Reinterpretation of music according to visual cues in virtual spaces

Simon Cutajar

Supervisors: Dr. Julian Togelius, Dr. Mark Nelson

Department of Games

IT University of Copenhagen

3rd June 2013

Submitted in partial fulfillment of the requirements for the degree of M.Sc.

Department of Games

Declaration

I, the undersigned, declare that the dissertation entitled:

Reinterpretation of music according to visual cues in virtual spaces

submitted is my work, except where acknowledged and referenced.

Simon Cutajar

3rd June 2013

Acknowledgements

First of all, I'd like to thank my supervisors Dr. Julian Togelius and Dr. Mark J. Nelson for the guidance they gave me while I was working on the project. I'd also like to thank Dr. Alessandro Canossa, Dr. Miguel Sicart, and Antonis Liapis for their helpful suggestions while working on the project.

I'd like to thank my friends (both my colleagues at the IT University of Copenhagen as well as old friends from Malta) for bearing with me while I worked on my thesis and for providing relevant feedback and criticism to the project. Special thanks to Ioana Marin for creating the 3D environment used in the project.

Lastly, I'd like to thank my family for supporting me through the 2 years of my Masters and for always believing in me. Special thanks to my mother Isabelle Galea, and to Nina Croitoru, for proofreading my thesis.

Abstract

Digital games rely on precomposed music, usually associated with certain moods or feelings that the composer or game designer wishes to express in particular sections of the game. By removing precomposed music and replacing it with procedurally generated music, we can have the music change according to different cues in the environment, thus creating a fresh, dynamic experience.

The aim of this paper is to demonstrate an algorithm that is capable of continuously creating procedural music for game environments in a real-time environment. It also demonstrates the feasibility of an experimental but flexible system that combines the procedural generation of music using 1D cellular automata with music→colour mappings taken from already existing literature. A third aim is to try and recreate a synaesthetic experience for non-synaethetes.

We have managed to build a simple system that demonstrates a continuous algorithm that creates music in a 3D environment, based on the surroundings. We evaluate the system by seeing which scales and colours are chosen in certain areas of our environment.

Contents

1. Introduction **1**
1.1 Motivation 1
1.2 Goal 2
1.3 Technologies Used 2
1.4 Project Structure 3

I Literature Review **4**

2. Procedural Music **5**

3. Music Generation **7**
3.1 Music Generation 7
3.2 Music Generation Using Cellular Automata 10

4. Mappings **14**
4.1 Music→Colour Mappings 14
4.2 Other Mappings 18

II Methodology **21**

5. Design and Implementation **22**
5.1 *MIDI-dot-NET* 22
5.1.1 MIDI Messages 22
5.1.2 Note Values 23
5.1.3 Time Signatures 23
5.1.4 Scales 24
5.1.5 Instruments and Channels 25
5.2 Generating Music Based On The Virtual World 25
5.2.1 Music→Colour Mappings 25
5.2.2 Extracting Colours from the Viewport 26
5.2.3 Generating Music Based On Character Speed 27
5.3 Cellular Automata 27

5.3.1 System Architecture 27
5.3.2 *GenerativeMusic* Class 28
5.3.3 *Conductor* 28
5.3.4 Instruments as Cellular Automata 28
5.4 Prototype 33
5.4.1 Game Environment 33
5.4.2 Instruments Used 33

6. Results **37**
6.1 *A Painted Picture of the Universe* by Roy de Maistre 37
6.2 Coloured Buildings 40
6.3 Graffiti 42

7. Future Work **44**
7.1 Improving the Music Generation Algorithm 44
7.2 Improving the Mappings 46

8. Conclusions **48**
8.1 Summary of Work 48
8.2 Revisiting the Goals 49
8.3 Conclusions 49

III Appendix **50**

A. Music Generation **51**
A.1 List of Scales 51
A.1.1 Major Scales 51
A.1.2 Natural Minor Scales 51
A.1.3 Harmonic Minor Scales 52
A.1.4 Melodic Minor Ascending Scales 53
A.1.5 Major Blues Scales 53
A.1.6 Arabic Scales 54
A.1.7 Phyrgian Dominant Scales 54
A.1.8 Lydian Dominant Scales 55

B. Music→Colour Mappings **56**
B.1 Music→Colour Mappings 56
B.2 Colour→Key Mapping 60
B.3 Scientific Music→Colour Mapping 61
B.4 Interval→Colour Mapping 61
B.5 Quantized Colours 62

References **63**

List of Figures

3.1 Result of a Class 4 Cellular Automata running with Rule 110 . . . 11
3.2 *Eutérpê Melōidía* . . . 12

4.1 Different colour scales . . . 14
4.2 A rendition of Scriabin's *tastiéra per luce* . . . 15
4.3 De Maistre's colour harmonizing chart . . . 16
4.4 Visual music paintings . . . 17
4.5 Plutchik's Wheel of Emotions . . . 19
4.6 Different ways of playing music depending on different emotions . 20

5.1 TimeSignature Class Diagram . . . 23
5.2 Mapping Class Diagram . . . 25
5.3 Converting Colours from the Viewport to Scales . . . 26
5.4 Music Generation System Architecture . . . 27
5.5 *Conductor* Class Diagram . . . 28
5.6 *Rule* Class Diagram . . . 29
5.7 The Output of a Cellular Automaton Rule . . . 30
5.8 *AutomataSettings* Class Diagram . . . 30
5.9 Creating a never-ending automaton . . . 31
5.10 Filling a measure . . . 32
5.11 Musical interpretation of the cellular automata . . . 32
5.12 Main areas of the city . . . 35
5.13 Prototype Screens . . . 35
5.14 City Render . . . 36

6.1 Test 1: Viewing *A Painted Picture of the Universe* by Roy de Maistre 37
6.2 Test 1: Analyzing Results . . . 38
6.3 Test 2: Viewing the Coloured Buildings in the City . . . 40
6.4 Test 2: Analyzing Results . . . 40
6.5 Test 3: Viewing Graffiti . . . 42
6.6 Test 3: Analyzing Results . . . 42

B.1 Music→Colour Mappings Table . . . 59

List of Tables

5.1 Note values 23

6.1 Test 1 Results 39
6.2 Test 2 Results 41
6.3 Test 3 Results 43

1. Introduction

This section introduces the motivation and the goal behind the project to the reader, as well as describing the technologies used and how the rest of the project is laid out.

When creating video games, more often than not, the music is precomposed and made to fit the location that the player is currently in. However, this more often than not results in the constant repetition of the precomposed music, especially if players tend to focus on exploring.

In this project, we attempt to rectify this situation by doing away with precomposed music and focusing on generated music. We are inspired by games such as *Proteus* (Key and Kanaga [2013]), where the music is created depending on where the player is in the environment, but unlike *Proteus*, we would like to avoid the use of prerecorded music.

1.1 Motivation

We wish to delve further into Collin's explanation of procedural music in games (Collins [2009]), by making a music generation algorithm that is completely reliant on the environment in order to play music. We are also inspired by Barthes [1967], since by not imbuing any meaning into the music being generated (since it has not been composed beforehand and it is being generated according to the surrounding environment), players are free to make up their own meaning for the music.

We are already aware of a large number of existing literature that deals with computer music, as well as the reinterpretation of music into different domains (such as colour and emotions) and vice versa. This is discussed in further detail in Part I. However, we are not aware of any work that attempts to combine the two, and thus the aim of this project is to try and fill in the gap in already existing literature.

We are also interested in trying to approximate a synaesthetic experience to

non-synaesthetes. Since synaesthesia is a broad topic and there are multiple types of synaesthesia, we intend to focus on chromesthesia (where people have inherent mappings between colour and sound).

1.2 Goal

We have three main goals for this project.

The first goal is to create a continuous and flexible algorithm that is capable of procedurally generating music for game environments. We aim for the algorithm to be continuous, as it must generate music constantly through the play session, and we aim for it to be flexible as it must react to the player's surroundings. This allows the music to still be relevant to what the player is doing.

The second goal is to investigate the use of 1D cellular automata to generate music. Most of the literature surrounding the use of cellular automata in music generation is about 2D cellular automata, and barely any that discuss the use of 1D cellular automata. We also aim to make use of music→colour mappings found in literature, creating a system that is able to take in different aspects of the environment around the player and use them as input to our music generation system. This allows us to close a gap in literature, since we could not find any papers that deal with generation of music based on environmental surroundings.

The third goal is to try and provide a synaesthetic experience to non-synaesthetes. Since we are mainly focusing on music→colour mappings, generating music based on colour would allow non-synaesthetes to see what colour "sounds" like to certain synaesthetes.

1.3 Technologies Used

The programming language that was used for this project was C# within the Unity Engine. C# was chosen due to previous experience with the language, as well as having external APIs available that allowed us to work with MIDI. The Unity Engine was chosen due to it being a 3D engine with a quick learning curve, as well as allowing scripts in C#.

An external library called *MIDI-dot-NET* was found that allowed us to do MIDI programming in C#.

1.4 Project Structure

After this introductory chapter, we proceed to the literature review, which is divided into 3 chapters: procedural music, music generation and mappings. These describe already existing literature and other systems that are relevant to the project. We then proceed to the methodology, which contains the design and implementation of our system, as well as our results, and any relevant future work that may be done on the project.

One may also find an appendix which contains the list of scales used in the music generation algorithm, and all the relevant music→colour mappings that are mentioned.

Part I

Literature Review

2. Procedural Music

Since we intend to have reactive, dynamic music based on the surrounding environment, we cannot simply record a soundtrack to be played. In this case, we decided to experiment with procedural music, which Collins describes as being "composition that evolves in real time according to a specific set of rules or control logics" Collins [2009]. She likens procedural music in games to gameplay itself; this is because she argues that every player plays the game in different ways, such as by taking different paths, or entering different areas or different menu screens at different times. Therefore, the music and sounds associated with the game (such as footsteps, gun shots, pre-composed music in different areas of the game and menu sounds) can be played back in different sequences during gameplay, creating procedural music. Wooller et al. [2005] describes four different paradigms of generative music, which are described below:

- **Linguistic / Structural**
  - Music created using generative grammars or other form of theoretical structure
- **Interactive / Behavioural**
  - Music generated by something that is inherently non-musical
- **Creative / Procedural**
  - Music created by processes that the composer starts (or has some control over)
- **Biological / Emergent**
  - Music that is not repeatable and not deterministic (such as wind chimes or birds)

Though it may seem unlikely, algorithmic music has existed for a long time, even before the invention of computers. Mozart made musical dice games (called

Musikalisches Würfelspiel). Here, dice were thrown and precomposed music was played depending on the outcome of the dice Becker [2005]. This was the use of open form, a type of aleatoric music. In the 1950s, Iannis Xenakis generated musical compositions using stochastic processes Järveläinen [2000] and by using Markov Models, Hiller created what is acknowledged as one of the first pieces of music composed with a computer, the *Illiac Suite* Becker [2005]. Terry Riley's *In C* is a semi-aleatoric generative piece of music where performers play bars in the sheet music in the way they want to, which means that performances are always different.

Wooller et al. [2005] goes on to describe two different classifications of algorithms that can create music. These are:

- **Transformational algorithms**
  - These algorithms usually affect the song's structure, such as by choosing which parts of the song to play next or adding or removing instrument lines. They can also change the way the notes are played such as by changing the pitch or the volume of the note.
- **Generative algorithms**
  - These algorithms create the actual sequence of notes to be played. They create the notes that are found inside the song's structure.

For the rest of this project, we will be focusing on music, rather than sound effects. We will also only be considering generative algorithms in our implementation.

3. Music Generation

3.1 Music Generation

An early example of a music generation algorithm being used in games is the "riffology algorithm", which was used in *Ballblazer* by LucasArts Lucasfilm Games [1984]. In Langston [1986], Langston explains that the algorithm chooses the easiest riff to play next from a list of riffs, such as a riff with a starting note closest to the previous riff's ending note. The algorithm also slightly tweaks the riffs by adding or removing notes and increasing or decreasing the speed of the riff. Langston argues that although the result is musical, the lack of structure in the generated riffs means that it doesn't remain interesting for long. This implies that generated music must contain some sort of inherent structure to be interesting.

In Miranda [2004], Miranda claims that there are three different approaches that one can take when using evolutionary computation in music: *engineering*, *creative* and *musicological*. In the engineering approach, generating music is treated as a problem that contains a huge space of possible solutions. Therefore, techniques such as genetic algorithms and genetic programming are used. In the creative approach, Miranda discusses the tendency of algorithms to generate music in a particular style, especially when using training examples. The creative approach tries to avoid this by using mathemathical models such as grammar systems and fractals. In the musicological approach, techniques are used that are inspired by studies on the origins of music. Some techniques that are discussed include simulating a critic when generating music, as well as using mimetic models.

For example, in Blackwell [2007], Blackwell uses swarming algorithms in a 3D environment to create music that is similar to free jazz in style. He does this by interpreting the position of the swarms as music. Blackwell was able to have swarms collaborate together to create music, as well as improvising with humans. In Schacher et al. [2011], Schacher describe a similar system that generates music using swarm systems, but models the problem by having primary and secondary

swarms with different characteristics. The music generated is combination of a light background ambience and bright sounds generated by the simulation. In Bisig and Neukom [2008], the authors discuss different ways of interpreting the swarm system musically, such as additive synthesis and granular synthesis, as well as discussing different ways of representing the physical representation of the swarm network.

Another technique is the use of evolutionary algorithms such as genetic algorithms to create music. Tokui and Iba [2000] describes the search space as "an infinite combination of melodies, harmonies, and rhythms", claiming that it is impossible to find meaningful music without some form of guidance. Husbands et al. [2007] details how an evolutionary algorithm would work with respect to both music generation and sound generation, as well as describing the new possibilities available. Tokui and Iba [2000] generate rhythm patterns that are represented using a combination of genetic algorithms and genetic programming, and these are constantly evaluated by the user. This allows the system to be able to generate specific types of rhythms, as well as rhythms that the user finds pleasing or interesting. Becker [2005] describes the use of genetic programming within interactive evolutionary computing techniques; Becker states that by asking users to guide the algorithm, the music generated has a human quality to it that would be too hard to create with a regular algorithm as the search space is too large.

Jacob [1995] tries to tackle the large search space problem, not by reducing the size of the search space, but by using larger building blocks to work with. Jacob first defines a set of motives to be used, creates phrases based on these motives, and then composes music by combining the phrases together. At a higher level, Jacob introduces three different modules; one that composes music, one that filters unsatisfactory music, and one that imposes order on whatever is left. Birchfield [2003] uses a coevolutionary genetic algorithm to generate a population of components that contain different features that describe frequency, harmony, rhythm, meter and other musical concepts.

One approach is to try and describe music by using grammars. García Salas et al. [2011] takes a linguistic approach by defining notes as a tone and how long the tone should be held for, and defining a musical composition as a group of notes in a particular arrangement. By evolving rules, and with the help of a frequency distribution matrix, the system is able to generate appropriate music.

Manousakis [2006] demonstrates the generation of music using L-systems. The author approached the problem by implementing different types of L-systems which are then interpreted by a "music turtle" in 3D space. The output can then be interpreted using interpretation and decomposition rules. Langston [1989] also describes how L-systems may be used in music generation; in particular,

describing the interpretation algorithm as being a depth first traversal of the resulting tree.

Another approach is to generate music based on fractals and other chaotic functions. Hinojosa Chapel [2003] describes four different composition tools that use chaotic functions (such as noise functions, Henon mappings and attractors, and the Julia set) and self-determined mappings to interpret the outputted numbers into musical values of some kind.

Brown and Kerr [2009] describes the use of adaptive music techniques where the music reacts to changes in the environment or to user interaction. The authors explain several techniques that could be used to make the music adapt to changes, such as transposition, tempo, rhythmic density and others.

Nierhaus [2010] also explains different algorithmic techniques and applies them to music, including other techniques such as Markov models, Petri nets and neural networks. For example, Nierhaus [2010] states that Markov models may be used to generate music that is similar in style to pre-examined music from a body of examples. Haus and Sametti [1991] uses Petri nets in music generation by connecting music objects together using transition functions and manipulating them in some way. The authors also discuss the use of a counter that serves as a synchronization point between music objects, since Petri nets do not usually cater for time. Todd [1989] discusses the use of neural networks, especially with regards to representing the concept of time that is inherent in a musical piece. One possible solution that is discussed is by using the memory of previous notes as input to the neural network itself.

3.2 Music Generation Using Cellular Automata

Cellular automata, first described by Ulam and Von Neumann in the 1940s, have only recently started to be used in music generation. The rules defining the self-organizing behaviour in the automata may be changed, leading to different results produced by the automata. Wolfram, who describes cellular automata as being "discrete dynamical systems with simple construction but complex self-organizing behaviour", categorizes cellular automata into different classes that describe the results that they output Wolfram [1984]. These are:

- Class 1
  - The automata generate patterns that eventually vanish or remain the same over time
- Class 2
  - The automata generate patterns that are periodic and repeat themselves
- Class 3
  - The automata generate patterns that never repeat, but are chaotic and unpredictable
- Class 4
  - The automata generate patterns that are complex. These patterns grow and evolve

Li et al. [1990] proposes a finer way of classifying cellular automata, claiming that their categorization is finer than the one presented by Wolfram. Their classification is presented below:

- Spatially homogeneous fixed points
- Spatially inhomogeneous fixed points
- Periodic behaviour
- Locally chaotic behaviour
- Chaotic behaviour
- Complex behaviour

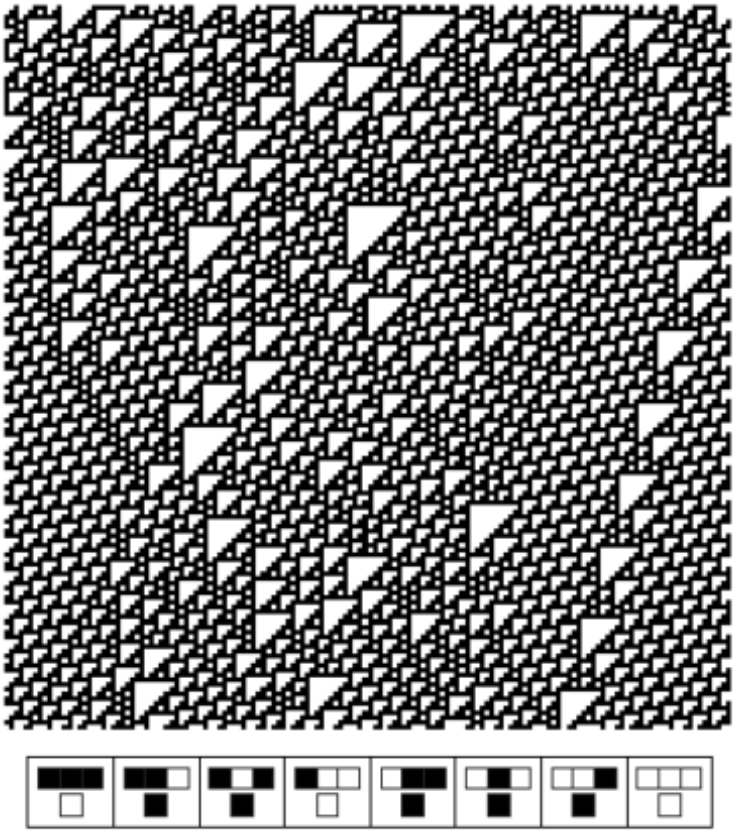


Figure 3.1: Result of a Class 4 Cellular Automata running with Rule 110. Taken from Burraston et al. [2004]

Fig. 5.7 illustrates the rule definition for Rule 110 and its result as a 1D cellular automata.

Burraston and Edmonds [2005] gives a historical overview of the use of cellular automata used in music, including their use in audiovisual installations. He also describes several systems that have been used cellular automata and MIDI, such as work by Beyls Beyls [1989], Millen Millen [1990] and Miranda Miranda [2003], as well as the use of cellular automata in synthesis.

Millen [1990] explains how cellular automata may be interpreted musically. Here, Millen describes how he maps the results of a cellular automaton in 2D space to the properties of notes, such as pitch. Millen also states that cellular automata music can be grouped into 2 categories: Category I-type, where nothing else is changed after the automaton starts working, and Category II-type, where cells in the 2D space or other state values may be modified while the automaton is running.

Jewell [2007] describes two different cellular automata arrangements: John Conway's "Game of Life" and the Demon Cyclic Space. Both automata operate in 2D space and are used together in the CAMUS 2D System (as explained in Miranda [2003]) to generate music. In this system, Miranda interprets the results from the Game of Life automaton as a three note chord and the results from the Demon Cyclic Space as the instruments used to play the notes in the chord. Miranda also extended his system with CAMUS 3D, which used a z co-ordinate to turn the three note chord into a group of 4 notes.

One system that we found that generates music from cellular automata is Eutérpê Melōidía by Juan Carlos Soriano Ramírez[1]. The system is capable of generating 1D or 2D cellular automata and interpreting the results musically. It is also able to interpret image data from pictures and use them as input to the 2D cellular automata, as well as being able to output the results as MIDI files or as a musical score. Fig. 3.2 shows a screenshot of the system working with a 1D cellular automaton using rule 90.

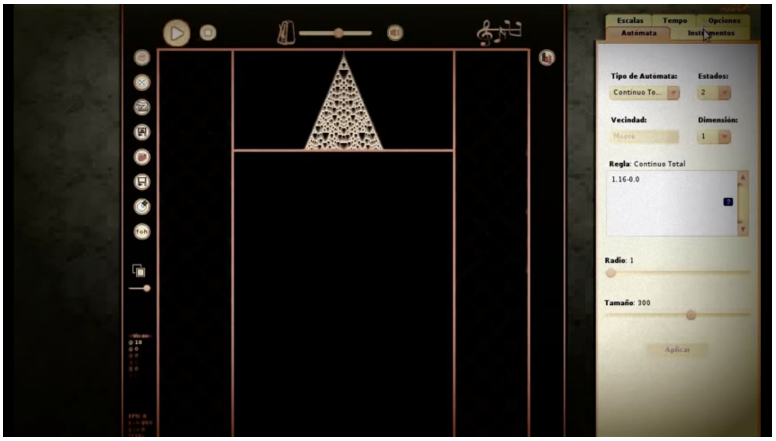


Figure 3.2: *Eutérpê Melōidía.* Taken from `http://www.youtube.com/watch?v=XCgqRoQJNMA`

Brown [2005] discusses the use of 1D cellular automata to generate rhythm lines by interpreting the result of the automata as having the duration of a sixteenth note (semiquaver), and triggering a note or a rest. Brown discusses how effective different CA classes are to generate proper rhythm lines, and states that there should be a balance between stability and novelty. He lists a couple of rules and techniques that he feels may be used to vary the generated rhythms, such as by inverting the result of the automaton or by staggering out different rhythms.

Beyls [1989] also discusses how cellular automata may be interpreted musically. Beyls points out several ways this can be done, such as using 1D automata

[1] `http://juankysoriano.com/euterpe-meloidia`

(which he calls *continuous automata*) and 2D automata, but also some other interesting ways, such as interpreting cellular automata results as wave propagation. He also mentions improving the musical output of an automaton by adding a history to it, by allowing an external object to give it feedback, or even by chaining multiple automata together.

4. Mappings

4.1 Music→Colour Mappings

In Poliniak [2012], Poliniak describes how different senses are used to teach singers, and it is surprising how some senses may be tied to music, such as balance and pain. Poliniak claims that being able to tie different senses together helps students to understand what sort of sound is needed, which sometimes goes beyond a note. Thus we can see that there exists certain mappings between music and different domains.

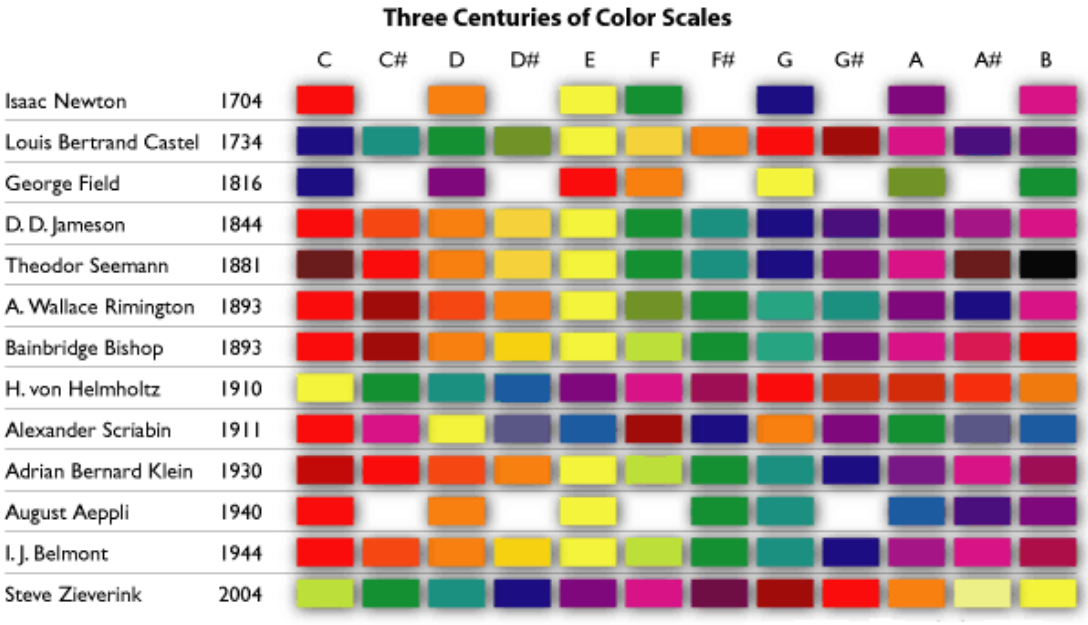


Figure 4.1: Different colour scales. Taken from Collopy [2009]

There have been multiple attempts throughout history that have tried to create associations between different objects. Jewanski [b] mentions Artistole as being one of the first people to form a colour scale, but it was Athanasius Kircher who attempted to create "associations between intensities of light, different de-

grees of brightness, types of tastes, the elements". Marin Cureau de la Chambre also applied the concepts of consonance and dissonance to notes. In Newton [2011 (originally published in 1704], Newton comes up with a seven step colour scale, now known as the colours of the rainbow, and maps it to the seven tone music scale (ascending whole notes starting from C). However, in Gerstner [1990], Gerstner states that Newton's choice of colours to map to particular notes was random and had no scientific background to it. Several other people after Newton have tried to map different colours to different notes, as can be seen in Fig. 4.1.

Jewanski [b] claims that Louis-Bertrand Castel was the "first to produce a pure color-tone analogy". He also wanted to combine music and art into an art form he called *musique muette*, and was one of the first people to design a colour organ, which he called the *clavecin oculaire*. Jewanski also describes how the frequencies for notes can be mapped to frequencies and the resulting wavelengths of colours. The scientific music→colour mapping can be seen in the Appendix in Section B.3.

There have been a number of projects and movements throughout history that have tried to investigate the simultaneous use of colour and music. Several instruments have been created to perform colour music Peacock [1988], and even though Giuseppe Arcimboldo designed a colour cembalo which he called "the harpsichord of colour", the first colour organ is generally attributed to Castel. Jewanski [a] states that a colour organ is "a device, usually controlled from a keyboard, with which music can be visualized or a pure display of colors presented as an autonomous art form". van Campen [1997] and Jewanski [a] both discuss the invention of several colour organs; some major ones include the *Farbenclavecymbel* by Johann Gottlob Kr'uger, the *tastiéra per luce* by Alexander Scriabin and the *Clavilux* by Thomas Wilfred.

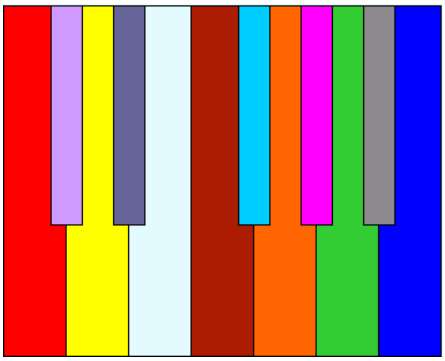


Figure 4.2: A rendition of Scriabin's *tastiéra per luce*. Taken from `http://upload.wikimedia.org/wikipedia/commons/6/68/Scriabin_keyboard.png`

Scriabin, who wrote the symphonic work *Prometheus: The Poem of Fire* for the *tastiéra per luce*, was a Russian composer and synaesthete. Influenced by theosophy and his synaesthesia, Scriabin rejected the correspondence between colours and single tones, and based his colour scale on tonality and on the circle of fifths, shown in Fig. 4.1 and Fig. 4.2. Scriabin based this correspondence on Newton's work in *Optics*, as well as theosophical readings.

Figure 4.3: De Maistre's colour harmonizing chart. Taken from `http://home.vicnet.net.au/~colmusic/maistre.htm`

Synchromism was a style of painting invented by Stanton Macdonald-Wright and Morgan Russell that was "dedicated to the physical and emotional effects of color" Collopy [2009]. Based on colour scales and inspired by music, Macdonal-Wright and Russell painted *synchromies*. Roy De Maistre was similarly inspired by music, and created a colour harmonizing chart (as seen in Fig. 4.3) to market his ideas and guide other artists Hutchison [1997]. However, while Howell finds the idea of synchromism to be interesting, Howell finds it to be a little restricting too. He claims that this is because of the "limited grasp of music theory" at the time, as well as referencing Macdonald-Wright [1924], which cites discordant intervals that are nowadays frequently used in popular music. Most experiments involving colour and music also used the major scales. Howell introduces *neosynchromism*, where he tries to "update the original principles of Synchromism" by adding other scales and accounting for modern taste.

Milicevic describes Wassily Kandinsky, an artist that was part of *Der Blaue Reiter (The Blue Rider)* art movement, who was also a synaesthete. Apart from integrating vision and music into theater, Kandinsky also orchestrated colour in his paintings, similar to the way music is orchestrated.

Abstract film is another area where mappings can be drawn from. Moritz [1996] for example, talks about Mary Ellen Bute and her abstract filmography, which is inspired by Joseph Schillinger's musical theories, as well as colour organs.

(a) *Composition IV*, by Kandisky. 1911

(b) *Airplane Synchromy in Yellow-Orange*, by Stanton MacDonald-Wright. 1920

Figure 4.4: Visual music paintings

Synchromy No. 4: Escape was an abstract colour film made in 1938 by Bute and Nemeth which featured Bach's *Toccata and Fugue in D minor, BWV 565* as music. Moritz [1995] goes into more detail and further discusses the visual music movement and prominent artists such as Walter Ruttmann, Alexander Laszlo and Oskar Fischinger.

In Kandinsky [2011 (originally published in 1912], Kandinsky compares several colours to different instruments, describing the different tones that the colour would produce. Examples include the flute for light blue, the church bell for orange and the tuba for red. Citing Gerstner [1990], Collopy produces a table that maps between hue, saturation, value and shape for colour, and pitch, amplitude, overtones, tempo, interval and mode for music Collopy [2001].

An interesting topic we would also like to investigate is sound-colour synaesthesia, also called chromesthesia, where people can see colours when listening to music or sounds. This is because we feel that there might be inherent mappings between colour and music that synaesthetes experience. For example, Ward et al. [2006] discusses that both synaesthetes and non-synaesthetes associate light colours with high pitched sounds and dark colours with low pitched sounds; however synaesthetes were reported to be more consistent and specific with their choice of colour association. Sidler however, states that not all synaesthetes necessarily have the same experience; Sidler describes 4 different people that all have different music→colour experiences.

4.2 Other Mappings

Another possible mapping is the mapping between music and space. Collopy [2001] demonstrates some parallels between music and the width of lines (citing Kandinsky [1979 (originally published in 1926], which states that high pitched instruments, like the flute and the violin, are associated with thin lines, while low pitched instruments, such as the double bass or the tuba, are associated with thicker, heavier lines). He also demonstrates parallels between music and the size of shapes (citing Kepes [1995 (originally published in 1944], which states that high pitches are associated with small shapes, while low pitches are associated with large shapes), as well as between music and the form of the shape (citing Karwoski and Odbert [1938], which states that the faster the music is, the more pointy and angular a shape is).

Another mapping that is relevant is one between music and physical space or virtual space. Ashley [2004], for example, investigates the concept of verticality in music, both in the gestures that musicians make as well as the position of notes on an instrument, and how it differs between cultures (in this case, Western and African cultures). In Western cultures, high notes are represented by raising their hands to an elevated position, while low notes are represented by lowering their hands. In African cultures, however, high notes are represented as being small, and low notes as being large. In Bonde [2008], Bonde explores the mappings between music and architecture, while in Winkler [1995], Winkler discusses the mapping between music and gestures. Winkler [1995] discusses the theremin, an electronic instrument made by the Russian inventory Léon Theremin, that could be played by using physical gestures and without having to actually touch the instrument. He also discusses *Variations V*, a dance performance where the dancers' movements were translated into sound and music.

One other possible mapping that we wish to consider is the mapping between music and emotions. Much work has been done already involving the classification of emotions by psychologists Ekman et al. [1972], Plutchik [1980], Parrott [2000]. Fig. 4.5, for example, shows Plutchik's Wheel of Emotions. We would like to eventually involve emotions in music generation process, either directly or indirectly. Rutherford and Wiggins [2002] describes a system called HERMAN, which generates scary music as a background for a program called *GhostWriter*, a children's educational program. This was done by analysing soundtracks from different films and making note of the styles of scary music used. Le Groux [2009] describes a system that generates music in response to the user's emotions, and thus describes several mappings used between musical parameters and the way the music is generated and played. Livingstone and Brown [2005] talks about a system that can change generated music dynamically based on certain emotions. Fig. 4.6 shows an example of how the way music is played can be mapped to different emotions.

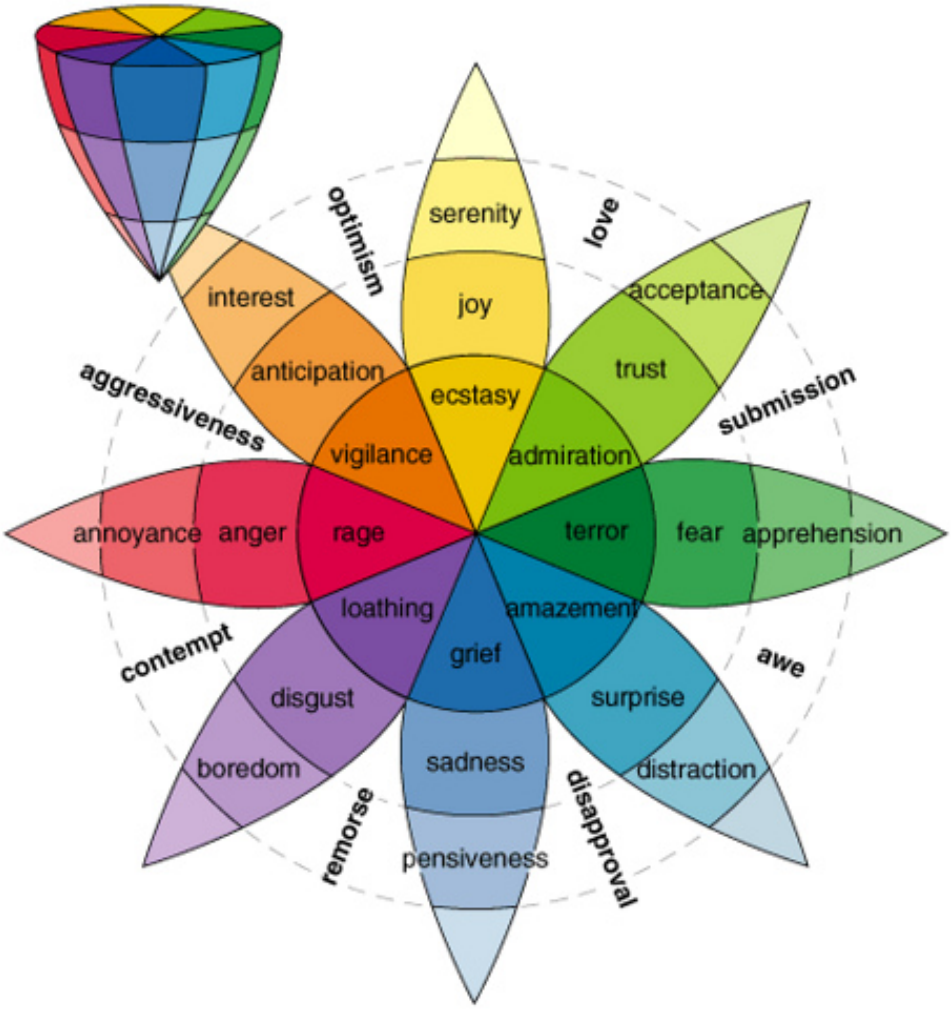


Figure 4.5: Plutchik's Wheel of Emotions. Taken from Plutchik [2001]

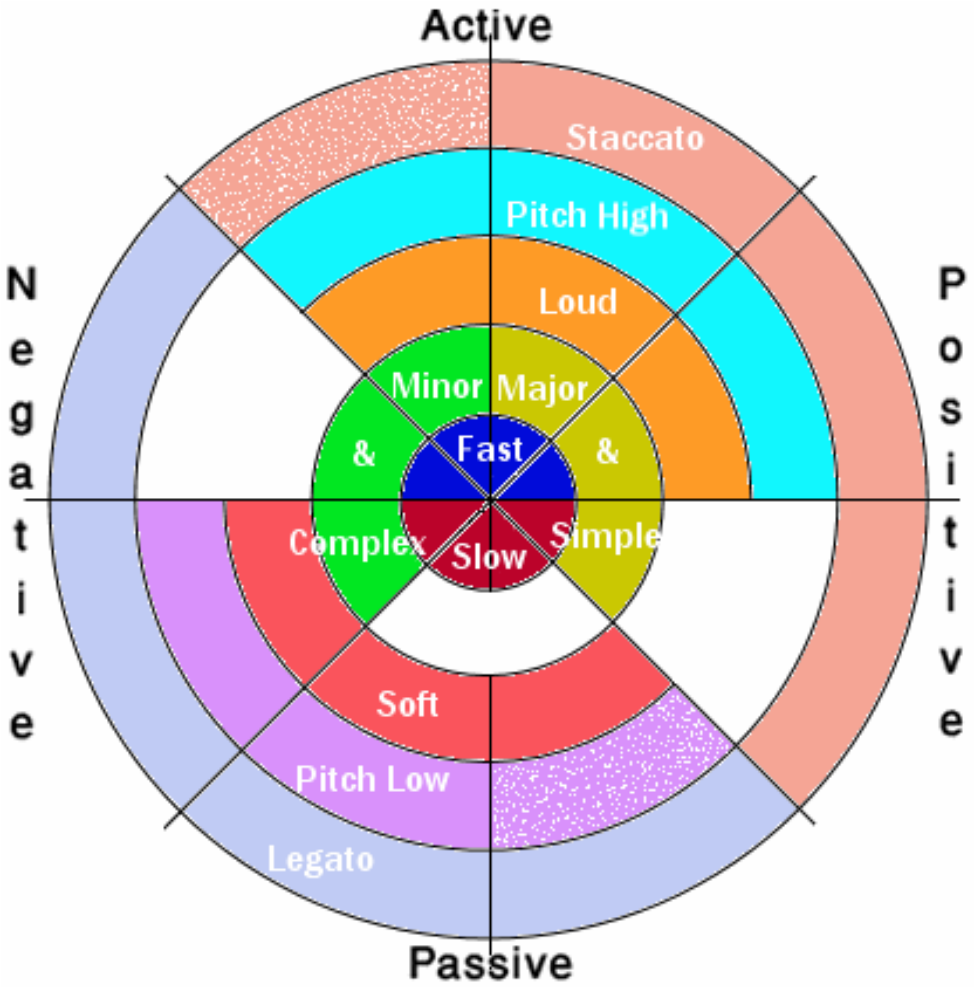


Figure 4.6: Different ways of playing music depending on different emotions. Taken from Livingstone and Brown [2005]

Part II

Methodology

5. Design and Implementation

In this section, we describe the steps we took to design and implement an algorithm that can procedurally generate music based on the environment around it. Note that this implementation builds upon an earlier version, although all aspects have been further developed.

5.1 *MIDI-dot-NET*

MIDI-dot-NET is an external library available under an open source BSD 3-Clause License. The library allowed us easy access to the computer's default MIDI output device, as well as allowing us to schedule messages to the device to play and stop notes. The library also allows for enharmonic equivalents between accidentals, including double sharps and double flats, as well as allowing us to create custom scales by supplying the necessary note progressions.

5.1.1 MIDI Messages

In order to tell the MIDI device what notes to play, messages must be sent to the device detailing which note to play, how to play it and the proper time to play it. In *MIDI-dot-NET*, two messages must be sent for the note to be played correctly, a *NoteOnMessage* and a *NoteOffMessage* immediately after it.

We noticed that *MIDI-dot-NET* also offered a *NoteOnOffMessage* to automatically schedule the *NoteOffMessage* in one call, but we did not manage to get this working and had to separate the individual calls.

Messages are usually grouped in bars and all scheduled together, rather than scheduling single notes. This is to allow greater structural control over the music generation, as well as allowing time for other processing that must be done in the background.

5.1.2 Note Values

Note values in *MIDI-dot-NET* are handled using float values, where a crotchet (or quarter note) has a value of 1. Knowing this, we could use the values of different notes in order to generate richer music. The values of the notes we used can be seen in Table 5.1.

British Name	American Name	Value
Dotted Minim	Dotted Half Note	3f
Minim	Half Note	2f
Dotted Crotchet	Dotted Quarter Note	1.5f
Crotchet	Quarter Note	1f
Dotted Quaver	Dotted Eighth Note	0.75f
Quaver	Eighth Note	0.5f
Dotted Semiquaver	Dotted Sixteenth Note	0.375f
Semiquaver	Sixteenth Note	0.25f

Table 5.1: Note values

5.1.3 Time Signatures

In music theory, time signatures serve as convention to show how many beats are in a bar, and what constitutes one beat. The concept of time signatures however, is not defined in *MIDI-dot-NET*. We attempted to get around this problem by creating a custom *TimeSignature* class, which neatly wraps several numbers into an understandable concept.

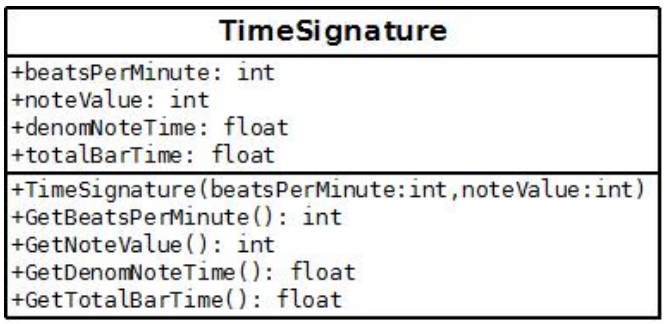


Figure 5.1: TimeSignature Class Diagram

In our implementation, the following time signatures were available: $\frac{2}{4}$, $\frac{3}{4}$, $\frac{4}{4}$, $\frac{5}{4}$ and $\frac{6}{8}$.

5.1.4 Scales

A scale is simply a sequence of notes, and *MIDI-dot-NET* comes with some scales already defined that one may make use of (these being the Major scale, the Natural Minor scale, the Harmonic Minor scale, the Melodic Minor Ascending scale, the Melodic Minor Descending scale and the Chromatic scale). To define a scale, one simply needs to provide the tonic note (the note that the scale starts from) and the pattern sequence of notes.

For our project, we made use of purely heptatonic scales[1] due to the colour→note mappings that we used. The scales' respective note progressions may be seen below. A full list of all the scales we used may be seen in the Appendix in Section A.1.

- **Major Scales**

  0, 2, 4, 5, 7, 9, 11

- **Natural Minor Scales**

  0, 2, 3, 5, 7, 8, 10

- **Harmonic Minor Scales**

  0, 2, 3, 5, 7, 8, 11

- **Melodic Minor Ascending Scales**

  0, 2, 3, 5, 7, 9, 11

- **Major Blues Scales**

  0, 2, 3, 5, 6, 9, 10

- **Arabic Scales**

  0, 2, 4, 5, 7, 8, 10

- **Phyrgian Dominant Scales**

  0, 1, 4, 5, 7, 8, 10

- **Lydian Dominant Scales**

  0, 2, 4, 6, 7, 9, 10

[1] Heptatonic scales contain exactly 7 notes.

Note that all notes in the scales had to be tweaked to fit the notes used by the mappings (natural notes or notes with sharps). In particular, the notes used were C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯ and B. Therefore, any notes in the above scales that are enharmonically equivalent are thus converted to the mentioned notes. For example, B♯ is enharmonically equivalent to C, D♯♯ is enharmonically equivalent to E, while F♭ is also enharmonically equivalent to E.

5.1.5 Instruments and Channels

Since each MIDI device has 16 independent channels to choose from, and Channel 10 is usually reserved for percussion, we are technically able to have 15 instruments play concurrently. In our example, we make use of 3 different music lines, where each line is generating music. Therefore, each one is assigned to its own channel, and the channel is set to play with a particular instrument.

5.2 Generating Music Based On The Virtual World

This section describes the design and implementation required to generate music based on the virtual world, such as generating music from colour.

5.2.1 Music→Colour Mappings

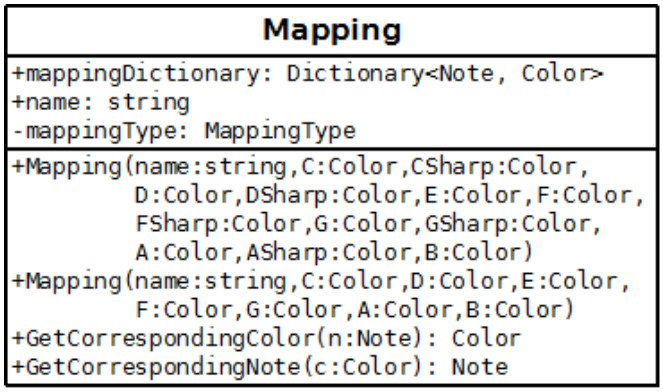


Figure 5.2: Mapping Class Diagram

The *Mapping* class, as illustrated in Fig. 5.2, is a wrapper class around a *Dictionary* that stores the relationship between colours and notes. It allows the creation of both twelve note scale mappings, as well as seven note scale mappings, as it offers two different constructors. The *Mapping* class also allows the retrieval of the corresponding colour or the corresponding note, depending on the input.

5.2.2 Extracting Colours from the Viewport

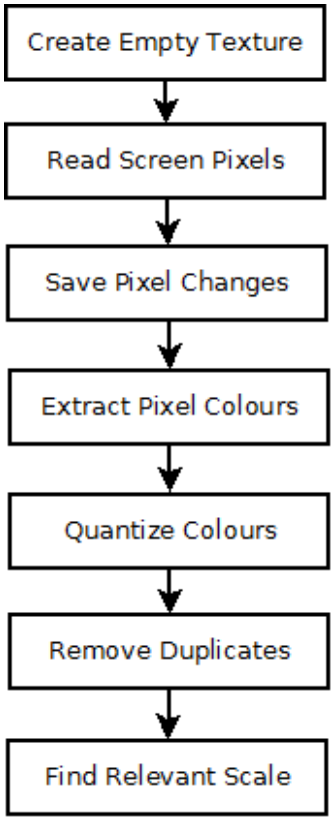


Figure 5.3: Converting Colours from the Viewport to Scales

Fig. 5.3 illustrates the steps that we take in order to convert colours visible on screen to music.

First, we create an empty Texture2D object with the height and width of the visible screen. The empty object is then populated using the *ReadPixels* method; taking the pixels shown on the screen and placing them on the empty texture. The pixels are then saved using the *Apply* method.

To extract colours from our obtained texture, we traverse through the pixels and every certain number of pixels, we keep the colour of the pixel found. The number of pixels needed to be traversed before sampling the texture can be changed easily in order to refine the algorithm's accuracy. After the whole texture has been successfully sampled, the retrieved colours are quantized by finding the closest colour from the chosen mapping. A list of all the colours from the mappings can be seen in the Appendix in Section B.5. Next, all duplicate colours are removed, so that the end result is a list of unique, quantized colours that have been found from the camera's viewport.

This list of colours is used to find the relevant scale. This is done by first converting the colours to their equivalent in music notes using the chosen music→colour mapping, Next, we look through the list of scales that were generated (which can be seen in the Appendix in Section A.1). If a scale does not contain all the notes in the list of notes, it is discarded.

The chosen scale is then given to the *Conductor*, which distributes it to the cellular automata it controls. This means that the next bar to be scheduled to be played will contain notes from the new scale.

5.2.3 Generating Music Based On Character Speed

We also decided to change the generated music's tempo based on the character's movement speed. If the character is not moving, the tempo is slow, and music plays at 80 beats per minute. If the character starts moving however, the tempo changes to 120 beats per minute and the music becomes quicker. This is done by first detecting whether or not the character is moving, and then sending the appropriate result to the *Conductor*.

5.3 Cellular Automata

The next section describes the way the music generation system was designed and implemented. We first take a top-down approach of the system by describing the system architecture and how things are meant to fit together, and then proceed to take a modular approach by describing how each part of the system works.

5.3.1 System Architecture

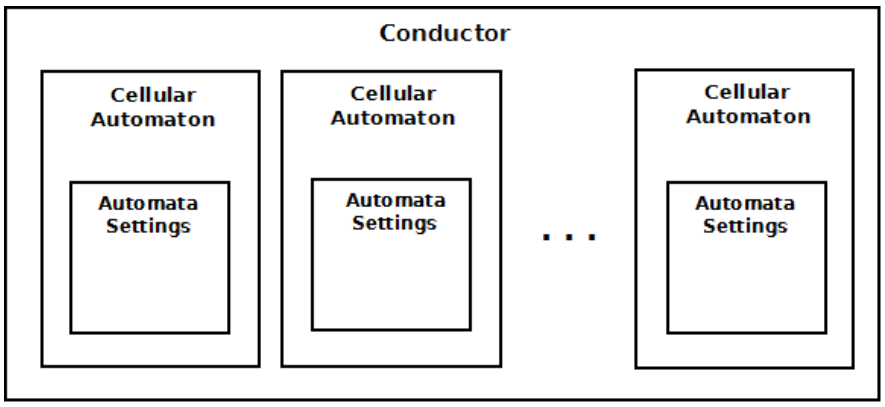


Figure 5.4: Music Generation System Architecture

Fig. 5.4 illustrates the system architecture of the music generation algorithm.

One can see that the system is capable of supporting multiple cellular automata, all controlled by a module called the *Conductor*.

5.3.2 *GenerativeMusic* Class

The *GenerativeMusic* class sets up the environment necessary for the music generation algorithm to work. This includes instantiating the *Conductor* class (explained below in Section 5.3.3) and all the necessary cellular automata, as well as setting up all the necessary mappings, time signatures, quantized colours, automata rules and scales. The *GenerativeMusic* class also opens up the appropriate output device to allow MIDI to be played.

5.3.3 *Conductor*

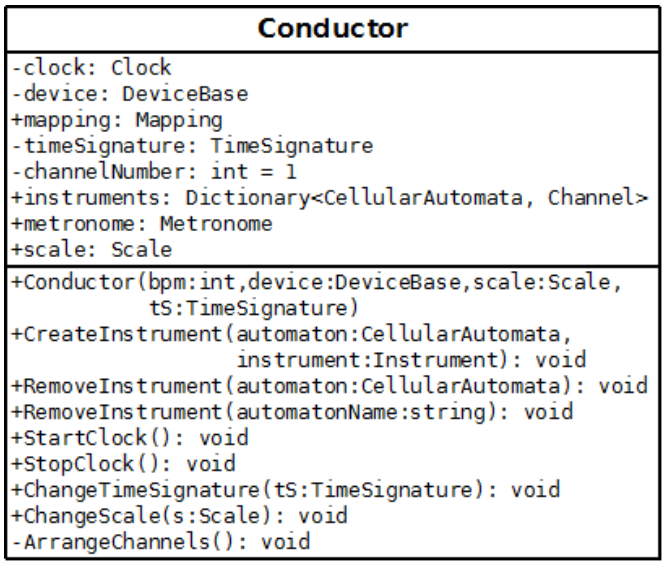


Figure 5.5: *Conductor* Class Diagram

As illustrated in Fig. 5.5, the *Conductor* class is a module that controls and "conducts" the cellular automata, serving as a wrapper class.

The *Conductor* takes a tempo, a time signature, a colour→note mapping, a beginning scale and a device where the cellular automata can output their results to.

5.3.4 Instruments as Cellular Automata

In our system, each instrument is played by a single automata, which is constructed with a particular rule and has knowledge of the global time signature,

scale and output device through the *Conductor*. On creation, we also assign a music generation mode to the automaton, which tells it how to interpret its population musically. We are also able to assign different automata to different channels, and assign different instruments to those channels; this is done through the *Conductor* on creation of the automaton.

Automata Rules

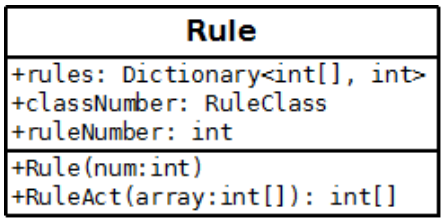


Figure 5.6: *Rule* Class Diagram

We implemented cellular automata rules as their own class, as can be seen in Fig. 5.6. The rules, which can be seen at the bottom of Fig. 5.7, were represented in an ⟨int[], int⟩ Dictionary, where an array of numbers corresponded to just one number. We defined black squares as having the value 1, and white squares as having the value 0. Therefore, the rule defined in Fig. 5.7 would be declared as having the array {1, 1, 1} produce 0, {1, 1, 1} produce 1, etc.

AutomataSettings Class

On initialization, the automaton requires certain settings that describe how the automaton's results will be interpreted musically. Fig. 5.8 shows a simple class diagram that shows 4 different modes.

PitchMapMode describes how the pitches are chosen from the automaton results. We only have one choice so far: *NoteToPitchDirectCorrespondence*, where the automaton results are directly mapped to the notes of the scale.

VelocityMapMode describes how the velocities of the notes played are chosen from the automaton results. We only have one choice so far: *FourLevelPitch*, which quantized velocity into 4 different levels. If the automaton results relevant to the *VelocityMapMode* is 00, then the note is played with a low velocity. 01 signifies a medium low velocity, 10 is a medium high velocity, and 11 is a high velocity.

ConductorMode describes whether or not each individual instrument depends on each other. There are 4 different options, although only one has been implemented so far. The first option is *NoConductorMode*, where each instrument reacts independently of one another. The second mode is *BassFirstMode*, where

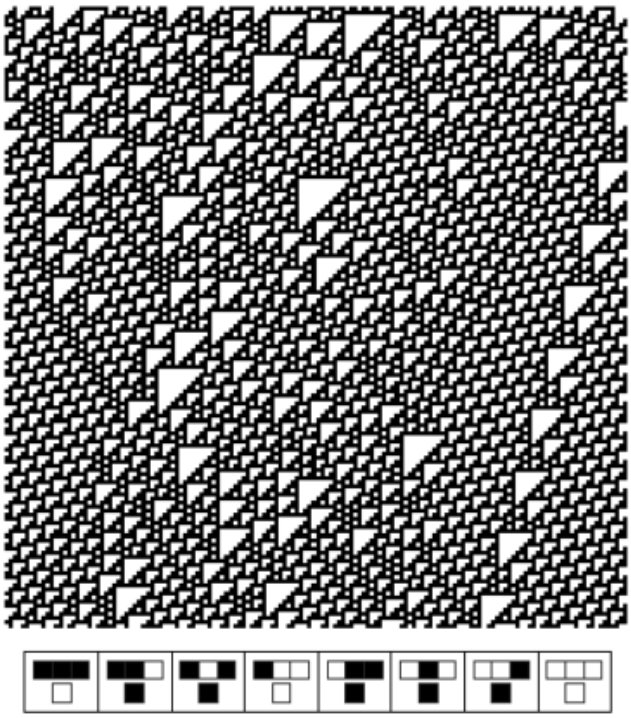


Figure 5.7: The Output of a Cellular Automaton Rule

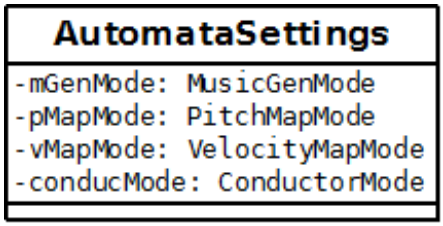


Figure 5.8: *AutomataSettings* Class Diagram

the bass instrument generates the melody first, and the other instruments based their created melody on the bass's melody. *RhythmFirstMode* and *LeadFirstMode* are similar, but the rhythm instrument and lead instrument generate their melody first, depending on the mode chosen.

MusicGenMode describes the length of the notes generated from the automaton rules; these are based on the note values as seen in Fig. 5.1. *EachNoteMadeFromArray* generates the length of the notes based on the result of the cellular automata. This is done greedily, where we take the length of the notes and rests generated by the automata until they no longer fit into the time allotted for the bar, and then add notes or rests of lengths that can fit into the remaining time. We don't use dotted notes in this case, because it causes technical problems when selecting them greedily. *EachNoteSolelyCrotchets* assumes that all notes

generated have the length of crotchet notes (fourth notes) or crotchet rests. This is similar for *EachNoteSolelyQuavers*, which handles quavers (eighth notes) and *EachNoteSolelySemiQuavers*, which handles semiquavers (sixteenth notes).

Interpreting Cellular Automata Musically

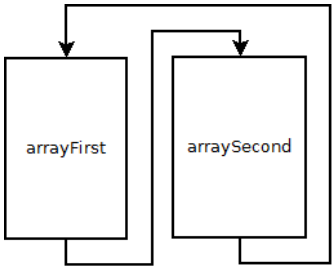


Figure 5.9: Creating a never-ending automaton

The automaton contains 2 lists of arrays which is used to store the generated output of the automaton. The first time it runs, the automaton initializes both lists and randomizes the elements of the first integer array with either 0 or 1. This represents the initial population of the cellular automaton. The size of this list and the size of the arrays are both chosen when automaton is initialized. The automaton then generates the population based on its rule until it fills the list, and then generates a second list so that it can switch between the first and second list and continuously generate a population until told to stop (as can be seen in Fig. 5.9).

Depending on the time signature and the *MusicGenMode* chosen, the algorithm traverses through the list and schedules notes to be played until it has filled a measure. The amount needed to fill a measure is calculated from the time signature, so for example, a time signature of $\frac{4}{4}$ would have a value of 4, whilst a time signature of $\frac{6}{8}$ would have a value of 3. This is because even though there are 6 quaver notes per measure, a quaver note is only worth 0.5, making a total value of 3.

For 3 of the *MusicGenMode* options (*EachNoteSolelyCrotchets*, *EachNoteSolelyQuavers* and *EachNoteSolelySemiQuavers*), the next line in the list removes the note value from the measure value, until it has been successfully filled and 0 is remaining. This can be seen in Fig. 5.10, where for a time signature of $\frac{4}{4}$, 4 crotchet notes are needed to fill a measure (shown in blue), 8 quaver notes are needed to fill the same measure (shown in red), and 16 semi quaver notes are needed to fill the same measure (shown in purple, spilling over to the next list). Once the first list is completed, a new population is added based on the last entry in the 2nd list.

The arrays can then be interpreted musically in different ways, giving different

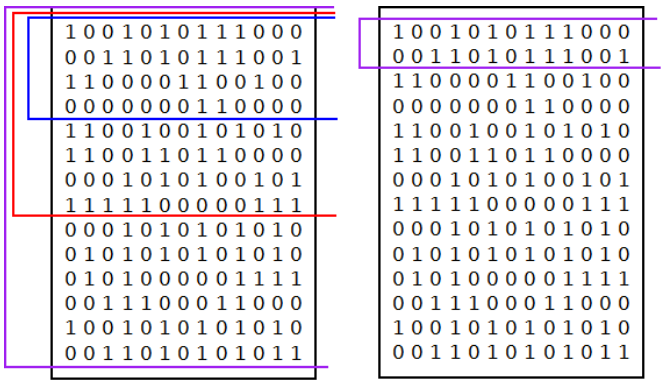


Figure 5.10: Filling a measure

results. From the result of the cellular automata, we can extract whether or not a note should be played, the length of the note to be played, the note's pitch, and the velocity (loudness) of the note. Fig. 5.11 is an example of the output given by a cellular automata. This is generated from the first line downwards.

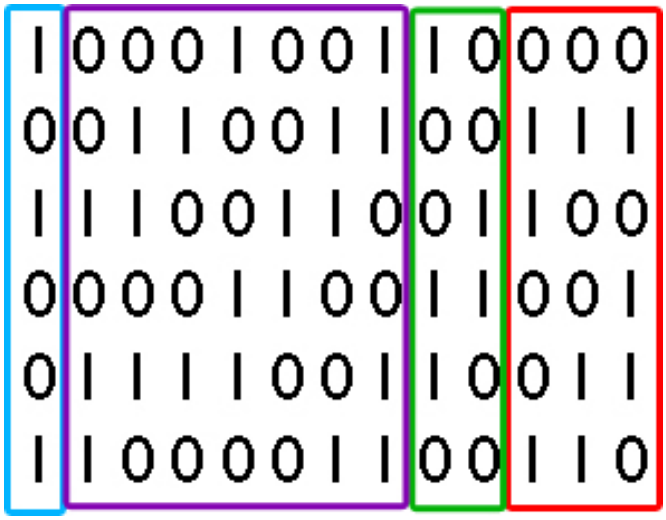


Figure 5.11: Musical interpretation of the cellular automata

The blue rectangle determines whether or not the note should be played, where 1 signifies that the note should be played, and 0 signifies that no note should be played (in musical terms, this is a rest). The notes on the scale are mapped to the numbers in the purple rectangle. If multiple pitches can be played, a selection is picked randomly. The green rectangle determines the velocity of the note played. We have 4 levels of velocities that can be selected; from 00 meaning a low velocity, to 11 which is a high velocity. The red rectangle is used in a certain music generation mode where the length of the note can be determined from the number, where 000 and 001 signify minims (half notes), 010 and 011

signify crotchets (quarter notes), 100 and 101 signify quavers (eighth notes) and 110 and 111 signify semiquavers (sixteenth notes).

In the examples that we have generated, we create three different cellular automata: one for a bass instrument, one for a rhythm instrument and another for a lead instrument. The instruments may change scales over time, as well as the type of music generation mode they use.

5.4 Prototype

The following section describes the prototype that was made.

5.4.1 Game Environment

Although the focus of our project is a music generation algorithm, we required a world environment in order to test our algorithm. Fig. 5.14 illustrates a city environment made by Ioana Marin in 3ds Max 2012 as part of a separate project.

The city environment consists of three main areas, as illustrated in Fig. 5.12. Fig. 5.12a shows the area containing graffiti, which is located in the east. Fig. 5.12b illustrates the park area, which also contains several colour→music paintings by some of the artists described in the Literature Review in Section 4.1. This is located in the centre of the city. Finally, Fig. 5.12c shows a section of the city that contains brightly coloured buildings, as well as a section with contains vivid neon signs.

When starting the prototype, players first encounter the Main Menu screen, as can be seen in Fig. 5.13a. Clicking on the Instructions button will take them to the Instructions screen where they can read more about the project (as illustrated in Fig. 5.13b), while clicking on the Begin button will take them to the Mappings Selection screen (as shown in Fig. 5.13c). Here, players may select the colour→music mapping they wish to explore by clicking on it. Clicking on the Begin button in this screen will then take them to a first person view of the city environment, where they may walk around and explore the city while listening to the generated music.

5.4.2 Instruments Used

For our project, we used 3 different cellular automata as instruments. Here are the settings we used:

- **Bass Cellular Automata**
- *Instrument:* Contrabass

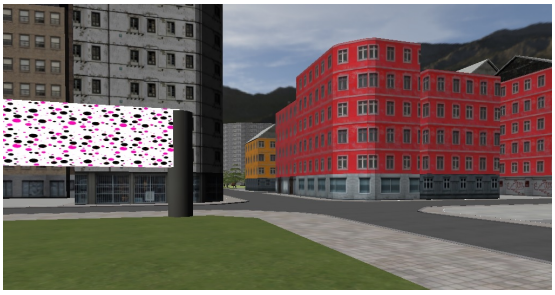
- *Rule:* 11
- *Octave:* 3
- *Music Gen. Mode:* Each Note Made From Array
- *Pitch Map. Mode:* Note to Pitch Direct Correspondence
- *Velocity Map. Mode:* Four Level Pitch
- *Conductor Mode:* No Conductor Mode

- **Rhythm Cellular Automata**
- *Instrument:* Viola
- *Rule:* 10
- *Octave:* 4
- *Music Gen. Mode:* Each Note Solely Quavers
- *Pitch Map. Mode:* Note to Pitch Direct Correspondence
- *Velocity Map. Mode:* Four Level Pitch
- *Conductor Mode:* No Conductor Mode

- **Lead Cellular Automata**
- *Instrument:* Trumpet
- *Rule:* 30
- *Octave:* 4
- *Music Gen. Mode:* Each Note Solely Semiquavers
- *Pitch Map. Mode:* Note to Pitch Direct Correspondence
- *Velocity Map. Mode:* Four Level Pitch
- *Conductor Mode:* No Conductor Mode

(a) Graffiti area

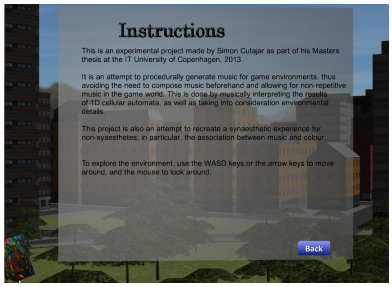
(b) Park area



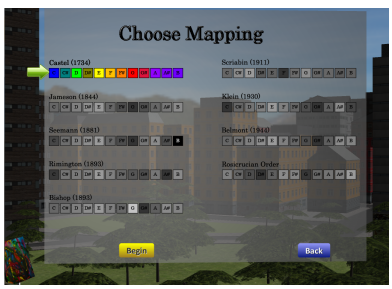
(c) Coloured buildings area

Figure 5.12: Main areas of the city

(a) Main Menu screen



(b) Instructions screen



(c) Mappings Selection screen

Figure 5.13: Prototype Screens

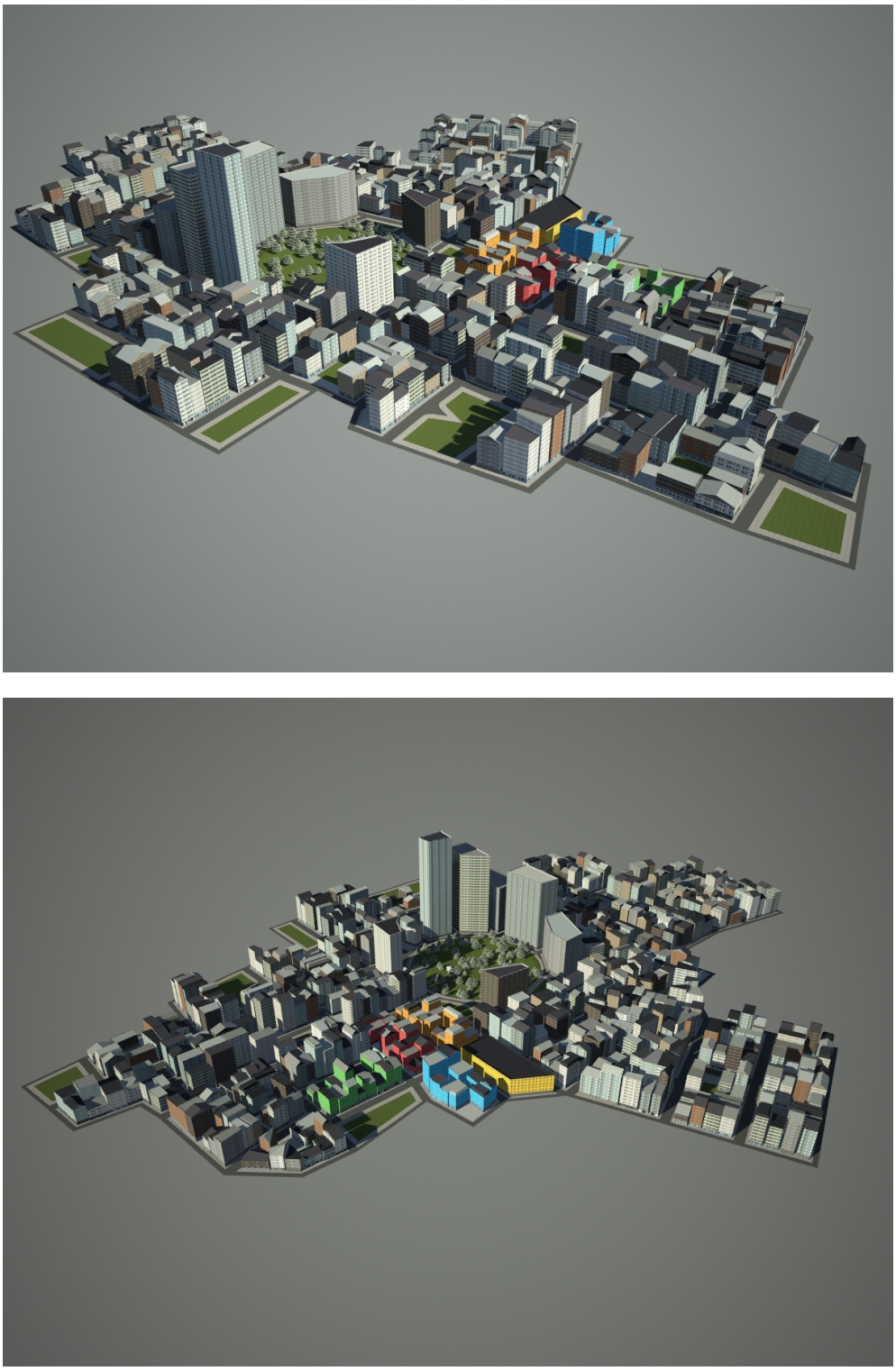


Figure 5.14: City Render

6. Results

In this section, we shall discuss the results of our algorithm and how it functioned using different mappings in different areas of the city.

The tests below describe the output of the algorithm according to certain views in the city. These tests are taken while remaining still, since the results would change drastically if the character would move around (since different colours would provide a different input to the algorithm). The output includes the colours found, the notes that are mapped to these colours, the relevant scales that include these notes and the final chosen scale.

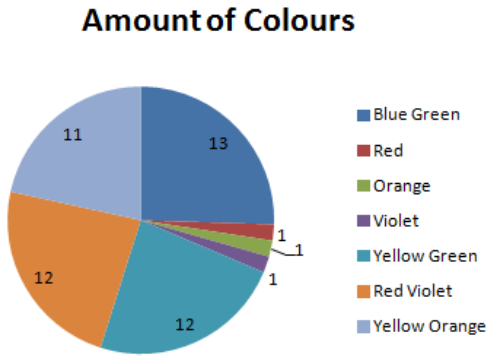
A list of relevant scales and mappings can be found in the Appendix in Sections A.1 and B.1 respectively. A reference of what colours we are referring to in the Colours column of Tables 6.1, 6.2 and 6.3 can also be found in the Appendix in Section B.5.

6.1 *A Painted Picture of the Universe* by Roy de Maistre

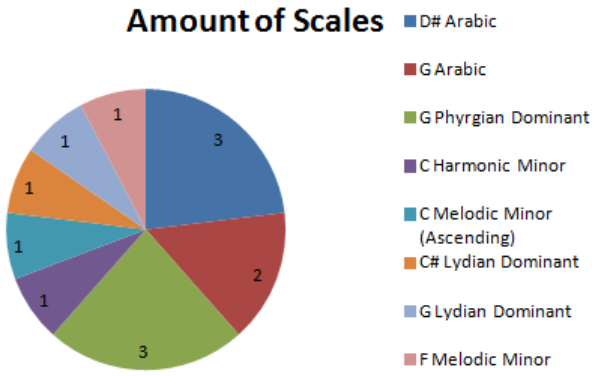
Figure 6.1: Test 1: Viewing *A Painted Picture of the Universe* by Roy de Maistre

We first tried out our algorithm on a painting by Roy de Maistre titled *A Painted Picture of the Universe*. The view we picked can be seen in Fig. 6.1. We picked the Belmont mapping for our test.

Table 6.1 shows the results obtained after looking at the painting in the game environment for some time. Fig. 6.2a shows the colours that were identified in the screen, and their amount. As we can see, the algorithm detects a large amount of blue green, yellow green, red violet and yellow orange, since it also takes into account the buildings and the grass around the painting. Fig. 6.2b illustrates which scales were chosen the most from the available colours; which in this case, happen to be the G Phyrgian Dominant and D# Arabic scales.



(a) Identified Colours, and Amount



(b) Scales

Figure 6.2: Test 1: Analyzing Results

Colours Found	Notes	Relevant Scales	Chosen Scale
Blue Green, Red, Orange, Violet	G, C, D, A#	14	F Melodic Minor
Yellow Green, Blue Green, Red Violet	F, G, B	15	G Lydian Dominant
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	G Arabic
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	G Arabic
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	G Phyrgian Dominant
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	C Harmonic Minor
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	G Phyrgian Dominant
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	C Melodic Minor (Ascending)
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	C# Lydian Dominant
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	G Phyrgian Dominant
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	D# Arabic
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	D# Arabic
Yellow Orange, Yellow Green, Blue Green, Red Violet	D#, F, G, B	9	D# Arabic

Table 6.1: Test 1 Results

6.2 Coloured Buildings

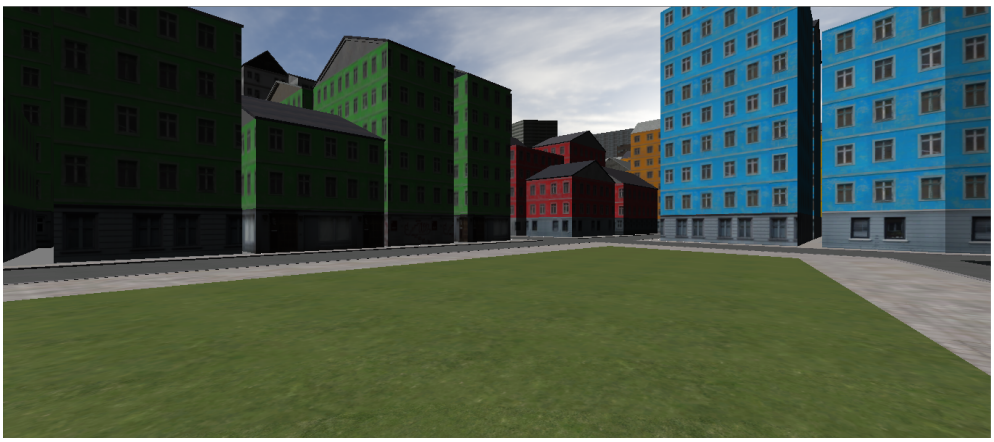
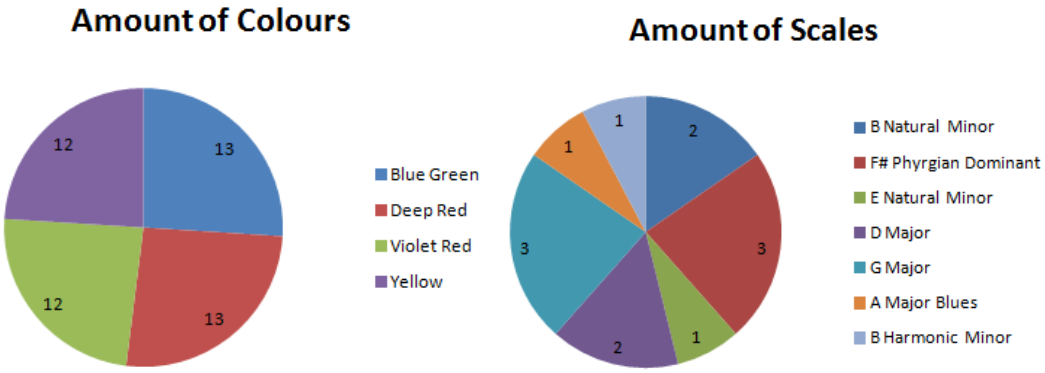


Figure 6.3: Test 2: Viewing the Coloured Buildings in the City

We then navigated to the area in the city containing buildings, as can be seen in Fig. 6.3. We picked the Rosicrucian mapping for this test.

Table 6.2 shows the results obtained. Fig. 6.4a shows the colours that were identified in the screen, and their amount. In this case, the algorithm detected a equal amount of blue green, deep red, violet red and yellow. Fig. 6.4b illustrates which scales were chosen the most from the available colours. The F# Phyrgian Dominant and G Major scales are the ones that appear the most, but we seem to have a lot of variety with the scales chosen.



(a) Identified Colours, and Amount (b) Scales

Figure 6.4: Test 2: Analyzing Results

Colours Found	Notes	Relevant Scales	Chosen Scale
Deep Red, Blue Green	G, D	36	F# Phyrgian Dominant
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	B Natural Minor
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	F# Phyrgian Dominant
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	E Natural Minor
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	D Major
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	G Major
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	A Major Blues
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	G Major
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	F# Phyrgian Dominant
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	D Major
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	G Major
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	B Harmonic Minor
Blue Green, Deep Red, Violet Red, Yellow	D, G, F#, B	7	B Natural Minor

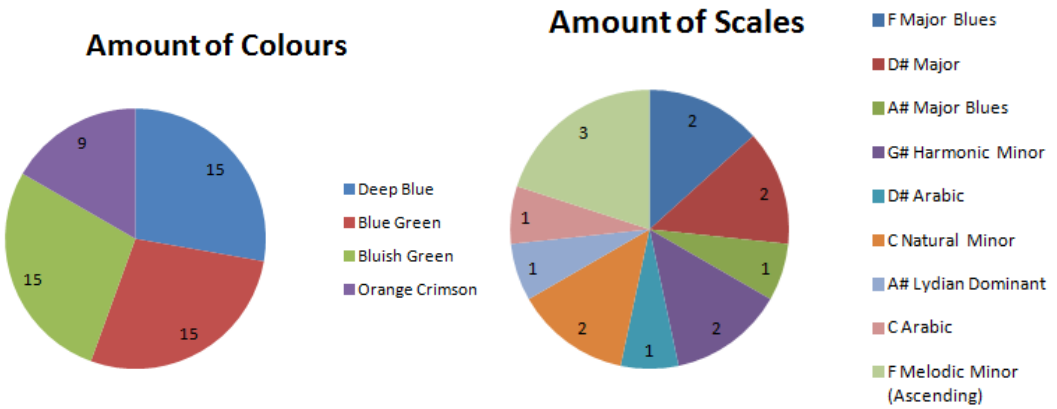
Table 6.2: Test 2 Results

6.3 Graffiti

Figure 6.5: Test 3: Viewing Graffiti

Our next test was to try out the algorithm in the area of the city containing graffiti, as seen in Fig. 6.5. We chose to use the Rimington mapping for this test.

Table 6.3 shows the results obtained for this test. Fig. 6.6a shows the colours that were identified in the screen, as well as their amount. Here, the most common colours seem to be Deep Blue, Bluish Green and Blue Green, even though most of the screen is greyish in colour. Fig. 6.4b shows that a large variety of scales seem to have been chosen while viewing the graffiti.



(a) Identified Colours, and Amount (b) Scales

Figure 6.6: Test 3: Analyzing Results

Colours Found	Notes	Relevant Scales	Chosen Scale
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	F Major Blues
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	D# Major
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	A# Major Blues
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	G# Harmonic Minor
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	G# Harmonic Minor
Deep Blue, Blue Green, Bluish Green	A#, G#, G	16	D# Arabic
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	F Major Blues
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	C Natural Minor
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	A# Lydian Dominant
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	D# Major
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	C Natural Minor
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	C Arabic
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	F Melodic Minor (Ascending)
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	F Melodic Minor (Ascending)
Deep Blue, Blue Green, Bluish Green, Orange Crimson	G, D, G#, A#	6	F Melodic Minor (Ascending)

Table 6.3: Test 3 Results

7. Future Work

In this section, we will be discussing potential improvements for our project, such as stuff that could be tweaked, added or improved.

7.1 Improving the Music Generation Algorithm

One area that might require future work is the musical interpretation of the cellular automata results. At the moment, we directly interpret the cellular automata results musically as explained in Section 5.3.4. Fig. 5.11 for example, explains the interpretations we chose, but these may not necessarily be the best interpretations and are open to experimentation. Other interpretations could be considered, such as those described in Beyls [1989], including multiple automata describing one instrument and having automata with memories.

Another area we could improve upon is the use of automata settings. At the moment, there are only a limited amount of options for each setting, and some only have one option. This could be improved on by adding more options to each setting, and possibly adding more settings. These could then be used further in the music generation algorithm, such as switching between settings depending on where the player is in the 3D world.

At the moment, the music generation system contains 3 cellular automata that represent 3 instruments (lead, rhythm and bass). Each instrument generates a melody regardless of what the other instrument is doing. A possible upgrade to the system would involve a hierarchical creation of music. For example, one situation could be where the bass first generates a melody, the rhythm generates a melody based on the bass, and the lead generates a melody based on the rhythm. This would be consider a "bottom-up" approach. A "top-down" approach would first generate the lead melody, and then have a rhythm line fit the generated lead melody, and have a bass line fit the generated rhythm line. More complex systems may be introduced if multiple instruments are used; one could potentially have a whole orchestra of instruments that start or stop based on what other members of the orchestra are doing.

Although we emphasized particularly on the use of structure in generated music, we only made use of low level structure in our project (through the use of the ordering of notes in bars). We believe that with a higher levels of structure, the music generated could be further improvement, by for example taking in to account valleys and peaks in the music, or by using conventional song structure such as choruses and verses.

We also intentionally chose certain rule classes and rules for each instrument. In our system, the bass instrument generated music using rule 11, a class 2 rule. The rhythm instrument also generated music using rule 10, another class 2 rule, while the lead instrument generated music using rule 30, a class 3 rule. Once initialized, these rules as the player explores the 3D environment. By changing these rules, the music generated could become more varied and more interested.

Another similar example is the octaves that instruments are playing in. When created, the bass instrument is set to play in octave 3, while the rhythm and lead instruments are set to play in octave 4. In comparison, middle C is referred to as C_4. These are not changed after initialization, and could result in more interesting music. Furthermore, when an instrument is assigned an octave, it can only play the 12 notes in that octave (from C to B), and is unable to play any notes higher or lower than that particular range. This might be changed in the future in order to allow a wider range of notes.

In this project, we focused only on the use of 1D cellular automata. Most of the literature discusses the use of 2D cellular automata, which we did not focus on. Future work might include a comparison between a music generation algorithm that uses 1D cellular automata, and an implementation that uses 2D cellular automata. This would mean that new musical interpretations would have to be devised, possibly inspired by the systems explained in Millen [1990], Jewell [2007], Miranda [2003]. There are very few systems that use 3D cellular automata to generate music (the most prominent one being CAMUS 3D), but this is also something that can used in a comparison between suitable musical cellular automata systems.

Another area of future work could be the type of cellular automata being used. At the moment, we are simply using the one-dimensional cellular automata rules explained by Stephen Wolfram in Wolfram [1984]. However, there are different rules that can be used, especially in higher dimension automata. Jewell [2007] uses the rules given in Conway's Game of Life, as well as the rules from the Demon Cyclic Space. One could potentially use continuous automata, where instead of representing states using discrete numbers (1 and 0), real numbers are used. Comparison between multiple rule systems in various different cellular automata systems and their musical interpretation may also be considered future

work.

One could also try and use external VST instruments to generate the music, instead of simply using the default MIDI library. A preliminary search shows that this can be done by writing a host in C# in order to communicate with VST instruments, possibly by using VST.NET[1]. The host would serve as an in-between, receiving MIDI information from the cellular automata and sending it to the VST to be played.

Finally, in our project, we considered the use of the music generation algorithm in 3D game environments. We have not considered the use of the music generation algorithm in 2D environments however. Another possible application might be the generation of music in other parts of the game, such as menu screens and cutscenes.

7.2 Improving the Mappings

Apart from the technical future work that may be done, there are also improvements that may be done to the mappings. Most of the project was focused on the mappings between colour and music, but there are several other mappings that are explained in the Literature Review in Section 4.2 (such as space→music and emotion→music). These have not been included in the final implementation, but have been left for future work.

To extract the relevant colours from the viewport, we currently use a simple system of extracting pixel colours. This is done every certain amount of pixels through the viewport. This system can obviously be improved on. One way of doing so is to inherently embed colours into the objects that are visible on screen. The system would then determine which objects were visible in the camera's frustum and use the colours of those objects. Some sort of image processing algorithm could also potentially be used.

Another potential improvement would be to also consider input from around the player, as opposed to just the front side. By weighting the inputs to the player, there could still be a small influence from other surrounding objects, such as behind the player.

Although we took it into consideration during the implementation, we didn't end up implementing the ability for synaesthetes to add their own mappings to the predefined list. It would be interesting to test whether the already existing mappings taken from theory clash with synaesthetes' mappings. Tests with non-synaesthetes are also planned. Another possibility is to include synaesthetes'

[1]Found at `http://vstnet.codeplex.com/`

mappings in the program and see if the music generated matches the colours that they are able to see.

8. Conclusions

In this chapter, we conclude our project by summarizing what we did, as well as revisiting the original goal of the project.

8.1 Summary of Work

We first took a look at already existing literature relevant to music generation. We first focused on a general search, considered all algorithms that were relevant (ranging from neural networks to swarm-based systems), and eventually settled on literature that discussed music generation using cellular automata. The bulk of the literature focused on 2D cellular automata, but some literature also focuses on different types of cellular automata, as well as how to interpret them musically. We also took a look at literature that was relevant to mappings involving music and other domains. In particular, we looked at music→colour mappings in a lot of detail, but we also investigated space→music and emotion→music mapping.

We then designed and implemented a system in Unity using C# that could procedurally generate music. This was done by having each cellular automaton act as an instrument, and be controlled by an externally conductor. Depending on the settings it was initialized with, the automaton would generate the appropriate line. We also worked on a way to extract colour information from the screen in order to properly choose the appropriate scale to play, which was done by comparing extracted colours to the colours associated to various scales according to the chosen mapping.

Finally, we took at look at the results of the algorithm in a city environment, and we determined the future work that could be done on the project by outlining its weaknesses and omissions and what could be improved.

8.2 Revisiting the Goals

Having discussed our goals for the project in Section 1.2, we revisit them in order to see whether or not they were met successfully.

Our first goal described creating an music generation algorithm for game environments. We believe that we have successfully created such an algorithm, as it continuously generates music in a 3D city environment. It is also flexible, because as the player explores the city, the music changes to fit what the player is seeing.

Our second goal described the exploration of music generation using 1D cellular automata, as well combining the music generation algorithm with music→colour mappings. We believe that we have met this goal, as explained in the Implementation chapter (Chapter 5).

The third goal was to provide a synaesthetic experience for people that do not experience music→colour synaesthesia. Although we managed to provide what we think is such as an experience, we did not test it with proper synaesthetes and do not know if their experience is accurately reflected.

8.3 Conclusions

The dissertation was an exploratory attempt in having algorithms that generate music for game environments, as well as algorithms using external factors available in the environment according to some sort of mapping. Although there exists literature that discusses music generation, and literature that discusses music→colour mappings, we are not aware of any literature that combines the two. We feel that their successful combination could prove to be a stepping stone for similar future work.

Part III

Appendix

A. Music Generation

A.1 List of Scales

A.1.1 Major Scales

C Major

C, D, E, F, G, A, B

C♯ Major

C♯, D♯, F, F♯, G♯, A♯, C

D Major

D, E, F♯, G, A, B, C♯

D♯ Major

D♯, F, G, G♯, A♯, C, D

E Major

E, F♯, G♯, A, B, C♯, D♯

F Major

F, G, A, A♯, C, D, E

F♯ Major

F♯, G♯, A♯, B, C♯, D♯, F

G Major

G, A, B, C, D, E, F♯

G♯ Major

G♯, A♯, C, C♯, D♯, F, G

A Major

A, B, C♯, D, E, F♯, G♯

A♯ Major

A♯, C, D, D♯, F, G, A

B Major

B, C♯, D♯, E, F♯, G♯, A♯

A.1.2 Natural Minor Scales

C Natural Minor

C, D, D♯, F, G, G♯, A♯

C♯ Natural Minor

C♯, D♯, E, F♯, G♯, A, B

D Natural Minor

D, E, F, G, A, A♯, C

D♯ Natural Minor

D♯, F, F♯, G♯, A♯, B, C♯

E Natural Minor

E, F♯, G, A, B, C, D

F Natural Minor

F, G, G♯, A♯, C, C♯, D♯

F♯ Natural Minor

F♯, G♯, A, B, C♯, D, E

G Natural Minor

G, A, A♯, C, D, D♯, F

G♯ Natural Minor

G♯, A♯, B, C♯, D♯, E, F♯

A Natural Minor

A, B, C, D, E, F, G

A♯ Natural Minor

A♯, C, C♯, D♯, F, F♯, G♯

B Natural Minor

B, C♯, D, E, F♯, G, A

A.1.3 Harmonic Minor Scales

C Harmonic Minor

C, D, D♯, F, G, G♯, B

C♯ Harmonic Minor

C♯, D♯, E, F♯, G♯, A, C

D Harmonic Minor

D, E, F, G, A, A♯, C♯

D♯ Harmonic Minor

D♯, F, F♯, G♯, A♯, B, D

E Harmonic Minor

E, F♯, G, A, B, C, D♯

F Harmonic Minor

F, G, G♯, A♯, C, C♯, E

F♯ Harmonic Minor

F♯, G♯, A, B, C♯, D, F

G Harmonic Minor

G, A, A♯, C, D, D♯, F♯

G♯ Harmonic Minor

G♯, A♯, B, C♯, D♯, E, G

A Harmonic Minor

A, B, C, D, E, F, G♯

A♯ Harmonic Minor

A♯, C, C♯, D♯, F, F♯, A

B Harmonic Minor

B, C♯, D, E, F♯, G, A♯

A.1.4 Melodic Minor Ascending Scales

C Melodic Minor Ascending

C, D, D♯, F, G, A, B

C♯ Melodic Minor Ascending

C♯, D♯, E, F♯, G♯, A♯, C

D Melodic Minor Ascending

D, E, F, G, A, B, C♯

D♯ Melodic Minor Ascending

D♯, F, F♯, G♯, A♯, C, D

E Melodic Minor Ascending

E, F♯, G, A, B, C♯, D♯

F Melodic Minor Ascending

F, G, G♯, A♯, C, D, E

F♯ Melodic Minor Ascending

F♯, G♯, A, B, C♯, D♯, F

G Melodic Minor Ascending

G, A, A♯, C, D, E, F♯

G♯ Melodic Minor Ascending

G♯, A♯, B, C♯, D♯, F, G

A Melodic Minor Ascending

A, B, C, D, E, F♯, G♯

A♯ Melodic Minor Ascending

A♯, C, C♯, D♯, F, G, A

B Melodic Minor Ascending

B, C♯, D, E, F♯, G♯, A♯

A.1.5 Major Blues Scales

C Major Blues

C, D, D♯, F, F♯, A, A♯

C♯ Major Blues

C♯, D♯, E, F♯, G, A♯, B

D Major Blues

D, E, F, G, G♯, B, C

D♯ Major Blues

D♯, F, F♯, G♯, A, C, C♯

E Major Blues

E, F♯, G, A, A♯, C♯, D,

F Major Blues

F, G, G♯, A♯, B, D, D♯

F♯ Major Blues

F♯, G♯, A, B, C, D♯, E

G Major Blues

G, A, A♯, C, C♯, E, F

G♯ Major Blues

G♯, A♯, B, C♯, D, F, F♯

A Major Blues

A, B, C, D, D♯, F♯, G

A♯ Major Blues

A♯, C, C♯, D♯, E, G, G♯

B Major Blues

B, C♯, D, E, F, G♯, A

A.1.6 Arabic Scales

C Arabic

C, D, E, F, G, G♯, A♯

C♯ Arabic

C♯, D♯, F, F♯, G♯, A, B

D Arabic

D, E, F♯, G, A, A♯, C

D♯ Arabic

D♯, F, G, G♯, A♯, B, C♯

E Arabic

E, F♯, G♯, A, B, C, D

F Arabic

F, G, A, A♯, C, C♯, D♯

F♯ Arabic

F♯, G♯, A♯, B, C♯, D, E

G Arabic

G, A, B, C, D, D♯, F

G♯ Arabic

G♯, A♯, C, C♯, D♯, E, F♯

A Arabic

A, B, C♯, D, E, F, G

A♯ Arabic

A♯, C, D, D♯, F, F♯, G♯

B Arabic

B, C♯, D♯, E, F♯, G, A

A.1.7 Phyrgian Dominant Scales

C Phyrgian Dominant

C, C♯, E, F, G, G♯, A♯

C♯ Phyrgian Dominant

C♯, D, F, F♯, G♯, A, B

D Phyrgian Dominant

D, D♯, F♯, G, A, A♯, C

D♯ Phyrgian Dominant

D♯, E, G, G♯, A♯, B, C♯

E Phyrgian Dominant

E, F, G♯, A, B, C, D

F Phyrgian Dominant

F, F♯, A, A♯, C, C♯, D♯

F♯ Phyrgian Dominant

F♯, G, A♯, B, C♯, D, E

G Phyrgian Dominant

G, G♯, B, C, D, D♯, F

G♯ Phyrgian Dominant

G♯, A, C, C♯, D♯, E, F♯

A Phyrgian Dominant

A, A♯, C♯, D, E, F, G

A♯ Phyrgian Dominant

A♯, B, D, D♯, F, F♯, G♯

B Phyrgian Dominant

B, C, D♯, E, F♯, G, A

A.1.8 Lydian Dominant Scales

C Lydian Dominant

C, D, E, F♯, G, A, A♯

C♯ Lydian Dominant

C♯, D♯, F, G, G♯, A♯, B

D Lydian Dominant

D, E, F♯, G♯, A, B, C

D♯ Lydian Dominant

D♯, F, G, A, A♯, C, C♯

E Lydian Dominant

E, F♯, G♯, A♯, B, C♯, D

F Lydian Dominant

F, G, A, B, C, D, D♯

F♯ Lydian Dominant

F♯, G♯, A♯, C, C♯, D♯, E

G Lydian Dominant

G, A, B, C♯, D, E, F

G♯ Lydian Dominant

G♯, A♯, C, D, D♯, F, F♯

A Lydian Dominant

A, B, C♯, D♯, E, F♯, G

A♯ Lydian Dominant

A♯, C, D, E, F, G, G♯

B Lydian Dominant

B, C♯, D♯, F, F♯, G♯, A

B. Music→Colour Mappings

B.1 Music→Colour Mappings

Newton — 1704

- C — Red
- D — Orange
- E — Yellow
- F — Green
- G — Blue
- A — Indigo
- B — Violet

Castel — 1734

- C — Blue
- C♯ — Blue Green
- D — Green
- D♯ — Olive Green
- E — Yellow
- F — Yellow Orange
- F♯ — Orange
- G — Red
- G♯ — Crimson
- A — Violet
- A♯ — Agate
- B — Indigo

Field — 1816

- C — Blue
- D — Purple
- E — Red
- F — Orange
- G — Yellow
- A — Yellow Green
- B — Green

Jameson — 1844

- C — Red
- C♯ — Red Orange
- D — Orange
- D♯ — Orange Yellow
- E — Yellow
- F — Green
- F♯ — Green Blue

- G — Blue
- G♯ — Blue Purple
- A — Purple
- A♯ — Purple Violet
- B — Violet

Seemann — 1881

- C — Carmine
- C♯ — Scarlet
- D — Orange
- D♯ — Yellow Orange
- E — Yellow
- F — Green
- F♯ — Green Blue
- G — Blue
- G♯ — Indigo
- A — Violet
- A♯ — Brown
- B — Black

Charles Fourier — 1704

- C — Violet
- D — Indigo
- E — Azure
- F — Green
- G — Yellow
- A — Orange
- B — Red

Rimington — 1893

- C — Deep Red
- C♯ — Crimson
- D — Orange Crimson
- D♯ — Orange
- E — Yellow
- F — Yellow Green
- F♯ — Green
- G — Bluish Green
- G♯ — Blue Green
- A — Indigo
- A♯ — Deep Blue
- B — Violet

Bishop — 1893

- C — Red
- C♯ — Red Orange
- D — Orange
- D♯ — Orange Yellow
- E — Yellow
- F — Yellow Green
- F♯ — Green
- G — Aquamarine
- G♯ — Blue
- A — Blue Violet
- A♯ — Violet
- B — Violet Red

Scriabin — 1911

- C — Red
- C♯ — Violet
- D — Yellow
- D♯ — Steely
- E — Pearly Blue
- F — Dark Red
- F♯ — Bright Blue
- G — Rosy Orange
- G♯ — Purple
- A — Green
- A♯ — Steely
- B — Pearly Blue

Klein — 1930

- C — Dark Red
- C♯ — Red
- D — Red Orange
- D♯ — Orange
- E — Yellow
- F — Yellow Green
- F♯ — Green
- G — Blue Green
- G♯ — Blue
- A — Blue Violet
- A♯ — Violet
- B — Dark Violet

Aeppli — 1940

- C — Red
- D — Orange
- E — Yellow
- F♯ — Green
- G — Blue Green
- A — Ultramarine Blue
- A♯ — Violet
- B — Purple

Belmont — 1944

- C — Red
- C♯ — Red Orange
- D — Orange
- D♯ — Yellow Orange
- E — Yellow
- F — Yellow Green
- F♯ — Green
- G — Blue Green
- G♯ — Blue
- A — Blue Violet
- A♯ — Violet
- B — Red Violet

Rosicrucian Order

- C — Yellow Green
- C♯ — Green
- D — Blue/Green
- D♯ — Blue
- E — Blue/Violet
- F — Violet
- F♯ — Violet/Red
- G — Deep Red
- G♯ — Red
- A — Red Orange
- A♯ — Orange
- B — Yellow

Figure B.1: Music→Colour Mappings Table

B.2 Colour→Key Mapping

Amy Beach

- **A♭ Major** — Blue
- **A Major** — Green
- **C Major** — White
- **D♭ Major** — Violet
- **E♭ Major** — Pink
- **E Major** — Yellow
- **G Major** — Red
- **F♯ Minor** — Black
- **G♯ Minor** — Black

Rimsky-Korsakov

- **A♭ Major** — Grayish Violet
- **A Major** — Rosy
- **B Major** — Dark Blue
- **C Major** — White
- **D♭ Major** — Dusky
- **D Major** — Yellow
- **E♭ Major** — Steely Blue
- **E Major** — Sapphire Blue
- **F♯ Major** — Grayish Green
- **G Major** — Brownish Gold

Scriabin

- **A♭ Major** — Purple Violet
- **A Major** — Green
- **B Major** — Bluish White
- **C Major** — Red
- **D♭ Major** — Violet
- **D Major** — Yellow
- **E♭ Major** — Steely Blue
- **E Major** — Bluish White
- **F♯ Major** — Bright Blue
- **G Major** — Orange Rose

B.3 Scientific Music→Colour Mapping

Note	Hertz	Equivalent Wavelength in Angstroms/10	Approximate Colour
A	440	619.69	Orange-Yellow
A♯	457.75	595.66	Yellow-Orange
B♭	472.27	577.34	Yellow
B	491.32	554.95	Yellow-Green
C♭	506.91	537.89	Green-Yellow
B♯	511.13	533.44	Green
C	527.35	517.03	Green
C♯	548.62	496.99	Green-Blue
D♭	566.03	481.7	Blue-Green
D	588.86	463.03	Blue
D♯	612.61	445.08	Blue-Violet
E♭	632.05	431.39	Violet-Blue
E	657.54	414.67	Violet
F♭	678.41	401.91	Ultra Violet
E♯	684.06	398.59	Invisible Violet
F	705.77	772.66	Invisible Red
F♯	734.23	742.71	Infra Red
G♭	757.53	719.86	Red
G	788.08	691.96	Red-Orange
G♯	819.87	665.13	Orange-Red
A♭	845.89	644.67	Orange

B.4 Interval→Colour Mapping

Athanasius Kircher

- Octave — Green
- Seventh — Blue Violet
- Major Sixth — Fire Red
- Minor Sixth — Red Violet
- Augmented Fifth — Dark Brown
- Fifth — Gold
- Diminished Fifth — Blue
- Fourth — Brown Yellow
- Major Third — Bright Red
- Minor Third — Gold
- Major Wholetone — Black
- Minor Second — White
- Minor Wholetone — Grey

B.5 Quantized Colours

The list below represents all the colours available in the music→ mappings, as can be seen in the appendix in Section B.1. The numbers associated with each colour is the colour's representation in RGB space.

Agate — 127, 0, 255
Aquamarine — 127, 255, 212
Azure — 0, 127, 255
Black — 0, 0, 0
Blue — 0, 0, 255
Blue Green/Green Blue — 0, 127, 127
Blue Purple — 64, 0, 191
Blue Violet — 71, 0, 255
Bluish Green — 0, 150, 127
Bright Blue — 80, 53, 242
Brown — 151, 72, 7
Carmine — 150, 0, 25
Cayan Blue — 6, 128, 175
Crimson — 220, 20, 60
Dark Red — 127, 0, 0
Dark Violet — 71, 0, 127
Deep Blue — 30, 38, 123
Deep Red — 134, 5, 0
Green — 0, 255, 0
Grey — 128, 128, 128
Indigo — 111, 0, 255
Olive Green — 128, 128, 0
Orange — 255, 127, 0
Orange Crimson — 237, 73, 30
Pearly Blue — 18, 106, 229
Purple — 128, 0, 128
Purple Violet — 135, 0, 191
Red — 255, 0, 0
Red Orange — 255, 63, 0
Rose — 255, 0, 127
Rosy Orange — 255, 63, 63
Scarlet — 255, 36, 0
Steely — 64, 64, 191
Ultramarine Blue — 18, 10, 143
Violet — 143, 0, 255
Violet Red/Red Violet — 199, 0, 127
Yellow — 255, 255, 0
Yellow Green — 127, 255, 0
Yellow Orange/Orange Yellow — 255, 191, 0
Yellow White — 255, 255, 127

References

R. Ashley. Musical pitch space across modalities: Spatial and other mappings through language and culture. In *Proceedings of the 8th International Conference on Music Perception and Cognition*, pages 64–71, 2004.

R. Barthes. The death of the author. *Aspen*, (5-6), 1967.

R. Becker. Genetic music. Master's thesis, Rochester Institute of Technology, 2005.

P. Beyls. The musical universe of cellular automata. In *Proceedings of the 1989 International Computer Music Conference*, 1989.

D. Birchfield. Generative model for the creation of musical emotion, meaning, and form. *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence - ETP '03*, 2003.

D. Bisig and M. Neukom. Swarm based computer music - towards a repertory of strategies. In *Proceedings of the Generative Art Conference*, 2008.

T. Blackwell. Swarming and music. *Evolutionary Computer Music*, pages 194–217, 2007.

J. Bonde. Sound-generated space. In *2008 Young Investigator's Forum on Culture Technology*, pages 34–40. KAIST, 2008.

A. R. Brown. Exploring rhythmic automata. In *EC'05 Proceedings of the 3rd European conference on Applications of Evolutionary Computing*, pages 551–556, 2005.

A. R. Brown and T. Kerr. Adaptive music techniques. *Improvise: The Australasian Computer Music Conference*, 2009.

D. Burraston and E. Edmonds. Cellular automata in generative electronic music and sonic art: a historical and technical review. *Digital Creativity*, 16(3):165–185, 2005.

D. Burraston, E. Edmonds, D. Livingstone, and E. R. Miranda. Cellular automata in MIDI based computer music. In *Proceedings of the 2004 International Computer Music Conference*, pages 71–78. International Computer Music Association, 2004.

K. Collins. An introduction to procedural music in video games. *Contemporary Music Review*, 28(1):5–15, 2009.

F. Collopy. Correspondences. RhythmicLight.com, October 2001. `http://rhythmiclight.com/archives/ideas/correspondences.html`.

F. Collopy. Playing (with) color. *Glimpse: The Art and Science of Seeing*, 2(3): 62–67, 2009.

P. Ekman, W. V. Friesen, and P. Ellsworth. *Emotion in the Human Face: Guidelines for Research and an Integration of Findings*. Pergamon Press, 1972.

H. A. García Salas, A. Gelbukh, H. Calvo, and F. G. Soria. Automatic music composition with simple probabilistic generative grammars. *Polibits*, 44:59–65, 2011.

K. Gerstner. *Forms of Color: The Interaction of Visual Elements*. The MIT Press, 1990.

G. Haus and A. Sametti. Scoresynth: a system for the synthesis of music scores based on petri nets and a music algebra. *Computer*, 24(7):56–60, 1991.

R. Hinojosa Chapel. *Realtime algorithmic music systems from fractals and chaotic functions: toward an active musical instrument*. PhD thesis, Universitat Pompeu Fabra, 2003.

J. Howell. Painting with Color Scales. Joey Howell – http://www.joeyhowell.com/. `http://www.joeyhowell.com/PaintingWithColorScales.pdf`.

P. Husbands, P. Copley, A. Eldridge, and J. Mandelis. An introduction to evolutionary computing for musicians. *Evolutionary Computer Music*, 2007.

N. Hutchison. Colour Music in Australia: De-mystifying De Maistre. Colour Music – http://home.vicnet.net.au/ colmusic/, 1997. `http://home.vicnet.net.au/~colmusic/maistre.htm`.

B. L. Jacob. Composing with genetic algorithms. In *Proceedings of the International Computer Music Conference*, 1995.

H. Järveläinen. Algorithmic musical composition. Tik-111.080 Seminar on content creation, Spring 2000.

J. Jewanski. Color Organs. See This Sound – www.see-this-sound.at, a. `http://www.see-this-sound.at/print/69`.

J. Jewanski. ColorTone Analogies. See This Sound – www.see-this-sound.at, b. `http://www.see-this-sound.at/print/43`.

M. O. Jewell. *Motivated music: Automatic soundtrack generation for film*. PhD thesis, University of Southampton, 2007.

W. Kandinsky. *Point and Line to Plane*. Dover Publications, 1979 (originally published in 1926).

W. Kandinsky. *Concerning the Spiritual in Art*. Empire Books, 2011 (originally published in 1912).

T. F. Karwoski and H. S. Odbert. Color-music. *Psychological Monographs*, 50 (2), 1938.

G. Kepes. *Language of Vision*. Dover Publications, 1995 (originally published in 1944).

E. Key and D. Kanaga. *Proteus*. [PC], 2013. Video Game.

P. S. Langston. (201) 644-2332 Eedie & Eddie on the Wire, An Experiment in Music Generation. In *Proceedings of the Usenix Summer'86 Conference*, pages 1–14, 1986.

P. S. Langston. Six techniques for algorithmic music composition. *15th International Computer Music Conference (ICMC)*, 1989.

S. Le Groux. Towards an emotion-driven interactive music system: ridging the gaps between affect, physiology and sound generation. In *6th Sound and Music Computing Conference*, 2009.

W. Li, N. H. Packard, and C. G. Langton. Transition phenomena in cellular automata rule space. *Physica D: Nonlinear Phenomena*, 45(1–3):77–94, 1990.

S. R. Livingstone and A. R. Brown. Dynamic response: real-time adaptation for music emotion. In *Proceedings of the second Australasian conference on Interactive entertainment*. Creativity & Cognition Studios Press, 2005.

Lucasfilm Games. *Ballblazer*. [Atari 800, Atari 5200], 1984. Video Game.

S. Macdonald-Wright. *A treatise on color*. Macdonald-Wright, Stanton, 1924.

S. Manousakis. Musical L-systems. Master's thesis, The Royal Conservatory, The Hague, 2006.

M. Milicevic. Experimental/Abstract Film and Synaesthetic Phenomena 1725 – 1970. School of Media Arts – https://soma.sbcc.edu/. https://soma.sbcc.edu/users/DaVega/FILMST_113/Filmst113_ExFilm_History/exp-film.pdf.

D. Millen. Cellular automata music. In *Proceedings of the 1990 International Computer Music Conference*, pages 314–316, 1990.

E. R. Miranda. On the music of emergent behavior: what can evolutionary computation bring to the musician? *Leonardo*, 36(1):55–59, 2003.

E. R. Miranda. At the crossroads of evolutionary computation and music: self-programming synthesizers, swarm orchestras and the origins of melody. *Evolutionary computation*, 12(2):137–158, 2004.

W. Moritz. Musique de la Couleur – Cinéma Intégral (Color Music – Integral Cinema). Center for Visual Music – http://www.centerforvisualmusic.org/, 1995. http://www.centerforvisualmusic.org/WMCM_IC.htm.

W. Moritz. Mary Ellen Bute: Seeing Sound. Animation World Network – http://www.awn.com, 1996. http://www.soundartarchive.net/articles/Moritz-1996-Mary%20Ellen%20Bute_%20Seeing%20Sound.pdf.

I. Newton. *Opticks: Or, a Treatise of the Reflections, Refractions, Inflections, and Colors of Light*. Lightning Source Incorporated, 2011 (originally published in 1704).

G. Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer, November 2010.

W. G. Parrott, editor. *Emotions in Social Psychology: Key Readings: Essential Readings*. Psychology Press, 2000.

K. Peacock. Instruments to perform color-music: Two centuries of technological experimentation. *Leonardo*, 21(4):397–406, 1988. http://www.paulj.myzen.co.uk/blog/teaching/voices/files/2008/08/instrumentstoperformcolor.pdf.

R. Plutchik. *Emotion: Theory, research, and experience: Vol. 1. Theories of emotion*. Academic Press, 1980.

R. Plutchik. The nature of emotions. *American Scientist*, 89(4), 2001. http://www.americanscientist.org/issues/page2/the-nature-of-emotions.

S. Poliniak. Singing with all nine senses. *Teaching Music*, 19(6):54–55, 2012.

J. Rutherford and G. A. Wiggins. An experiment in the automatic creation of music which has specific emotional content. In *Proc. for the 7th International Conference on music Perception and Cognition*, 2002.

J. C. Schacher, D. Bisig, and M. Neukom. Composing with swarm algorithms — creating interactive audio-visual pieces using flocking behaviour. In *Proceedings of the International Computer Music Conference 2011*, pages 100–107, 2011.

N. Sidler. Synesthesia and Colorlight Music. Natalia Sidler – www.nataliasidler.ch. `http://www.nataliasidler.ch/download/Synesthesia-and-Color-light-music.pdf`.

P. M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

N. Tokui and H. Iba. Music composition with interactive evolutionary computation. In *Third International Conference on Generative Art*, pages 215–226, 2000.

C. van Campen. Synesthesia and artistic experimentation. *Psyche*, 3(6), 1997.

J. Ward, B. Huckstep, and E. Tsakanikos. Sound-colour synaesthesia: To what extent does it use cross-modal mechanisms common to us all? *Cortex*, 42(2): 264–280, 2006.

T. Winkler. Making motion musical: Gesture mapping strategies for interactive computer music. In *ICMC Proceedings*, pages 261–264, 1995.

S. Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, pages 1–35, 1984.

R. Wooller, A. R. Brown, E. Miranda, R. Berry, and J. Diederich. A framework for comparison of process in algorithmic music systems. In D. Burraston and E. Edmonds, editors, *Generative Arts Practice*, pages 5–7, 2005.