather more if he collects them. Every time he activates the *rocks*, he will throw one in the direction he is facing that will make noise where it lands, which can be on a wall or the maximum range of the throw. The noise created by the rock hit will only trigger effects in the enemies near it, even if the rock hits the enemy itself. A rock will not change any state of the game and can only make the enemies briefly change their path, which can be useful.

All items except the *hands* can be collected, and they are shown in the game world with the same images as in the inventory. During their generation procedure they are placed randomly around the map, with the only limitation of avoiding being too close to each other. Each type of item has a percentage of appearing, changing accordingly to the game play, providing more of one item if it can be used more often in that particular level.

- **Alert System** — The alert system, which has already been explained before, is triggered by the first time one fainted body is spotted by another guard or the cameras. During this alert, every camera will rotate faster and every enemy will move with twice the speed they had and will see further. Also, they will be rotating faster to make their movements smoother. Every guard when entered in the alert state will create a new path to follow that would be also longer than the normal route defined before.

  While the alert is maintained the player cannot end the level and go by the final spot will not produce any effect. It will last until every guard has finished the alert route created before. When they all finish, the system returns to the normal one, and all the affected ones will have their attributes back to their initial state.

3.4 Game Parameters

In this section it is described the input parameters that the game receives from the analysis of the features of its song segment. It always receives all the parameters and generates the map according to them. These are all the ways that each song can interfere with the generation process, still, the major differences come with the accumulation of different values for each parameter, leading to a variety of gamestyles.

- **Map Scale** — This parameter is the one that will define the dimension of the map. Even though a level can take longer to pass due to the size of the map, it will not interfere with its difficulty. The value received is multiplied for the initial dimension of the map, so it will be scaling its size in a linear way.

- **Number of Map Blocks** — A block is each rectangle or square of impassable terrain placed along the map. They will create different paths for the level and will define their width.

  The blocks are generated following a simple Dungeon-building algorithm [38], like the ones used in Roguelike mentioned previously (2.1.2). This algorithm was adapted to match

the requirements of this type of game, mainly by reversing the *wall* tiles to *floor* tiles and vice-versa. It will be explained later (3.5)on how the generation process will proceed step-by-step.

- **Number of Enemies** — As the name of the parameter indicates, it will define how many enemies will be wandering through each level.
- **Number of Cameras** — Following the same idea as stated above, this will indicate how many cameras will be supervising the paths of the map.
- **Number of Traps** — This parameter will define how many traps will be spread all over the map.
- **Number of Items** — The number of items available in the map to collect is defined by this parameter. Although it does not directly define which type of item will be generated, the items created are limited by this parameter.
- **Number of Boxes** — As referred before, this indicates how many boxes will be created and spread around the map.
- **Enemies Speed** — Every enemy will have the same speed indicated in this parameter. This speed is applied to every frame of the game using the time passed between current frame and last frame. Its value is given as the amount of pixels covered by frame, so it will have small values ranging approximately between 0.01 and 0.05. Despite the speed varying between slow and fast, the rotation speed of the enemies stays the same for every level and will only be changed during an alert system.
- **Enemies Vision Range** — The range of the vision of the enemies is also something that is defined by the input parameters. It only changes the distance and will not interfere with the angle. This range value is measured in pixels and is applied as the radius of the vision.
- **Cameras Rotation Speed** — Similar to the enemies, the speed of cameras is applied to every frame, but it is only applied as a rotation effect.
- **Cameras Vision Range** — This parameter is exactly the same as the enemies vision range described before, but it will be applied to the cameras.
- **Colors** — The color is a parameter that will change the visual aspect of the level. It is applied as an overlay to the *wall* tiles of the map, and will not have any impact in the gameplay as seen in 3.2. To increase performance, instead of rendering on the top of each tile, it creates a new tile where the color occupies a new layer and uses transparency to give the desired effect.

These parameters are defined for each song and can be changed in order to change the game content that will be generated. All of them must have a value assigned, since the algorithm of generating each level will require it to be able to create the content needed.

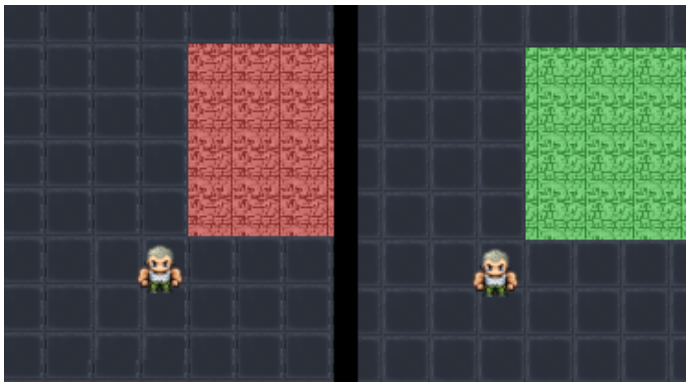

Figure 3.2: Color changes between different levels

3.5 Level Generation

When a level is about to start, it first procedurally generates all its content based on the input parameters. First, it creates an open map with the right dimensions. Then, the dungeon-building algorithm, mentioned before, starts running to create the blocks that will fill the map. The normal way the algorithm works is as follow: a first room is created randomly with also random dimensions; after that a random path is generated and connected to another generated room; this keeps looping until every room is created. In the changes made to the algorithm, instead of rooms, blocks are created and connected with other blocks, with walkable paths. This process keeps going until all blocks, limited by the parameters, are placed and it will automatically create some paths that will allow the player to go through.

When it is finished, a random position in the top-left of the map is chosen for the player to start, as well as a place in the bottom-right of the map is chosen for the final spot. The algorithm confirms that there is a path between these two and that it is possible to go through. If it is not possible, new positions are given or if needed, which is unlikely due to the paths already created in the dungeon-building algorithm, the generation starts again from an open map. The algorithm used to calculate the path between the player and where the level finishes is a simple A* algorithm to find the shortest path, using an estimated cost of the distance for the heuristic method based on a map without obstacles. At the same time it also considers the distance already travelled, balancing the two as it moves from the starting point to the goal [39].

After this step of the generation process, it starts placing the elements based on their number limitations given by the input parameters and following the generation process for each element. The algorithm keeps trying to generate possible levels, but if some generation would not allow it, e.g. a map with no blocks and trying to place traps between paths, the algorithm will restart and use a different allocation for each element. After some tries, the algorithm stops and the level will not be created. This might happen in case of changes to the input parameters after being defined by the song analysis explained in the next section (4).

All this process can be checked in the diagram in the figure 3.3 with each element following its own generation process that will limit or enhance the place where it will be placed.

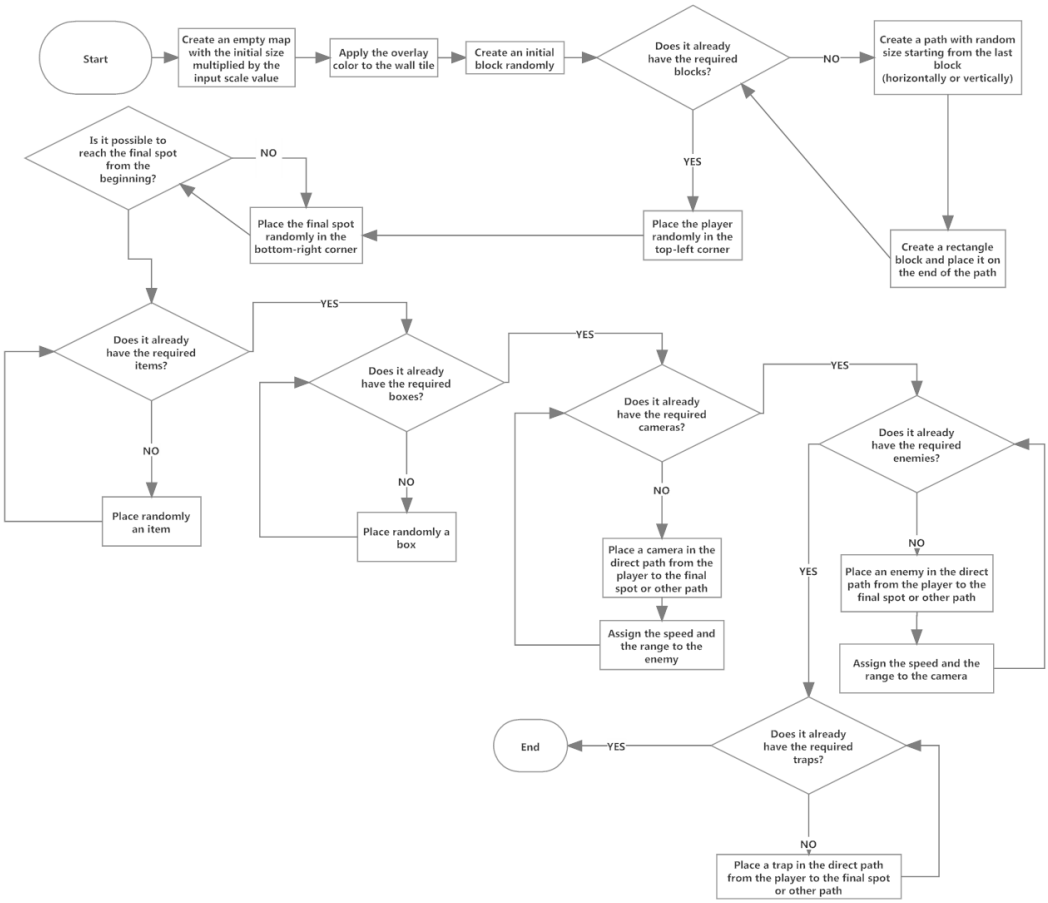

Figure 3.3: Process Diagram of the level generation algorithm

Chapter 4

Music Features Extractor

This chapter presents the music features extractor developed with the same goal as the previous chapter. There will be a detailed description of the music features extracted.

This is the second module of this thesis and is based on the idea of receiving a song and extract some music features from it. For that, it uses two main APIs, Meapsoft API[1] and The Echo Nest API[2] used in its java client version jEN[3]. It allows the user to choose his own song and go step by step until the game is ready to be played with that song. Other than that, it also allows the user to use songs already analysed before.

4.1 Extraction Process

The extracting process is divided into various steps to enhance the features extracted and to get better results. For the process to start, it is needed that the user first chooses a song that must be in *.mp3* or *.wav* formats. The file name of this sound will be used as the base name for all subsequent file I/O operations. After it is chosen, the program allows the user to segment the song.

4.1.1 Segmentation Process

For this step, the program analyses the input sound file and outputs a list of segments representing events present in the sound. This events mode simply detects sudden, substantial changes. The user can change two values that will affect how the software will segment the song into different parts. To qualify the events, it is used a first slider, also nominated sensitivity. A low sensitivity will result in fewer segments than a higher sensitivity. It is used as a sensitivity to change; if the sensitivity is high, then everything may seem like an event. The other value, designated time window, can also be set by a slider and will define what the size of the window is to check for a

[1] http://www.meapsoft.org/
[2] http://the.echonest.com/
[3] https://github.com/echonest/jEN

change. It determines how closely spaced events can be; a small window will allow very closely spaced events, while a high window will ignore events that occur too close together. Each segment calculated will be analysed to generate a level on the game.

To notice a change of event, it first runs a Short-Time Fourier Transform (STFT), which is used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time, on the input sound signal. After having its Fourier transform, using the time window defined, it runs an onset detection [40]. This is a technique in signal and audio processing to detect the position in the signal where a new note begins. And for each window, it looks for a local maximum of the onsets, where that value is also higher than the sensitivity threshold defined. In case that happens, it maps the power of the frame into a mel scale, which is a perceptual scale of pitches judged by listeners to be equal in distance from one another.

In the end, a first order difference equation is computed over the values obtained and it checks where the changes occur. So, in a detailed description, the sensitivity will interact with what is considered a maximum value of an onset, and the time window will define what the range in frames is where a maximum can be found. This process is implemented in the software of this thesis by using the Meapsoft API. After outputting the list of segments, because a small segment is not appropriate to play each level, it aggregates the small ones into bigger ones, in order to create segments with adequate sizes and still different.

All the previous process may take several seconds and in the end it will be shown in a color time-line how many segments were calculated and in what time intervals they occur. The segmenting process can be run multiple times in case of pretending to change the defined values for it.

4.1.2 Analysing Process

As soon as the segmenting process is finished, if the segments are similar to what the user expects, he can start the process of analysing the sound features. This process has two main parts, each one using a different API, the first part uses the Meapsoft API and the second part uses the Echo Nest API.

Initially, by using each segment start time and its duration, the program will analyse the correspondent part of the sound and will retrieve different features such as: Power, Spectral Centroid and Pitch. These features will be explained later on section 4.2. The Meapsoft API will output in a *.feat* file all the values of the features for each segment and then splits the initial song that was input in the program in sound files with the same time as each segment. The output file occupies an average of *2kb* and will have its content displayed, for example, in the following way:

```
1 #filename  onset_time chunk_length AvgChunkPower(1)  AvgPitch(1) AvgSpecCentroid(1)
2 Trinity.mp3   0.0 64.96000036 -21.900351088 49.498308205  1095.3029109
3 Trinity.mp3   64.959999084  39.600000143  -21.5851679279  54.99830820 964.65337295
4 Trinity.mp3   104.55999755  12.767999649  -21.3085386683  50.2483082  923.02438313
5 Trinity.mp3   117.32800292  20.368000179  -21.438346040 57.998308205  1033.62484048
```

Each line represents a segment and it has the value for each feature displayed in sequence. The *onset_time* represents the seconds where the segment starts and the *chunk_length* its respective duration.

In the second part, already using the Echo Nest API, the program will upload each sound file, generated after the split, and retrieve the following features: Loudness, Energy, Tempo and Valence. All these features are also explained in the section 4.2. After this process, the program proceeds to save each value of the features in the same file of the first part.

Due to the need of uploading each file, this process may take a couple of minutes depending on the upload speed. If the file has already been uploaded, the process is largely accelerated and takes only some seconds.

Finally, after retrieving all the values, the software will map each feature into parameters and display them. This procedure is explained in the section 5. The user can see what parameters were output for each segment and if he seeks to edit them, it is possible by clicking them and change the desired values. However, in case this is done, the parameters may end up being not related with the sound used and if the parameters are changed in the wrong way, i.e. abnormal values, the game will not be able to generate the corresponding levels.

When the user is ready to play, after the analysis is over, he can click *Start Game* and each level will be generated accordingly to its segment, as well as the segment sound will be looping while the level is played. Before launching the first level, the program will proceed to change the format of each sound segment to *.aiff* in order to be better supportable in the game application. It will also save the feature values for the sound analysed, which will allow the player to re-use it anytime he wants without the need to do the previous described process. As a result of using an already analysed music, it will allow the user to start the game in only some seconds.

4.2 Music Features

In this section, each feature extracted is described more meticulously, how it is done and how it can be interpreted. It is important to notice that even if each feature is independent of another, they might share similar characteristics and can be grouped together to better interpret the music.

- **Power** — The power is roughly known as the intensity of the sound. So, it distinguishes sounds with a higher volume than others.

  This power is shown in dB, which is a logarithmic unit used to express ratio between two values. In order to retrieve this value, the program first applies the Short-Term Fourier Transform on the audio signal and then it acquires all the transformed frequencies of each frame of the sound. It depends on the amplitude of signal, which can be understood as explained in the figure 4.1.

  After having a transformed sample of the desired segment, it calculates the square of the amplitude of the signal for each frame. Because it wants the average power in the segment, it will calculate the mean of the squares of the amplitudes, following the formula:

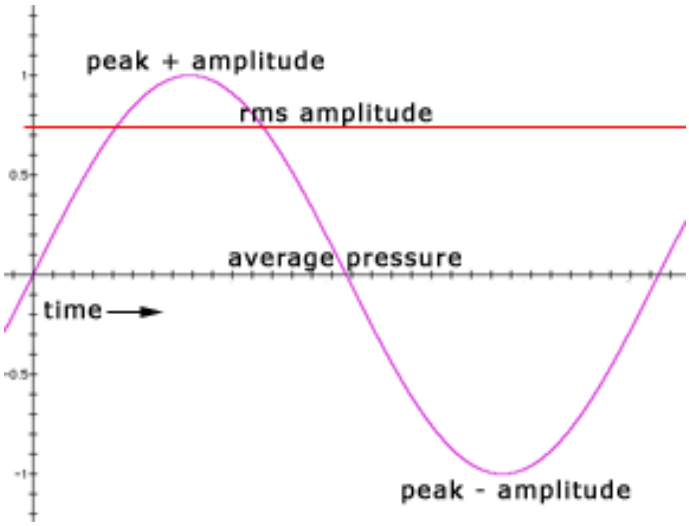

Figure 4.1: Amplitude and RMS amplitude of a signal

$$\bar{A} = \frac{\sum_{k=1}^{n} A_k^2}{n} \tag{4.1}$$

After retrieving the average square amplitude of the signal, it will then proceed to calculate the power of the signal in decibels by:

$$P_{db} = 10 \log_{10} \bar{A} \tag{4.2}$$

Independently on the values obtained by these formulas, there must be a knowledge that the dB is a relative measure. Since there is no calibration and any reference, this measurement is only relative to itself. This way, it helps to create a relationship between segments or even between different songs in the same program, still it will not be able to conclude how loud is the sound in real life without the proper calibrations.

The values obtained will be negative and the further it distances from zero, the louder the sound gets.

- **Loudness** — The Loudness is the intensive attribute of an auditory sensation in terms of which sounds may be ordered on a scale extending from soft to loud. Loudness depends primarily upon sound pressure but it also depends upon frequency and waveform of the stimulus. The units on the scale of loudness should agree with common experience estimates about the magnitude of the sensation. The measurement of loudness is a significant part of the audio art because the loudness of a sound or noise plays an important role in the reproduction of sound [41].

It is calculated in a similar way as the power, but using the amplitude instead of the square of the amplitudes and also using the ratio to the maximum intensity in a sound.

$$L_{db} = 20\log_{10}\left(\frac{\bar{A}}{A_0}\right) \tag{4.3}$$

Here, loudness will also be represented with a negative value and the closer it is from zero, the louder is the sound.

The use of both power and loudness of the sound is to have two references for the intensity, the former to compare between the analysed songs and the latter to have a comparison in a larger dataset, stored by the Echo Nest.

- **Pitch** — The pitch is a music property that allows the classification of a sound as relatively *high* or *low*. Pitch is determined by the frequency of sound wave vibrations. Although the frequency is a precise scientific unit of measurement, the pitch also has a subjective component which takes into account the relative placement of the frequency within the context of an established tuning system and in relation to other frequencies.

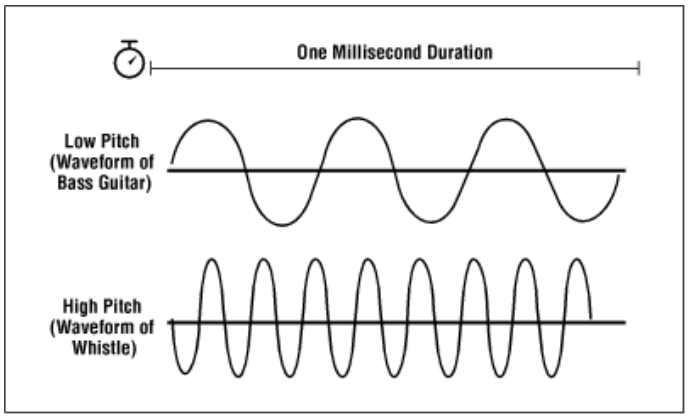

Figure 4.2: Two different waveforms and their approximate frequency rate [1]

A common way to identify the pitch of a sound is to use the MIDI standard to map the frequency to a real number. It will create a linear pitch space in which octaves have size 12, and a semitone has the size of 1, where the A440 is assigned the number 69. In this context, the A440, also known as the A above the middle C or A4, is used as reference. Also, as the Human perception of musical intervals is approximately logarithmic with respect to the base or lowest frequency of a periodic waveform, the perceived interval between the pitches of A220 and A440 is the same as between the pitches of A440 and A880. And so, motivated by this perception, it is used this formula to represent the pitch based on the MIDI standard:

$$p = 69 + 12log_2\left(\frac{f}{440}\right) \tag{4.4}$$

It is possible to visualize in the figure 4.3 where the middle C is located in the piano notes and how the pitch changes according to which notes are played.

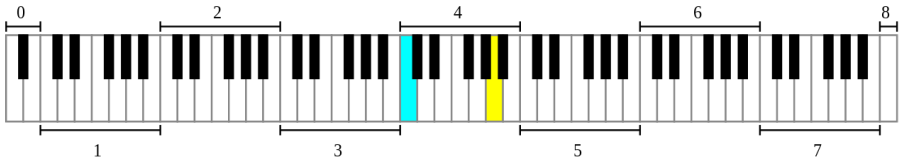

Figure 4.3: Representation of the octaves in a piano. Middle C is marked as the blue note and the A440 or A4 is marked as the yellow note. The pitch increases with the octaves.

In this program, the pitch is computed in a different but similar way. First, it applies a logarithmic-frequency template to the average of the frames together, in order to enhance the harmonic sets. With this, it is able to pick the biggest peak of each frame. Next, it will choose what the most common value is between the retrieved frequency peaks and will use an adaptation of the previous standard formula to calculate the MIDI value corresponding to the dominant pitch in the segment.

Furthermore, by knowing the base pitch is correspondent to the C0 with a frequency of 16.35Hz, it is possible to use the frequency of a wave to calculate the octave and the semitone of a sound. It is calculated by:

$$v = log_2\left(\frac{f}{f_C0}\right) \tag{4.5}$$

, where the integer part of v is the octave and the fractional part is the semitone inside that octave. Because it is represented from 0 to 1, it can be calculated by interpolation from 0 to 11. For example, if this formula is applied to the frequency of 440Hz, the result is 4.75, so the note with that frequency is located in the octave 4 with the semitone being the number 9 that represents the A in the octave.

Although this is particularly important to get dominant semitone within the octave or the dominant octave in the sound, it is not easy for the normal human ear to distinguish semitones without any reference. That would imply that even if there was a large difference of dominant semitones between two songs or segments of a song, this would be barely noticeable, and would have no effect in the outcome of the content generation since it would have no prediction before. This feature is also known as chroma of a sound.

- **Spectral Centroid** — The Spectral Centroid is commonly associated with the measure of the brightness of a sound. An instrument like a tuba or bass guitar will have a lower Spectral Centroid than an instrument like a violin.

  This measure is obtained by evaluating the "center of gravity" using the frequency of the Fourier transform and magnitude information [42]. The individual centroid of a spectral frame is defined as the average frequency weighted by amplitudes, divided by the sum of the amplitudes, as shown by the formula:

$$Spectral\ Centroid = \frac{\sum_{k=1}^{N} kM(k)}{\sum_{k=1}^{N} M(k)} \tag{4.6}$$

  , where the M(k) is the amplitude corresponding to the bin k, which is usually specified as a consecutive non-overlapping interval of a variable.

  In practice, centroid finds this frequency for a given frame, and then finds the nearest spectral bin for that frequency. The centroid is usually a lot higher than one might intuitively expect, because there is so much more energy above (than below) the fundamental which contributes to the average.

  In this feature extractor of this thesis, it calculates the spectral centroid for each frame of the segment and outputs the average of them.

  High values for the spectral centroid usually are due to the high frequency components from sound. And even if the fluctuation of the spectral centroid over time looks stable it does not seem to imply anything for music genre classification and it only depends on what kind of instruments are playing in time [43].

  Although the spectral centroid is usually associated to the brightness of the sound, it is largely used to calculate the timbre of it, also known as the tone color or quality. The timbre allows to distinguish different types of instruments even if they have the same pitch and loudness.

- **Tempo** — The tempo is one of the features that can define a rhythm, along with the intensity and the regularity [44]. Tempo is the speed or pace of a given music, which can also be described as Beats per Minute (BPM). It is one of the most important features that can be extracted from songs, mainly in this thesis, because it is what most people relate it to when listening to music. It is perhaps best described by analogy to the human activities of foot-tapping or hand-clapping in time with music, tasks whose average human listeners are capable of. Despite its apparent intuitiveness and simplicity compared to the rest of music perception, beat tracking has remained a difficult task to define, and still more difficult to implement in an algorithm or computer program [45].

  When describing musical tempo, it is often useful to make a distinction between notated tempo and perceptual tempo. Notated and perceptual tempo can differ in that, for a given

excerpt of music, there is only a single notated tempo while listeners unfamiliar with the score can perceive the tempo to exist at different metrical levels [46]. For some pieces of music, the perceptual tempo is quite ambiguous, while for others it is not. Still, it is often desirable to have a representation of perceptual tempo rather than notated tempo [47].

The primary information required for beat tracking is the onset times of musical events, and this is sufficient for music of low complexity and little variation in tempo. However, when the complexity of the music increases, it becomes more difficult to distinguish the beats from other elements of the music, mainly because the peak-detection is rather sensitive to the threshold parameters and frame window size.

Other main problem of computing the tempo is if the music does not have obvious onsets on the beats, i.e. if there are no drums or a bass playing and keeping the rhythm. In these cases, the tempo must be deduced by the periodic notes or timing the pitch variations.

Another dilemma that still is not easily solved nowadays regarding the tempo is the half-twice BPM issue, especially with live tracking. To explain this issue, first it should be understandable the definition of tatums, which represent the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events. What this issue shows is that even if a song has a fixed BPM, it is possible to listen to the tatums and infer a BPM with twice the value it really has. Even though it was used a half-twice term, the same thing can happen if the subdivisions of each beat are a different number than two.

There are two main problems that emerge with this. First, using the already mentioned definitions, the algorithm might calculate a correct tempo, which is the notated tempo of the song, still when a listener tries to deduce its tempo, his perceptual tempo would be twice the notated tempo. Although the algorithm managed to succeed and get the right tempo, for the user and this thesis, it would seem that the tempo deducted was too slow. In other words, even if two songs have the same tempo, one may seem much faster than the other because of audible tatums between beats and the algorithm would not detect them.

The other problem is the opposite of the one stated previously. It happens when the algorithm treats each tatum as if it was a beat. This way, two songs that have the same BPM and also their perceptual tempo is the same, the algorithm would get it wrong because it would consider each tatum as a beat and would deduce a faster tempo than it should. This may also be caused by the sensitivity issues already mentioned.

These last problems may be easily solved with the user interaction, but when letting the computer automatically calculate it, it is not easy to have a full confidence of the tempo output.

- **Energy** — The energy of a music is a combination of other features in order to obtain a value that can more easily be related with that feeling on the listener. It becomes a very important feature in this matter, mainly due to the fact that it can match with the listener's feelings when listening to the analysed song, even if he has little knowledge in music.

The most important features that are included in the mix to compute energy is loudness and tempo, or a even more specific one that is the duration of each note. It also includes rhythmic regularity, structural changes and the timbre.

Although there are many features included to calculate the energy, it can be more easily explained as a relation between the loudness, or intensity, and the duration of a beat. For two different sounds with the same intensity, if one's bpm is shorter, its energy is higher. The same happens the other way round, if two sounds have the same bpm, if one of them has high intensity, it will also have higher energy. It can, then, be described as the amount of intensity existent in a short time interval.

This energy attribute outputs in a range from 0 to 1, where 1 is the most energetic and 0 is the least.

- **Valence** — The valence is a feature, also in psychology, to evaluate emotions. It is usually described as a negative or positive valence, going from suffering to attractiveness, respectively. A negative valence includes feelings such as anger and fear, and on the other hand a positive valence includes feelings such as happiness and pleasure.

  The valence dimension is inter-correlated with the arousal dimension, which refers to how excited or apathetic the emotion is, ranging from sleepiness or boredom to frantic excitement [48].

  Russell [49] proposed a model of affections based on two bipolar dimensions: pleasant-unpleasant and arousal-sleepy, theorising that each affect word can be mapped into this bi-dimensional space by a combination of these two components.

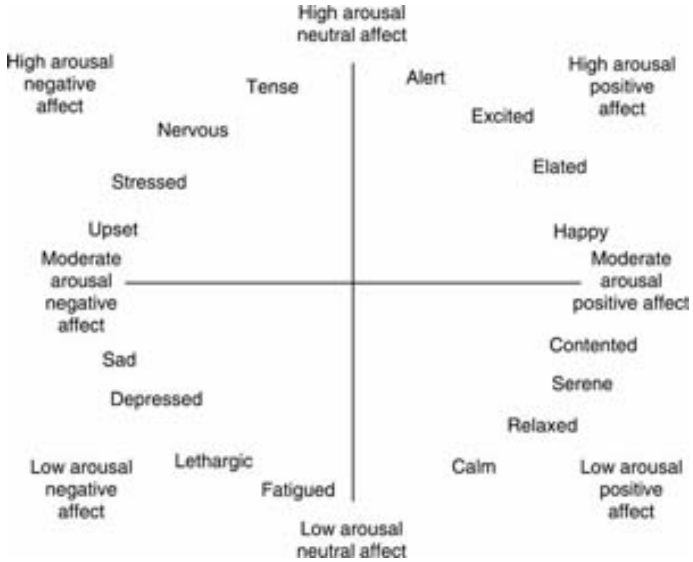

Figure 4.4: Valence-Arousal space labelled by Russell.

In this diagram, it is possible to see how the valence increases from unpleasant feelings to the opposite. The valence, in music, maps the feeling the sound transmits to the listener

into this diagram. In order to do so, there have been plenty of studies in this field to better understand how the feelings can be extracted from sound waves.

Although conclusions on that matter diverge one from another, the main features used to compute a valence value are the tonality of the music, energy, rhythm and harmonics [50]. The tonality is also understood as the main tone of the music described previously as the chroma. The rhythm is a junction of mainly the tempo and beat regularity, and the harmonics are the combination of different musical tones, creating an audible effect. In monophonic music, harmonics are easily observed in the spectrogram. However, it is hard to find harmonics in poly-phony, because many instruments and voices are performed at once.

Having these features, it is possible to compute a valence value that will range from 0 to 1 and it will give an estimated feeling in the valence-space that can be connected to the feeling created by the music in the listener.

This described process makes all the game content be generated before the start of the game, still when using the intensity of the music to increase or decrease the vision of some elements in the game, it is updated in real-time. With this, every frame of the game retrieves the current intensity of the segment of the song playing in the background and maps it in the moment with a little of lag in the synchronization.

There were other music features, like the chroma feature mentioned that could be extracted and used in this thesis. Those features include the Mel-frequency cepstral coefficients that represent a power spectrum of a sound based on the mel-scale and largely used in speech recognition, and others similar. These features were not used mostly because they are hardly, or nearly impossible, to be recognizable by a listener and that would mean that it would have no impact in the content generated to what was expected.

Chapter 5

Mapping Music Features into Game Parameters

In this chapter, a final description is given regarding the way the music features relate to the parameters of the game.

After having the two main modules, the game and the music feature extractor, it is possible to fill out the purpose of this thesis that is relating the two of them. This is achieved using two major ways. One is to use a segment's feature compared to the other segments of the same song and the second is to use that feature compared to a previous standard value.

For the first one, after all the segments of a song were analysed, the program will calculate the minimum and maximum value for each feature between all the segments. With those boundaries established, it is easier to have a ratio of a segment's feature and its impact in the whole song. This will enable to better distinguish segments of the same song, even if compared to other songs that difference should not exist. For example, there is a song that has 4 segments with their tempo ranging from 100-120 bpm, and other song that has all the segments with their tempo of 160bpm. Although the parameters generated based on the tempo should all be "lower" in the first song, using this strategy would make the segment with 120bpm may look similar to the song with 160bpm in order to greater increase the distance between the segments. This is used mainly to enhance the user's fun and be able to better distinguish parts of the same song than make that difference exist only between other songs. If the segment's features are close to each other, they will not be really different otherwise, the one different will have impact in the content generation.

The second way is to control the first one from getting too distanced values. In this one, each feature instead of being compared to the same feature in the other segments, it is compared to a default value that would probably coincide with the listener's opinion. With this, for example, a segment with 60 bpm is always consider a slow part of the music, because it is below the value used by most people as the point that distinguishes a slow from a fast music.

These two ways are used together in order to better evaluate each segment feature using a standard approach and, at the same time, still be able to create divergences inside the same song.

5.1 Mapping Process

The mapping of the features start by first defining what are the dimensions of the map. For that value it is used the pitch of the segment that by making the map bigger, the higher is the sound. The middle tone used in the pitch to distinguish high from low sounds was the A3 with the value of 57 on the standard MIDI values for pitch. This tone was chosen because it is one octave lower than general tuning standard A4, and to use the A4 would not be correct since it is already too high to most songs and too distanced from the middle C, C4. The scale of the map was the first parameter to be assigned, because it will also scale all other parameters to fit in a bigger or smaller map.

Likewise, the pitch is used to define how narrow or wide are the paths. To establish that, it is needed to assign a value to the number of blocks that will be generated in the map. To counter the previous parameter, the higher the pitcher, the more blocks are created, so the paths will be smaller. To define the number of blocks generated, it will not take into account what is the ratio of this segment's pitch regarding the other segments of the same song. It will only use the distance to the A3 tone to define how many blocks are generated. This strategy will allow that the segment with the highest pitch in the song but a low pitch for the listener, will have, after the content generation, a normal sized map with wide corridors, otherwise every small map would have wide corridors and every big map would have narrow corridors.

The next step is to determine the value of the range of the cameras and enemies vision as well as their speed. For the vision's range, the mapping process uses both power and loudness features, giving more emphasis to the latter. It calculates using the two ways described in the section 5, where the standard value is an estimated value in between what is considered a loud music or a quiet music, based on the songs analysed. For both of them, the loudest the music is, the further the enemies and the cameras will be able to see and vice-versa. This aspect is also changed in real-time, which will make that, after a previously defined number of frames, the loudness value of the sound currently played is calculated and then mapped during the gameplay. The resulting parameter value is based always on the first calculated in order to match that segment's position for that feature.

On their speed, the mechanism is the same but it is used the energy feature to define this parameter. The more energetic a segment of the song is, the faster the enemies will walk and the rotation of the cameras will be faster as well. To decide what makes a song more or less energetic without comparing them to each other, it uses the mean value of the energy of songs from a bigger dataset [51], which is 0.5. Because the energy mainly relates the tempo to the intensity, the tempo will be according to the speed/range of each enemy.

In the last steps of the mapping process, it is taken into account the tempo and the numbers of enemies, cameras, traps, items and boxes are defined. All of them have their value related to the tempo, because they also relate among one another. For the tempo, previous studies have shown that listeners tend to prefer a tempo near a resonance of 120 bpm [52]. So, it is based on this timing that it starts being evaluated the difference of each segment's tempo to 120 bpm, where if

positive, the tempo is faster and there will be more elements, and if negative, a slower tempo will cause the opposite. It is largely related with the difficulty since the less number of enemies and cameras, the easier the level gets.

During this process of mapping the tempo into parameters, the program will use the energy of the same segment, together with the tempo, to decide which main elements are to be created. The aim of this is to make the generation of content capable of changing the gamestyle. A segment with low energy and slow tempo will probably have no enemies and a large number of traps and cameras. Following this, other elements will be placed towards this type of level and in the end a slow music will be represented by a slow level, with more stealth and careful moves. On the other side, a fast music, would represent a fast level, with many guards and more shooting. Like this example, it is desired to use the analysis of the music to generate content that would condition the way the user will play the game.

To conclude the mapping process, it is defined the color of the content created. This color will be related to the emotional state transmitted by each segment. Because the emotional state is represented by the Valence value, it is a process of mapping this valence into a color. Colors are for many people an important source of information, yet, colors are also part of a personal preference. Therefore, it is influenced by cultural differences and other factors.

There are some models that map an emotional state to colors, but they all maintain some differences between them making it impossible to define a true model that would fit every one [53]. In this thesis, to better relate a single value from 0 to 1 from each segment's valence to a color, it uses a model based on the traffic light colors, which are worldly accepted as an increase of pleasure from red to green, passing by the yellow. To make it happen, it is used the model to represent colors HSB, and defined the Hue, which is a property of a color, represented in the figure 5.1. It only uses the values that range from red to green, roughly from 0 to 0.4 in this scale.

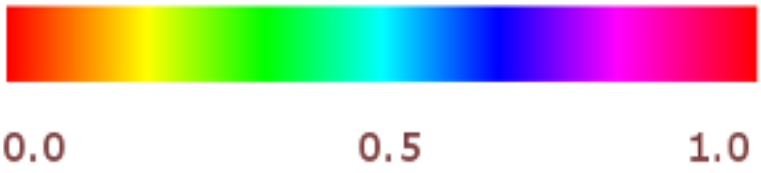

Figure 5.1: Hue Scale

In the model HSB, other than the Hue it is also represented by the Saturation and the Brightness, which are defined by the spectral centroid feature of the music. Still, the main feature that defines a color is the valence of the music.

To better understand the algorithm of mapping the features in this game, it can be visualized in pseudocode. The important parts of the algorithm are the usage of two different values before assigning a definitive one for each parameter and how each feature obtained is used to change different parameters. This way it increases and maximizes the possibility of the player to notice

the changes made into the level based on his song chosen. In the next pseudocode, each *calculate* method used is based in different values for each parameter, but following the structure of the main one.

```
function CALCULATE(feature)
    valueInSegments ← COMPARETOLIMITS(featuresMin, featuresMax)
    valueDefault ← COMPARETODEFAULT(standardValue)
    return FINALVALUE(valueInSegments, valueDefault)
end function


function MAPPINGFEATURES
    featuresMin ← GETMIN()
    featuresMax ← GETMAX()

    for all Segment do
        mapScaleValue ← CALCULATESV("Pitch")
        numberBlocks ← CALCULATEB("Pitch")              ▷ Only use the valueDefault

        visionEnemies ← CALCULATEVE("Loudness","Power")
        visionCameras ← CALCULATEVC("Loudness","Power")

        speedEnemies ← CALCULATESE("Energy")
        speedCameras ← CALCULATESC("Energy")

        color ← CALCULATECOLORINHUE("Valence","SpectralCentroid")
        numberEnemies ← CALCULATENE("Tempo")
        numberCameras ← CALCULATENC("Tempo")
        numberBoxes ← CALCULATENB("Tempo")
        numberItems ← CALCULATENI("Tempo")
        numberTraps ← CALCULATENT("Tempo")

        if CHECKGAMEMODE("Energy","Tempo") then
            APPLYTONUMBERS()          ▷ It only applies to the quantities of each element
        end if

    end for
end function
```

Chapter 6

Experimental Results

In this chapter, the results of the performed experiments used to evaluate the effectiveness of the proposed approach are presented and discussed. The experimental setup is first described, followed by the results of the experiments performed both in the segmentation and in the evaluation of music. Finally, a last approach on the results of the mapping algorithm is described and how it satisfied the creation of two different game modes.

6.1 Experimental Setup

To obtain the following results, they were chosen six different songs that vary in all their features. They were segmented in different parts and each part was analysed and noted. Two of the songs have segments with different states varying inside the same song and the songs left have all the segments similar to each other.

For the segmentation of each song, the sensitivity and the size window were tuned manually in order to gather the most consistent results which are similar to the reality of its segmenting. The criteria used to tune those variables was to resemble the natural breaks expected when listening to a song. For example, in one of the chosen songs, "Wiz Khalifa - See you again ft. Charlie Puth", it was expected to have the segments matching the breaks recognized in the minutes 0:39, 1:14, 1:55, 2:30, 2:53. So, by changing the variables accordingly it was possible to have the most approximate best results with the sensitivity tuned to 1.0 and the window size to 0.7.

In the end, there were used 33 segments in the experience of mapping the features into parameters. These segments cover slow and fast, low and high and also quiet and loud sounds. These were the main criteria when choosing the songs, which are able to cover the most different possibilities while also being recognizable in the segmenting to separate those possibilities.

6.2 Segmenting Results

Each song was segmented in different parts ranging from 3 to 8 and for each one of them it was noted if they correspond to the natural break expected or if they fail to recognize the changes in the music. It was considered that if the program takes 5 seconds or more to recognize a break, it counts as if it failed, while if it was less than 5 seconds, it counts as success with a specified margin error.

After analysing all the segments and compare them to the natural breaks expected in each music, the results are as followed:

Table 6.1: Segmenting Results

Success		Failed	
No Error	With Margin Error	Above Error	Miss
13	12	4	4
25		8	

This table is better translated in the figure 6.1.

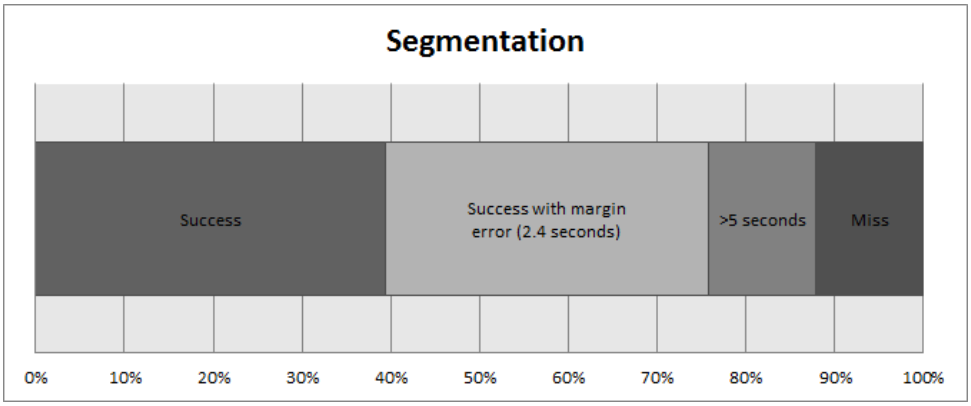

Figure 6.1: Segmenting Results Chart

In this figure it is easily seen that the segmentation of the algorithm can roughly detect 88% of the breaks, with a success of 75%, under an acceptable time of 5 seconds. The mean time a segment is detected, if not in perfect breaks, is around 2.4 seconds while in the ones it fails to detect in time is around 7.15 seconds.

Almost half of the segments detected have been with a small delay or advance in time. This is due to the fact that most of the times a change in the music is not clear enough and the transition itself takes time to happen. Also because sometimes the algorithm may detect few small changes that quickly return to the same state, it will not trigger a break in the segment in the first change it detects. This happens mostly when a song changes its structure while not changing its rhythm.

6.3 Mapping Results

Regarding the mapping process, the resulting parameters were consistent with the pre-analysed features. As it is possible to notice in the figure 6.2, the number of enemies plus the number of cameras, as it happens with the other elements, increases linearly with the tempo. Due to the usage of the adapting algorithm in this process, where it compares what is the impact of the each segment's features to the other segments of the same song, it is possible to visualize the effect created when the tempo is 120 bpms or 180 bpms. Even if the segments have the same tempo, as it shows, some of them will be evaluated as having a bigger or lower bpm, diverging around the linear line.

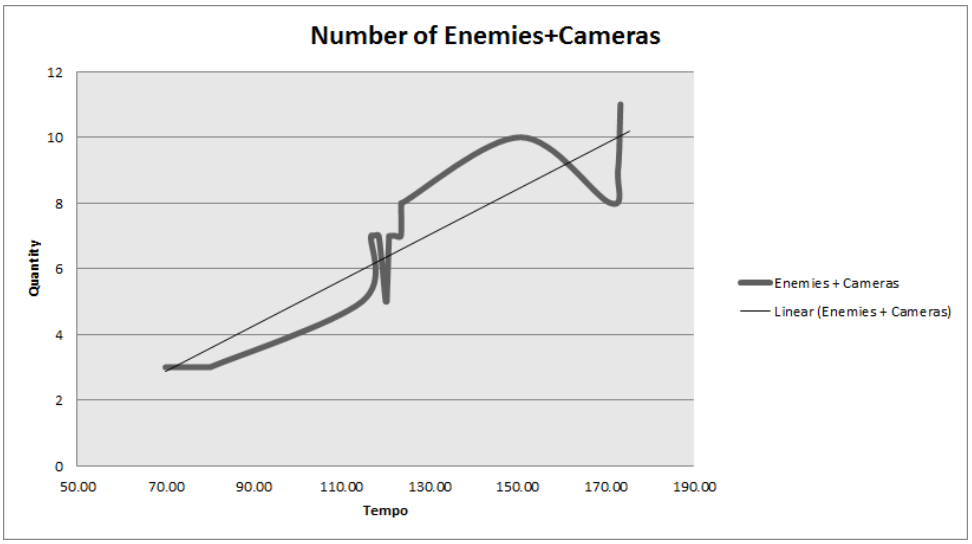

Figure 6.2: Number of Enemies+Cameras generated based on Tempo

The same effect, previously mentioned, also happens when calculating the distance that each enemy or camera will be able to see, using the Loudness and the Power. In the figure 6.3 it is possible to see how the distance increases with the Loudness, but because most of the segments have similar Loudness, from -15 to -5, the algorithm works in order to create a bigger difference between segments so, there are some discrepancies from the linear evolution. Instead, it varies around it.

In both previous figures it can be seen the usage of the most important parameters, as the ones that will mostly change the environment of the level. Both of them are the ones that are easier to look for in changes and the songs chosen could better represent a variation in these aspects, leading to these results.

It is important to notice that in these cases and in the mentioned algorithm it is not affected by other segment's features except its own. For example, two segments of the same song having similar but distinct tempo can be largely separated, but this effect on the tempo will not affect other

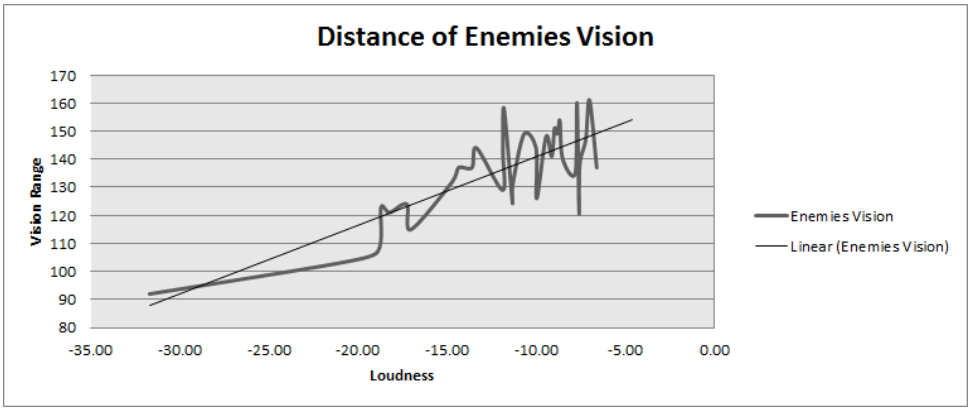

Figure 6.3: Distance of the Enemies Vision based on the Loudness

features even if they are involved. Each feature compares to itself in the other segments, otherwise it would lead to a needlessly wrong increase of other features.

When analysing the results of the mapping, it is possible to notice the presence of the most important component of this thesis that is the capability of the mapping algorithm to somewhat detect and distinguish different game modes using the features of each segment. These game modes are changes in the playstyle of the player due to the content generated. The biggest difference is possible to visualize in the figure 6.4, in which the number of traps created while the tempo is slow is absurdly larger than the number of enemies. As mentioned in section 5, the algorithm uses the energy together with the tempo to distinguish a slow segment and possibly change its gamestyle. As seen in this picture, those changes are explained in levels with no enemies and full of traps, this way it requires the player to crouch and avoid the traps and the cameras carefully but with time to act. On the other hand, a fast paced level will have more enemies than traps, requiring the player to act accordingly to the enemies' path and move quietly.

Although it seems as a relation between the tempo and what is considered a slow or fast music, because of the problems related to the calculation of the tempo, it is important to use the energy to better analyse and detect those speed differences. Even in the previous figure it was possible to see that between the slow tempo, there is still a usage of the normal game mode.

In the next figure it is shown how the energy and the tempo are related to for each segment.

It is feasible to imagine a line that separates the two game modes based on the relation Energy-Tempo and that the algorithm should follow. As stated before, this is an important aspect of the mapping process, since it will allow, based on the most recognizable music features, to address a different type of level without changing only the quantity or difficulty of the level by scaling all the elements in the game.

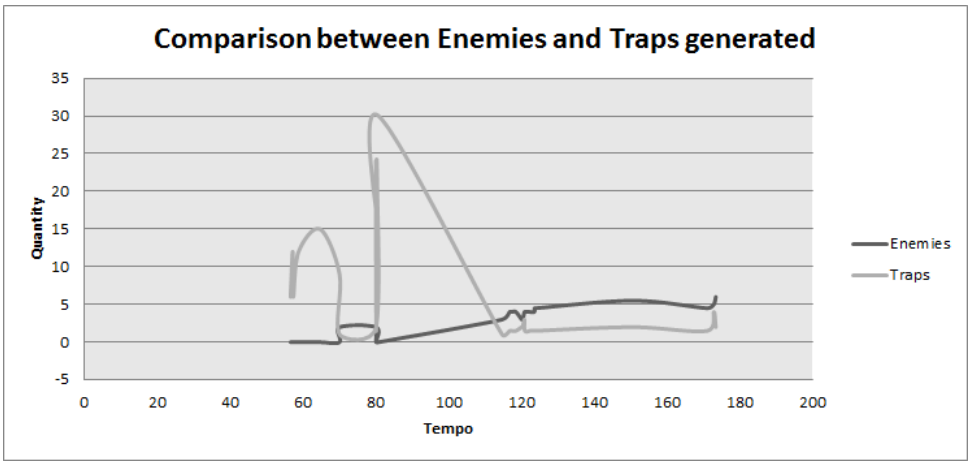

Figure 6.4: Comparison of the number of Enemies and Traps generated based on the tempo and on the Game Mode

6.4 Methods Comparison

With these results it is possible to compare this method of processing the features and map them into game parameters with other possible methods.

The easiest change was to use the energy for the number of elements created instead of tempo. Because the energy also depends on the tempo, at first the changes would seem the same, still, after some attempts it could be possible to notice that the game modes would disrupt the existence of easier levels compared to harder levels. This is due to the fact that the energy is used with the tempo to decide which elements to generate, but the energy does not have a high factor of misleading as the tempo. Also, if each element's quantity was based on the energy, this quantity could never be low since it would already change game mode.

Another method applied at first was to use the spectral centroid feature in the color choosing along with the valence. However, the numbers obtained by the spectral centroid, even if they could resemble as the sound is different and show its *brightness*, the changes were not easily recognizable by the listener, and could go against the valence values, since to say a sound shines or that it is different, does not always mean that it is more or less satisfying. So, in order to maintain the parameters generated easily recognizable to the normal listener and user, it was preferred to keep the colors affected only by the valence and have the spectral centroid affect the other values of the color as the saturation and brightness.

It was considered and tried to use the intensity of the sound, represented by the power and loudness, to have an impact in the speed of the elements in the game. Although it worked and the changes in the game could be visualized, it was not accepted for the same reasons as the previous method, since it is implied that the user will not have knowledge in music and that the user will react easily to the changes that are similar to what he listens to. So, when applying a speed to an element of the game, the most logic way is to use the speed of the music as well. Because the tempo can mislead and it was already used to generate quantities, it is better to apply the energy

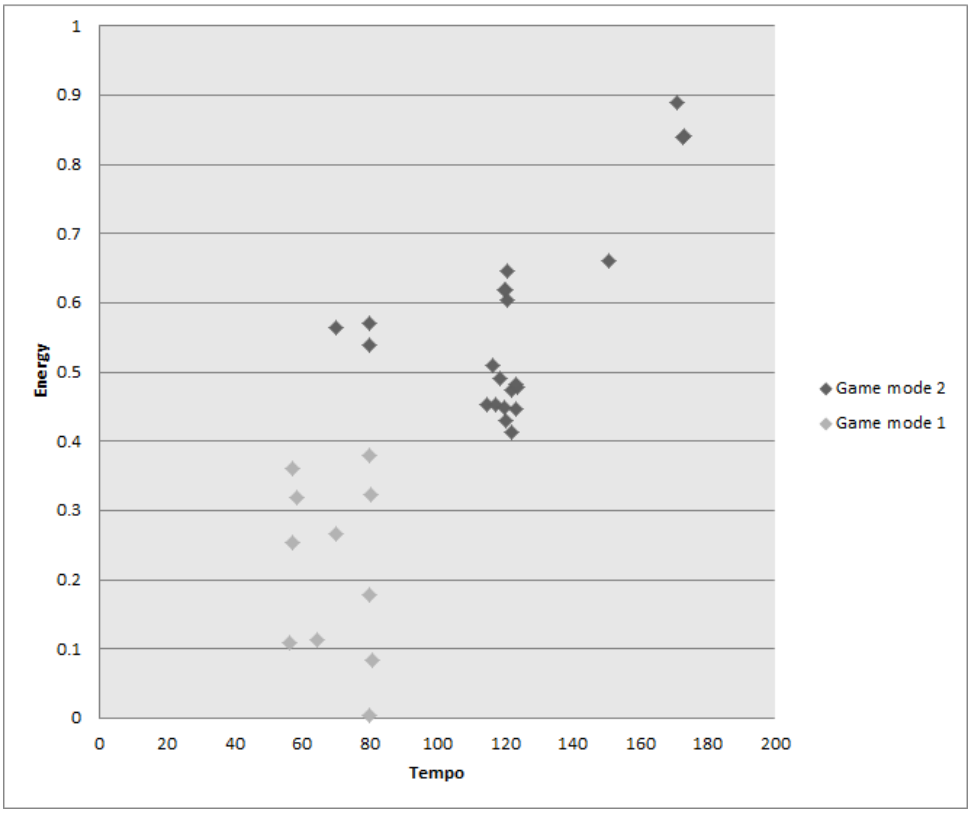

Figure 6.5: Comparison of the Energy and the Tempo for each segment and its changes on the Game Mode

here, since it can also represent the pace of the sound and at the same time it will make more parameters less dependent on the same feature. Another reason and also being in agreement with the listener and user is that it is common to relate the volume of a sound, which is connected to the intensity, to triangles with increasing intensity from one corner to the others. That could be applied in game in the visions of the enemies and cameras and making it easily perceptible that an increase in the sound intensity would lead to an increase of the distance of their sight. Finally, because it was intended to apply the real-time changes on the vision of those elements, it should use a changing feature throughout the music. Since all the other features change only a little or abruptly, it could not be applied with this effect, so the usage of the intensity here was a must.

As already stated in the section 5, it was planned to use the chroma of a sound to change the dimensions or the number of blocks created, but it was impracticable since it is highly difficult for a normal listener to recognize the differences in the music, and so it would make it seem randomly generated. As a result of this, the pitch of each segment was responsible for assigning a value to these parameters. Even though it is possible to distinguish two different sounds if their pitch is too different, it is hardly distinguishable if they are not that different. This would make the pitch the least important feature along with the spectral centroid, because they were the features less recognizable by the listener. To solve these problems, the best way was to apply these features on

the least important parameters as well and that is why the pitch affects the dimensions and could not be used to assign quantity, vision or speed, since it would not have any comparison between the visual impact and the music listened.

The last changes to the method of mapping based on the results obtained were about the game modes detected. For that it was considered to use the most perceptible features: Tempo, Intensity, Valence and Energy, since it relates both Tempo and Intensity. Since there are four main features, it was possible to create at least four different game modes based on how each feature relates to one another. The experiments of those game modes worked, still there was the problem that it was not always understood why a game mode is applied and not other, if they were all seemingly different. To better overpass those problems, the mapping process used the two game modes implemented and two more that would use only cameras or only enemies in between. For those game modes it already used the intensity along the energy and tempo used now. Despite the effectiveness and perceptiveness in noticing why each game mode was chosen, it led to another major problem in the matter of this thesis. The usage of more different game modes would reduce the impact that the music chosen would have, because it could also lead to intermediate game modes that would not imply a distinctive sneaky game style or a fast paced sneaky-shooting style. This ends up being a limitation in this thesis since it could be overcome with a deeper analysis and a game with more different features that could be visually easier to distinguish. In a best case scenario it could use all of the features related to each other to create more game modes, but it was not feasible to do it in this thesis' limited time.

6.5 Final Results

Once every song used was segmented and each of the segments analysed using the chosen method and algorithm, the results obtained are summed up as followed in the Table 6.2 in relation to the two game modes.

Table 6.2: Game mode results

Songs			Segments	
Mostly Game mode 1	Mostly Game mode 2	Both	Game mode 1	Game mode 2
1	2	3	11	22
6			33	

In this table it is possible to conclude that, because of the process used (5.1), the same song can have both game modes in different segments if the differences between them are perceptible and since the biggest changes are connected to the most important features, the player will understand these changes.

As shown in this experiment, depending on the song, the generation of the content will be different but not always linear with the increase or decrease of each music feature. A deeper analysis can oversee another importance of this process, as it can negate and limit the attempt of disrupting the gameplay. Although the algorithm already works with a limit to each element

created, the use of a song whose features are a possible minimum could lead to an empty level, with no content to be generated. This algorithm prevents this as it works on the opposite direction with different game modes, changing different parameters when a feature decreases or when it increases.

Chapter 7

Conclusion

The conclusion of this research is presented in this chapter. The initial defined research questions are answered according to the obtained results as well as it is given an outlook on open questions for future research in this area.

7.1 Goals

A1. Is it possible to generate a stealth game procedurally only based on input parameters?

To create any game content procedurally based on input parameters is always important to keep all the factors in the game or the elements included in the content possible to be changed by the parameters defined. If everything is already parametrized, it will come down to the decision making about the game and polish what the generation algorithm will be able to do. Since every element can be built inside boundaries already established, the content of the game created is what will make the type of game that it is going to be played.

As for stealth games, creating levels using these methods will excel due to the fact that the fun that a stealth game imposes on the player decreases abruptly after the first time each level is passed. This is mainly because this type of games is seen as a challenge and the priority is to pass that challenge at least once. If it is possible to keep the levels being generated procedurally, it will keep the player triggered, as well as it allows the player to keep changing his levels according to the difficulty he expects or wants.

B1. What are the most important music features to be used in game content generation?

Although there are a lot of features that can be extracted from music when it comes down to the analysis of a sound wave, many of them are imperceptible to the listener as it is, for example, the frequency of the wave. Even if many of the music features can be detected and used by applying

the proper music knowledge to make them more user friendly, they still can end up being non-relevant when applied to a context where the players will not have that music knowledge and will not be able to detect the difference. Example of that is using the frequency of the sound wave to extract the tonality of the sound. It is easier to see changes affected by the tonality than by the frequency, still both of them end up being undetectable for most of the users.

Following this idea, the most important features are the ones that can easily be detected by the user, which are the tempo and the intensity. In games where the music chosen will weight on the content generated, these mentioned features are the ones that people will be looking for when choosing the song. However, there are more that can also be used, but they are the result of combining other extracted features in order to make a solo feature that can be detected by the player, such as the energy and the valence. Adding these to the initial ones, it is possible to have a kit of the features that will mostly be the ones that the player will be looking for in the game. Also, as secondary features to be used are the pitch, the tonality and the regularity.

This is confirmed by the results given in section 6.3, the tempo and the loudness/intensity vary among all the songs, since they are the features people look for and the easiest for a listener to detect.

C1. How viable is the usage of music features in the generation of game content?

As seen in the literature review (2.1.1), generate game content procedurally is important because it can keep the player triggered while making the game less annoying. Being able to allow the player to opt for the experience he wants when playing the game is something that is considered the future of videogames.

When using the music features to do so, as stated in the section 6.4, it is relevant to combine the most important features with the most important elements in the game. That is what will make the player notice the difference he can make with his choices and by mixing some of those features, a big role can be played in the evolution of the game design, while keeping the player aware that it is his song chosen that generated the end result.

The fact that some songs can look different for the player but end up having the same features, as seen in the given results (6.3), allied with the problem that many of the features cannot be easily or can even be wrongly recognizable, depending on the song, brings an issue difficult to overpass since the world of music is vast and is nearly impossible to perfectly create the right environment for each song. And if the connection between the player and the music chosen is not created inside the game and he cannot see the effects of his choice, the game will not have the same impact.

Concerning this connection created with the player, this will be where the usage of music excels, mainly due to the fact that many people create a special bound with the music they listen to and feel as if the music expresses their feelings and desires. If it is possible to use that in the game they are going to play, that is what makes using music unique and different from using other input sources.

In this thesis, to relate the music features with the game, the process follows an algorithm

reported in section 5. That algorithm will evaluate each feature from the segment of the song in two distinct ways, which are: comparing it to the other segments and also comparing it to a standard common value. This way, it is possible to distinguish segments in the same song, creating a bigger diversity throughout the levels generated, and at the same time the respective level will have its content similar to what a listener would expect it to. After the evaluation of each feature, it will proceed to assign values to the game parameters, each one depending on one or two features, and together create the content to the level.

C2. Is it possible to change the gamestyle based on the content created?

Given the results obtained (6.3), it is possible to see that combining some features can decide where a game should change its generation algorithm, described in section 3.5, and consequently change gamestyle according to the content created. That will only work if the game type initially designed is not much linear, for example, the content generated in a simple racing game can change the difficulty and the environment of the game, but hardly changes what the player has to do to keep racing and avoid obstacles until the end.

In this thesis, this was achieved by changing the way the respective level is played. Although the ultimate goal keeps being the same, there are different ways to reach the end of the level. One of the game modes activated requires a stealth game more focused on the player's movement and being able to check every step because there are traps around. The other game mode already gives the player more mobility in exchange of having more difficult elements to overpass. It brings out a stealth game more focused on eliminating or avoiding the potential threats to keep going.

7.2 Future Research

Although the proposed approach was able to prove it is possible to generate game content procedurally based on music, there are still enhancements that could be made to properly increase the performance of the game as well as increase the precision of the segmentation process along with the analysis of the features. The game developed in this thesis is enough to run the tests that it was submitted to, but in order to be completely playable and to improve in further and more demanding tests, some optimizations mainly in the rendering algorithms would lead to better performances overall.

Regarding the segmentation process, some future changes that could be included would be algorithms that could possibly detect repeated sequences and with that be able to avoid breaks among equal segments where they could be merged together. If that algorithm was implemented it was possible to increase the sensitivity of the segmentation process and knowing that repeated parts would be merged together instead of having multiple small ones.

When it comes down to the core of this thesis, the main enhancements that are already being researched and in the future will be easier to implement are the improvements in the music feature

extractions. Mostly, being possible to analyse thoroughly the tatums and the beats of a sound and better detect what the tempo of the song is, would greatly enhance the desired effect of using this feature in games.

An interesting topic to look for in the future based on the work developed during this thesis would be to allow the game to change the sound while it is being played. To visualize changes in real time, both in the music and in the game while they are dependent on each other, would be an interesting study in which features would the player focus on.

Other interesting topic based on the results of this thesis would be to study what better combinations could be made of music features in order to better correlate with a listener's perception and feelings. Working with solid features that could be easily rated by the player would change the issue on how to map "meaningless" features into the game.

References

[1] J. Beggs and D. Thede, "Chapter 2. the science of sound and digital audio." http://docstore.mik.ua/orelly/web2/audio/ch02_01.htm, 2001.

[2] J. Juul, *A casual revolution: Reinventing video games and their players*. The MIT Press, 2012.

[3] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Affective Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 147–161, 2011.

[4] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, "Procedural content generation for games: A survey," *TOMCCAP*, vol. 9, no. 1, p. 1, 2013.

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

[6] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 241–248.

[7] M. Naumenko, "Tomáš dvořák interview: Beauty beneath the rust and dirt." http://game-ost.com/articles.php?id=494&action=view, 2010.

[8] K. Collins, "An introduction to procedural music in video games," *Contemporary Music Review*, vol. 28, no. 1, pp. 5–15, 2009.

[9] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, M. Slaney, *et al.*, "Content-based music information retrieval: Current directions and future challenges," *Proceedings of the IEEE*, vol. 96, no. 4, pp. 668–696, 2008.

[10] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," 2010.

[11] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," 2011.

[12] J. Togelius, N. Shaker, and M. J. Nelson, *Introduction*. Springer, 2014.

[13] A. C. Nuno Barreto and L. Roque, "Computational creativity in procedural content generation: A state of the art survey," 2014.

[14] C. Crawford, "Process intensity," *The Journal of Computer Game Design 1*, vol. 5, 1987.

[15] A. Doull, "The death of the level designer," 2008.

[16] F. C. Pereira, "Creativity and artificial intelligence a conceptual blending approach." http://www.ebrary.com, 2007.

[17] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, *Procedural Content Generation: Goals, Challenges and Actionable Steps*, vol. 6 of *Dagstuhl Follow-Ups*, pp. 61–75. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[18] D. M. D. Carli, F. Bevilacqua, C. T. Pozzer, and M. C. d'Ornellas, "A survey of procedural content generation techniques suitable to game development," in *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*, pp. 26–35, IEEE.

[19] Y. Takatsuki, "Cost headache for game developers." news.bbc.co.uk/2/hi/business/7151961.stm, 2007.

[20] A. Jordan, D. Scheftelowitsch, J. Lahni, J. Hartwecker, M. Kuchem, M. Walter-Huber, N. Vortmeier, T. Delbrugger, U. Guler, I. Vatolkin, and M. Preuss, "Beatthebeat music-based procedural content generation in a mobile game," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pp. 320–327.

[21] M. Toy, G. Wichman, K. Arnold, and J. Lane, "Rogue," *Computer Science Research Group, UC Berkeley*, 1980.

[22] G. Wichman, "A brief history of rogue." http://www.wichman.org/roguehistory.html, 1997.

[23] M. B. Loguidice and B., "The history of rogue: Have @ you, you deadly zs." http://www.gamasutra.com/view/feature/4013/the_history_of_rogue_have_you_.php, 2009.

[24] M. Nitsche, C. Ashmore, W. Hankinson, R. Fitzpatrick, J. Kelly, and K. Margenau, "Designing procedural game spaces: A case study," *Proceedings of FuturePlay*, vol. 2006, 2006.

[25] M. M. Stern and Andrew, "Procedural authorship: A case-study of the interactive drama façade," *Digital Arts and Culture: Digital Experience: Design, Aesthetics*, vol. Practice (DAC 2005), 2005.

[26] G. N. Yannakakis, "Game ai revisited," 2012.

[27] C. Browne, G. N. Yannakakis, and S. Colton, "Guest editorial: <newline/>special issue on computational aesthetics in games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 3, pp. 149–151, 2012.

[28] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 252–259.

[29] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pp. 111–118.

[30] S. Colton and C. Browne, *Evolving Simple Art-Based Games*, vol. 5484 of *Lecture Notes in Computer Science*, book section 32, pp. 283–292. Springer Berlin Heidelberg, 2009.

[31] M. Hull and S. Colton, "Towards a general framework for program generation in creative domains," *Programme Committee and Reviewers*, p. 137, 2007.

[32] K. Sims, "Evolving 3d morphology and behavior by competition," *Artificial life*, vol. 1, no. 4, pp. 353–372, 1994.

[33] N. Gillian, S. O'Modhrain, and G. Essl, *Scratch-Off : A Gesture Based Mobile Music Game with Tactile Feedback*. 2009.

[34] M. Csikszentmihalyi and M. Csikzentmihaly, *Flow: The psychology of optimal experience*, vol. 41. HarperPerennial New York, 1991.

[35] A. Rollings and E. Adams, "Fundamentals of game design," *New Challenges for Character-Based AI for Games. Chapter 20: Artificial Life and Puzzle Games. Prentice Hall*, pp. 573–590, 2006.

[36] M. Ian, "Artificial intelligence for games," 2006.

[37] R. B. Games, "2d visibility." http://www.redblobgames.com/articles/visibility/, 2012.

[38] R. van der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 1, pp. 78–89, 2014.

[39] A. Patel, "Introduction to a*." http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html, 2015.

[40] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A tutorial on onset detection in music signals," *Speech and Audio Processing, IEEE Transactions on*, vol. 13, no. 5, pp. 1035–1047, 2005.

[41] H. F. Olson, "The measurement of loudness," *Audio Magazine*, pp. 18–22, 1972.

[42] U. Nam, J. O. Smith III, and J. Berger, "Automatic music style classification: towards the detection of perceptually similar music," *Proceedings of Japanese Society of Music Cognition and Perception, Fukuoka, Japan*, 2001.

[43] N. Unjung, "A short study on music classication." https://ccrma.stanford.edu/~unjung/AIR/final4web.pdf, 2001.

[44] D. Liu, L. Lu, and H. Zhang, "Automatic mood detection from acoustic music data," in *ISMIR*.

[45] S. Dixon, "Automatic extraction of tempo and beat from expressive performances," *Journal of New Music Research*, vol. 30, no. 1, pp. 39–58, 2001.

[46] M. F. McKinney and D. Moelants, "Deviations from the resonance theory of tempo induction," in *Proc. Conference on Interdisciplinary Musicology*, 2004.

[47] M. F. McKinney and D. Moelants, "Extracting the perceptual tempo from music.," in *ISMIR*, 2004.

[48] M. A. Nicolaou, H. Gunes, and M. Pantic, "Continuous prediction of spontaneous affect from multiple cues and modalities in valence-arousal space," *Affective Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 92–105, 2011.

[49] J. A. Russell, "A circumplex model of affect.," *Journal of personality and social psychology*, vol. 39, no. 6, p. 1161, 1980.

[50] B.-j. Han, S. Ho, R. B. Dannenberg, and E. Hwang, "Smers: Music emotion recognition using support vector regression," 2009.

[51] J. Sundrams, "Danceability and energy: Introducing echo nest attributes." http://runningwithdata.com/post/1321504427/danceability-and-energy, 2010.

[52] D. Moelants, "Preferred tempo reconsidered," in *Proceedings of the 7th international conference on music perception and cognition*, pp. 580–583, Causal Productions Adelaide, 2002.

[53] N. A. Nijdam, "Mapping emotion to color," *)ˆ(Eds.):'Book Mapping emotion to color'(2009, edn.)*, pp. 2–9, 2009.

[54] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 653–668, 2005.

[55] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead, "Pcg-based game design: enabling new play experiences through procedural content generation," in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, p. 7, ACM, 2011.

[56] M. Pichlmair and F. Kayali, "Levels of sound: On the principles of interactivity in music video games," in *Proceedings of the Digital Games Research Association 2007 Conference" Situated play*, Citeseer.

[57] M. Scirea, M. J. Nelson, and J. Togelius, "Moody music generator: Characterising control parameters using crowdsourcing," in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pp. 200–211, Springer, 2015.

[58] J.-J. Aucouturier, F. Pachet, and M. Sandler, ""the way it sounds": timbre models for analysis and retrieval of music signals," *Multimedia, IEEE Transactions on*, vol. 7, no. 6, pp. 1028–1035, 2005.

Appendix A

Song Analysis

In the figure A.1 are detailed the results of the analysis that was run for each segment.

		Features								Parameters												Segmenting		Game Mode
		Segment	Power	Pitch	SpecC	Tempo	Loudness	Energy	Valence	Scale	Blocks	Enemies	Cameras	Items	Boxes	Traps	EVR	CVR	Espeed	Cspeed	Color	Success	Difference	0 = Normal / 1 =Traps
Wiz Khalifa – See you again	Sensitivity: 1.0 Window: 0.7	1	-22.00	65.25	718.8	80.70	-17.02	0.084	0.66	2	165	0	16	6	0	60	115	115	0.017	434	green	Y	0	1
		2	-19.00	36	383.7	80.10	-7.70	0.571	0.72	1	56	2	1	7	1	2	160	160	0.035	634	green	Y	0	0
		3	-22.60	36	410	70.03	-7.80	0.564	0.534	1	56	2	1	6	1	1	135	135	0.035	630	green/yellow	Y	3	0
		4	-20.46	36	626	70.00	-10.00	0.266	0.59	1	56	0	2	2	0	9	144	144	0.024	533	green	Y	4	1
		5	-21.77	36	320	80.00	-7.00	0.54	0.53	1	56	2	1	7	1	2	161	161	0.035	645	green/yellow	N	0	0
		6	-21.60	50	2167	80.00	-8.50	0.38	0.11	2	155	0	12	5	0	48	140	140	0.028	592	red	Y	0	1
Dick Dale – Miserlou	Sensitivity: 1.0 Window: 0.25	1	-22.38	45.49	940.8	171.01	-6.59	0.89	0.666	2	152	9	7	19	5	3	137	137	0.041	757	green	Y	0	0
		2	-21.87	47.25	790.1	173.34	-7.55	0.841	0.925	2	153	12	10	27	6	4	138	138	0.033	600	green	Y	2	0
		3	-22.33	42.5	816.1	172.77	-7.58	0.84	0.815	1	60	5	4	12	3	4	121	121	0.033	587	green	Y	2	0
Franco Micalizzi – Trinity	Sensitivity: 1.0 Window: 0.5	1	-21.90	49.5	1095	57.27	-14.35	0.2544	0.668	1	65	0	0	1	0	6	137	137	0.02	533	green	Y	0	1
		2	-21.60	54.99	964.6	122.26	-8.96	0.475	0.919	2	159	8	6	22	5	3	151	151	0.033	673.9	green	Y	1	0
		3	-21.30	50.2	923	122.26	-10.67	0.413	0.75	2	155	8	6	22	5	3	149	149	0.03	633	green	Y	4	0
		4	-21.44	58	1.034	123.54	-8.65	0.448	0.858	2	161	9	7	22	5	3	153	153	0.032	656.14	green	Y	3	0
		5	-21.80	50.25	807.7	123.75	-9.44	0.4788	0.862	2	155	9	7	22	5	3	148	148	0.033	675	green	Y	0	0
		6	-24.80	49.5	2014	56.55	-21.07	0.11	0.28	1	65	0	0	1	0	6	103	103	0.018	442.1	orange	Y	0	1
Mozart – Turkish March	Sensitivity: 0.89 Window: 0.42	1	-23.76	70.75	600.2	114.66	-18.90	0.454	0.756	2	169	6	4	15	3	2	107	107	0.03	648	green	Y	0	0
		2	-20.80	56.99	564.3	58.73	-18.70	0.32	0.97	2	160	0	0	3	0	24	123	123	0.027	571.22	green	Y	0	1
		3	-18.28	56.99	483.3	118.44	-11.80	0.49	0.89	2	160	8	6	21	5	3	158	158	0.03	672	green	Y	0	0
		4	-21.50	56.99	586.9	123.48	-17.26	0.483	0.963	2	160	8	6	22	5	3	124	124	0.03	665	green	N	0	0
		5	-21.10	70.75	530.5	116.68	-13.60	0.51	0.87	2	169	8	6	20	4	3	137	137	0.03	682	green	Y	0	0
		6	-21.42	71	588.9	57.12	-18.24	0.36	0.974	2	169	0	0	3	0	24	121	121	0.03	591	green	Y	0	1
		7	-20.02	57	518.4	117.34	-13.34	0.453	0.748	2	160	8	6	21	4	3	144	144	0.03	647.3	green	Y	3	0
		8	-23.64	567	1408	64.65	-11.88	0.113	0.26	2	160	0	4	3	0	30	129	129	0.02	442	orange	Y	2	1
Mariah Carey – I wanna know what love is	Sensitivity: 1.2 Window: 0.57	1	-25.40	61	1134	80.16	-31.70	0.005	0.46	2	163	0	12	4	0	48	92	92	0.016	420.9	yellow	Y	0	1
		2	-19.48	37.75	892.8	80.03	-14.70	0.178	0.3	1	57	0	2	2	0	9	132	148	0.02	497.9	orange/yellow	Y	1	1
		3	-21.19	46	548.5	80.60	-11.86	0.323	0.4	1	56	0	4	3	0	15	143	143	0.026	562	yellow	N	0	1
		4	-21.89	84	1295	150.66	-7.21	0.66	0.66	2	178	11	9	26	6	4	147	147	0.037	712	green	N	0	0
The Doors – Roadhouse Blues	Sensitivity: 0.95 Window: 0.36	1	-22.50	42.49	1006	120.20	-9.96	0.619	0.902	1	60	3	2	8	2	2	126	126	0.035	683.7	green	N	0	0
		2	-21.45	42.75	974.3	120.78	-8.80	0.604	0.875	1	60	4	3	11	2	3	149	149	0.035	669.6	green	N	0	0
		3	-21.87	76.25	1156	120.88	-9.12	0.646	0.96	2	173	8	6	22	5	3	141	141	0.037	709.2	green	Y	2	0
		4	-21.89	42.75	774.4	120.08	-11.33	0.449	0.854	1	60	3	2	8	2	2	125	125	0.026	523.7	green	N	0	0
		5	-21.49	42	804.4	120.28	-11.31	0.43	0.862	1	60	3	2	9	2	2	131	131	0.025	506	green	Y	2	0
		6	-21.14	42.75	1.038	120.00	-8.69	0.619	0.87	1	60	3	2	7	2	2	154	154	0.035	684.3	green	N	0	0

Figure A.1: Detailed description of the results of the analysis