Computationally generated music using reinforcement learning.

Kristopher W. Reese
University of Louisville

COMPUTATIONALLY GENERATED MUSIC USING REINFORCEMENT LEARNING

By

Kristopher W. Reese
B.S., Hood College, 2009
B.A., Hood College, 2009

A Thesis
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

Department of Physics
University of Louisville
Louisville, Kentucky

May 2011

COMPUTATIONALLY GENERATED MUSIC USING REINFORCEMENT LEARNING

By

Kristopher Wayne Reese
B.S., Hood College, 2009
B.A., Hood College, 2009

A Thesis Approved on

February 21, 2011

By the following Thesis Committee:

[illegible]

Adel Elmaghraby, Thesis Director

[illegible]

Roman Yampolskiy

[illegible]

Charles Hardin

ACKNOWLEDGEMENTS

I would like to first and foremost acknowledge, and thank, Dmitri Tymoczko of Princeton University for allowing me to read the manuscript of his new book *The Geometry of Music* before its release date in December of 2010. The concepts laid out in his book make up the basis of this research thesis. Without his book, I would have had a much more difficult time learning the concepts and likely building unnecessary data structure.

Next, I would like to thank my advisor, Adel Elmaghraby, for his support with this research project. It was his idea to do something a bit more artistic in the computer field; I simply took his idea and molded it into what it is now. I would also like to thank Roman Yampolskiy for his continued support in the development of these ideas, and helping to develop the project into what it is now.

Finally, I must acknowledge the musicians who were gracious to spare me their time in order to help me with various aspects of the thesis. These musicians include: David Conway, Taylor DiClemente, Diana McLaughlin, and Jennifer Huntoon. I cannot thank them enough for the time that they gave me to help me with my thesis.

ABSTRACT

COMPUTATIONALLY GENERATED MUSIC USING REINFORCEMENT LEARNING

Kristopher W. Reese

February 21, 2011

Computers and music have shared a rich history since the 1950s. Many languages and standards have been built around music. Yet even before the advent of the computer, music shared algorithmic ideas with mathematics which brought about many new styles over the centuries. Today's computers provide even more power, and with Intelligence algorithms, are able to create complex systems for generating art. Music is no exception, but very little has been done in generating music using such algorithms.

Reinforcement Learning provides a means of learning good motions of chord progressions in music theory. Dmitri Tymoczko's Latent model for the underlying chord structure creates a mesh orbifoidal network capturing voice leading and surrounding chords. This presentation discusses experimentation in the latent model with a combination of the ideas taught in traditional Tonal Harmonic theory. Unlike David Cope's work in mimicking composer styles using machine learning, this approach attempts to tackle the problem head on through experimentation with Tymoczko's latent model for chords.

Reinforcement Learning provides a means for learning this network and reward states in order to reach a terminal goal (taught in music theory as cadencing chords). Using Reinforcement Learning we are then able to use the reinforced model to generate chord progressions which have a tonal center (a center of gravity pulling the chords towards a certain pitch class). Further, a discussion of the implemented algorithm is also given.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

With the aid of electronic computers the composer becomes a sort of pilot: he presses the buttons, introduces coordinates, and supervises the controls of a cosmic vessel sailing in the space of sound, across sonic constellations and galaxies that he could formerly glimpse only as a distant dream.

– Iannis Xenakis, 1965

CHAPTER I

INTRODUCTION AND MOTIVATION

A Problem Statement

Since its inception, the computer has provided scientists in varying fields a tool for doing complex computations. However, today's computers have become a widely used instrument for media art, as well as in algorithmic composition. This notion of the computer being a tool for artistic expression dates as far back as Charles Babbage's concept for the Analytical Engine. It was Ada Lovelace that noted in her translation of Luigi Menabrea's Sketch of the Analytical Engine [1]:

> Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

The use of the computer in current algorithmic composition of music dates back to the late 1950s with Hiller's Illiac Suite composed by the University of Illinois' Illiac computer in 1957. Since then, various systems of music theory have been implemented into algorithmic compositions, including serial, stochastic, and chance music - each of these are described in chapter two. However, very little has been done in algorithmic composition to create music that we can define as "tonal".

With the power of computing today and with algorithmic paradigms, such as dynamic programming or heuristic algorithms, it seems surprising that little work has been done in the areas of algorithmic tonal compositions. For this reason, we will step back from current models of algorithmic composition and propose varying techniques that are based on the relatively new field of geometric music theory and provide algorithms that follow modern algorithmic paradigms to create an algorithmic approach to generating tonal compositions.

In addition to the above mentioned, we hope to provide a new approach to algorithmic composition that would be beneficial in areas of psychoacoustics, neurosciences, and even in game development. Unlike many previous algorithmic music programs, which use a sort of dice-game to musical composition, or approaches composition from more atonal styles, our hope is that this new technique will provide computer music with algorithmic tonal music generation.

B Defining Tonality

In the previous section, the use of the word tonal is used very frequently as a problem that this thesis attempts to solve, but what is tonality? The word "tonal" is an oft-contested word. Some music theorists and musicologists use the word very restrictively, defining only music of the 18th and 19th centuries as "tonal". This restricts all music of the 20th century as "post-tonal", including the harmonies of jazz music with the sonic sounds of composers like Xenakis. It is hard to believe that both of these very different genres of music could be lumped into a single category.

Other theorists use the word more expansively defining certain elements as essential for "tonality". In this use of the word, we can look to the Koftka et al. definition of tonal harmony. Kostka et al. [2] define tonal harmony as "refer[ring] to music with a tonal center, based on major and/or minor scales, and using tertian chords that are related to one another and to the tonal center in various ways."

In this definition of tonal music, we make use of a specific tonal center, a pitch-class that provides a heavy center of gravity in the music. From this tonal center, we can begin building chords from various pitch classes in a specific major or minor scale using tuples of intervals in the scale, for example a tertiary chord built on a tonal center of C could be built using the pitch classes of C-E-G as shown in figure 1. Through the use of this definition of tonality, we break this classification of music from the 20th century as being all post-tonal. We are then able to look to modern genres as inspiration for our algorithmic composition generator.

C A Style to Imitate

Much of the 20th century is filled with approaches that imitate previous generations of music including the neo-classical and neo-romantic classification of art music. We also find heavy usage of various atonal approaches that grew out of the post-war eras of the early 20th century. It was during these times of atonality that algorithmic composition using computers began to grow into existence.

More recent composers have begun to move back towards tonal harmonies. One of the musical genres of the 20th century, which has been called the leading musical style of the late 20th century, has been the minimalist movement. This minimalist movement in the art of the 20th century grew out of the media art movement of the early 20th century. In this, artists reduced materials and form to basic

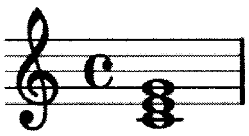


Figure 1. A rendering of a C-major chord which is built on the pitch classes, from lowest to highest note, of C - E - G

fundamentals and never intended to express feelings or convey their state of mind.

Despite the art movement, minimalist music grew to become one of the 20th century's most popular techniques, which was able to contain a wide range of expressive content. In this movement of art music, composers attempted to reduce materials in the composition to a minimum and simplified procedures in the music so that the musical content of the piece was immediately apparent [3].

Because of this style's simplification of musical content, it provides a unique testing ground for algorithmic tonal music. By reducing much of the content to its simplest form, we would more easily be able to classify the output from the algorithms as tonal or atonal. Because of this, much of this thesis will approach algorithmic music from a minimalist standpoint.

D Synopsis

This thesis addresses ideas that span many fields of study. It bridges the gap of the mathematical and computational with the more artistic field of music. Despite this overlap, this paper will focus primarily on the algorithms and mathematics behind the developed system. Any important musical terminology that occurs in the thesis will be explained. However, this paper assumes an understanding of the most basic music theory (such as how to read music) due to the length it would take to describe all of the topics in music theory. For those computer scientists or mathematicians who have little or no understanding of basic music theory and/or how to read music, the author has provided the reader a section, Appendix A of this document, for a short discussion on the assumptions of understanding of music that are held throughout this thesis.

The rest of this document explores a brief history of electroacoustics and algorithmic composition, including a handful of programming languages designed as a tool for generating sounds and music (Chapter II). A discussion of the role of mathematics in music theory will be presented, and we will examine the recent developments in Geometric Music Theory by Tymoczko et al. (Chapter III). We then present an explanation of the current practices of Reinforcement Learning techniques that are used throughout this document (Chapter IV). To follow this, this paper will present the modified models for generating tonal music (Chapter V). This will lead into the discussion on possible techniques that could be used for generating personalized and interactive real-time audio using the generated engine, with an emphasis on biometric personalization (Chapter VI). The conclusion presents the contributions and possible future work related to this project (Chapter VII).

CHAPTER II

A HISTORY OF ALGORITHMIC MUSIC

The history of music is full of examples in which mathematics has played an extremely important role in what is now known as Music Theory. The earliest examples show that even the Greek philosophers who worked in attempting to analyze music used mathematics as an explanation for harmonics, creating scales, and more. Pythagoras' work in harmonics is probably one of the best known examples of mathematics used to describe harmonics.

Algorithmic music follows much of the same rich history as mathematics in music. Karlheinz Essl describes algorithmic music as, "A method of perceiving an abstract model behind the sensual surface, or in turn, of constructing such a model in order to create aesthetic works." [4] This definition and explanation of algorithmic music fits, and, using this definition, we find an extremely rich history of algorithmic music dating back long before the creation of the computer. However, today this style of music is generally associated with music in which computers generate.

No matter how you look at algorithmic music, whether solely music generated by a computer, or music generated using some methodical "algorithm", both styles share one common theme. Creators of this music have a desire to create a sound which is infinite, exceeding the finite limitations of human knowledge; a way for music to overcome barriers which are either inherent in our minds or created by generations of social stigma. [4]

A The Algorithm

When discussing algorithmic music, the first word, algorithmic, becomes one of the most unfamiliar in the area of music. Boolos & Jeffrey [5] informally define the word algorithm as a means of giving "explicit instructions for determining the nth member of the set for an arbitrary finite n. [The] instructions are to be given quite explicitely, in a form in which they could be collowed by a computing machine, or by a human who is capable of carrying out only very elementary operations on symbols." In essence, an algorithm becomes a set of precise rules for a fast and efficient means of solving problems.

The word algorithm is derived from the name of the Persian mathematician, Muhammad ibn Mūsā al-Khwārizmī, who introduced the use of Hindi-Arabic notation into what is now known as Algebra. The original definition of the latinized version of al-Khwārizmī, "algorism" was used to refer to only this particular form of Algebra. Later translations of the latinization became what is now referred to by the word algorithm. The word algorithm is defined, informally, as a set of definite procedures for solving problems or performing various tasks. [6]

Today, algorithms are more generally used in the field of computer science as a way to allow a computer to solve problems efficiently. Algorithms are used in computing for all sorts of tasks, including path planning, sorting numbers, scheduling, and many other areas. Generally algorithms have efficient solutions which compute problems in a polynomial time. However, there do exist problems in which finding a solution becomes difficult to find in a finite amount of time. A common example of this is the "Traveling Salesman Problem". The problem becomes intractable for extremely large numbers of cities. In this case, the computation that is needed to be done in order to solve this using standard methods grows exponentially for each city added. This is a classic example of a NP-hard problem in combinatorial mathematics.

The field of Artificial Intelligence grew out of an attempt to find solutions which can find a solution, or more often a "good enough" solution for these types of problems, quickly. However, much of the field of artificial intelligence uses forms of algorithms, such as the genetic algorithm and particle swarm algorithms, which may not be classified as "correct" algorithms, meaning they do not necessarily yield a correct result all the time.

More advanced fields of "Artificial Intelligence" yield even more complex algorithms to compute certain aspects. Computer Perception is a more advanced field in this domain of "Artificial Intelligence" which includes Computer Vision and other Perceptive techniques. Even this field of computing uses combinations of algorithms to find information in digital information, which can be used to perceive aspects in which the computer is searching for, such as the location of a specific item in an image.

The bulk of this thesis focuses on use of Machine Learning algorithms and other less intensive algorithms, for problems which can be computed relatively quickly, to generate new music. Machine Learning, and more specifically Reinforcement Learning, are simply a continuous algorithm which are used to observe specific information and use statistical inference to make complex decisions in the domain of the data. More about the specific algorithms used will be discussed in a later chapter in this thesis.

The point to take out of this section is that almost all of computing is associated with algorithms in almost all domains. This is no different in music. However, music is full of a rich history of mathematics and algorithms that are generally lost due to the little use of the information. The rest of this chapter will discuss the history of Algorithms and Mathematics in Music from the Ancient Greeks to modern computationally generated music.

B Formal Processes in Music

Since music's early recorded history, music has always closely been associated with mathematics and formal structures. It was the Greeks who first delved into attempting to understand music using mathematical processes. During the Baroque era, we see formal structure begin to help build the music of the era. These structures grew into what musicians now know as the various forms of the classical era; during the classical to the modern era, the idea of using randomness to build music was brought about using what is now known as the "dice-rolling game". It wasn't until the 20th century that determinisitc processes were brought about in music.

1 Music, Mathematics, and the Ancient Greeks

The Greeks philosophers were the first, in recorded history, that attempted to understand music in a mathematical way. It was Pythagoras who is best known for his work in music during the ancient era of music. Pythagoras recognized the ratios between the tones that are played in music. He was able to prove his work by using a simple stringed instrument and folding the string to produce tones. He found, for example, that what is now know as an octave (12 tones up or down) was a ratio of 1:2. By folding in various ways, we can produce different tones that can be used. This was the first written example of music being combined with mathematics.

Pythagoras was not the only philosopher of the greeks who associated music with mathematics. Aristoxenus was one of the first greek philosophers who had the idea of geometrization of musical space around 320 BCE. His ideas were radically different from Pythagoras in that rather than using discrete ratios, Aristoxenus used continuously variable quantities. [7] Aristoxenus was the first arithmetician who proposed why a slight mistuning of notes are still perceived as categorically invariant. This led him to believe that the principal of consonance of the scale had a narrow, but acceptable range of variation. Aristoxenus' work was important in that it later influenced the theory of Greek Orthodox, Hellenized Persian, and Arab music which gives the appearance of direct descent to the arithmetician's work. [7]

Ptolemy is mostly associated with mathematics, astronomy and geography. However, Ptolemy wrote an influential work on music theory entitled "Harmonics". Ptolemy spent much of this work criticizing his predecessor and arguing for a basis of musical intervals based on mathematical ratios. This was in complete contrast to Aristoxenus and his followers, following more closely with that of Pythagoras. The difference between Ptolemy and Pythagorus was that Ptolemy based his work on empirical observations. Ptolemy believed that a musical not could be translated into mathematical equations and vice versa. [8]

Ptolemy's work later bled into what is known as "Musica universalis". This is an ancient philosophical concepts that regards the ratios and proportions of the movements of celestial bodies as forms of "musica". The term "musica" is not usually thought of as audible music in this philosophy, but as a harmonic, mathematical, and/or religious concept.

Though there are only a handful of Greek philosophers presented here that discuss music, there remain many which are not mentioned; these three are the most prominent philosophers of music during their time in ancient Greece, and helped to shape modern music theory. Later composers would return to these philosophers for ideas in composition during the 20th century. From just this brief sample of philosophies, we see that since early recorded history, mathematics has played a major role in the understanding music.

2 Chance Music from the Classical Era to the Modern Era

Much of the Renaissance and Baroque era began to step further away from using mathematics as a way to understand music and is where we begin to see a theory solely dedicated to music take shape. However, even during this period we see formal structures take shape in the understanding of making music sound developmental. This is minor to the subject discussed in this thesis, and mention of it is a nod to it as a part of the era which can be viewed as remotely mathematical or algorithmic.

This lack of interest in mathematical formulations of music continued into the classical and romantic eras, where little was done relating to music and mathematics. The major contribution during these two eras exist in chance music better known as *Musikalisches Würfelspiel*, which can be literally translated as "Musical Dice Game". These games were popular throughout Western Europe during the 18th century. It provided a system for using dice to randomly compose music long before the computer system was invented.

The most well-known dice game was published in 1792 by Nikolaus Simrock in Berlin. Because Nikolaus Simrock was Mozart's publisher, the game is often attributed to Wolgang Amadeus Mozart, however this attribution has yet to be authenticated by any musicologist. [9] In this game, the dice is rolled which randomly selects a small section of music. Each section that is rolled is then patched together with the previous ones to create a musical piece.

Despite the lack of authentication of for the published game, Mozart did seem interested in the game. An autographed genuine musical game by Mozart can be found in the Bibliothètique Nationale in Paris and designated K 516f, written in 1787. [10] This musical piece contains no instructions and no evidence that dice were involved in the composition of the piece, leaving the creation process of the piece up for debate between musicologists.

This dice game was perhaps the earliest known example of some form of chance being used in music. During the 19th century, very little was done again with music and mathematics. It wasn't until the 20th century that we begin seeing compositions being created using elements of chance. During this time, John Cage created numerous algorithmic systems to employ chance in creating music which was based on the 'I Ching', star atlases, or other such means.

John Cage employed these methods in order to overcome the habitual methods of composers themselves. Cage felt that by using methods of chance instead of representing order systems or expressing subjective sentiments, the sound of the music is freed from any prior meaning or historical connotations, free to 'come into their own'. [11]

It was during this time that composer John Cage began developing ideas for graphical representations of music, which left much of the music to chance and choice by the musician themselves. By providing these graphical representations, Cage was able to lay a groundwork which leaves the song nearly open ended, and all of the parameters of the music free. This allows all parameters of the music to change from each performance by changing the times for note, such as starting and stopping of notes, as well as the frequency, amplitude, use of filters and distribution of sounds in the musical space. [11]

C Deterministic and Stochastic Processes

In the previous section, the methods of mathematics were discussed as they were applied to music during a period of relative order in music. This section continues by discussing methods which employed sets of random operations within the context of an overlying algorithmic model to gain control over the direction of the music. In this sense music falling into this category is both deterministic and stochastic processes were used to create a sort of aleatoric classification of music during the early 20th century.

1 Serialism

The elements of World War II left not only many of the cities of Europe in ruin, but nearly eradicated the music of that century. Soon after the eradication of the Nazi's in europe, younger composers gathered together to create a new musical grammar, free of the traditional practices of music in the years prior to the war. Serialism was the result of this gathering of composers. This form of music is primarily attributed to the composer Schoenberg, who was the first to employ its techniques with relative success.

In this form of music, Schoenberg's dodecaphonic technique created music whose pitches are predetermined in all serial music. This technique was later extended to other "parameters" of music such as pitch duration, dynamics, and timbre. The dodecaphonic series of the music becomes the unifying principal of the music which allows to music to sound less like a random selection of a subset of the 12 pitch classes, and more like an organized form of music.

This series that makes up the unifying principal is, more simply, a random set of values from the set of twelve tones in western music. Variations of this series can

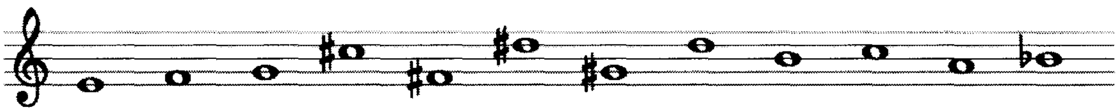


Figure 2. The 12-tone serial row series used by Schoenberg in Suite, Op. 25.

be created by applying various transformations, such as transpositions, inversion, retrograde, and permutations. These mathematical operations can be obtained by transforming the symbolic representation of the row into a numeric representation.

When observing or creating serial music, it is often beneficial to have all 48 possible forms of the tone series. To create this, it can be represented by a 12x12 matrix. If for example we choose the tone series shown in figure 2, we can build this 12x12 matrix by transforming the row series into prime series (denoted using P) and inversional series (denoted by I).

Calculating the Inversion of the prime theme is relatively simple when thought about in geometric terms. If we take a circle which connects each of the notes to their next neighboring notes on two sides, we can draw a line between the primary themes starting note, E in the case of the primary theme shown in figure 2, and the notes tritone interval, or a jump of 6 notes- the starting notes tritone. From this, all intervals between notes can be thought of as a line segment between the starting note and the second note. Figure 3 shows this technique. In this figure, we see line segments between E and G on the top. If we take this line segment and draw the same line segment on the bottom, we can determine the inversion of a note. In this example, the inversion of G is a $C\sharp$.

If we then take our primary theme, P_0, we can find the first inversion series by following this technique. Doing this results in the first inversion I_0 as shown in table 1. With the information found in this table, we can begin building a matrix to give

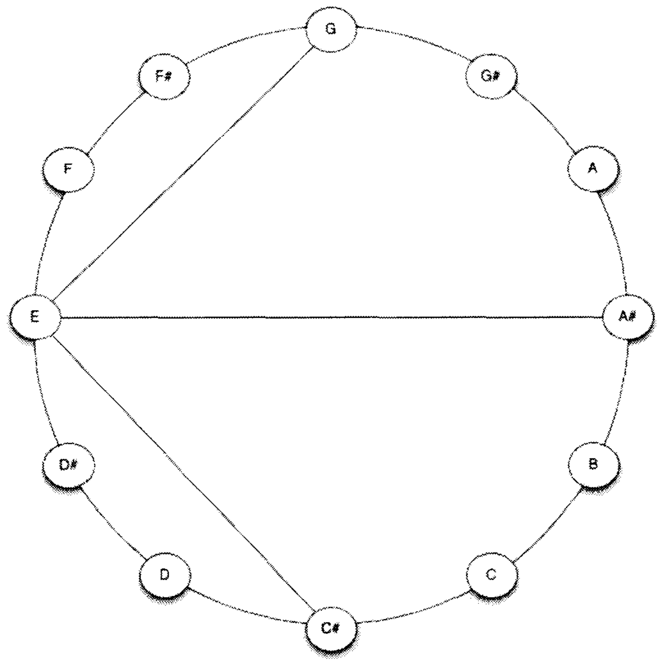


Figure 3. A geometric representation of Serial Inversion, The line segment between E and $A\sharp$ is the inversion line while the other lines represent the inversion between G and $C\sharp$.

all 48 possible forms.

To begin the matrix, we must first convert the Prime series into a set of numerical representations. For the purpose of this example, we use E as the 0 note and continue by determining the number of jumps up it would take to reach the next note. Therefore our prime series becomes the numerical series: 0, 1, 3, 9, 2, 11, 4, 10, 7, 8, 5, and 6. By doing this, the subscripted version of the prime, inverted, and retrograde version of the series are represented using this number sequence.

Now we can begin building the matrix of sequences that can be used in a piece created using this technique. The prime series is written as the first row of the matrix and the Inversion is written as the first column. From here we can calculate all of the rows by adding the total number jumps to the notes in the first prime sequence. Therefore, if we are calculating the prime 1 row, P_1, we simply move the

	1	2	3	4	5	6	7	8	9	10	11	12	
P_0	E	F	G	$C\sharp$	$F\sharp$	$D\sharp$	$G\sharp$	D	B	C	A	$B\flat$	R_0
I_0	E	$D\sharp$	$C\sharp$	G	D	F	C	$F\sharp$	A	$G\sharp$	B	$B\flat$	RI_0

TABLE 1

Primary, Inversion and Retrogrades of the theme found in Figure 2

notes in the P_0 row up by 1 note. Therefore our P_1 row becomes: F, $F\sharp$, A, D, G, E, A, $D\sharp$, C, $C\sharp$, $B\flat$, B.

By continuing this for all tone rows, we generate an entire matrix which can be used to represent all possible forms of the music. The Columns from top to bottom represent inversions of the Prime, The rows from left to right represent the primes themselves. By reading the primes backwards, we can represent the Retrogrades of the primes (R_x) and by reading the columns from bottom to top, we can represent the Retrograde inversions (RI_x). The matrix for the theme found in figure 2 is shown in table 2.

During the composition process, composers will start their composition with the P_0 line, however, after using the P_0 line in the composition, the composer would choose any of the forms that result in the matrix to create their composition. The decisions for the composition itself however was generally left up to the composer. The same technique used in choosing forms of the notes was later applied to other parameters of music, in what is more formally called "Total Serialism." In "Total Serialism", the composer was left with fewer attributes in the music to decide upon, however the general direction of the piece and selection of the series forms were still left solely to the composer.

By composing a piece with strictly predetermined material, we see the first move towards a more algorithmic style of composition. Karlheinz Essl views

	I_0	I_1	I_3	I_9	I_2	I_{11}	I_4	I_{10}	I_7	I_8	I_5	I_6	
P_0	E	F	G	$C\sharp$	$F\sharp$	$D\sharp$	$G\sharp$	D	B	C	A	$B\flat$	R_0
P_{11}	$D\sharp$	E	$F\sharp$	C	F	D	G	$C\sharp$	$B\flat$	B	$G\sharp$	A	R_{11}
P_9	$C\sharp$	D	E	$B\flat$	$D\sharp$	C	F	B	$G\sharp$	A	$F\sharp$	G	R_9
P_3	G	$G\sharp$	$B\flat$	E	A	$F\sharp$	B	F	D	$D\sharp$	C	$C\sharp$	R_3
P_{10}	D	$D\sharp$	F	B	E	$C\sharp$	$F\sharp$	C	A	$B\flat$	G	$G\sharp$	R_{10}
P_1	F	$F\sharp$	$G\sharp$	D	G	E	A	$D\sharp$	C	$C\sharp$	$B\flat$	B	R_1
P_8	C	$C\sharp$	$D\sharp$	A	D	B	E	$B\flat$	G	$G\sharp$	F	$F\sharp$	R_8
P_2	$F\sharp$	G	A	$D\sharp$	$G\sharp$	F	$B\flat$	E	$C\sharp$	D	B	C	R_2
P_5	A	$B\flat$	C	$F\sharp$	B	$G\sharp$	$C\sharp$	G	E	F	D	$D\sharp$	R_5
P_4	$G\sharp$	A	B	F	$B\flat$	G	C	$F\sharp$	$D\sharp$	E	$C\sharp$	D	R_4
P_7	B	C	D	$G\sharp$	$C\sharp$	$B\flat$	$D\sharp$	A	$F\sharp$	G	E	F	R_7
P_6	$B\flat$	B	$C\sharp$	G	C	A	D	$G\sharp$	F	$F\sharp$	$D\sharp$	E	R_6
	RI_0	RI_1	RI_3	RI_9	RI_2	RI_{11}	RI_4	RI_{10}	RI_7	RI_8	RI_5	RI_6	

TABLE 2

Matrix of the possible forms of the prime theme found in figure 2.

serialism as "highly ordered by predetermination", with the results appearing as a statistical collection of points in both space and time. [4] Serialism, though created prior to the advent of the computer, was the basis for many of the first computer programs to generate musical structures.

2 Stochastic Composition

Despite the changes in the approach to composition of music, there were extreme critics of the serial style of music. One such composer who criticized the strict pre-determinism of serial music, was Iannis Xenakis. Xenakis wrote in his paper, "The Crisis of Serial Music," about the complexity of this style of music which shaped the music as "auditive and ideological non-sense." [12]

It was Xenakis who suggested replacing the determinism that was brought about with serial music with a general concept of probabilistic logic. Through this means, Xenakis could contain the entirety of the serial music as a strict particular case. Even the definition of "stochastic music" comes from Xenakis. Xenakis defines stochastic music as based on random operations within time-variable constraints. His stochastic music was used to generate music using the statistical representations of the structures that can be found all over nature. Xenakis states that stochastic music is built in an attempt to model "natural events[,] such as the collision of hail or rain with hard surfaces, or the song of cicadas in a summer field." [13, 14]

This area of stochastic composition really breaks down into two separate categories. The first school of thought in stochastic music was Xenakis' ideology for stochastic music. In this, Xenakis implemented stochastic methods like the Gaussian distributions or Markov Chains. This gives the music much more deterministic qualities while still employing levels of chance to music as well. The second school of thought revolves around Gottfried Michael Koenig's ideology for composition. Koenig replaced the serial permutation mechanism with a non-deterministic, yet promising strategy of using aleatoric principles. The term, aleatoric, is used to describe a process who's outline is predetermined and fixed, but the details of which are left to chance.

Koenig's work is perhaps some of the most important work in combining computers with music. In 1963, Koenig began work on a composition that was based on an algorithmic model and was implemented as a computer program called *Projekt 1* (PR1). Koenig assembled lists of parameter values, and used psuedorandom operations in order to select a value for the each of the parameters. [15]

PR1 in its original form was unable to convert any of the resulting parameters into actual music. Instead the composer was forced to interpret the results in order to produce music which was playable. Later versions of the program get rid of a lot of the limitations of the program itself, due to further development of the program and by other composers who used the application. Yet even in its final version, determining the input data is limited, requiring only a handful of parameters from the composer. [15]

D Music and Computers

Many of the above topics discussed the influence of mathematics on music, and the use of Algorithmic techniques for creating music. We see many of the techniques coming to fruition in the early part of the 20th century. In later parts we start seeing the use of computers alongside the compositional process of composers to simplify the algorithmic techniques which were used. But it wasn't until the latter half of the 20th century that music was able to take full advantage of the computer, allowing it to not only compose music based on algorithms but also to play the music which was created or output musical scores which were able to be read by a performer without the need of human intervention.

1 Generative Music

Many of the topics discussed above were first created in an attempt to free music from the societally created structures that limited music to something which seemed to go against the structures of soundscapes created in nature. It was pop artist Brian Eno who first became interested in ripping the bonds of time limitations in music. He saw that the natural soundscapes created in the world had no beginning and no end, yet music always seemed to start and stop.

In 1978, Brian Eno created a non computer based system for generating an unending, evolving sound environment for the LaGuardia Airport, which he called "Music for Airports". For this, Brian Eno used the phasing of tape loops with different lengths to create different instrumental tracks, allowing the music to at some point clump all of the sounds together and at other points spread the instruments through the music. By using this simple looping technique, Brian Eno was able to create an infinite number of soundscapes which were based on only a handful of elements.

Brian Eno's work later inspired work to be taken to software engineers Pete and Tim Cole to create a computer program using the same techniques. This was one of the first steps in which computers were used to generate music for certain situations. Brian Eno's idea of using Ambient Music began a surge of computer related media on the Internet.

In 1997, Maurice Methot and Hector LaPlante began to contemplate what type of medium could best be utilized to listen to music such as Eno's composition. Because the music no longer consisted of a beginning or end, Compact Disks were highly inappropriate for the music. Because of this, Maurice and Hector began 'The Algorithmic Stream', one of the earliest audio streaming systems on the internet which provided non-repeating computer generated music. Though this later died down, many project still exist which revolve around this idea.

Though Eno's work is of little importance to computer music, it is of extreme importance as it helped to influence streaming media on the internet during the era. Because of this, it deserves special mention in this thesis for both the simple looping algorithm and streaming audio.

2 Programming Music

The first era of true computer programming language dedicated to sound synthesis was called MUSIC, appearing in 1957. It was developed by Max Mathews at AT&T Bell Laboratories. This language was build in order to provide a model for specifying sound synthesis modules, connections, and time-varying controls. During the development of MUSIC, the language was compiled on a series of punchcards and implemented as a low level assembly language. Several further developments of the MUSIC programming language were released in further iterations of the language, MUSIC I-IV.

It wasn't until 1968 that a programming language dedicated to music composition was able to break its previous limitations and be implemented within another programming language. MUSIC V was released as an implementation of FORTRAN. Unlike previous MUSIC languages, this meant that MUSIC V was able to be used on any computer system capable of running FORTRAN instead of being limited to specific hardware. MUSIC V also provided a model for later music programming languages an environments making mention of this language important. [16]

Around the time of the advent of the modern operating system, we see several other languages and extensions of languages begin to appear. In 1972, development of the CARL System was developed as a series of open source, interconnectable programs for Signal Processing and Signal Analysis. We also see during this time a library for the C Programming Language which was modeled after the MUSIC-N languages mentioned above. The most widely used descendant of the MUSIC-N languages today is CSound which was developed in the late 1980s by Barry Vercoe and his colleagues at MIT Media Labs. This further developed the compositional and audio playback that we now use on modern computers today. [16]

During the 1980s we see yet another type of music playback system developed. The MIDI (Musical Instrument Digital Interface) specifications were published in August 1983 with the purpose of bringing different digital instrument makers together under a single standard. [17] This was primarily built out of the synthesizer needs of progressive rock bands. By using a MIDI based synthesizer, a performer was able to play multiple sounds from a single keyboard, rather than the many keyboards that are often seen from the early progressive rock concerts.

After the ratification of the MIDI standards, we begin seeing MIDI implemented in many of the Operating Systems of the era. This development in the Operating Systems allowed for powerful and inexpensive tools for computer based MIDI sequencers. Though during its early development, the quality of the hardware and the unsophisticated methods for the synthesis methods used for audio playback resulted in giving MIDI a poor reputation with some critics. Yet today, MIDI sound synthesis results in often higher quality sound which is driven by MIDI data proves that MIDI is an overlooked method of sound generation.

E Decisions for this Thesis

This thesis extends many of the techniques used by previous composers in an attempt to both better understand the aesthetic benefits of Stochastic composition. This thesis also maintains a level of scientific value in both the abilities of the computers and the viability of the model developed by Tymoczko in the realm of Computers. We see through the history of the computer itself that it has become a compositional tool that have been used by many to help aid in their compositional process.

Today's computers have the ability to do much more than just act as a random number generator for the composer. We see during the last decade of the 20th century the development of Intelligent systems dedicated to writing in the compositional style of the composer by David Cope in his works entitled "Experiments in Musical Intelligence". [9] Unlike David Cope's work, this thesis is an attempt to create a general model that computers can use to create music. Yet even still, the models used by David Cope for machine learning and music and the models presented here, which use Reinforcement learning, are not contrary to one another. More about this will be discussed in the Conclusion.

This thesis focuses primarily on the development of a tonal harmonic progression algorithm using Tymoczko's underlying structure of chords, presented in Chapter III. This model is implemented using Java with a Reinforcement Learning technique known as Q-Learning. This technique provides a way of learning about and traversing the environment in the model. It is a stochastic decision making algorithm which allows us to use stochastic techniques first used by composers such as Xenakis, but also allows for complex decision making that composers such as Mozart or Beethoven might have take advantage of naturally.

For this thesis, all of the generated music also uses the MIDI Specification. This provides the most robust and simplest way of generating music using the JAVA programming language. There is no reason a library such as CSound could not be used to develop this thesis. It was simply a due to the ease of the JAVA MIDI API that pushed the decision towards using the MIDI Specification. Subsequent chapters of this thesis discuss the development of the thesis further.

CHAPTER III

A SCIENTIFIC APPROACH TO MUSIC THEORY

The previous section discusses how mathematics has played a major role in the development over the millennia, staring with the ancient greeks and working towards the stochastic processes we understand today. It seems relatively reasonable that music would follow some stochastic decision making process, which is discussed later in this thesis; however, in order to use stochastic decision making, we must have some underlying, latent model which music can follow.

Much of music theory is riddled with more of the discussion of the language of music and what will sound good to their listeners even when incorporating chromaticism into the language. In this sense, Tonal Music theory has become more like an english grammar class than a science. Despite this, there are proponents of music theory as a science. A recent development in understanding tonal harmonies was provided in 2006 by Dmitri Tymozcko. This article [18] was the first article on Music to be published in a scientific journal.

Dmitri Tymoczko provides an understanding of tonal harmonic movement through a latent model which can be used to model any n-dimensional chord in an n-dimensional model using geometry. The rest of this chapter focuses on the works of Dmitri Tymoczko as this model is the model which makes up the program that is developed in this thesis. This model combined with the stochastic decision making algorithms provide a means for allowing computers to generate harmonies which are

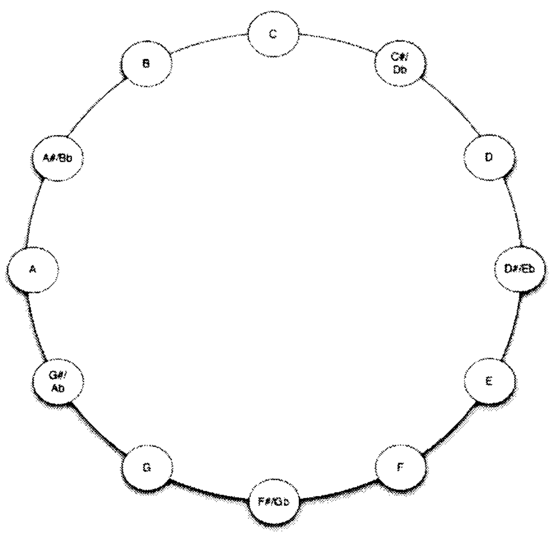


Figure 4. The Circular Pitch class space described by Dmitri Tymoczko.

centered around a tonal center.

A One-dimensional Space

Dmitri Tymoczko explains in his original article and his book [18–20] any n-dimensional chord space. The first space that is discussed by Tymoczko is the one dimensional chord space. However, he simply calls this the "Circular pitch-class space" [20]. This name is best suited for musicians who find it hard to represent a one-dimensional chord space is, by all means, simply a representation of the pitch classes in music that were described in appendix A.

Tymoczko says that, geometrically, this one-dimensional space can best be thought of as a line. [18, 20] Though the pitch class represents only a single dimension, the reason it is best represented as a circle is caused by the shared pitch classes repeat after completion of the notes. For example, if we start at the pitch of a C and move up one semitone every step, we eventually reach the pitch class of B. Because of the nature of the circle, returning to the pitch class of C is like returning

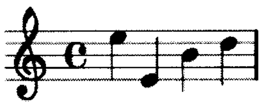


Figure 5. A simple intervalic passage.

to the C that we had started on.

Using this pitch-class space helps in representing particular ways of moving from one pitch class to another. [21] Using this geometric language, we can capture the movement from one pitch to another by modeling the intervalic movement between the two pitches. If we view the movements shown in figure 5, we can represent the figure using common language.

The first intervalic jump can be represented in this language as, "E moves down 12 semitones (or an octave)." More formally we could write this as $E \xrightarrow{-12} E$. This simply means that from the E we are moving -12 semitones to get to the interval E. A negative movement is a counterclockwise movement on the circle shown in figure 4. From here we see $E \xrightarrow{+7} B$, which is a clockwise movement by 7 semitones to the B; and lastly $B \xrightarrow{+3} D$, a clockwise movement by 3 semitones.

This understanding of the one dimensional space is important not only for being able to create a language in which melodic lines might be described, but also for understanding higher dimensional spaces and how they work relative to the one dimensional space.

B Two-dimensional Space

Much of the emphasis by Dmitri Tymoczko is put into understanding the two dimensional musical space. This is likely because understanding this is much simpler than attempting to describe musical space in 3 or 4 dimensions. By understanding

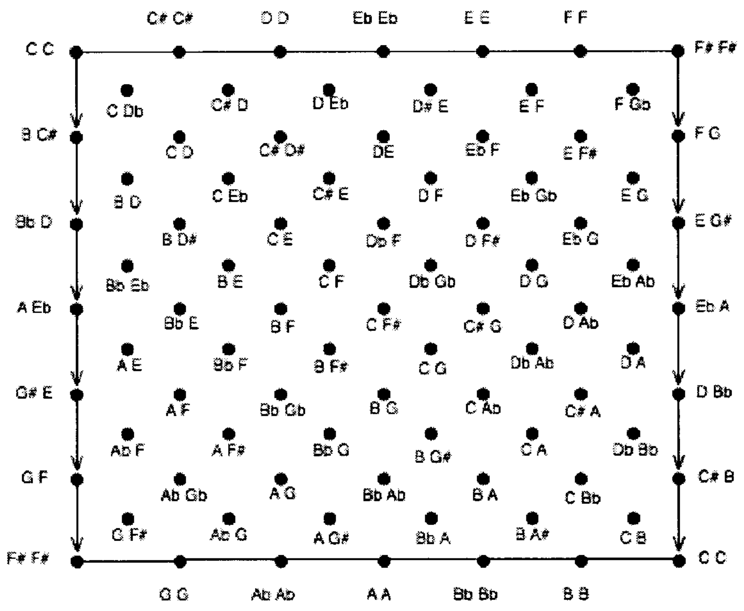


Figure 6. The Two dimensional orbifold described by Dmitri Tymoczko.

the two dimensional space, you begin understanding the higher dimensions as well.

Figure 6 represents the two dimensional interval space between two note passages. This model is built from an understanding of the combination of the one dimensional spaces of each singular note. In this case, the first note and the second note represent the way in which interval movement can be plotted on a cartesian graph. In this mesh of notes, the movement of singular notes is moved to a 45 degree angle. Therefore in order to move a single note, we move along the diagonal of the

Dmitri Tymoczko also describes his model with regards to the contrary and parallel motion of intervalic movement. Contrary movement in music is described as movement of the interval in different directions. This can be described by vertical movement from one interval to another where movement upwards represents the notes moving towards one another and moving downward represents moving away from one another. Parallel motion is different in that both notes move in the same direction, which is represented as movement to the right or left. Moving to the right

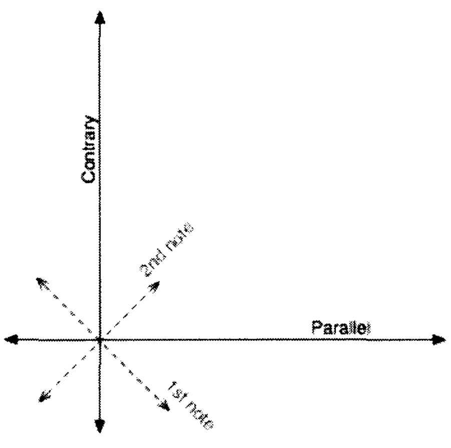


Figure 7. Movements possible in the two dimensional interval space.

results in parallel movement upwards and moving to the left is parallel movement downwards. [18, 20] The possible movements in this two dimensional space is summed up in figure 7.

Now that we understand the movement, an understanding about what happens when we reach the end of the graph. Tymoczko describes this two dimensional space as repeating on the right and left in the same way an mobius strip works. The right an left sides of this plot are brought back around and twisted so that the [$F\sharp$,$F\sharp$] pairs match up and the [C,C] pairs match up. [18, 20]

Using the same formal language that we had mentioned in the one dimensional section, we can understand movements in this space as well. The musical passage in figure 8 contains a two voice intervalic passage. We can represent the movement of the two voices on the graph. We notice that the first movement to the second is in parallel motion but they do not move the same distance. Therefore we move in the direction of parallel until we reach one of the voice's notes and then move the other voice's note to the proper note along the diagonal. The 2nd to third interval is in contrary motion so we move vertically and then fix the note movement. The next

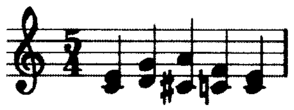


Figure 8. A simple two voice passage containing various intervalic jumps.

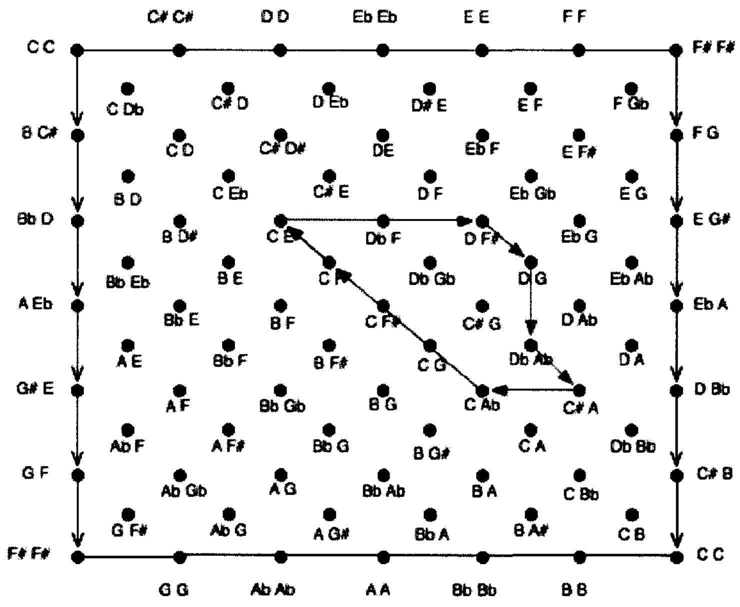


Figure 9. Movements representation on the two dimensional space and representing the movements of figure 8. The Blue lines are the first to second interval; Green is second to third; Red is third to fourth; Black is fourth to fifth.

interval is parallel with some voice fixing, and lastly the 4th to 5th interval are a simple movement of the top voice. These movements along the diagram are shown in figure 9.

We can also formally defined these movements as $(C, E) \xrightarrow{+2,+3} (D, G)$ for the first to second; $(D, G) \xrightarrow{-1,+2} (C\sharp, A)$ for the second to third; and so on. In this instance, we can extend the formal language to simply include two notes and two values for change. This is further extended for any n-dimensional chord space discussed by Tymoczko in [20].

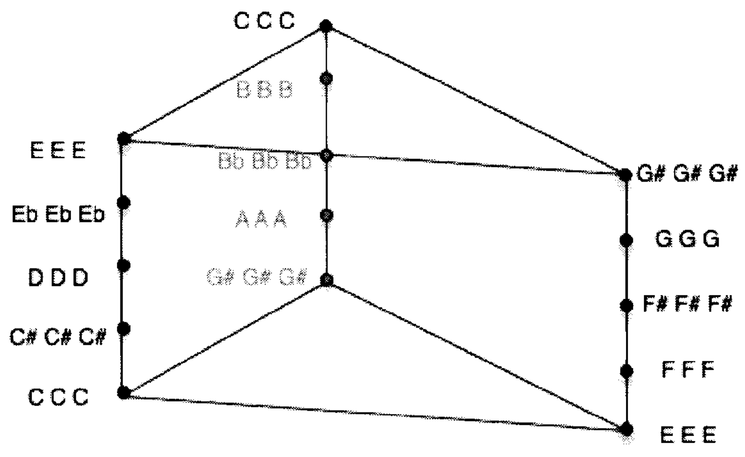


Figure 10. The Three dimensional orbifold described by Dmitri Tymoczko.

C N-dimensional Space

Up to this point, the models have dealt with one dimensional chords, or pitch classes, and two dimensional chords, or intervals. It isn't until we get to the 3rd and 4th dimensional spaces that we begin seeing what is traditionally understood in music to be a chord. However, understanding these previous dimensional spaces helps us in understanding the movements in the 3rd or 4th dimensional spaces. It is often hard for humans to visualize shapes in dimensional spaces beyond the third dimension.

The third dimensional space is shown in figure 10. This space is very similar to the two dimensional space described in the previous section except containing a third note in the model. Because of this, Tymoczko concludes that the shape of the model is that of a triangular prism. This model contains two folds that are used to connect the edges of the prism together. In figure 10, the (C, C, C) pairs match up as well as the (E, E, E) pairs and $(G\sharp, G\sharp, G\sharp)$ pairs. [20]

Moving along the same horizontal slice of the model is movement of the middle

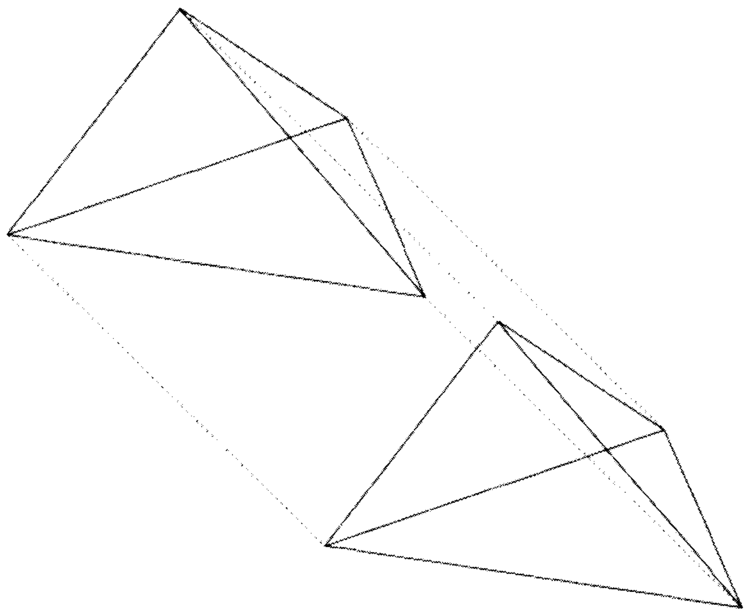


Figure 11. A visual representation of the shape of the four dimensional orbifold described by Dmitri Tymoczko.

or last notes in the plane We can also contain movement of the last two notes. Each movement to a vertical plane is then movement of the first note where diagonals from that contain contrary and parallel motion of the voices in the chord.

It is interesting to note that Tymoczko has said that many composers choose a smaller subset of chords from this model and use them throughout their compositional career. This subset of chords rarely changes over the life of their career. [18, 20] Therefore, choosing a smaller subset of chords in the model might be a viable option for the computer if the model can be used for generating chord progressions.

Describing higher dimensional chord progressions goes beyond the limits of what the experiments in this thesis will need. However, it is good to note that higher dimensional chord spaces would be extremely useful in music outside of art music. The fourth dimensional space is shown in figure 11 and other higher dimensional chords would exist in music such as Jazz. These spaces are harder to visualize, and this thesis will not attempt to explain these spaces. If you are curious, Dmitri Tymoczko's book, [20], does an excellent job of explaining these spaces.

D Applications in Stochastic processes

From the previous three sections, we began understanding the principles of movement in the model created by Tymoczko. We can see very mathematical principles to the movements of the chords and begin structuring the model to work in the stochastic processes described in the following chapter.

For the rest of this thesis we will work with the third dimensional space and with a smaller subset of possible actions limiting motion to the same distance between the movement of notes. This will allow the experiment to work solely with the "pure" actions, movement of single notes, contrary movement of two or three notes, and parallel movement of two or three notes. Later experiments and extensions to the algorithm may be needed to work with combined actions of movement to chords. Now that we have limited the experiment to a smaller subset, we can begin looking at the possibility of using Tymoczko's model for generating tonal harmonic chord progressions on the computer.

CHAPTER IV

CONCEPTS IN MACHINE LEARNING

Machine Learning has become a major field in computers, allowing the computer to learn about data and make inferences based on the data. The primary development of this project relied heavily on Reinforcement Learning, a subfield of Machine Learning. In this, we allow the algorithm to learn what "sounds" good, or rather works in this case, as sounding good varies greatly on the audience.

This chapter will discuss many of the concepts that are needed to understand how the engine is able to generate harmonic progressions based on Tymoczko's latent model for Chords. This chapter focuses on two topics in particular: Markov Decision Processes and Q Learning. Markov Decision Processes make up the understanding of the world and is necessary in this instance to help us better understand how the Q Learning algorithm works for these models.

A Markov Decision Processes

A Markov Decision Process is a dynamic programming algorithm first presented by Richard Bellman in 1957 [22,23]. The Markov Decision Process (MDP) is a mathematical process for modeling complex decisions. This model was later brought to the computer in order to solve complex, sequential, and stochastic processes. Unlike its path planning predecessors, such as Dijkstra's algorithm, this algorithm is specifically designed for taking probabilistic actions into account in order to find the most suitable path for the traversing agent to take in the world.

A MDP consists of a 4-tuple of inputs into the algorithm. It consists of a set of States in the world, a set of Actions that can be take from a state, a transition model, and a reward function. More formally it can be written as a 4-tuple in mathematics:

$$(S, A(s), P(s, a, s'), R(s)) \tag{1}$$

where S is the set of all states in the world, $A(s)$ is the actions possible at state s, $P(s, a, s')$ is the transition model which represents the probability of moving from one state s to the next state *s'* given a specific action a, and $R(s)$ represents the Rewards at state s. [24, 25] Russel and Norvig prove that the definition of the Rewards system as being dependent on the action and outcome, or more formally, $R(s, a, s')$, is unnecessary as it does not change the problem in any fundamental way. This thesis will use the notation presented by Russell and Norvig. [24]

The solution to the problem is unlike other path planning algorithms in that a fixed action solution will not offer the flexibility in the world that is needed to take stochastic problems into account. The solution to this problem then becomes an policy in which the traveling agent could use at any state in the world. Therefore, we need to take all states into consideration and determine how to best move from one state to the next with the possibility that we might move out of the "optimal path" as we would call it in a path planning algorithm. We generally denote the optimal policy as π and the recommended action as $\pi(s)$, the policy, π, for the state s.

Let us look at an example of a world in which we might use a Markov Decision Process to determine the best path to take in the world. Figure 12 shows a 4 x 4 world with a single obstacle and two terminal states, or states which end the traversal. With this information, we can begin using an algorithm knowns as Value

Iteration to determine the optimal policy for the agent moving in the world.

The Value Iteration algorithm is used for calculating an optimal policy. In this algorithm, we want to calculate the utility of a state and then use the utilities to determine the best policy. Richard Bellman's equation described in [22, 23] can be used for calculating the utility. The Bellman Equation, as it is known, calculates the utility by taking the immediate rewards for the current state and adds the discounted utility of the next state. Using this definition we can see that finding the utility for the current state depends on all of the future states until you reach a terminal point. The Bellman equation can be written more formally as shown in equation 9.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{2}$$

In equation 9: $U(s)$ represents the Utility at a current state; $R(s)$ represents the reward of the current state; γ represents the discount factor of the model; a is an action of the set of Actions, $A(s)$; P(s' — s,a) is the transition matrix; and $U(s')$ is the utility of the next state, s'. Using this equation, we can calculate the utility of the state by using simple dynamic programming. This algorithm essentially works from the terminal states backwards and iterates through this algorithm until the world converges on their utilities. Bellman [22, 23] provides mathematical proofs as to why this will always converge.

Since we now know this simple mathematical equation can be used to solve the equation, we can write an algorithm which takes advantage of this by adding an acceptable error rate and continuing to iterate until completed. This algorithm can be written as shown in Listing IV.1

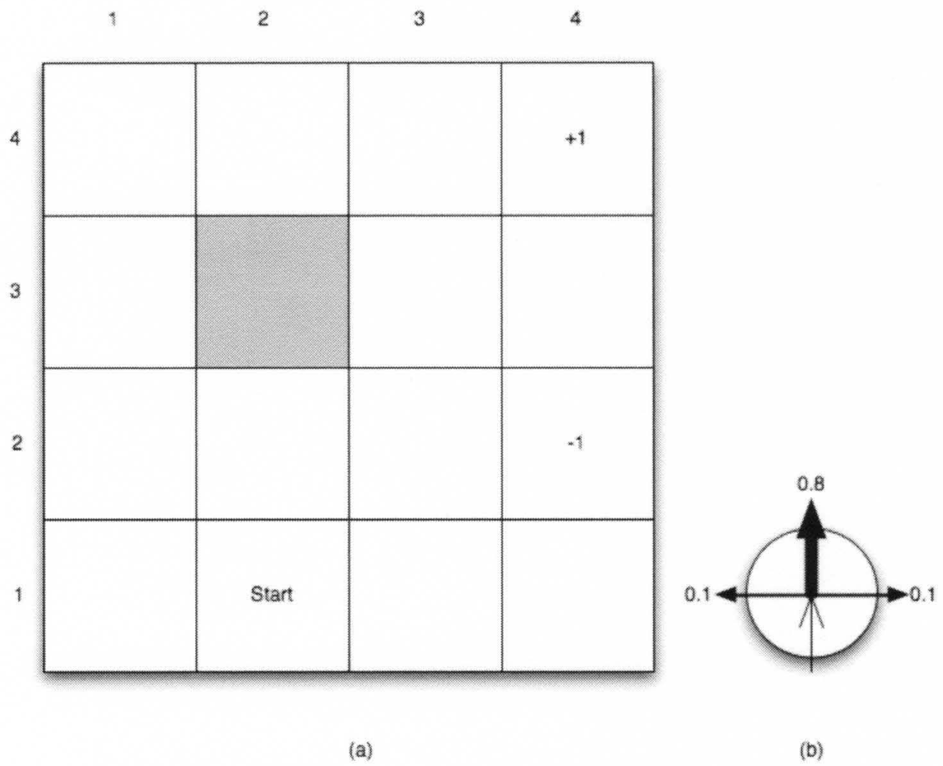


Figure 12. (a) A 4 x 4 world that would be presented to an agent to traverse through the safest possible route in order to maximize the final Reward of the system. The grey area represents an unpassable obstacle in the world. (b) The transition model of the world. Traveling forward results in an 80% likelihood of moving to the next state and 10% likelihood of moving to the right or left of the forward direction. This does not represent any particular direction in the world, but rather forward could be a move to the left of their current square and simply put means the probability of travelling to the desired state.

ALGORITHM IV.1. The MDP Value iteration Algorithm which uses the Bellman Equation to determine the Utilities of each state

```
function MDP( S, A(s), P(s' | s, a), R(s) )
    do
        U = U';
        delta = 0;
        for each (s in S) do
            U'[s] = R[s] + gam * maxAction(summation(P(s' | s, a)*U[s']));
            if (abs(U'[s] - U[s]) > delta)
                delta = abs(U'[s] - U[s]);
            end
        end
    while ( delta < epsilon(1 - gam)/gam );
    return U;
end
```

By running this algorithm, we are able to solve the Utilities for each state. Figure 13 shows the utilities of the world first presented in figure 12. Initially, all nonterminal states have a Utility of 0, and therefore only terminal states contain non-zero initial values. This algorithm iterates until the change in values is extremely small, or a user defined value for epsilon in the algorithm.

With the values that are returned from the Value Iteration algorithm show in figure 13, we can then determine the best policy from the results. In order to determine the policy, we simply move to the highest possible number on the board. Where ties result, such as in in square (3,3), with the utility 0.9000, and square (1,2), with the utility 0.6312, the agent is allowed to move to either of the tied squares. Therefore in square (3,3) it is allowed to move to either (3,4) or (4,3).

The best policy for this world is shown in figure 14. Notice as mentioned that squares (3,3) and (1,2) contain two possible directions to travel. Since we now have an optimal policy using the parameters which we passed the algorithm, we can look at the original world and determine how an agent would attempt to solve the world. If we look back to figure 12, we see the starting location of the agent is at (1,2).

We can look at our policy and we know that we can move in one of two

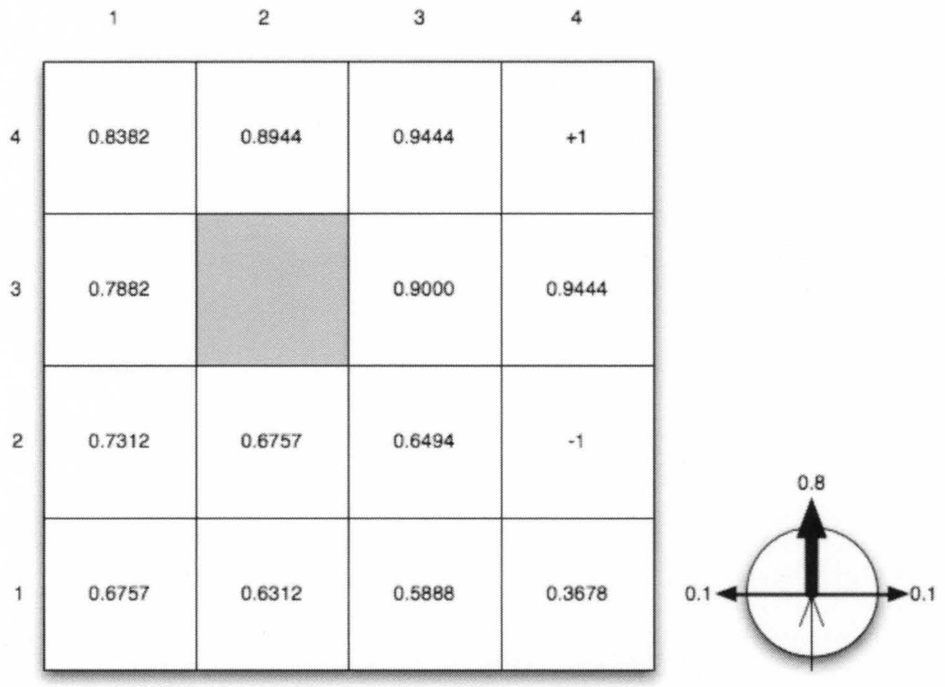


Figure 13. The utilities of every state in the world which was presented in figure 12. This algorithm was run using a $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states in the world.

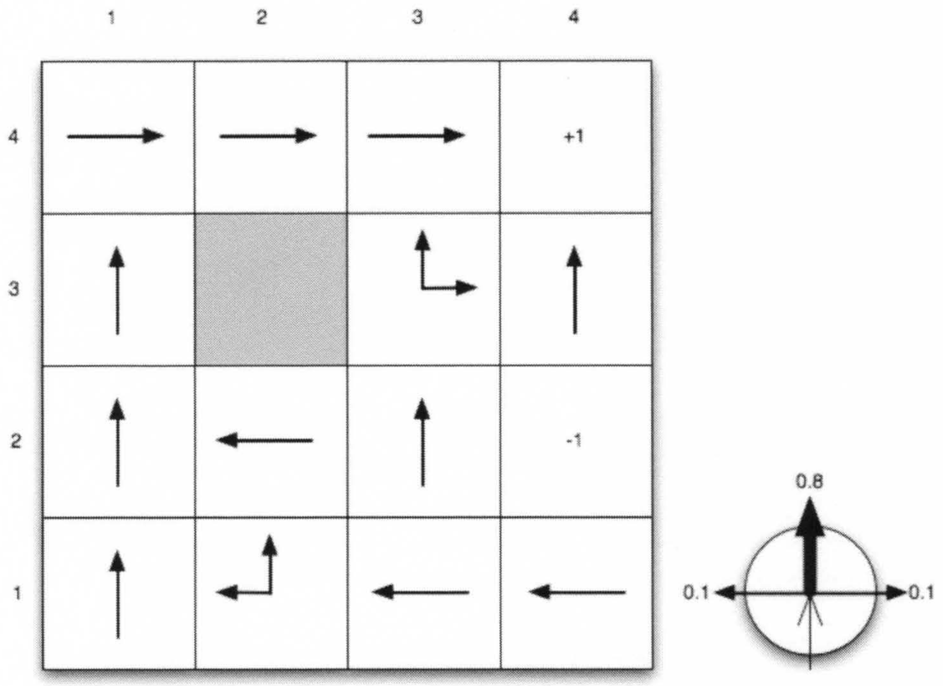


Figure 14. The resulting policy of the world shown in figure 12 and using the Utilities calculated in figure 13. The arrows represent the way in which the agent in the world should travel for an optimal solution.

directions, up or left. Following the arrows and assuming we are always successful, we can say that the optimal policy for the agent to travel is *[up, left, up, up, right, right, right]* or *[left, up, up, up, right, right, right]*. As mentioned previous in this section, this is not a shortest path algorithm. In a shortest path algorithm we could make a series of 5 moves, *[up, right, up, up, right]*, to reach the goal. However, this does not account for the -1 terminal point which would minimize the rewards in the system if travelled to accidentally. The MDP problem takes this into consideration and does not risk the possibility of hitting the negative reward terminal square.

We see now that this algorithm can be used to take into consideration the possibility of moving from one state to another given a stochastic world. This is ideal in Tymoczkos model where we want certain terminal states and how the agent in that world gets there is left up to randomness. In music, there are general rules which say that certain things should be avoided, such as jumps of tritones. Using a negative reward system, we can have the algorithm avoid these jumps entirely by negating the reward if it does jump a tritone in any way.

Since we now understand the basic concepts of the Markov Decision Process, we can look at algorithms which allow for much quicker determination of a policy. The MDP Value Iteration algorithm is well suited for smaller worlds but can be slow when presented with a much larger world. There is however an algorithm known as Q-Learning which is an extension of the MDP model used for Reinforcement learning.

B Q Learning

Q-Learning is a reinforcement learning technique first discussed by Christopher Watkins in his doctoral thesis in 1989. [26] It wasn't proven that his algorithm would converge on an answer until a few years later when Watkins and Dayan proved the convergence. [27] Q Learning works by learning an action-value function that gives the expected utility of taking a given action in a given state. The algorithm is different from the MDP in that unlike an MDP which gives a nearly exact utility, the Q Learning algorithm is an approximation of the MDP.

The Q Learning algorithm uses the Bellman update equation as a part of the algorithm itself. However, unlike the MDP which uses the Transition times the Utility, the Q Learning algorithm implicitly defines this as a part of the Q matrix. This is beneficial in many respects where a model of the transitions might not be known. Therefore, the bellman equation portion of the Q Learning algorithm can be written as shown in equation 3. [26]

$$Q(s,a) \leftarrow R(s) + \gamma \underset{a}{max} Q(s',a) \tag{3}$$

Later developments in the algorithm extended this model to include a learning rate. This variation of the Q Learning algorithm is called delayed-Q Learning and has substantial improvements over the traditional Q Learning. This brings a technique called "Probably approximately correct (PAC) learning" to the bounds of MDP. [28] The extension to this Q learning algorithm can be written formally as equation 4.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(s,a)\left[R(s) + \gamma \underset{a}{max} Q(s',a) - Q(s,a)\right] \tag{4}$$

This equation can be simplified as equation 5. [29]

$$Q(s,a) \leftarrow Q(s,a)(1-\alpha(s,a)) + \alpha(s,a)\left[R(s) + \gamma \underset{a}{max} Q(s',a)\right] \tag{5}$$

This mathematical foundation gives us oversight in understanding how Q Learning is related to the Markovian Decision Process. Similar to the MDP, we are again left with a highly mathematical algorithm. The Q Learning algorithm can be seen in Listing IV.2.

In this algorithm, we define any number of episodes to iterate and build the Q Matrix. The Q matrix is initialized to zero on all nonterminal states and set to the reward for the terminal states. We iterate through all episodes choosing starting states and actions at random and calculating the likelihood of the state action pair using equation 5. The selection of random actions for the current continues until a terminal state is reached. Once the episode has finished, the next episode continues and we start with a new random state location.

ALGORITHM IV.2. The Q Learning iteration algorithm for building the Q Matrix

```
function QLearning(S, A(s), R(s))
  Q = zeros();
  for each episode
    s = random(S);
    while(s != goal)
      a = random(A(s));
      Q(s, a) = Q(s,a)*(1 - alpha) +
        alpha*(R(s) + gam * maxAction(Q(s',a)));
      s = s';
    end
  end
  return Q;
end
```

After completing all episodes in this fashion, we are left with a Q matrix containing the estimated utilities for each state-action pair. Using this state action pair, we can traverse the world through the estimated utilities by simply choosing the action which maximizes the estimated utility for that state. This is by far the simpler of the algorithms presented here, as shown in Listing IV.3.

ALGORITHM IV.3. An algorithm to use the generated Q matrix

```
s = initial;
while (s != goal)
    a = maximumQValue(s);
    s = nextstate(s, a);
```

If we use the Q Learning algorithm on the world shown in Figure 12, we can determine the approximate utilities with the algorithm in Listing IV.2. Since this is a relatively small world, it normally converges on an answer quickly. Once the Q matrix is built, we can begin choosing the direction in which the agent in the world will move. Table 3 shows the Q Matrix built using the Q Learning algorithm with 45 episodes.

If we, for instance, start in the location shown in Figure 12, coordinate (1,2), we would look at the Q matrix and determine the best direction we can travel. For this, the highest number is 0.6312 on the Left Action. This means that from (1,2) we would move to (1,1). From here we again choose the highest number, 0.6757 on action up, and move in that direction to (2,1). We continue this until we reach the goal. If during the movement, a movement fails and we move to another square which was not our intended state, the Q matrix is still able to get us to the goal by creating what is, essentially, an optimal policy.

Figure 15 shows the estimated policy for the world. If we compare this q learning policy to the policy of the value iteration algorithm for the MDP, we see very few changes to the overall path. We do see that the Q Learning algorithm is slightly more conservative in that it takes no risks at (2,3) since it moves to the left in the Q Learning algorithm instead of up as in the Value Iteration algorithm. We also see that in (1,2) and (3,3) we are left with a single direction in the Q Learning algorithm, but the direction in the Q Learning algorithm is one of the directions shown for the value iteration policy as well. Therefore, only a single major

State	ACTION_UP	ACTION_RIGHT	ACTION_DOWN	ACTION_LEFT
(1,1)	0.6757	0.6057	0.6312	0.6412
(1,2)	0.6270	0.5617	0.5914	0.6312
(1,3)	0.5794	0.3780	0.5309	0.5888
(1,4)	-0.7443	0.1910	0.3499	0.3678
(2,1)	0.7312	0.6469	0.6412	0.6914
(2,2)	0.6386	0.6102	0.6031	0.6757
(2,3)	0.6476	-0.6911	0.3986	0.6494
(2,4)	-1.0000	-1.0000	-1.0000	-1.0000
(3,1)	0.7882	0.7475	0.7026	0.7475
(3,2)	0.8444	0.8370	0.6694	0.7476
(3,3)	0.9000	0.8749	0.6640	0.8394
(3,4)	0.9444	0.7156	-0.6556	0.6800
(4,1)	0.8038	0.8382	0.7638	0.7932
(4,2)	0.8538	0.8944	0.8538	0.8094
(4,3)	0.9050	0.9444	0.8694	0.8600
(4,4)	1.0000	1.0000	1.0000	1.0000

TABLE 3

The Q matrix built using the algorithm in Listing IV.2 on the world shown in Figure 12 using 45 episodes directional change occurs using the Q Matrix.

If we apply the Q Learning algorithm to the latent model for chord progressions using what is known about tonal harmonies, we should be able to use this Q Learning algorithm on Tymoczko's model by using the starting point of tonal harmonies a *I* chord and several positive terminal states, such as the *IV* chord, the *V* chord, and the *vii^o* chord. Using this information and rewarding major, minor, and diatonic chords, we should be able to use the traversal of the world to produce tonal harmonic progressions quickly and using techniques that are likely applied by composers without any thought to the compositional process. Chapter V discusses how the Q Learning algorithm has been altered to work in a way in which harmonies can be generated.

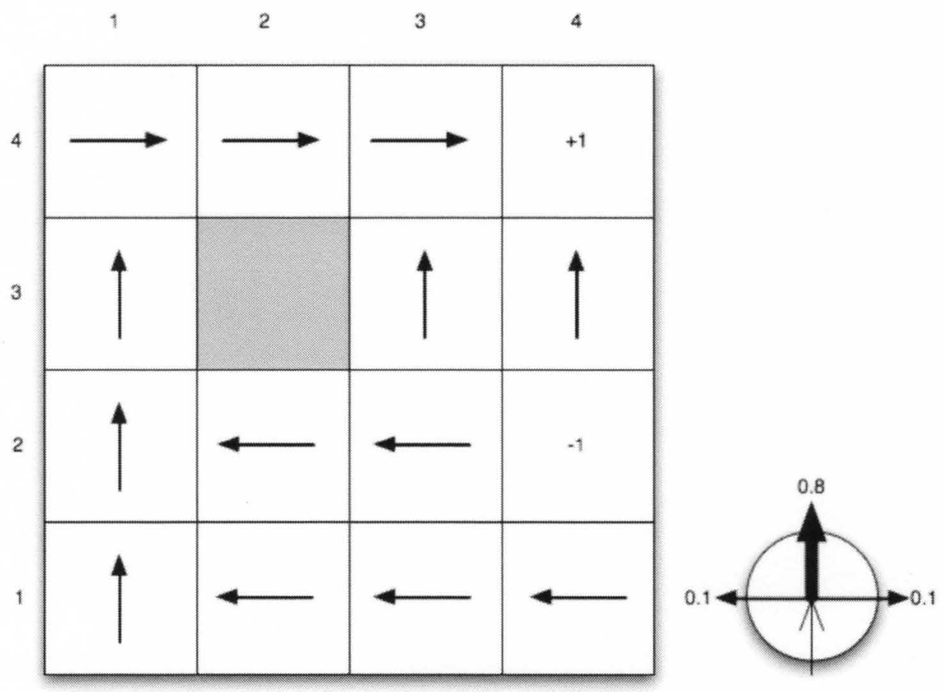


Figure 15. The resulting policy of the world shown in figure 12 and using the Q Matrix shown in Table 3. We see only minor changes in the overall policy between the Q Learning algorithm and the Value Iteration algorithm.

CHAPTER V

AN ALGORITHMIC TONAL MUSIC GENERATOR

Using the theoretical music concepts discovered by Dr. Dmitri Tymoczko and the computer science algorithms discussed in Chapters III and IV respectively, we can further develop these ideas to approach algorithmic composition with a new set of algorithms. Using these new set of algorithms for music composition, we leave the stochastic genre which was made prominent by Iannis Xenakis for more intelligent, decision making algorithms.

This chapter will discuss the algorithms and approaches that were taken for generating the music in the software created for this document. The chapter is laid out to discuss each of the parameters that have been included in this software. You will notice that a majority of the chapter focuses on the Chord Progression algorithm. This algorithm is the driving force behind the software. However, parameters such as scales and rhythms will be discussed as well.

A The MuseGEN engine

The MuseGEN engine (short for Music Generation Engine) is the engine that contains all of the primary functionality for generating music. In this iteration of the MuseGEN engine, three primary components exist. The Music Database, which consists of the database of possible scales in the program; the Rhythm Generator which generates all musical rhythms; and the Tonal Harmonic Progression

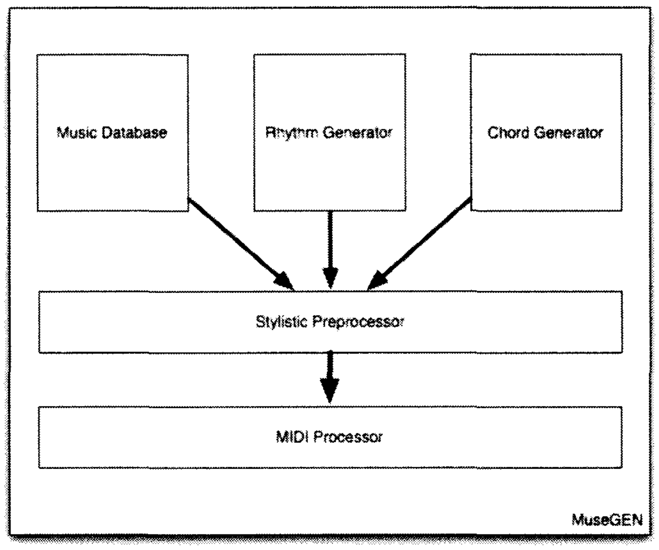


Figure 16. A flowchart of the MuseGEN engine during its iteration at the writing of this document.

generator, which is the set of classes for generating musical harmonies.

The engine was designed from the ground up to be modular. Each component of the engine is broken into separate modules which can be used by the stylistic preprocessor to generate music. Currently, only a single stylistic pre-processor class exists within the MuseGEN engine, used to generate music of the Minimalist style. This stylistic pre-processor then transfers MIDI information for the generated tones to the MIDI processor. The MIDI processor is a dumb component, meaning that it has no source of error checking within the processor. The MIDI processor simply takes the MIDI information and adds the information to the track that is being generated. A visual representation of the layout of the engine, during the current iteration of this writing, can be found in figure 16.

Subsequent sections break down each of the components written for this document and the algorithms and techniques that were used to create the various components. The next section discusses the fundamental algorithm used for generating musical rhythms in this generator. This is followed by a discussion on the approach for storing scales and other musical parameters which are not easily created using algorithmic techniques in a form of Database using XML. The next section is a discussion on the algorithms and the mathematics that make up the generation of Tonal Harmonic Progressions in the music. Since MIDI programming is relatively obscure, the next section discusses both the Stylistic Preprocessor and the MIDI processor portions of the MuseGEN engine. The chapter ends with a discussion on the possible future directions for the MuseGEN engine and the modules that make up the engine.

B Generating Musical Rhythms

Musical Rhythms are, more simply, nothing but mathematics patterns of accents and unaccented beats in music. Because musical rhythms have only two states, they can be represented on the computer as boolean, or binary, states, zeroes and ones or true and false. Though we know that storage can be done using boolean states, this does not account for the evenness of the mathematical pattern that makes up musical rhythms around the world.

Godfried Toussaint [30] first proposed the use of the Bjorklund algorithm for generating musical rhythms which create an evenness between the accented and unaccented beats of the rhythm. The Bjorklund algorithm is an extension of the Euclidean algorithm which is designed to distribute bits of 1s and 0s evenly.

The Euclidean Algorithm is one of the oldest and most well known algorithms. Euclid described this algorithm in "Elements" in Proposition 2 of Book VII. The algorithm is a recursive algorithm used to calculate the Greatest Common Divisor of two integers. [31] The algorithm works by repeatedly replacing the larger number of the two numbers by their difference.

Consider as an example that we want to determine the greatest common divisor for the numbers 11 and 8. First, we subtract 8 from 11, which equals 3; then 8 minus 3 equals 5; 5 minus 3 equals 2; 3 minus 2 equals 1; and finally 2 minus 1 equals 1. Therefore, using Euclids description, we know that the greatest common divisor between 11 and 8 is 1. Listing V.1 shows the euclidean algorithm as written using a recursive technique.

ALGORITHM V.1. The Euclidean Algorithm as written using recursion

```
function euclid(m, k)
if k = 0
    then return m
    else return euclid(m, k % m)
```

The Euclidean algorithm itself is not enough to generate rhythms. The Bjorklund Algorithm extends this concept that was laid out in the Euclidean Algorithm. The Bjorklund Algorithm was originally designed to be used to even out the repetition rates evenly for Spatial Neutron Source (SNS) timing. In Bjorklund's papers, [32] and [33], the author lays out a detailed algorithm for generating symmetry between pattern width and repetition rate for SNS timing.

In this algorithm the author defines "Ugliness" in a pattern as any pattern which is not symmetrically separated. In [32], the author uses a pattern with a width of 8 and a repetition rate of 2. In this case, a symmetrical distribution of the repetitions would be [1000100] where 1 is a repetition. Bjorklund goes on in the same paper to say that since these cycles continually repeat, any rotation of the pattern is also optimal. Toussaint's [30] contends the same occurs naturally in music as well. Rotations of an optimal pattern gives various rhythms from around the world, which the author lists during the majority of the latter portion of the paper.

Bjorklund's Algorithm is relatively simply to understand and to implement using Boolean or Byte Arrays. Bjorklund [33] and Toussaint [30] describe the algorithm using a series of bits. In order to begin the algorithm, we need to know the maximum width of the set as well as the repetition rate. If for example the algorithm is given a width of 9 and a repetition rate of 5, the algorithm will compute the number of 1s needed in the sequence as 5 and fills the rest of the sequence with 0s. Therefore the algorithm builds a sequence which looks as such:

111110000

From this, we begin by, in a sense, moving the 0s underneath of the 1s.

11111

0000

Since this algorithm has no more 0s, no more steps need to be done. We can simply merge the columns into a single sequence from left to right, top to bottom. We therefore are left with a sequence which looks like:

101010101

If we choose a sequence which has more 0s than 1 (where pattern width minus repetition rate is less than the repetition rate), we simply continue the second step. So for example if we have a pattern width of 12 and a repetition rate of 5, we would have a sequence as such:

111110000000

From here we move 5 of the 0s under the 1s:

1111100

00000

we repeat this step again:

11111

00000

00

Since three of the columns have less rows than the other two, we then move 2 more of the columns under the first two columns:

111

000

00

11

00

Now that only 1 column has less rows, we stop here. We then combine the columns so that the symmetrical sequence is:

100101001010

Knowing this, we can easily create an algorithm that suits the needs of the algorithm. Listing V.2 is the code found in MuseGEN that is used for generating Musical Rhythms using the Bjorklund algorithm. This algorithm is written using the Java programming language and uses a Boolean Array to represent the accented (truths) and unaccented (false) beats.

ALGORITHM V.2. The Bjorklund Algorithm

```
private int Bjorklund(int m, int k)
{
    if(k == 0)
        return m;
    else if(k == 1)
        return this.Bjorklund(k, m % k);
    else {
        int location = 0;
        int searcher = (m - k < k) ? m - k: k;

         for(int i = 0; i < searcher; i++) {
             int newColSize = this.colSizes.get(i)
                + this.colSizes.get(this.colSizes.size() - 1);
             int oldLastSize = this.colSizes.get(
                this.colSizes.size() - 1);
             location += newColSize;
             this.colSizes.remove(i);
             this.colSizes.remove(this.colSizes.size()-1);
             this.colSizes.add(newColSize, i);

             for(int j=0; j < oldLastSize; j++) {
                 this.rhythm.add(location - oldLastSize,
                 this.rhythm.get(this.rhythm.size() - 1));
                 this.rhythm.remove(this.rhythm.size() - 1);
             }
         }
         return this.Bjorklund(k, m % k);
     }
}
```

C A database of scales

Unlike the other sections of the engine, scales posed a slightly different obstacles. Unlike the Rhythm generation or the tonal harmonic progression generation, generating world scales could not be represented using a single mathematical formula. For this reason, a different approach had to be taken in order to generate suitable musical scales.

To meet the requirements, a semistructured database was created using XML. The flexibility of XML allows future scales to be added to the list while keeping the data organized in a way where information might not be consistent between each scale. The primary object of the created database is the scale itself. Each scale then contains a set of fields that must be filled in. This is accomplished using XML Schema to validate the insertion of new information and that the file has been created properly.

Currently the only information that is stored in the XML database for each object are the fields of Title, Alternative Names of the scale, and the notes that can be used to build the scale. The Title field represents the primary name for that scale. The Alternative Name is used when a scale is referred to by any other name. For example, a Double Harmonic Major scale (which is it's primary name) is often called the Arabic scale as well. The Notes that can be used to build the scale are exactly the same between the two scales. The notes field itself comprises of any number of notes that MuseGEN then uses to build the scales.

The title field and the Notes field are single occurrences for a scale. This is done because the primary name for the scale is only a single name. The Alternative Names however is slightly different. Since a scale could theoretically have any number of names, the alternative names of the scales is let unbounded. This means that a user can insert any number of names to the database for this scale.

The notes that make up the scale is slight more complicated. Unlike what one may expect, the scale itself does not contain MIDI integers as representations of the scale. Instead, the database is created using an additive model. In this case, we can build the scale on any MIDI note based on the additive information that is retrieved from the database. Therefore each number in the database is a sum of the previous note in the scale. Therefore is we saw the Notes to be added as [0, 2, 3, 2] we would take any midi note we would like to build the scale on and add those numbers to the previous note in the scale. If we choose 46 as the starting note, we would add 0, then add 2 which would give us 48. We then add 3 to 48 which gives us 51, and then we add 2 to the 51 to get the note. This is a simplified example of how this information is used.

ALGORITHM V.3. XML Schema for created the Musical Scale Databse

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="scales"
        xmlns:tns="scales"
        elementFormDefault="qualified">

<!-- Simple Elements -->
<xs:element name="title" type="xs:string"/>
<xs:element name="alt" type="xs:string"/>
<xs:element name="note" type="xs:int"/>

<!-- Complex Elements -->
<xs:element name="notes">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="note" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="scale">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="title" maxOccurs="1"/>
            <xs:element ref="alt" minOccurs="0"
                        maxOccurs="unbounded"/>
           <xs:element ref="notes" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="root">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="scale" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</schema>
```

The XML schema for the created database can be found in Listing V.3. Using this schema, we create a database of scales by simply adding scales to an XML file which uses the schema as validation. After this file is created, MuseGEN reads the XML file using Java's built in XPath API to interpret the information in the database. These scales are then built using the above mentioned additive method and the MIDI notes are stored in an array. This array is then Mapped to the scale names using a Java Map data structure, using the scales name (and alternative names) to map to an integer MIDI note array.

Understanding how these scales are built and stored is important for the next section which uses the scale in multiple ways. First, the scale can be used to determine the root note of the scale from which the chords generated can be changed to reflect this information. The scale is also used to determine if a chord is a diatonic chord or not. Diatonic chords are simply chords that are based on the notes within the scale that has been selected. More information on how the scales are used will be described in one of the next section.

D Tonal Harmonies

Tonal Harmonic Progressions represent the major focus of this thesis and will be broken down and explained in detail. Many of the concepts that are explained here are extensions of the concepts of Geometric Music Theory, Markov Decision Processes, and Reinforcement Learning laid out in chapter III and chapter IV. Therefore it is expected that you have read these chapters before delving into the understanding of the tonal harmonic generation algorithm.

1 Minimizing Space Complexity

The latent model that is laid out in Dmitri Tymoczko's idea of geometric music theory has the ability to model all possible chords in western music based on the twelve pitch classes that musicians are extremely familiar. The downside to this model is that, though it captures all possible permutations within any n-dimensional torus, the number of permutations grow exponentially. Therefore the space complexity requirements for storing the number of chords that are defined in Tymoczko's model is equivalent to the notation in equation 6, where n is the number of notes in the chord.

$$O(12^n) \tag{6}$$

However, Tymoczko's model also captures the the voice leading aspect of the chordal progression as well. For this reason, the space complexity for storing the number of chords is expanded greatly. This is neither a good nor bad thing. It is simply a portion of the model which causes the need for excess storage of chords. However, in a subsequent paper, Tymoczko [21] defined a way to model various transpositions of chords using a dynamic programming technique.

Since chords in music theory are defined as a series of pitch classes played together, chords which share the same inversion of pitch classes can be combined into a single category of chords. Therefore a chord which is defined as [0, 4, 7], or [A, $C\sharp$, E], would be equivalent to [4, 7, 0]; [7, 0, 4]; or [7, 4, 0], as well as all other permutations of the 3 numbers. To simplify the space complexity, we can therefore simplify the space complexity using combinatorics.

In this example, we can therefore simplify the permutations into a combinations of pitch classes. However, we must also take into consider repetitions of pitch classes since Music theory defines [1,1,1] as a chord as well as [1,1,2].

Therefore we notice that there are repetitions of notes that make up chords as well as independent pitch classes. In combinatorics, we know that combinations with repetitions can be solved using an extension of the binomial coefficient. This combination is defined in equation 7.

$$\binom{k+n-1}{n} \tag{7}$$

In equation 7, n would represent the number of notes in the chord and k is defined as the number of possible pitch classes. In western music, k is statically defined as 12 pitch classes. Therefore, we can define the minimized space complexity using the modified binomial coefficient by replacing the variable k with 12. This space complexity is shown in equation 8. Also note that the top portion of the binomial coefficient is simplified by subtracting 1 from k immediately.

$$O\left(\binom{11+n}{n}\right) \tag{8}$$

Since Dimitri Tymoczko's model changes based on the number of notes in the chord, this simplification shrinks the world space significantly as the world space grows exponentially. In the case of 3 note chords, using Tymoczko's model, we would have a world of 1728 chords. Using the minimized world, we have a total of 364 chords in the world using 3 note chords. This is approximately 89% savings in the size of the world. This savings grows exponentially as the number of notes in the chords grow. If we build a map of 4 note chords, we have a map of 20,736 chords using Tymoczko's model and 1,365 chords using the modified combination process, a 93.4% savings in space.

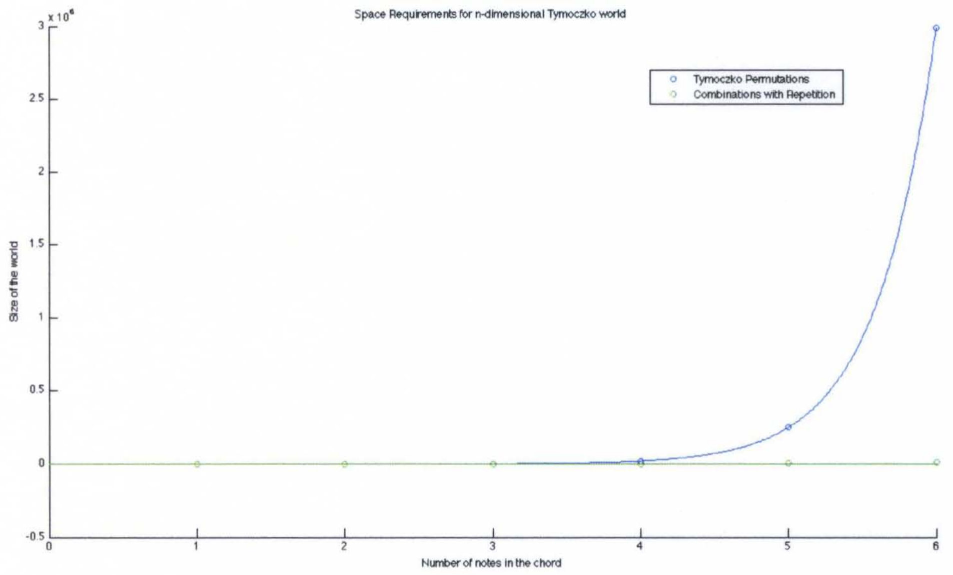


Figure 17. A plot showing the space requirements for the Tymoczko permutation world and the combination with repetition world. Plot generated in MATLAB.

If we continue to plot this out, we see that the savings become extremely significant as the number of notes, n, in the chord grows. Figure 17 shows the differences in growth between the two maps. there is a large significance in the growth until about 4 note chords where the Tymoczko world begins to grow significantly while the combination process remains relatively small at the higher orders.

Minimizing this space complexity does force the need for several additional computations in MuseGEN engine. The most significant of which is the addition of a Voice Leading algorithm which works to smooth the voicing between the playing of chords in the MIDI sequence. We also need to add additional connections between chords in order to capture the same information that might have been captured in Tymoczko's model. The latter of these is relatively simple to do, and is done while generating the world by looking at possible edges in the world from that chord and determining if a new edge is needed for a chord which has already been added. The Voice Leading algorithm is slightly more complex and is described in detail in a subsequent section of this chapter.

2 Altering the Markov Decision Process

Having discussed how the world that the algorithms will be using is built, we can begin to understand how the Markov Decision Process and Q-Learning discussed in chapter IV can be used to generate tonal harmonies. Despite this, neither Markov Decision Processes nor Q-Learning algorithms themselves offer a solution to the complete problem without some minor alterations to the discussed algorithms.

Recall from the previous chapter that a Markov Decision Process (MDP) is a 4-tuple graph algorithm containing the tuples of $(S, A, R(s), P(s'|s, a))$. In this tuple, S represents a finite set of states, A represents a finite set of actions, $R(s)$ represents the immediate rewards received for transitioning to a give state, s, and $P(s'|s, a)$ represents a transition matrix using the probabilities of moving to a state given the current state and the action that was taken. The primary goal of the Markov Decision Process is to find a policy of transitioning to reach a goal state.

Also recall, that the oft used method for solving a Markov Decision Process was defined by Bellman [22] . Richard Bellman defines what is most commonly referred to as the Value Iteration approach to find an optimal policy as equation 9. In this equation the actions from any state make up a large portion of the resulting Utility vector for finding an optimal policy. In most cases, these actions are predefined for all states and never change. The actions for the chord progression algorithm is slightly different from the actions that would make up a vector as defined in the Bellman equation, where γ is the discount factor.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{9}$$

In the chord progression algorithm, we have a large number of possible actions. For the purpose of this experiment, only a smaller subset of actions were chosen. However, calling the movement an action is slightly incorrect as the actions for this world are are themselves a tuple. Therefore, it may be better to call the action for the chord progression algorithm a velocity vector. It should be noted that in a MDP, $P(s'|s,a)$ can be simplified. Since it can be recognize that we are trying to find the probability of moving to state prime, s', and that it is known that s and a are independent variables, we know that solving the probability becomes the probability of being in the current state, s, times the probability of choosing an action (as shown in equation 10).

$$P(s'|s,a) = P(s)P(a) \tag{10}$$

However, it should also be noted that since this is a fully observable MDP, the probability of being in an incorrect state would be 0, and therefore P(s) would always equal 1. Therefore, the probability of transitioning to a new state simple becomes the probability of the action, as shown in equation 11.

$$P(s'|s,a) = P(a) \tag{11}$$

Since we have defined the action for chord progression as a velocity vector, $\vec{v}_x$, we can rewrite this equation replacing all action as the velocity vector (equation 12):

$$P(s'|s,\vec{v}_x) = P(\vec{v}_x) \tag{12}$$

The value of x in this equation for the velocity vector is dependent on the number of notes in the chord. If for example we have 4 notes in the chord, we would need to make 4 separate velocity vectors. Since it is easier to think about using 3 note chords, the following examples will use 3 note chords. Assuming we are using 3 note chords we might rewrite this equation as equation 13.

$$P(s'|s,\vec{v}_{ijk}) = P(\vec{v}_{ijk}) \tag{13}$$

It is easier to think about the equation by writing this Probability as equation 14.

$$P(s'|s,\vec{v}_{ijk}) = P(\vec{v}_i,\vec{v}_j,\vec{v}_k) \tag{14}$$

At this point, we can use the chain rule to expand this probability and simplify finding the probability. It is also important to note that $\vec{v}_i$ is independent of $\vec{v}_j$ is independent of $\vec{v}_k$. However, $\vec{v}_j$ is dependent on $\vec{v}_i$; $\vec{v}_k$ is also dependent on $\vec{v}_j$ and $\vec{v}_i$. Therefore we can expand this probability using the chain rule, equation 15, and then simply by removing variables which are not dependent on another in that chain, equation 16.

$$P(\vec{v}_i,\vec{v}_j,\vec{v}_k) = P(\vec{v}_i|\vec{v}_j,\vec{v}_k)P(\vec{v}_j|\vec{v}_i,\vec{v}_k)P(\vec{v}_k|\vec{v}_i,\vec{v}_j) \tag{15}$$

$$P(\vec{v}_i,\vec{v}_j,\vec{v}_k) = P(\vec{v}_i)P(\vec{v}_j|\vec{v}_i)P(\vec{v}_k|\vec{v}_i,\vec{v}_j) \tag{16}$$

It should now be noted that the tuple of $\vec{v}_x$ contain the direction of the action and magnitude of the speed. In this, a_x represents the probability of moving in a direction towards the goal while $speed_x$ represents the probability of a direct hit on the state. The latter portion relates to undershooting or overshooting a note in the chord. This was added to the chord progression MDP to encourage stochastic exploration of the world. We also know that each of the probabilities for the tuples are independent of one another, and therefore can simply the equation further. Since the probability of $\vec{v}_x$ contains independent tuples, we can redefine the Probability as shown in equation 17.

$$P(\vec{v}_x) = P(a_x, sp_x) = P(a_x)P(sp_x) \tag{17}$$

Knowing this information we can continue where we had left off in equation 16. All of the velocity vector probabilities can be replaced with the tuples, as in equaton 18. Then simplifying the equation becomes simple. Since the Speed are always independent of one another, and only actions make up the dependency in the velocity vector dependency, we can simplify the transition to its simplest form in equation 19.

$$P\left(\vec{v}_{ijk}\right) = P\left(a_i, sp_i\right) P\left(a_j, sp_j \mid a_i, sp_i\right) P\left(a_k, sp_k \mid a_i, sp_i, a_j, sp_j\right) \tag{18}$$

$$P\left(\vec{v}_{ijk}\right) = P\left(a_k \mid a_i, a_j\right) P\left(a_j \mid a_i\right) P\left(a_i\right) P\left(sp_i\right) P\left(sp_j\right) P\left(sp_k\right) \tag{19}$$

3 Q Learning

With this equation, we can begin to solve the Bellman Equation for the MDP. This extends into the Q Learning algorithm which uses an extension of the Bellman equation to solve the MDP. In the Q Learning Algorithm, the Q update function is defined as equation 20. Where the Q function $Q(s,a)$ is the Q matrix, the α function , $\alpha(s,a)$ is the learning rate, and γ is the discount rate.

$$Q\left(s,a\right) \leftarrow Q\left(s,a\right) + \alpha(s,a)\left[R(s) + \gamma \underset{a}{max} Q(s',a) - Q(s,a)\right] \tag{20}$$

This is equivalent to the equation in equation 21.

$$Q\left(s,a\right) \leftarrow Q\left(s,a\right)\left(1-\alpha(s,a)\right) + \alpha(s,a)\left[R(s) + \gamma \underset{a}{max} Q(s',a)\right] \tag{21}$$

In the case of Q learning, we can extend this function by simply replacing the actions, a, with the velocity vectors. The α function must always return a value between [0, 1]. For the purpose of this algorithm, all chords contain the same learning rate. Therefore this function can be more simply represented as a static number. For this reason, the Altered Q update function becomes the equation shown in equation 22.

$$Q\left(s,\vec{v}_{ijk}\right) \leftarrow Q\left(s,\vec{v}_{ijk}\right)\left(1-\alpha\right) + \alpha\left[R(s) + \gamma \underset{\vec{v}_{ijk}}{max} Q(s',\vec{v}_{ijk})\right] \tag{22}$$

We can continue to keep the Q matrix as a 2-dimensional matrix by writing a function to us the combination of velocity vectors to identify each uniquely and using that id in place of the velocity vector. Using this, we can modify the Q learning algorithm described in chapter IV. This modified algorithm is shown in listing V.4. This algorithm is modified from the original algorithm described by Watkins and Dayan [26,27].

ALGORITHM V.4. The modified Q Reinforcement Learning Algorithm

```
set gamma;
set R(S);
Q(:,:) = 0;

while(episodes)
    s = Rand(S);
    while(s != goal)
        v = Rand(V);
        Q(s,v) = Q(s,v)(1-learning_rate)+learning_rate*
            [R(s)+gamma*max_action(Q(next_state,v))];
        s=next_state
    end
end
```

One important aspect to the Q Learning function, and especially to the Q update portion of the function, is the rewards vector for various states in the music. Dmitri Tymoczko gives the basis for the rewards scheme in his book "A Geometry of Music" [20] states that all of the central chords in the model are the major, minor, augmented and diminished chords. These chords are defined by Tymoczko as dividing the octave nearly evenly. These central chords are the primary chords that are used by composers to create music. So when generating music, these chords are the chords that are most desirable.

For this reason, each element of the world can be weighted using several classifications of chords. For the purpose of this experiment, these classifications were broken down into: major chords, minor chords, augmented chords, diminished chords, diatonic chords, and other chords. A major chord is any chord who has two intervals which consists of an interval of a major third (a jump of 4 tones) and followed by a minor third (jump of 3 tones). A minor chord is any chord which consists of two intervals of a minor third (jump of 3 tones) followed by a major third (jump of 4 tones). An augmented chord is any chord which consists of two intervals of a major third (jump of 4 tones) followed by a major third (jump of 4 tones). A diminished chord is any chord which consists of two intervals of a minor third (jump of 3 tones) followed by a minor third (jump of 3 tones). Other chords are defined as chords which do not fall into any of the other categories above.

Diatonic scales are slightly more difficult to define in the scope of this project. In traditional tonal harmony, diatonic chords are chords which consist of notes that exist within the major and/or minor scales that are being used and is generally used to define the four types of chords explained above. Since the definition of a chord and the word tonal that Tymoczko defines in his book [20] are slightly altered from their traditional usage, it then comes to question whether the word diatonic can be used to escape the traditional meaning of the word.

For the purpose of this experiment, it was decided that the word diatonic should be altered very slightly to fit in with the larger scope of the tonal progression algorithm. In the scope of this experiment, the word diatonic has been used to explain any chord who contains pitch classes from within any defined scale. This therefore means that any chord, whether it is a traditional major, minor, augmented, or diminished, or whether it is simply a chord which contains notes from within the scale, becomes a diatonic chord. This allows the chords that are generated to contain a much larger gamut of sounds without modulation than limiting the word diatonic to its traditional usage.

The rewards scheme that was used for this experiment can be found in listing V.5. These numbers cannot be altered from within the application and must be altered from within the code if they are to be altered. These numbers presented were chosen arbitrarily. This scheme was chosen since Major chords represent the largest portion of chords that we wanted the algorithm to use. Minor chords follow that, but to avoid making the music sound too dark, this is significantly less than major chords. Augmented and diminished chords are extremely rare in music so these numbers, though still positive, have a much lower number than the other.

Since we primarily want to remain in the same key, an additional 1000 are added to all diatonic chords. All other chords get a negative reward in order to avoid ever playing those chords unless it occurs by random chance.

ALGORITHM V.5. The rewards system used by the Q Learning Algorithm

```
    private static double MAJOR          = 1500;
    private static double MINOR          = 150;
    private static double AUGMENTED      = 8;
    private static double DIMINISHED     = 8;
    private static double DIATONIC       = 1000;
    private static double OTHER          = -50;
```

The goals for the music were also given an addition score, at a significantly higher rate than any of the other chords listed. These goals are defined in music theory as the chords which result in cadences. These chords are better known as the major IV chord, the major V chord, and the diminished VII chord. These are simply stop points in the algorithm which allow the movement of the chords to continue on a chord which is used in cadences, primarily the tonic chord, the initial starting chord of the algorithm.

With this information, the Q Matrix can be generated using the Q Learning algorithm found in Listing V.4. This Q matrix can then be used to begin generating tonal harmonic progressions. The way in which the Q Matrix is used is defined in the next section.

4 Using the Q Matrix

Using the Q matrix is relatively simple. If you recall algorithm IV.3 in chapter IV, no significant changes need to be made to this algorithm in order to properly use the matrix. However, this algorithm itself does not provide a method for exploration around the world. For this reason, the algorithm is modified slightly to allow for some stochastic exploration of the world.

In this case, a velocity vector, $\vec{v_x}$, is used to allow for the stochastic exploration. Since the velocity vector consists of both the direction and the magnitude of speed, we can allow one of these to follow a distribution to allow for some slight randomness to allow for exploration. In this experiment, the speed was given an empirical distribution that closely resembles a normal distribution. This empirical distribution is given as the values [0.05, 0.15, 0.6, 0.15, 0.05]. Each of these resembles a final offset for each note in the chord. The offsets are additive to the final velocity vector that is calculated. These offsets represent undershooting, overshooting, or hitting the desired target. The offset order is: undershooting by 2, undershooting by 1, hitting the target, overshooting by 1, overshooting by 2. In an additive manner this is [-2, -1, 0, 1, 2].

By using this empirical distribution on all of the notes in the chord we add some stochastic exploration to the entire world. This does however remain a MDP problem, as opposed to a Partially Observable Markov Decision Process (POMDP). This remains an MDP simply because adding this offset does not change the fact that the process understands that it undershot or overshot a specific chord. A POMDP would require that the model has no way to determine if it has reached the next state.

ALGORITHM V.6. A modified algorithm to use the generated Q matrix using probabilitic offsets

```
p[ ] = {0.05 => -2,
        0.20 => -1,
        0.80 => 0,
        0.95 => 1,
        1.00 => 2}
s = initial;
while (s != goal)
    v = maximumQValue(s);
    for each(v element as n)
        r = U[0,1];
        v[n] = add offset where r < p[i]
    end;
    s = nextstate(s, v);
end;
```

Using this empirical distribution is done by generating a Uniform Random Variate between 0 and 1 (U[0,1]). The number that is generated is compared to the Cumulative Distribution Function, or in this case, adding the probabilities up until the generated random number is less than the summed probabilities.

Therefore, we can rewrite the algorithm that uses the Q Matrix to reflect this simple change in the algorithm. This algorithm can be found in Listing V.6. To allow the program to generate MIDI tracks based on the generate stated, an iteration of this loop is done when the Chord is requested by the program. This allows the program to generate a good voice leading for the music and to add the chord to the MIDI tracks without needing to generate excess amounts of chords before this is done. In that sense, the Java class that was created for this works more as a service to the rest of the program.

5 Solving Voice Leading

Now that we are able to generate chord progressions based on the Q Learning algorithm to solve the Markov Decision Process, we are left with one more musical issue in the program. If you recall from the "Minimizing Space Complexity" section in this chapter, the topic of voice leading was left open at the end of the chapter. In order to generate music which is good to a listener, the topic of voice leading must now be explored for multiple reasons. Firstly, voice leading plays an important part in make music sound smooth during transitions of chords. Secondly, Voice Spacing which are extremely far apart can lead to sounds which are awkward to most listeners.

Solving Voice leading can be solved simply by generating a matrix of the distances between the current note and the next note. We take a single note from the current chord and calculate the distance in both directions to each of the notes in the next chord. We continue this for all of the notes in the current chord. Therefore if we want to move from the [A, $C\sharp$, E] chord to the [$C\sharp$, F, $G\sharp$] chord, we can build a matrix as shown in table 4. To calculate the distance, we simply count the number of notes between the current note and the note we are calculating to in a direction, whether that direction is up or down. You will notice that both Up and Down are listed in the matrix. Calculating the opposite is done by subtracting the number that is opposite from 12.

Generating this matrix is relatively simple in comparison to using the matrix. There are two approaches that can be taken to use this matrix: Greedy and Brute Force. The MuseGEN engine has implement both types of approaches for users to

	$C\sharp$ (Up)	F (Up)	$G\sharp$ (Up)	$C\sharp$ (Down)	F (Down)	$G\sharp$ (Down)
A	4	8	11	8	4	1
$C\sharp$	0	4	7	12 (0)	8	5
E	9	1	4	3	11	8

TABLE 4

The Voice Leading Matrix for generating smooth voice leadings. This is generating by calculating the number of notes between one note given the direction that is desired.

use.

The Greedy Approach is quicker for larger matrices, but in most cases, the matrices are going to be relatively quick to brute force as well. The Greedy approach is more likely to find a suboptimal solution to the problem as well. In music, an optimal equation is not necessary in most cases, however it might be beneficial to have an optimal solution to prevent the voices from leaving becoming distant from one another over time.

The Greedy approach looks at the first note, in this case, we could assume the note A would be the first note. If we choose this note as the first note, we would choose the item in the matrix that contains the lowest number. Therefore we would choose $G\sharp$ as the note that the A would move to. The algorithm then prevents the other notes from being able to choose $G\sharp$. $C\sharp$ would then choose $C\sharp$ since the movement is a value of 0. E would then be left with F. In this voice leading, the Greedy approach does return the optimal solution, but again, this is not always the case. If for example, we use the chord [C,C,D] and move to [A,B,D] choosing the voice leading for the Cs first, we get the suboptimal solution of [D,B,A] where an optimal solution would return [B,A,D]. The greedy algorithm is shown in Listing V.7.

ALGORITHM V.7. The Greedy Algorithm used to generate smoother voice leadings

```
set VLMatrix;
for each(voice)
        choose lowest element from VLMatrix[voice];
        remove element chosen from VLMatrix
end;
```

Since it was decided that this experiment would only work with 3 note chords, The optimal algorithm uses an iterative approach, using a series of three loops to choose the most optimal approach. Essentially, every voice receives its own loop to help choose the optimal solution. Because of this, it becomes unusable as the number of notes in the chord raises. The algorithm has a running time shown in equation 23.

$$O(n^n) \tag{23}$$

However, the running time is insignificant for low order chords. Because of this, using the optimal algorithm for chord progression is possible. As mentioned, the algorithm in MuseGEN uses a series of 3 loops which loop over each of the voices and chooses the most optimal solution. This optimal solution is found by summing the voicing distances and choosing the voice leading which offers the lowest voicing distance. This algorithm for finding the solution for a three note chord is shown in Listing V.8.

ALGORITHM V.8. The Optimal Algorithm used to generate smoother voice leadings

```
set VLMatrix;
for(int i=0; i < VLMatrix[voice].length; i++)
    for(int j=0; j < VLMatrix[voice].length; j++)
        for(int k=0; k < VLMatrix[voice].length; k++)
            if(i==j || j==k || i==k)
                continue;

            if(sum(voiceLeading) < sum(minimumVL))
                minimumVL = voiceLeading;
        end
    end
end

return minimumVL;
```

No matter which method was used, the main purpose of this algorithm is to reintroduce the voice leading back into the latent model that was minimized during the generation of the chords in an earlier section from this chapter. This is also used to keep the chords from moving in a direction that spreads the voice far away from other voices in the chord. Using all of the algorithms and modifications to algorithms discussed in this section of the chapter, we can generate harmonic progression which seem to have some order and tonal center to the human ear. Despite this order, there are still many parameters that can be added to help give direction and more order to the tonal progressions. The future work that can be done with this project is discussed a little later in this chapter.

E Programming MIDI tracks

Up to this point in the chapter, most of the discussion has been on the three modules that have been created for this experiment. This section focuses on the two core components that are in charge of developing the MIDI files that are created in this experiment. The first of these classes is the stylistic preprocessor. This discusses the way in which the current stylistic preprocessor works. The final portion of this section of the chapter discusses the MIDI processor itself. This section will discuss how all of the modules that were discussed earlier are used to create sound, and how Java's MIDI API works to create MIDI sequences.

1 Stylistic Preprocessor

The stylistic preprocessor is perhaps the most important class in the MuseGEN engine. The original intention for this program was to make the program easy to switch between preprocessors. Since the preprocessor is the portion of the engine that is used to generate various styles of music, the preprocessor would ideally be able to replace one another. However, for the purpose of this experiment, the stylistic preprocessor is a static class that lacks any ability to be switched.

The current stylistic preprocessor was designed to be very minimalistic so that each of the generated chords could be heard clearly without the disruption of other sounds in the sequence. For this reason, the preprocessor only requires the three modules that were discussed in this chapter: the music database for scales, the rhythm generator, and the tonal progression generator.

The implemented stylistic preprocessor takes in a set number of measures and generates that many number of measures worth of chord progressions. The program therefore loops through all of the measures, continuing to prepare the MIDI information to be processed. In this stylistic preprocessor, a measure is defined as the length of the generated rhythm. Therefore, for each of the measures, the stylistic preprocessor loops through the length of the measure array. When a true value is hit in the rhythm a new chord is produced and stored for processing.

The Chord storage is produced as a multidimensional matrix containing every beat. Therefore the length of the preprocessed MIDI sequence becomes the number of voices by the number of measures times the length of the rhythm. When a false is present in the rhythm matrix, a -1 is stored in the preprocessed MIDI sequence. When a chord is generated, the information is translated from pitch class representations (zero to 11) to a MIDI integer which represents a pitch played at a certain octave. The initial octave of the MIDI integer is statically defined in the implemented preprocessor.

Generating the MIDI integer is relatively simple when the initial octave is statically defined. First, the base pitch must be calculated. To do this, we can take the base pitch representation (0-11), add one to the value and multiply the octave. This should give the initial base pitch for a voice. To play the first chord, we need to add the pitch which represents the chord's pitches for each of the voices. This becomes the first chord that is heard by the listener. From here, we take the returned chord and the path with which to reach the new chord for each of the voices. We take the the shortest distance to the new note and add or subtract the distance, depending on whether the note is moving up or down respectively. Once all information has been generated for that preprocessed MIDI sequence, the sequence is returned to the calling class so that the information can be passed to the MIDI processor.

2 MIDI Processor

The MIDI processor is much simpler in comparison to the Stylistic Preprocessor. The MIDI processor uses Java's MIDI API to generate a MIDI sequence which can be played back by the MIDI player class built for MuseGEN. The MIDI processor is a very simple class designed to process the information that is received from the Stylistic Preprocessor.

Since the information received from the Stylistic Preprocessor is a 2 dimensional array, the MIDI processor simply loops through the multidimensional array to add information to the MIDI sequence. The preprocessed MIDI information is a dual array containing an Y axis which contains the voices that are being played and a X axis that contains the time ticks for each note being played.

If you recall that the preprocessed information contains MIDI notes and a stop code (-1) where no changes are being made in the voice. Currently, this is the only stop code available in the MIDI processor. The MIDI processor interprets this code by simply skipping over the code and allowing the voice to continue playing through the new chord. The only changes that are made to the MIDI sequence occur when a new MIDI note is played in a voice.

MIDI sequences in Java are simple to create and adding notes to the MIDI sequence are fairly simple as well. For the current iteration of MuseGEN, all velocities on the note remain constant. These velocities represent the loudness and softness at which the note is play. The MIDI information then takes information about the length of the Y axis of the multidimensional array. The Processor then translates that by creating an n-dimensional ArrayList that contains all of the voices as an individual track built on the same sequence.

In order to first create a MIDI sequence to be played back, we must create a new Sequence object. Since our tempi may be user defined in the Stylistic preprocessor, we need to create an sequence object which relies on Ticks rather than a frame rate. This is done using the Sequence object constructor and the Sequence.PPQ division type in Java. From this created sequence, we can create a number of tracks which are dedicated to playing the notes of the music for each of the voices for the generated song.

For all of our experiments, we create 4 tracks, meaning the generated music has four separate voices in the music. This was modeled after the quartet musical groups where four players perform on different instruments; however, the sound that is heard in the MIDI is non-reflective of the voices. Each track uses a piano sound, which gives the illusion that each note that is heard is a single instrument.

After creating the tracks, we simply loop through the entire generated MIDI song from the stylistic preprocessor and tell each voice to turn the note on or the current note it is playing off. This is done by creating a Note Event using the Java createNoteEvent function and using either the $ShortMessage.NOTE\_ON$ or $ShortMessage.NOTE\_OFF$ to turn a note on or off respectively. The below code, in listing V.9, is used to generate a MIDI sequence which can then be returned and played back or saved to a MIDI file for later playback.

ALGORITHM V.9. The MIDI Processor class used in Java to convert the multidimensional array created in the Stylistic preprocessor to a MIDI sequence for playback.

```java
import javax.sound.midi.*;

public class processor
{
    private int[][] song;
    private Sequence sequence;
    private Track[] tracks;

    public processor(int[][] midipre)
    {
        try {
           this.song = midipre;
           this.tracks = new Track[this.song[0].length];
           this.sequence = new Sequence(Sequence.PPQ, 1, this.song.length);

           for(int i=0;i<this.tracks.length;i++)
           {
               this.tracks[i] = this.sequence.createTrack();
           }
        }
        catch(Exception e)
        {
            System.out.print(e.getMessage());
            System.exit(1);
        }
    }

    public void process()
    {
        int violin1curr = this.song[0][3];
        int violin2curr = this.song[0][2];
        int violacurr   = this.song[0][1];
        int cellocurr   = this.song[0][0];

        int violin1 = 57 + this.song[0][3];
        int violin2 = 57 + this.song[0][2];
        int viola   = 57 + this.song[0][1];
        int cello   = 45 + this.song[0][0];

        this.tracks[0].add(createNoteOnEvent(violin1 , 0));
        this.tracks[1].add(createNoteOnEvent(violin2 , 0));
        this.tracks[2].add(createNoteOnEvent(viola , 0));
        this.tracks[3].add(createNoteOnEvent(cello , 0));

        for(int i=1;i<this.song.length;i++)
        {
            if(this.song[i][0] == -1)
                continue;


            if(this.song[i][3] != violin1curr)
            {
                this.tracks[0].add(createNoteOffEvent(violin1 , i));
                this.tracks[0].add(createNoteOnEvent(violin1 +
                  this.shortestDistance(this.song[i][3] , violin1curr), i));
            }

            if(this.song[i][2] != violin2curr)
            {
                this.tracks[1].add(createNoteOffEvent(violin2 , i));
                this.tracks[1].add(createNoteOnEvent(violin2 +
                this.shortestDistance(this.song[i][2], violin1curr), i));
          }

          if(this.song[i][1] != violacurr)
          {
              this.tracks[2].add(createNoteOffEvent(viola, i));
              this.tracks[2].add(createNoteOnEvent(viola +
                this.shortestDistance(this.song[i][1], violin1curr), i));
          }

          if(this.song[i][0] != cellocurr)
          {
              this.tracks[3].add(createNoteOffEvent(cello, i));
              this.tracks[3].add(createNoteOnEvent(cello +
                this.shortestDistance(this.song[i][0], violin1curr), i));
          }

          violin1curr = this.song[i][3];
          violin2curr = this.song[i][2];
          violacurr   = this.song[i][1];
          cellocurr   = this.song[i][0];

      }
  }

  private static MidiEvent createNoteOnEvent(int nKey, long lTick)
  {
      return createNoteEvent(ShortMessage.NOTE_ON,
                             nKey,
                             VELOCITY,
                             lTick);
  }

  private static MidiEvent createNoteOffEvent(int nKey, long lTick)
  {
      return createNoteEvent(ShortMessage.NOTE_OFF,
                             nKey,
                             0,
                             lTick);
  }

  private static MidiEvent createNoteEvent(int nCommand,
     int nKey, int nVelocity, long lTick)
  {
      ShortMessage message = new ShortMessage();
      try {
          message.setMessage(nCommand, 0, nKey, nVelocity);
      }
      catch(InvalidMidiDataException e)
      {
          System.out.print(e.getMessage());
          System.exit(1);
      }

      MidiEvent event = new MidiEvent(message, lTick);
      return event;
  }

  private int shortestDistance(int note, int newNote)
  {
      int t = 0;
      if(newNote < note)
          newNote += 12;

```

```
        t = newNote - note;
        if((12 - (newNote - note)) < t)
            t = -1 *( 12 - (newNote - note) );

        return t;
    }

    public Sequence getMIDISequence()
    {
        return this.sequence;
    }

    private static final int    VELOCITY = 64;
}
```

F The future of MuseGEN

A lot of improvements still remain to be made in MuseGEN. Firstly, there are still many parameters of music that have not been added to this iteration of the program. One module that would be an excellent addition to the next iteration would be a dynamics module. This module would be used to change the loudness and softness of the notes to make the music feel more musical. This is an important aspect of all music as music is rarely found to remain at a constant level of dynamics.

A lot of improvements still remain to be made in the Chord Progression algorithm as well. This shows a great step towards a model which allows the computer to mimic the deterministic process that composers might use to generate creative new works. This is of course done by learning which chords lead well into new chords while still progressing the music towards a musical goal in a sentence. While this model does well in learning which chords lead well into other chords, it currently lacks the ability to create a progression towards a goal.

The first thing that would be added to MuseGEN Harmonic progression algorithm would be the ability to consider the stochasticism of the model. For this iteration, calculating the probabilities of the move was left out. However, this is an incorrect way of finding the best way to a goal. For this reason, adding the stochasticism would allow the computer to choose paths which more closely model the compositional determinism. This could be done by adding one simple line to the Q Learning algorithm, as shown in equation 24.

$$Q\left(s,a\right) \leftarrow Q\left(s,a\right)\left(1-\alpha\right) + \alpha\left[R(s) + \gamma \underset{a}{max} P(s'|s,a)Q(s',a)\right] \tag{24}$$

The second thing that would need to be changed would be the need for adding a temporal aspect to the compositional process. This would of course modify the Q Learning problem significantly which may cause issues when attempting to learn about rewards and goals. Using a temporal aspect in the chord progressions, we could have a changing rewards system allowing us to give more direction towards a specific goal. To do this, we might modify the Q Learning equation to appear as shown in equation 25, where τ is the symbol for the added temporal aspect.

$$Q\left(s,a,\tau\right) \leftarrow Q\left(s,a,\tau\right)\left(1-\alpha\right) + \alpha\left[R(s,\tau) + \gamma \underset{a}{max} P(s'|s,a)Q(s',a,\tau')\right] \tag{25}$$

And lastly, we would want to allow the learning algorithm to explore more of the world. We could do this by allowing the progression to modulate to a new key. This modulation would simply change the stop points. By changing the stop points we need to change the reward system as well. This is slightly hard as changing the rewards system would require the computer to learn more information. However, it may be possible to change the already learned information to fit into the modulated rewards system.

In addition to these changes, it would be beneficial to add new Stylistic Preprocessors into the code and modify the Stylistic Preprocessor to work independently of the program. This would simply require an overhaul of the current method for building Stylistic Preprocessors, which is not difficult, but rather time consuming.

Overall, MuseGEN was built as an experiment to show the possibilities of using Reinforcement Learning algorithms in the scope of computer generative music. The results show a fantastic start towards allowing computers to begin choosing Chord Progressions in a larger scope for generating new music. However, there still remains a lot of research that needs to be done before a computer could use this model effectively to generate new works that would sound extremely pleasing to a listener.

CHAPTER VI

PERSONALIZING THE MUSIC GENERATION SYSTEM

The overall idea of this project was to create a music generation system that would use a level of personalization to help create the music. This personalization would capture certain aspects of the music to help determine the parameters to use. One of the ultimate goals would be to tie this personalization into a random number generator to help guide the music in a personalized direction. However, for the purpose of this project, little was done in this area. The primary contribution to the engine through these personalization methods was through the creation of a specified file type which can be interpreted to help set various parameters.

A The .MGX file type

The MGX filetype was created for the purpose of this experiment to set the various parameters in the music for the MuseGEN. The MGX file extension was chosen due to it being a rare file type used mostly for save games in an older game and for Micrographx Picture Publisher Clipart files. The MGX extension is an easy to remember acronym for "Music Generation XML".

This file is based off of XML using a schema created for this file type specifically. The schema is found online at the author's personal website. In order to create the .MGX file, you simply give an XML file the .MGX extension and add the schema to the root element of the XML file. For this file type, the noNamespaceSchemaLocation should be used on this element. Therefore to initially create the .MGX file, you should include the XML tag and the root element "musegen". The code should then begin by looking like the code in Listing VI.1.

ALGORITHM VI.1. Start of the .MGX file type

```
<?xml version="1.0" encoding="UTF-8"?>
<musegen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation=
             "http://kreese.net/2010/MGXSchema.xsd">


</musegen>
```

The current location for the .MGX file types can be found in the listing, Listing VI.1, as well. This location is "http://kreese.net/2010/MGXSchema.xml". A web location for this file schema was chosen so that any future changes to the file type can be reflected in the MGX file. This allows validations to be done without the need of software designed to work with each type of MGX schema. This Web Location allows any XML validator, which uses XML schema, to check the validity of any .MGX files.

Much of the rest of the .MGX files were created with the future of MuseGEN in mind. Five major elements currently exist with only 3 of them being used to their fullest at this iteration of MuseGEN. The five elements are "root", "timesig", "scales", "modulations", and "data". The first four are representative of their musical counterparts. The "root" element contains 2 elements information about the root pitch of the scales that are being used, "note" and "octave". The "timesig" element contains a 3-tuple of integer elements which can be used to build the time signature in the music as well as the rhythm for the music. The "Scales" contains any number of "scale" elements which are either strings or integers representing one of the scales built into MuseGEN.

These three are the elements that are currently used by MuseGEN. In the future, modulations will be added to MuseGEN which will then take into consideration the fourth element in the .MGX file. The “modulations” element contains any number of “root” elements. This “modulations” element would be used to change the root pitch of the music and force the key of the scales that are being used to change. This would allow more exploration of the latent model that was discovered by Tymoczko.

The final element can be used to store information that the creator of the .MGX file deems important. This might be important facial feature locations or IP addresses of people hitting a server. The limits of this data are nearly limitless; however, the information must be stored as integers. This was put into place so that older .MGX files could be forwards compatible with future iterations of MuseGEN. In those iterations, if an element is missing, the engine would use all of the data elements to generate the missing information.

We therefore can see the final schema as is shown in Listing VI.2. Making an MGX file by hand is not a problem so long as it correctly follows the Schema below. An example MGX file can be found in Appendix C. The sample file is a complete sample that was built by hand with the exception of the “data” tags which were built using a random number generator using a normal distribution.

ALGORITHM VI.2. The XML schema for the .MGX file type.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <!-- Simple Elements -->
   <xs:element name="octave" type="xs:integer"/>
   <xs:element name="beat" type="xs:integer"/>
   <xs:element name="bpm" type="xs:integer"/>
   <xs:element name="accentedbpm" type="xs:integer"/>
   <xs:element name="var" type="xs:decimal"/>
   <xs:element name="note" type="xs:string"/>
   <xs:element name="scale" type="xs:integer"/>

   <xs:element name="root">
       <xs:complexType>
           <xs:all>
               <xs:element ref="note" minOccurs="0"
                      maxOccurs="1"/>
               <xs:element ref="octave" maxOccurs="1"/>
           </xs:all>
       </xs:complexType>
   </xs:element>

   <xs:element name="timesig">
       <xs:complexType>
           <xs:all>
               <xs:element ref="bpm" maxOccurs="1"/>
               <xs:element ref="beat" maxOccurs="1"/>
               <xs:element ref="accentedbpm" maxOccurs="1"/>
           </xs:all>
       </xs:complexType>
   </xs:element>

   <xs:element name="scales">
       <xs:complexType>
           <xs:choice>
               <xs:element ref="scale" maxOccurs="unbounded"/>
           </xs:choice>
       </xs:complexType>
   </xs:element>

   <xs:element name="modulations">
       <xs:complexType>
           <xs:sequence>
               <xs:element ref="root" maxOccurs="unbounded"/>
           </xs:sequence>
       </xs:complexType>
   </xs:element>

   <xs:element name="data">
```

```
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="var" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="musegen">
        <xs:complexType>
            <xs:all>
                <xs:element ref="root" maxOccurs="1"/>
                <xs:element ref="timesig" maxOccurs="1"/>
                <xs:element ref="scales" maxOccurs="1"/>
                 <xs:element ref="modulations" maxOccurs="1"/>
                 <xs:element ref="data" maxOccurs="1"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

B Biometric Personalization

One proposed way of generating personalized music using the technique in this document is through the use of Biometric Techniques. Perhaps the simplest to take from is using facial recognition. Since more and more laptops are beginning to include cameras built into the system, it is becoming more probable that a user of the system will have a web camera that could be used for such programs.

Facial Recognition is not the only biometric technique that personalization could be limited to. There is no reason that other biometric techniques could not be used just as effectively as Facial Recognition. For example, fingerprint scanners on laptops could offer yet another solution to biometric personalization in the system. There is also a possibility of using behavioral biometrics to personalize the output music.

All three of the techniques above offer viable solutions on modern computer systems. Unfortunately due to time constraints and running into unforeseeable issues in the coding of MuseGEN, no biometric personalization was implemented in the system. Biometric personalization is of interest in this system, and the original intention was to include such personalization. It is therefore one of the first things to do within the next iteration of the system.

C Massive Online Personalization

Another topic of interest has been the idea of using this system in an online type community. Much of radio is now turning to the internet. There are radio channels on the internet which provide listeners a personalized station for their favorite music, such as Pandora. And even so, little has been done in the area of online personalized generative music.

This section provides a brief discussion of the approach used for an independent study that will be conducted during the Fall semester of 2010. In it, we will begin generating music to be played on the internet that can be influenced by the listenership directly. For this, we must first set up a station that can be used to play the music. This can easily be done using Shoutcast or Icecast, daemons that run on the system allowing you to stream music to any number of listeners on the internet.

After setting this up, we can begin using IP addresses to determine the country in which the most listeners are coming from and using these IP addresses to influence the probabilities of moving from one chord to another. In this respect, the model becomes more of a Naive Bayesian model than a reinforcement learning model. However, the reinforcement learning model that is generated from Q Learning is still of use in determine how the changes should affect each chord.

In this example, we normalize the Q Learning Matrix so that each number represents a number between zero and one. We can then use these probabilities with some basic mathematics to change the numbers in the matrix dynamically during the generation process. We could then normalize again and use a uniform random number generator to process information to determine the next chord.

Since MuseGEN generates MIDI files, a little more needs to be done in order to play the music through shoutcast. However, there are applications on Linux which convert MIDI to WAV or MP3 files. Timidity++ is the most commonly used program for such tasks. With it, you pass the MIDI information into the program which then converts the program into a WAV file. Though this cannot be done in real time, we can generate portions of the song for every so many minutes and then convert and add the song to the Icecast buffer so that the generated music sounds continuous.

This personalization is a completely different view on the way in which music is generated than those proposed in this thesis. The intention of the personalization is the same though. More experiments will need to be run to validate whether this personalization offers the same type of result as those performed for this thesis. This personalization technique has little to do with the end results of the thesis, and is more of an interest in the possibilities of generative music than scientific results.

CHAPTER VII

CONCLUSION

The previous sections of this thesis discussed the possibilities for using the computer to emulate the compositional process that many composers use to create new music. None of the research was truly new in regards to the mathematics or theoretical aspects, but rather provides a unique implementation of the Q Learning algorithm in the context of musical composition. The research pushes towards the viability of using Tymoczko's latent model, discussed in Chatper III, to computationally generate harmonic progressions.

Though using stochastic processes is not a new task in music, no research I had run across had attempted to generate tonal harmonic progressions in such music. Using Markov Decision Processes to create such progressions does not seem to be a highly researched area. It does however seem to be possible to create tonal harmonic progressions using Q Learning. The Outcomes section below discusses some of the resulting music and compares it to a short passage written by composer Jennie Huntoon. The Future Work section is a culmination of all of the possible future work that exists and was discussed in several other sections, as well as other work that might be possible for the future of MuseGEN.

A Outcomes

The results of the experiment show some promising results. The fact that the music was able to built chords that center around the tonic of the scale proves that this might be a viable option for generating harmonic progressions. Figure 19 shows a generated passage from MuseGEN. The first chord is built on a $< C, D, G >$ chord and resolves to the chord at the end of the passage.

Throughout the passage in figure 19, we see the tonic appear as well. Starting in measure six of this passage, we see the alto voice holding the C pitch through the rest of the passage. We also see this appear in the bass voice in measures 11 - 14. The fact that these notes appear throughout the piece is indicative of the music being focused around the pitch class of C.

Figure 18 shows a similar passage written by composer Jennie Huntoon and used to compare the generated piece with a true piece composed by a human. This passage is in the key of e minor which means the central focus of the piece should be on the E pitch class. We notice that the $< E, G, B >$ chord is present in the beginning of the piece and is returned to later in the end of the passage. Though in the human composed piece, there are no measures where the pitch of E is held for mutliple measures, we do see the piece return to chords containing the E pitch frequently.

Because of the central focus of the pitch class C and E in figures 19 and 18 respectively, we can consider both passages to be tonal in harmony. However, there are still many things that make the generated passage no where near the depth of a human generated passage. One very obvious visibility is that the computer generated passage seems to repeat chords throughout the passage and sometimes for measure at a time. The human generated piece is much more flowing and free

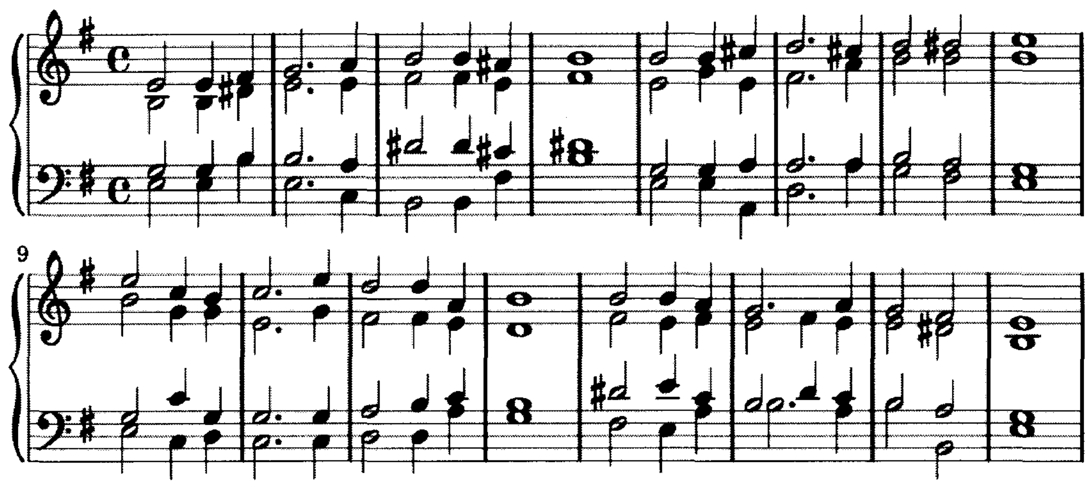


Figure 18. A passage written by composer Jennifer Huntoon using a four voice block chord style. The chord progression shows tonal harmonic progressions throughout the piece with some minor chromaticism as well. Music copyright of Jennifer Huntoon, used with permission from composer.

moving between chords.

This may be caused by a lack of temporal rewarding in the piece. Humans likely have the ability to comprehend the distance between the beginning and the end of the piece and base their entire selection of chords on this temporal reward system. The current implementation of MuseGEN does not take this into consideration, meaning the rewards system is purely static and does not change over time. As mentioned in the next section, this is something that should be looked into during the next iteration of this project.

Despite these issues, there is evidence that Tymoczko's model captures tonality and that by introducing stochastic decision making processes we can have a computer generate tonal progressions. The next section discusses where this project needs to go next in order to generate music which might be something that users would enjoy listening to. After introducing some of these additions to the

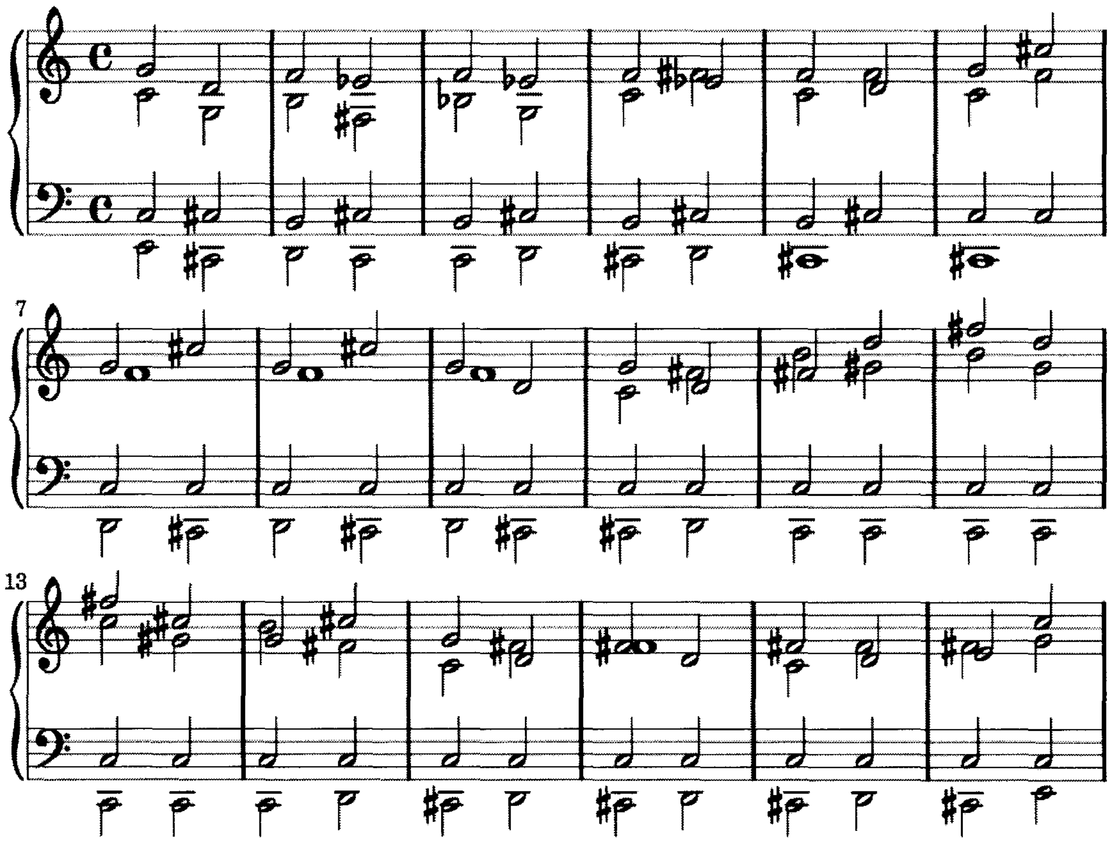


Figure 19. A passage generated by the MuseGEN engine. The passage uses 4 part voice block chords. It shows a tonal center around the pitch class c.

experiment, it would be worthwhile to have musicians analyze the music for potential errors and run a statistical analysis between the computer generated and human composed music in order to further validate tonality in the passages. There was not enough time or analyzers to do this in this thesis.

B Future Work

The Music Generation Engine created for this thesis still leaves a lot to be desired. It will not be writing music in the style of Mozart, Beethoven or other great composers any time in the near future. However, this was not the intention of this experiment. With the evidence provided in this thesis, MuseGEN has made a step forward in a self sufficient agent for generating possible chord progressions. Yet even with this evidence there still remains a lot of work.

The next step for this project would be to give the progression some form of direction. Currently the music seems to wander around its environment aimlessly. This provides an interesting sound for the music, but without direction, the music is not something so captivating that an audience might enjoy listening to for extended periods of time.

There are several ways in which it has been contemplated for giving the music a direction. The first would be through a combination of other Machine Learning algorithms to create a dynamic rewards system that changes with new training data. This would be relatively simple, and might rid the music of the dark sounds that are commonly heard in the music generated in this audio. Yet this still does not ensure that the music is given proper direction.

Related to the above possible future work of this engine is adding a temporal element to the Q Learning algorithm. Time is an important parameter that was omitted in this experiment because it posed many problems in the coding that would have been difficult. By adding these elements of time, we create the need for a major overhaul to the Q Learning algorithm where the Q Matrix exists as a multidimensional matrix containing not only state-action pairs but state-action pairs through a period of time. We also need to add the temporal element to the Rewards matrix. Because of this temporal element here, it might be better to use some training data from other composers to build the Rewards matrix. These topics are contemplated at the end of Chapter V.

Another possible derivative experiment with this project might be the understanding and learning of composer's preferred chordal movements. In very much the same way that David Cope's work with Musical Intelligence learns various aspects of the composer, we could observe chord progressions made by composers using a Bayesian network in order to allow the application to generate chord progressions in the style of Bach or Mozart.

These additions are perhaps the most important to the overall goal of this project. These are not, however, the limiting future work for the engine in general. There are still many parameters of music that are not taken into account by the engine. Firstly, though the element of Musical Rhythms are touched upon during this iteration, the musical rhythms are static and do not allow for the robust rhythms that are heard in music. Perhaps some combination of the mention Bjorklund algorithm and some form of genetic algorithm would suit the musical rhythm generation well.

The engine is also missing a melody generator. Melodies make up the flowing lines that exist in music, and without these flowing melodies, music is often looked at as boring. It is hard to take the melody into account until further developments are made on the chord progression algorithm. Once a chord is generated, choosing a melody is simpler. Yet even still, the development of the melody into actual music is something that would need to be observed as well.

Dynamics are yet another parameter of music unobserved by this engine currently. It would be a start to generate dynamics for the music in much the same way that stochastic music might generate chord progressions. However instead of jumping around, we might use a random number generator to choose whether to move the dynamic up, down, or remain the same so as to create smooth and flowing dynamics in the music. At some point, the dynamics might want to be guided probabilistically using a smaller version of the MDP through the movement of the melodic lines.

As one can see from this list of items which are possible for the engine, there remains a lot of work before music can be generated which might be pleasing to a typical listener. As mentioned several times in this thesis, these early experiments were used to prove the possibility of generating tonal harmonic progressions using Markovian Decision Processes. It would be the hope of this project to create a system which could generate music that any audience member could listen to and enjoy. However, from a purely scientific point of view in music, this experiment far exceeded the expectations that one might hope for generating tonal progressions. It may be possible with further development to create an intriguing progression of harmonies in future iterations of this engine.

REFERENCES

[1] L. Menabrea, "Sketch of the analytical engine," *Bibliotheque Universelle de Geneve. Translation and Notes by Ada Lovelace*, no. 82, pp. 1–59, Sep 1842.

[2] S. Kostka and D. Payne, *Tonal Harmony: with an introduction to Twentieth-Century Music*, 5th ed. McGraw-Hill, 2003.

[3] J. Burkholder, D. Grout, and C. Palisca, *A History of Wester Music*, 7th ed. W.W. Norton, 2005.

[4] K. Essl, *The Cambridge Companion to Electronic Music*, 1st ed. Cambridge Press, 2007, ch. Algorithmic Compositon, pp. 107–125.

[5] G. Boolos and R. Jeffrey, *Computability and Logic*, 4th ed. Cambridge Press, 1999.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. McGraw-Hill, December 2003.

[7] J. H. Chalmers, *Divisions of the Tetrachord.* Frog Peak Music, 1993, ch. 3. Aristoxenos and the geometrization of musical space, pp. 17–23.

[8] Ptolemy, *Harmonics*, J. Solomon, Ed. Brill, 2000.

[9] D. Cope, *Experiments in Musical Intelligence.* A-R Editions, Inc., 1996.

[10] H. Noguchi, "Mozart - musical game in c k. 516f," *Mitteilungen de ISM*, vol. 38, pp. 89–101, 1990.

[11] J. Cage, *Silence: Lectures and Writings.* London: Marion Boyars Publishers, 1978.

[12] I. Xenakis, “The origins of stochastic music,” *Tempo: New Series*, no. 78, pp. 9–12, 1966.

[13] ——, “La musique stochastique: elements sur les procedes probabilistes de composititon musicale,” *Revue d'Esthetique*, vol. 14, no. 4-5, pp. 294–318, 1961.

[14] ——, *Formalized Music: Thought and Mathematics in Music*, rev. ed. Pendragon Press, 1992.

[15] G. Koenig, “Composition processes,” *Computer Music Reports on an Intenational Project*, 1980.

[16] G. Wang, “The chuck audio programming language ”a strongly-timed and on-the-fly eviron/mentality”,” Ph.D. dissertation, Princeton University, Aug 2008. [Online]. Available: https://ccrma.stanford.edu/ ge/thesis.pdf

[17] G. Loy, “Musicians make a standard: The midi phenomenon,” *Computer Music Journal*, vol. 9, no. 4, pp. 8–26, 1985.

[18] D. Tymoczko, “The geometry of musical chords,” *Science*, vol. 313, no. 72, pp. 72–74, Feb 2006.

[19] ——, “The geometry of musical chords - supporting material,” *Science*, vol. 313, no. 72, pp. 1–29, May 2006. [Online]. Available: www.sciencemag.org/cgi/content/full/313/5783/72/DC1

[20] ——, *A Geometry of Music.* Cambridge Press, Sep 2010.

[21] ——, "Scale theory, serial theory and voice leading," *Music Analysis*, vol. 27, no. 1, pp. 1–49, Mar 2008.

[22] R. Bellman, "A markovian decision process," Rand Corp, Tech. Rep. AD0606367, April 1957.

[23] ——, *Dynamic Programming.* Princeton: Princeton University Press, 1957.

[24] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River: Prentice Hall, 2010.

[25] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 2005.

[26] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, 1989.

[27] C. Watkins and P. Dayan, "Q learning," University of Edinburgh, Tech. Rep., 1992.

[28] A. Strehl, L. Li, E. Wiewiora, J. Langford, and M. Littman, "Pac model-free reinforcement learning," *Proceedings of the 23rd ICML*, pp. 881–888, 2006.

[29] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* Boston: MIT Press, 1998.

[30] G. Toussaint, "The euclidean algorithm generates traditional musical rhythms," *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, 2005.

[31] Euclid, *Elements*, T. Heath, Ed. Dover, 1956.

[32] E. Bjorklund, "A metric for measuring the evenness of timing system rep-patterns," Los Alamos National Labs, Tech. Rep., 2003.

[33] ——, "The theory of rep-rate pattern generation in the sns timing system," Los Alamos National Labs, Tech. Rep., 2003.

APPENDIX A

Musical Foundations for Non-Musicians

Much of this document makes assumptions that the reader has at least a basic understanding of how to read music. I recognize that a majority of the readers of this document will be computer scientists and mathematicians who may not have the level of knowledge in Music Theory that this thesis skips over for the purposes of flow. This Appendix is meant to be a reference guide for those non-musicians who are interested in this topic but might be overwhelmed by the music terminology and musical renderings in the document.

A The Physics of Pitch

Pitch is one of the most basic foundations of general music theory. Kostka and Dorothy [2] define pitch as a reference to the highness or lowness of the sound produced. From a physical sciences standpoint, pitch is simply a musical word for the frequency of notes. The higher the frequency of a tone, the higher the tone sounds to the human ear.

The human ear has the ability to hear sounds between the spectrum of 20 Hz to 20 KHz (or 20,000 Hz). Because of this, there is an extremely large range of possibilities of pitches that the human ear can hear. Distinguishing between small differences in tone, especially at the higher range of human hearing, becomes difficult. Because of this, music theory defines a set of pitch classes, which separate frequencies into easily distinguishable sets of frequencies that are closely related to other pitches in the class.

The pitch classes in music theory are described using the first seven letters of the alphabet: A, B, C, D, E, F, and G. Further pitch classes are represented by lowering or raising each pitch by half a step. Since, for example, lowering a B and raising an A result in the same sound, and since raising a B is equal to a C (and and a lowered C sounds like a B) and E and F work the same way, we have a total of 12 possible pitch classes. Each pitch class is representative of a large set of possible notes which have a frequency ratio of 2:1 to its neighboring octaves. In simpler terminology, a pitch class of A contains tones with a frequency of 440Hz, 880Hz, or 220Hz (and all other tones which are $\frac{1}{2}$ or 2 times any tone frequency in that set). Each tone in the pitch class represents an octave of that pitch class.

Using this idea of pitch classes containing octaves, you can further represent the example given in the previous paragraph using various octaves numbers. The 440 Hz pitch can be represented using the characters A5 (an A in the 5th octave), whereas a 220Hz is represented as A4 (an A in the 4th octave) and 880 is represented as A6 (an A in the 6th octave). For the purpose of this thesis, our octave designation system is a version of the MIDI Octave Designation System. In this system, the lowest note available to Midi, Midi note 1 (8.176 Hz), is defined as octave zero and the tone itself is represented as C0, and the highest available tone, Midi Note 128 (12,543.854 Hz), is Octave 10 and represented as G10.

Understanding this concept of pitch class lays the groundwork for further elements of reading and understand pitches in music. The next subsection discusses how these pitches are represented in musical renderings, including alterations of the pitch class and enharmonic tones.

B Reading Pitches in Music

Each of the pitch classes that were discussed in the previous section can be represented in musical staves which can be viewed as a 2 dimensional plot where there Y axis is representative of the pitch and the X axis is the duration of the notes. However, instead of using traditional mathematical plots, musicians use a series of 5 lines with a total of 4 spaces separating each of the lines, as shown in figure 20. When a note is placed on any line or space within the staff, various pitches and note lengths in music can be represented.

In order to determine which pitch is being represented, musicians use clefs to indicate certain pitches on the range. Musician use three different clefs: the Treble clef or the G-clef, the Bass clef or the F-clef, and the Alto/Tenor/etc. clefs or the C-clef. In this document, all figures use either the Treble or Bass clefs. For this reason, the rest of this section will discuss pitches on the Treble and Bass clefs.

Figure 33 is an engraving of both the Treble (21(a)) and Bass (21(b)) clefs. These clefs have points on them which represent the note for which they are named which make it easy to remember, and to determine the pitch representations on and off of the staff. In the case of the Treble clef, the pitch of G is represented by ending the swirl portion of the clef on the line representing the pitch of G. The Bass clef works in a similar way by representing the pitch of F on the staff. In this case, the top curvature portion of the clef ends on the line representing the pitch of F. Using

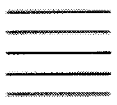
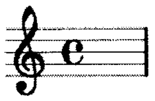
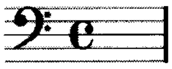


Figure 20. Rendering of a completely empty staff



(a) Treble Clef



(b) Bass Clef

Figure 21. An engraving of the (a) Treble and (b) Bass clefs on the staff.

these two clefs, you would now be able to derive any pitch representation on the staff by incrementing - or decrementing - from this pitch representation until the pitch you are attempting to find is reached.

One simpler way to remember the notes on the staff is using mnemonic phrases. We can remember all of the notes represented on the lines of the Treble clef staff using the mnemonic phrase "Every Good Boy Does Fine" where the first letter represents the pitch representation from the bottom line to the top line from right to left (E, G, B, D, F) in the mnemonic phrase. The spaces on the staff can be easily remember by using the mnemonic word FACE, where again, the letters of the word represent the spaces from the bottom up. The Bass clef mnemonic phrases work in the same way as the mnemonic phrase for the lines in the treble clef. The Bass clef mnemonic phrases for lines and spaces are, "Good Boys Do Fine Always" and "All Cows Eat Grass" respectively.

Since we now have two methods to determine and remember the pitches on the

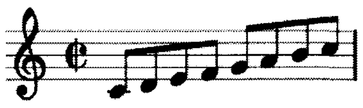


Figure 22. A C major scale which shows all pitch classes within one octave.

staff, we should be able to understand and read a simple scalar passage. Figure 22 shows a C major scale which shows all of the pitch classes on Treble clef staff. You can ignore the numbers written after the Treble Clef, these will be discussed in a later section in this appendix. Using the first method discussed, the pitch of G can be determined, which in this case is the fifth note in the figure. From here, all of the other pitches can be determined as well. By moving upwards on the staff, we increment the G by one. Since G is the end of our pitch class list, we return to the beginning of the list; therefore the note following the G is the pitch of A. Subsequent notes in that direction are B and C respectively. Moving downwards, we work backwards on the list. The note just prior to G is the pitch of F and we can continue this to determine all of the notes in the scale.

Using the second method allows us to figure out 6 of the notes in the scale without counting. Since our mnemonic phrase for lines is "Every Good Boy Does Fine" from bottom up, and our spaces have a mnemonic word, FACE, from the bottom up, we know that the third note in the scale in figure 22 is an E since this is the first line; the fourth note in the scale is on the first space which we know is F. We can continue this up to the 8th note in the scale, or C. However, since our mnemonic phrases only tell us between the first and fifth line, the mnemonic phrase does not help us in determining the first two pitches in the scale. To do this, we can go back to the first method and decrement the first line, E, to determine that the first and second notes are C and D respectively. Notes will occasionally go off the staff and, to the best of the authors knowledge, there is no easy way to determine the pitches of those notes without using the first method described.

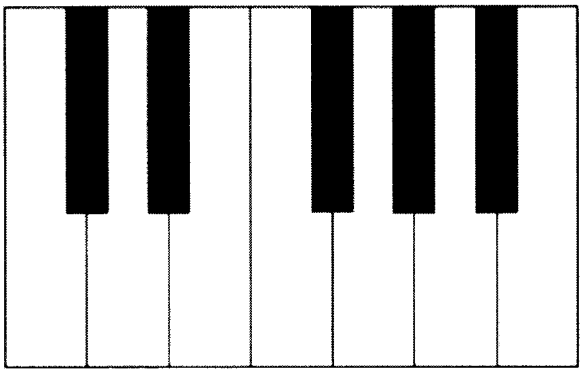


Figure 23. A visualization of a single octave on a keyboard from C to B. (Image Courtesy of "Jonathan Diet" and Public-Domain-Photos.com, Licensed under the Creative Commons)

C Altering Pitch Representations in Music

In the previous section, it was shown how musicians represent seven of the pitch classes that have been discussed. If you remember, there are a total of 12 possible pitch classes. In order to achieve the other possible pitch classes, we must alter notes by raising or lowering pitches. To better understand this, it would help to visualize an octave on a keyboard (See figure 23). The seven pitch classes that were discussed in the previous section are the white keys shows in figure 23. The rest of this section will discuss when the black keys are shown on the musical staff.

Musicians use two symbols to alter pitches in music. The sharp symbol ($\sharp$) is is used to raise the note by have a step. Conversely, the flat symbol ($\flat$) is used to lower the pitch by half a step. In modern music, both flats and sharps coexist as enharmonic equivalents. For example, raising a C and lowering a D result in the same pitch sounding. Generally the decision of which note to use is left up to the composer of the music.

If you refer to the keyboard image again, figure 23, you will notice the first two white keys - which are C and D respectively - are only seperated by one black key.

This is a C♯ or D♭ depending on which note the composer has decided to write in the music. However, both notes sound the same since they are both played on a single key.

1 Altering Pitches Directly

When a note in music needs to be raised or lowered half a step, the sharp and flat symbols can be used directly on the staff. When placing the note on the staff, the symbol is placed to the left of the note which needs to be raised. Figure 24 shows the C♯ major scale, in which all notes in the scale are raised by half a step from the C major scale.

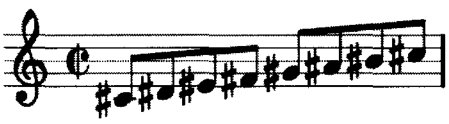


Figure 24. A C major scale which shows all pitch classes within one octave.

There is a special case, which should be discussed here. If you remember the visualization of the keyboard (figure 23), you may recall that there is not a black note separating the E and the F keys (third and fourth white keys respectively). In this case, raising an E by half a step will result in an enharmonic sound of an F while lowering an F by half a step will result in an enharmonic sound of an E. This same special case happens between the B and the C keys. These cases are caused by the tuning system used by music which splits the octave into 12 tones, where pythagorean theory determines various triads which make up the white keys on the keyboard.

Similar to the C sharp major scale, we have the C flat major scale which is constructed by lowering all of the notes by half a step. This scale is shown in figure

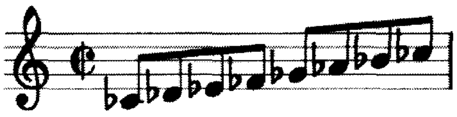


Figure 25. A C♭ major scale which shows all pitch classes within one octave.

25. Both figures 24 and 25 are shown by altering the notes in the C major scale directly.

Another rule that should be remembered is that when an accidental - an altering of the pitch directly - is used, the alteration affects the note throughout the measure it is written. In order to bring the accidental back to a white note, we use the natural symbol (♮). However, if instead of bringing the note back to a natural in the measure that contains the alteration, the alteration ends in the next measure, unless the alteration is written again and the natural symbol is not needed. The concept of a measure is further explored in section E.

In figure 26, the passage shows the first portion of Beethovens "*FürElise*" written in a slightly different time signature to show the effects of accidentals on a measure of music. Notice that in this measure of music, the first D is raised by half a step and that the second D has no accidental next to it. In this instance, the performer is expected to play the second D as a D♯. However, the third D has a natural sign written next to the note. Therefore, the first two Ds are to be played as a D♯, while the final D is meant to be played as a D (or D♮).

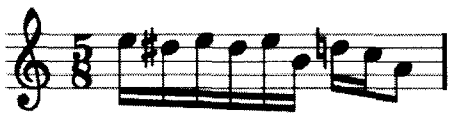


Figure 26. The opening passage of Beethovens "*FürElise*" written in 5/8 time to show the effects of accidentals on a measure.

If we were to rewrite Beethoven's passage again, as a two measure system and ignoring Beethoven's originally written pickup measure, we can see how accidentals would affect the appearance across a two measure system. Figure 27 is an engraving of exactly this; a bar separates measures. None of the nones were changed, therefore the first and second Ds are raised half a step despite the second D not having an accidental placed in front of it. Notice now in the second measure that the D has no natural sign written next to it. Since this D is now a part of a new measure, it is assumed by the performer or analyst that the D is a D natural and no symbol is needed.

2 Key Signatures

In much of music, altering all the necessary pitches can become a daunting task which leaves music nearly unreadable by the performers or analysts. Generally using a scale which is built on a specific note makes use of various flats or sharps to make a specific type of scale. For example, the C major scale shown in figure 22 is built on the pitch class of C. If we were to alter this scale so that the scale begins on D, the scale would no longer be a major scale. This is because scales are a static musical device; if you remember the visualization of the keyboard once more, using the C major scale we can determine the jumps of pitch classes required to create a major scale.

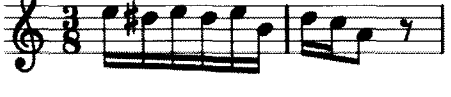


Figure 27. The opening passage of Beethovens "*FürElise*" written in 3/8 time and ignoring the pickup measure originally written by Beethoven to show the effects of accidentals across two measures.

If we tie each of the notes in the C major scale to the piano, we can see that the jump from C to D is of two pitch classes (in other words skip over one pitch class and play the second). The same happens between D and E. From E however, we now have a jump of only one pitch class, E to F. We can continue this to determine that a scale is built using the following jumps: 2-2-1-2-2-2-1. Now that we know this, all scales of the same general type (major, minor, etc.) must be built using these number of jumps.

If we return to the major scale which begins on D, we find that we no longer have just white keys in the scale. Building the scale using the jumps, we now see an F♯ and C♯ in the scale. If we were to write music based on this scale, we would required to write each note's sharps and flats into the sheet music. There is however an easy way to tell the analyst or performer that specific notes in the music should be sharp or flat.

In order to do this, we simply place the sharps or flats at the beginning of the music, next to the possible staffs. This lets the performer or analyst know that every note in the music will be sharp or flat. If we return to our C♯ major scale shown in figure 24, we can rewrite this so that the scale becomes easier for a performer or analyst to read. Figure 28 shows how this would be done. Notice that the engraving now looks very similar to the C major scale; however, since there are sharps written to the right of the staff, we know that every note in that system will be sharp.

We can do the same for the C♭ major scale shown in figure 25. This same

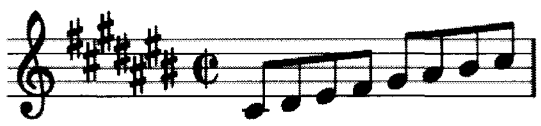


Figure 28. A C major scale which shows all pitch classes within one octave.

concept for the key signature would be used. Figure 28 shows how this key signature would appear in the music. Again, you will notice that the scale once again looks similar to the C major scale. The key signature does, however, make every note in that scale flat. The C♯ and C♭ scales were used as examples to show the extremes of altering pitches as these scales contain all of the sharps and flats in western music.

Every scale built on various pitches has a key signature which makes building the scales easy to remember. To remember the number of flats and sharps in the scales, musicians have created what is known as the Circle of Fifths. This tool starts with the C major scale at the top and works by jumping by seven pitch classes when moving to the right, or by five pitch classes when moving to the left. Each jump adds a sharp when moving to the right and adds a flat when moving to the left. To better visualize the circle of fifths, you can refer to figure 30, which also shows the associated key signature(s) for each of the jumps.

You may notice that in the Circle of Fifths image, the bottom elements of the circle contain more than one key signature associated with the element. These key signature are enharmonically equivalent, which simply means that when the scales for each of the key signatures are played, they are indistinguishable to the ear. These element's key signatures are use nearly interchangeably, depending on the enharmonic tone used to generate the scale. For example, D♭ and C♯ are enharmonically the same (as discussed in a previous section). If the scale is built using the C♯ as the primary note in the scale, we would use the seven sharp key

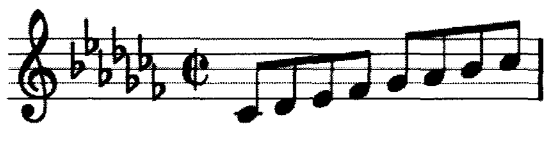


Figure 29. A C♭ major scale which shows all pitch classes within one octave.

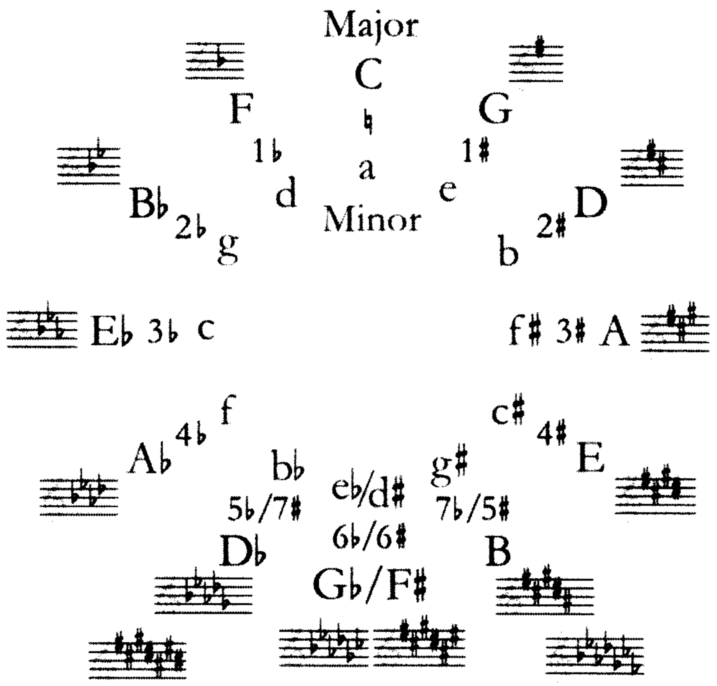


Figure 30. The Circle of Fifths which is used to show how closely related scales are and their respective key signatures. (Image Courtesy of user "Just Plain Bill" on Wikimedia Commons, Licensed under the Creative Commons)

signature. However, if we chose to use D♭ instead, we would use the five flat key signature.

As a note, key signatures can be changed in the middle of a piece. None of the examples used in this thesis will contain changes in key signatures. However, it is good to note such things as a possibility in other documents, related or unrelated to this thesis, containing musical examples. The way in which this is done is by either simply writing the new key signature, if you are adding sharps or flats, or by writing the natural symbol in the key signature to emphasize to the user that those particular tones are now flat. If you are changing from sharps to flats (or vice versa), you would first write the entire flat or sharp key signature as naturals to tell

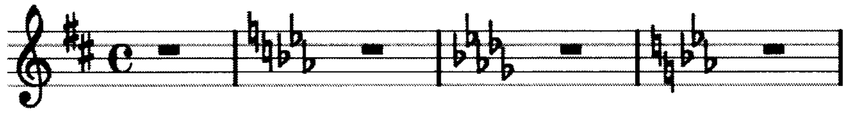


Figure 31. Moving between key signatures. This example shows 2 sharps (D major) moving to 3 flats (E♭ major) and then adding 2 more flats (D♭ major) and then moving back to 3 flats (E♭ major)

the user that key signature is no longer valid to this portion of the music, then write the new flat or sharp key signature in its entirety. Figure 31 shows how a composer might move from one key signature to another.

D Time Value of Notes

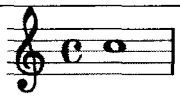
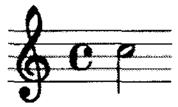
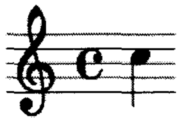
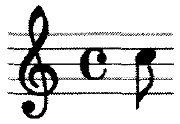
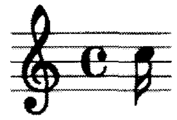
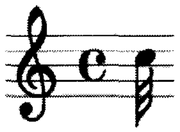
Prior sections in this appendix focused on one aspect of music, pitch. The rest of the appendix will focus on the second aspect of music, time. Musicians use a series of symbols in music notation which allow the performer or analyst to understand the timing of the start and end of notes at a specific point in the music. Table 5 shows the various symbols that will be discussed as well as a reference to the names which will be used in the discussion.

The table shows the note values from the Whole Note to the Thirty Second note. You may notice that as the notes get higher, they are generally some addition to the previous note to change the note value. For example, the half note is simply the addition of a bar connected to the whole note symbol and the quarter note is simply changing the half note so that the head, the portion that looks similar to the whole note, is colored in. From the quarter note, the addition of flags on the bar are used to cut the note values.

Generally, these notes are what you will find in almost all music. However, occasionally other shorter notes may be seen in western music. These notes are

TABLE 5

The Engraving of note lengths and the name of the symbol; table displays Whole note to thirty-second note.

Music Notation	**Note Name**
	Whole note
	Half note
	Quarter note
	Eighth note
	Sixteenth note
	Thirty-Second note

similar to the 32nd notes but contain more flags. By adding an additional flag to the 32nd note, we get a 64th note. Computer Scientists may notice that these notes fall in the typical binary system as all values of notes are multiplications to the power of 2. This makes understanding the values of the notes slightly easier to remember as Computer Scientists are used to working in the twos power domain.

These note values are generally not the only portion that represents time in the music. In fact, the note values discussed represent fractions of higher order notes. These symbols are simply used to make understanding the music as a performer or analyst possible. Any of the notes discussed can be potential beat values in the music. This portion of time is discussed in the next section on Time Signatures. The important portion of this section is to understand that each of the higher order notes can be broken into smaller segments using smaller order notes.

For example, a whole note can be broken down into half notes by including 2 half notes. A whole note could be further broken down into quarter notes by including 4 quarter notes. In comparison, a half note can be broken into 2 quarter notes and 4 eighth notes. To think about this in mathematical terms, which may generally be easier for mathematicians and computer scientists, we can use simple mathematics to determine how many of a smaller order note make up a larger order note. This equation can be found in equation 26.

$$2^{n_{low}-n_{high}} \tag{26}$$

In equation 26, n_{low} and n_{high} are the order of the note, or in otherwords the exponent portion of the power of 2s that were discussed earlier. For example, a half note is 2^1, therefore n_{high} would be 1, while an eighth note is 2^3, therefore n_{low} would be 3. Using this equation, we can find the number of eighth notes the half note can be broken into:

$$2^{n_{low} - n_{high}}$$

$$2^{3-1}$$

$$2^2 = 4$$

Therefore, we know that we can break a half note into 4 eighth notes.

On occasion, these note values are not enough and we need to extend the notes being used to add length. This is done by adding a dot to the right of the note. This dot adds a half of the notes original value. So if the dot is added next to the half note, our note becomes the length of 3 quarter notes. This works for any note where we want to add length. Figure 32 shows the notation for adding length to a note. The example shows a dotted half note, but the dot can be applied to any note length discussed in this section.

E Time Signatures

The final element that this appendix will discuss is the concept of time signatures and speed of music. The time signature is an important aspect of music, as the time signature is used to determine how the music is to be broken down. The time signature itself is a symbol which appears as a numerical fraction near the beginning of the staff. The time signature is used to allow the musician or analyst

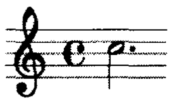


Figure 32. The Musical Notation for adding length to notes, this example shows a dotted half note.

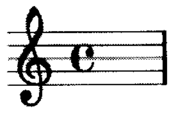
to better understand how the composer wants the "meter" of the music to feel. In rare cases, the time signature is left out and the music is unmetered. This gives the music a specific effect which is commonly found in ancient or medieval music, such as chant music.

It is not necessary to understand numerical fractions in order to interpret time signatures. A time signature is comprised of 2 numbers which are placed on the staff one above the other. The numerator of the time signature is the portion that tells the analyst or performer how many beats there will be in a measure. The denominator is the portion that represents the type of note which gets the beat.

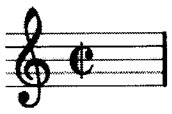
If you return to table 5 in the previous section, each of these notes can be used in the denominator of the time signature. The time signature is written as the twos power interpretation of the note. For example, a half note would be written as 2^1 or 2, and an eighth note would be written as 2^3 or 8. The numerator of the time signature can be written as any mathematically natural number.

Understanding both the numerator and the denominator give you an understanding of how the music is broken into measures. Each measure is separated by a bar on the staff. Before a bar can be placed, the sum of the value of all of the notes in the measure must be equal to the number of beats given in the time signature. If the time signature is written as $\frac{3}{8}$, we must have 3 eighth notes, an equivalent number of smaller order notes, or some combination thereof.

There are two special symbols used in place of time signatures that should be understood, as they may be frequently used in this thesis. The first symbol can be found in figure 33(a). This symbol is known as the Common Time symbol, and looks like a C. The symbol is generally used in place of a $\frac{4}{4}$ fractional time signature, as the common time symbol is equivalent to $\frac{4}{4}$. The second symbol can be found in figure 33(b). This symbol is known as the Cut Time symbol.



(a)



(b)

Figure 33. An engraving of the (a) Common Time and (b) Cut Time symbols used in place of a fractional time signature. These are equivalent to $\frac{4}{4}$ and $\frac{2}{2}$ fractional time signatures.

In mathematics, $\frac{4}{4}$ and $\frac{2}{2}$ would both be equal to one another. In music, this holds true for the number of beats within the measure as well. Though it is generally accepted in music that $\frac{2}{2}$ is twice as fast as $\frac{4}{4}$, given the number of beats total in the measure. However, they can both sound the same with changes in tempo as well.

Tempo is another important aspect of timing in music which is closely associated with time signatures. You may have noticed that time signatures had nothing to do with the speed of the music. The tempo is written at the start of the piece, either as a word representing the approximate speed or as a natural number representing the number of Beats per Minute (BPM). With the BPM, the music can be played as fast or slow as needed by playing the number of beats per minute as accurately as possible. The beat which is played is closely associated with the denominator of the time signature. For the purpose of this thesis, most of the examples will not have a tempo associate with the music engraving. This is for simplicity as the tempo has no effect on the purpose of the examples.

F Final Words

With an understanding of each of these sections, a non-musician should be able to understand and observe the various musical factors in this thesis. Though most of this thesis focuses on the mathematical properties of music and using various algorithmic approaches to take advantage of the mathematical properties, there are still musical concepts which cannot be avoided in this thesis.

To review what was discussed, we briefly went over the acoustical properties of music. This section has very little to do with the thesis itself, however having an understanding of the acoustical properties allows us to discuss pitch in a scientific manner rather than in a purely artistic way. These acoustical properties might be used for such things as signal processing which is a potential future project for this music generation engine.

The next two sections went over the twelve pitch classes found in music and how these are written so that musicians and analysts can understand what the composer wanted with respect to pitch. We discussed the Staff and various clefs used, while putting more emphasis on the two clefs which are frequent in this thesis (the bass and treble clefs). These clefs allow the analysts or musician to determine which pitch each note on the staff symbolize. Without these clefs, music would be unreliable. We also discussed the sharp symbol ($\sharp$) and the flat symbol ($\flat$) which are used to alter pitch classes. The natural symbol ($\natural$) was discussed as well for canceling an alteration in the middle of a measure.

The next two sections focused on the other important aspect of music, time. We discussed the symbology of various notes used in music as well as how to alter each of the pitches to included larger time values. The time signature was discussed to give the reader a concept of beats per measure. The concept of beats per minute were also introduced in this section to discuss the difference between time signatures and tempo, or speed of the music.

With an understanding of these topics, a non-musician reading this thesis should be able to understand each of the examples used throughout the thesis, as well as the majority of the concepts used in the software application built for this thesis. As such, only the important parts used for understanding this thesis were presented. Any further musical inquiries should be directed to a more thorough source, such as a book focused on music theory.

APPENDIX B

List of Scales used in the Program

The below table lists the scales which have been included in the program for the software to help guide the generation of music in the program. All musical engravings use C as the root pitch of the scale.

TABLE 6: List of Scales included in the MuseGEN package

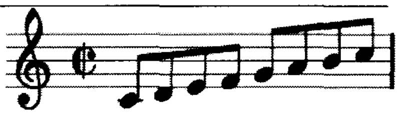
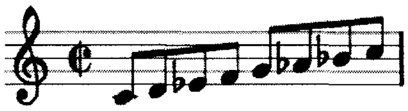
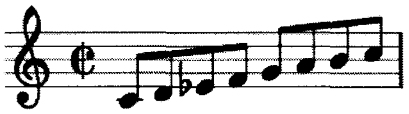
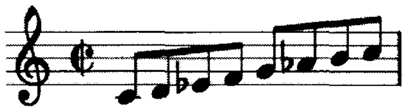
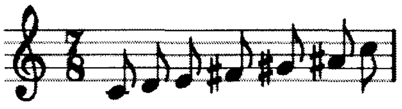
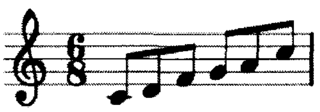
Scale Name	**Pitch Class Jumps**	**Musical Notation**
Major	0-2-2-1-2-2-2-1	
Natural Minor	0-2-1-2-2-1-2-2	
Melodic Minor	0-2-1-2-2-2-2-1	
Harmonic Minor	0-2-1-2-2-1-3-1	
Whole Tone	0-2-2-2-2-2	
Major Pentatonic	0-2-3-2-2-3	

TABLE 6 – Continued

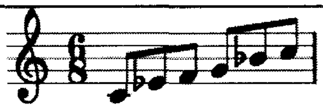
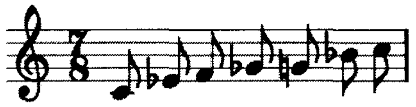
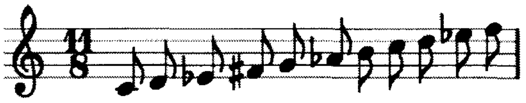
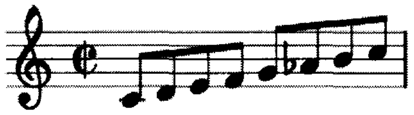
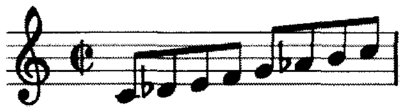
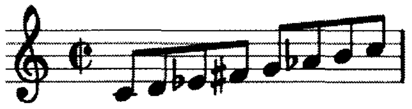
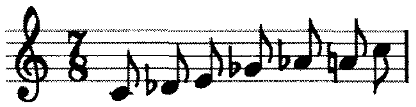
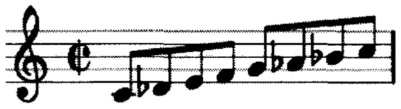
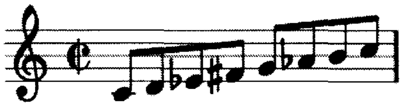
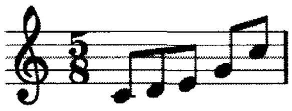
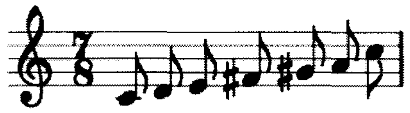
Scale Name	Pitch Class Jumps	Musical Notation
Minor Pentatonic	0-3-2-2-3-2	
Blues	0-3-2-1-1-3-2	
Algerian	0-2-1-3-1-3-1-2-1-2	
Harmonic Major	0-2-2-1-2-1-3-1	
Double Harmonic Major Arabic	0-1-3-1-2-1-3-1	
Double Harmonic Minor Hungarian Gypsy	0-2-1-3-1-1-3-1	
Hungarian Folk	0-1-3-2-2-1-3	
Phrygian Dominant Jewish	0-1-3-1-2-1-2-2	
Egyptian	0-2-1-3-1-1-3-1	
Eskimo Tetratonic	0-2-2-3-5	
Eskimo Hexatonic	0-2-2-2-2-1-3	

TABLE 6 – Continued

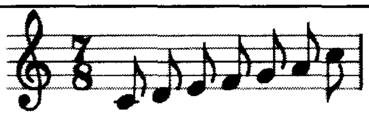
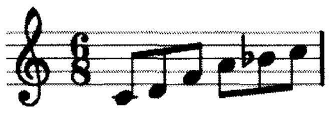
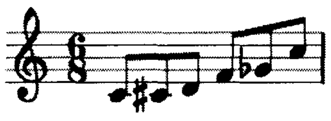
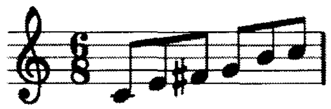
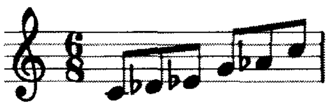
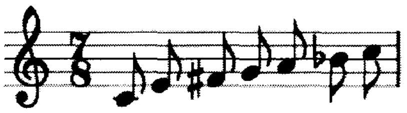
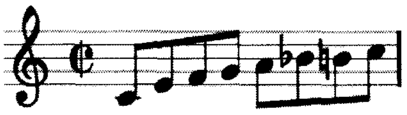
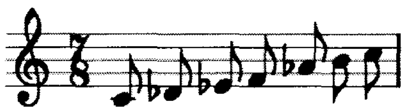
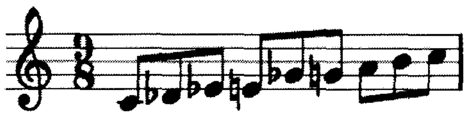
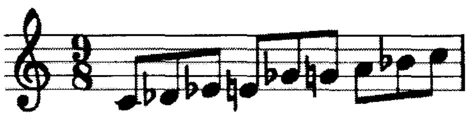
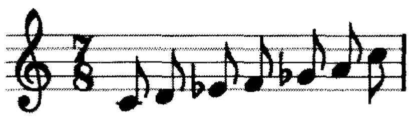
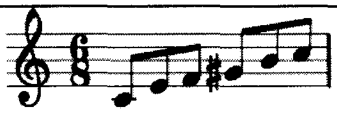
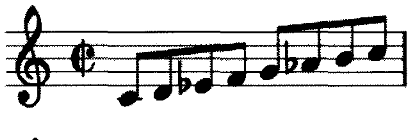
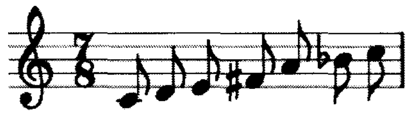
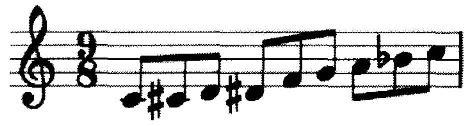
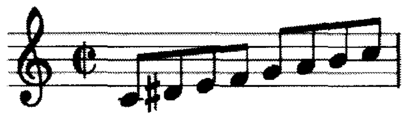
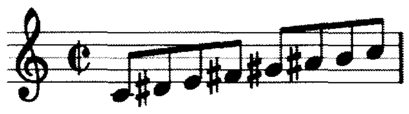
Scale Name	Pitch Class Jumps	Musical Notation
Scottish Hexatonic	0-2-2-1-2-2-3	
Oriental	0-2-3-4-1-2	
Oriental Pentacluster	0-1-1-3-1-6	
Chinese	0-4-2-1-4-1	
Balinese	0-1-2-4-1-4	
Raga Vutari	0-4-2-1-2-1-2	
Raga Madhuri	0-4-1-2-2-1-1-1	
Raga Viyogavarali	0-1-2-2-3-3-1	
Shostakovich	0-1-2-1-2-1-2-2-1	
Blues Octatonic	0-2-1-2-1-2-2-1-2	
Pyramid Hexatonic	0-2-1-2-2-3-3	

TABLE 6 – Continued

Scale Name	Pitch Class Jumps	Musical Notation
Romanian	0-4-1-3-3-1	
Gnossiennes	0-2-1-2-2-1-3-1	
Prometheus	0-2-2-2-3-1-2	
Adonai Malakh	0-1-1-1-2-2-2-1-2	
Houzam	0-3-1-1-2-2-2-1	
Rock 'n' Roll	0-3-1-2-2-2-1-1	

APPENDIX C

MGX File Sample

The below MGX file was generated as a sample for experimenting with making the MGX file work properly in MuseGEN. It was generated mostly by hand for the primary parts: the “root” tag, the “timesig” tag, the “scales” tag, and the “modulations” tag. The data var tags were generated using a normal distribution variate using a $\mu = 45$ and $\sigma = 12$.

ALGORITHM C.1. A sample .MGX file provided in the MuseGEN engine.

```
<?xml version="1.0" encoding="UTF-8"?>
<musegen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation=
                "http://kreese.net/2010/MGXSchema.xsd">

   <root>
      <note>Eb</note>
      <octave>5</octave>
   </root>

   <timesig>
      <bpm>9</bpm>
      <beat>4</beat>
      <accentedbpm>6</accentedbpm>
   </timesig>

   <scales>
      <scale>2</scale>
      <scale>4</scale>
      <scale>9</scale>
      <scale>6</scale>
   </scales>

   <modulations>
      <root>
         <note>Bb</note>
         <octave>6</octave>
```

```
        </root>

        <root>
            <note>F</note>
            <octave>5</octave>
        </root>

        <root>
            <note>C</note>
            <octave>5</octave>
        </root>
    </modulations>

    <!-- e.g. normal dist- mu:45 sigma:12 -->
    <data>
        <var>55.346080</var>
        <var>48.825183</var>
        <var>29.307740</var>
        <var>39.796896</var>
        <var>49.111494</var>
        <var>87.940763</var>
        <var>78.233244</var>
        <var>28.801357</var>
        <var>81.419082</var>
        <var>53.704851</var>
        <var>44.243342</var>
        <var>53.576915</var>
        <var>42.540407</var>
        <var>43.510268</var>
        <var>62.876371</var>
        <var>61.908414</var>
        <var>62.006309</var>
        <var>53.057966</var>
        <var>30.510157</var>
        <var>53.606864</var>
        <var>64.562823</var>
        <var>50.866725</var>
        <var>57.416316</var>
        <var>53.722622</var>
        <var>41.358709</var>
        <var>48.526458</var>
        <var>35.552606</var>
        <var>55.660748</var>
        <var>31.235159</var>
        <var>32.173555</var>
        <var>35.286016</var>
        <var>9.668590</var>
        <var>62.260564</var>
        <var>48.902286</var>
        <var>35.940860</var>
        <var>61.443582</var>
        <var>24.461803</var>
        <var>43.773091</var>
        <var>42.102636</var>
        <var>48.830481</var>
        <var>48.754303</var>
        <var>34.621441</var>
        <var>44.639384</var>
        <var>43.021452</var>
        <var>52.532487</var>
        <var>58.119188</var>
        <var>58.311280</var>
        <var>34.636166</var>
        <var>45.928309</var>
        <var>30.430595</var>
        <var>31.637991</var>
        <var>...</var>
    </data>

</musegen>
```

COLOPHON

The original source for this thesis was created in TeXShop, processed using the LaTeX engine, and output as a PDF. The musical figures in the thesis were created using an extension to TeXShop which allows for the Lilypond Music Notation language to be included and processed from within the LaTeX engine. These TeXShop engines were provided by Nicola Vitacolonna.

Any figures or images not created using Lilypond or original to this document exist as public domain, licensed under the creative commons, and are attributed within the caption for these figures. Such figures and images are copyright of their respective owners. Original images were created by the author and for the sole purpose of this text. Images which are not music engravings were created using various software tools, such as Omnigraffle, GIMP, and Dia. These images have no attributions and are copyright of the author.

Proofreaders: Jennie Huntoon, Kendra and Rick Reese (parents)

Committee Members: Adel Elmaghraby, Roman Yampolskiy, Charles (Tim) Hardin

CURRICULUM VITAE

NAME: Kristopher W. Reese

ADDRESS: Computer Engineering and Computer Science
University of Louisville
Louisville, KY 40292

EDUCATION: B.S. Computer Science
Hood College
2009

B.A. Music Performance
Hood College
2009

PUBLICATIONS: Reese, K., A. Salem, G. Dimitoglou. 2009. "Gaming Concepts in Accessible HCI for Bare-Hand Computer Interaction." In Proceedings of the 14th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES '09). pp. 40-46. Louisville, KY. August 2009

Reese, K., A. Salem, G. Dimitoglou. 2008. "Using Standard Deviation in Signal Strength Detection to Determine Jamming in Wireless Networks." In Proceedings of the 21st International

Conference on Computer Applications in Industry and Engineering (CAINE '08). pp. 250-254. Honolulu, HI. December 2008

Reese, K., A. Salem. 2009. "A Survey on Jamming Avoidance in Wireless Ad-Hoc Sensory Networks." Journal of Computing Sciences in Colleges (CCSCE '08). Vol. 24. Iss. 3. pp. 93-98. Frederick, MD. January 2009

Reese, K., G. Dimitoglou. 2008. "A Survey of Path Planning Algorithms for Autonomous Robotics." In Proceedings of the Consortium for Computer Sciences in Colleges, Eastern Conference 2008 (CCSCE '08). Frederick, MD. October 2008

PRESENTATIONS: *Doctoral Seminar*, March 27, 2011. "Generative Chord Progressions using Reinforcement Learning." University of Louisville, Louisville, KY.

PROFESSIONAL EXPERIENCE:

Yakabod, Inc., Frederick, MD (Feb. 2008 - Aug. 2009)

Web Programmer

- Worked on a Highly Secure application (The Yakabox), which is used by over 20,000 users worldwide and by government agencies such as the NSA.
- Researched and Developed Server- and Client-side systems for Large-Scale Information Retrieval and

Enterprise Social Networking using various web-based languages, frameworks, Web Services, and APIs.

- Automated the updating process for the Yakabox.

Blazer Radio (Hood College), Frederick, MD (Jun 2006 - Jun 2008)

Web Manager

- Managed the Blazer Radio website, maintaining updated information such as organizational data, schedules, and announcements.
- Developed web surveys using ColdFusion.
- Developed a program to store DJ play lists in a MySQL database and XML using PHP.

Volvo Powertrain, Hagerstown, MD (Jun 2006 - Aug 2007)

Summer Intern - Chief Project Management

- Helped to plan project within Volvo Powertrain
- Organized the Sharepoint portal to help resolve file-sharing issues in the company
- Utilized tools such as Microsoft's Excel, Microsoft Project, and Microsoft Access to show statistics, charts, and other data for projects.

Mosaic Sales, Frederick, MD (Oct 2005 - May 2007)

Epson Sales Representative

- Communicated the benefits of Epson products while

	assisting customers with identifying the product that best fit their needs. - Received training on products and took monthly exams on Epson's product lines.
TEACHING:	*2009-2010, University of Louisville*, Program Design in C – GTA *2010-Present, KCTCS* Introduction to Computers – Instructor *2010, KCTCS* Computer Maintenance Essentials – Instructor *2010, KCTCS* Advanced Computer Maintenance – Instructor *2011, KCTCS* Program Design & Development – Instructor *2011, KCTCS* Introduction to Database Design – Instructor *2011, KCTCS* Introduction to JavaScript – Instructor
AWARDS:	Summer Research Institute, Hood College, Frederick, MD, June 2008
ASSOCIATIONS:	Association for Computing Machinery, *since 2007* Institute for Electrical and Electronics Engineers, *since 2008* American Mathematical Society, *since 2010*