2000

MasterPiece: Computer-generated Music through Fractals and Genetic Theory

Amanda Broyles
Ouachita Baptist University

SENIOR THESIS APPROVAL

This Honors thesis entitled

"MasterPiece: Computer-generated Music through Fractals and Genetic Theory"

written by

Amanda Broyles

And submitted in partial fulfillment of the
Requirements for completion of the
Carl Goodson Honors Program
Meets the criteria for acceptance
And has been approved by the undersigned readers.

Dr. Terry Sergeant – Thesis Director

Dr. George Keck – Second Reader

Mr. Eric Phillips – Third Reader

Dr. Doug Reed – Honors Program Director

April 14, 2000

MasterPiece: Computer-generated Music through Fractals and Genetic Theory

Amanda Broyles

15 April 2000

Contents

Abstract

A wide variety of computer-generated music exists. I have written a program which will generate music by using genetic theory and fractals. The genetic theory is used to mold input pieces into a musical motif. The motif is then elaborated by the fractal formula into a composition. A brief introduction to the world of genetic theory and fractals is given. Analysis of a musical work produced in this manner shows coherent patterns and also emotion.

1 Introduction

MasterPiece is a program which generates music. The program is given example input music pieces to use as a basis. The lines from the input pieces are mutated together through genetic theory. The line produced is then fed into a fractal formula to generate an entire composition. By the use of mathematical formulas to perpetuate the musical output, I can generate structured compositions.

2 Previous Examples of Computer-generated Music

Computer-generated music examples are prevalent, especially on the Internet. The Internet is where my research began. I wish to review some examples in this section.

The first example has interesting applications in biology. The idea is to produce music from the RNA sequence of the AIDS virus. The RNA peptide sequence is used to determine the sequence of notes played. While, at first glance, this concept seems to be more of a curiosity, it does have useful applications. Patterns in the coding are easier to detect through the music than by simply viewing the lists of peptides. This technique can be applied to any sequence of RNA, not just the AIDS virus. The source code is written in LISP, a highly functional language which works well in this situation. The work was obtained from an email posted by Peter Stone [10].

Genetic algorithms and fractals have also been used to compose music, though I have never seen an example where they were used simultaneously until my work. An example of music produced by applying a genetic algorithm can be found in *Composing with Genetic Algorithms* by Bruce L. Jacob [6]. Jacob explains his implementation and approach to the problem of composing music with a computer. One interesting feature here is Jacob's use of an `EAR` function. It serves as a fitness function. The `EAR` function evaluates the music, and decisions concerning composition are based on this evaluation. It is used as a filter for unpleasant sounds. My first approach to the computer-generated music problem also included such a function. However, after a more in depth study of musical concepts and styles, I realized that there can be no definitive `EAR` function. In other words, beauty is in the eye of the beholder (or the ear of the listener, in this case). Many twentieth century techniques are based on upsetting the norm and questioning the boundaries of music. Therefore, I decided not to include boundaries with regard to musical style but, instead, to let the music take a free form.

3 Genetic Theory

3.1 Genetic Algorithms

A genetic algorithm is a method for solving problems in artificial intelligence. Genetic algorithms are algorithms which incorporate the theories of genetics and natural selection. They are usually applied to optimization problems.

A chromosome is composed of genes. A gene is a trait, a characteristic. In genetic algorithms it is sometimes useful, depending on the implementation, to consider a gene to be a single bit (either 0 or 1). If the chromosome has that characteristic, then the trait is turned on; the bit is a 1. Similarly, if the chromosome does not have the given trait, then the bit is turned off; it is a 0. A group of individual chromosomes make up a population. Genetic algorithms seek to optimize chromosomes within the population. A chromosome is a potential solution to a problem. For example, the chromosomes could represent different choices for scheduling events. However, they are not limited to such problems. They have been used to model economic systems, biological systems, and mechanical systems (product development).

There are five steps to a genetic algorithm.

1. Generate a random initial population.
2. Test the population to see if a solution is found.
3. While there is no acceptable solution, select the members to keep.
4. Generate new members.
5. Repeat from Step 2 until an acceptable solution is found.

There may be multiple solutions that are acceptable. For example, an acceptable solution may be any chromosome with a fitness rating of over 80%.

The fitness function, specified by the situation and data, is used to test the members of the population, the chromosomes. Chromosomes are generated in two ways. The first is through crossovers. As in genetics a new individual is created from two parent chromosomes. Part of the new chromosome's genes come from one parent, and the rest come from the other parent. The pattern in which the genes come from the parents is left to each genetic algorithm. Mutation of a chromosome occurs when one or more genes are mutated. A gene that was previously inactive may become active, or a trait that was turned on may be turned off. Due to the algorithm's unique combination of randomness and determinism, genetic algorithms are often very nice methods for solving a variety of problems [3].

3.2 Modified Genetic Theory

MasterPiece uses a modified version of a genetic algorithm, incorporating the genetic theory but not the fitness function or testing. In music there is no singular, widely-accepted standard to determine the quality or beauty of a musical composition. Hence, I deemed a fitness function to be an unnecessary complication. I only perform crossovers and mutations on my input musical chromosomes.

4 Fractals

Fractals have been a "hot topic" for mathematicians in recent years. The concept is not that difficult to understand and can even be generalized into one word: self-similarity. In 1977 Mandlebrot formed the mathematical representation, but philosophers and scientists have pondered the concept for years. Jonathan Swift wrote about fleas on fleas on fleas [4]. Even Doctor Suess had the right idea with "Horton Hears a Who." In a fractal a small section will look exactly like the whole. Thus, a city within a city, a flea on the back of a flea, etc., all exhibit self-similarity.

4.1 Cantor's Dust and Fractal Dimensions

Consider a thought exercise proposed by Cantor. Take a line segment.

Divide it into thirds and then remove the middle third. The result would look something like this.

Now, take out the middle third of these line segments.

Continuing . . .

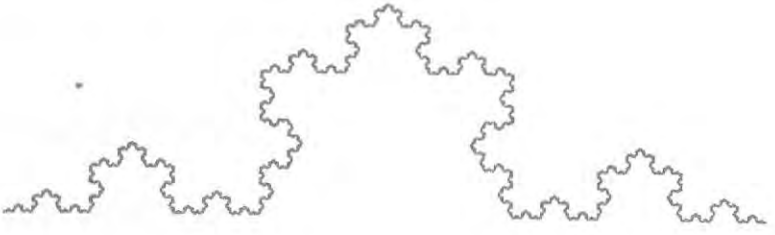
This became known as Cantor's Dust. At infinity, mathematicians might consider this to be a set of points, thus, having dimension 0. However, these really represent infinitely small pieces of a line segment. Line segments have dimension 1. So, one can see that it may be logical to give this set of Cantor's Dust a "fractional dimension" (somewhere between 0 and 1).

After n iterations the line segments would have a total length of $(2/3)^n$. It's fractional, or fractal, dimension is defined to be $\log(2)/\log(3) \approx 0.6309$ [8].

4.2 Koch Curve

Koch's Curve, proposed by the mathematician Helge von Koch in 1904, is an example of a curve in a finite amount of space that is infinitely long. How? Take a line segment to serve as the base. Then "break" it into four equal pieces and reassemble them as shown. This is called the motif of the fractal, or the building block on which it is formed.

Now, let each of the four segments (0-3) serve as a base, and repeat the process. Eventually, the figure will look like the following [8, page 28].



It is self-similar. At infinity, it will occupy a finite space, but it will have an infinite length!

4.3 The Classic Example and Everyday Life

If one looked at a map of a coastline, one might see its major inlets and the general curve of the shore. However, if one were to walk along that same stretch of coastline, it would be farther than the measure on the map because one would be walking around the smaller inlets that were too small to include on the map. Similarly, if an ant walked along the shore, he would have to walk around smaller indentations in the shore, further increasing his journey. This can be carried into the atomic and subatomic levels. Hence, it is a finite coast with infinite length. It all depends on the point of view [4].

Fractal formulas are very powerful. Due to their self-similarity, or recursion, the formulas can be fairly simple mathematically. This allows them to be coded easily. Scientists have started entertaining the notion that formulas for a human's biological makeup are stored as fractal formulas in DNA. If the position for every cell was specifically stored, it seems impossible that human DNA would be sufficient to hold all the information. However, if a formula which stated how often a blood vessel should branch was stored and if this same formula applied proportionally to smaller and smaller veins and capillaries, then it seems quite possible for a DNA sequence to be adequate [4].

In fact, many fractal patterns can be observed in nature. For example, the branches of a tree or the fronds of a fern may correspond to such a fractal pattern. "The bronchi of the human lung exhibit self-similarity over at least 15 levels [4]." Even in not so natural settings, such as the fluctuations in cotton prices, a fractal pattern of self-similarity was found to hold [4].

For a better explanation of fractal dimensions consult "Math and Real Life: a Brief Introduction to Fractional Dimensions [4]" or "Fractals [8]".

5 MasterPiece

5.1 Introduction

MasterPiece is a program designed to generate music through genetic theory and fractals. Example music pieces are input into MasterPiece. The pieces are then mutated together using the genetic theory discussed in section 3. The resulting line of music becomes not only a musical motif but a motif for the fractal formula to follow. The fractal formula expands the motif into a complete composition. The piece, though originated through metamorphasis and mutation of the motif, should be coherent due to the fractal quality of self-similarity. Of course, there are small flaws and approximations due to my approach. This will be discussed in section 7. The aesthetic quality of my endeavor is discussed in section 6.

5.2 In Detail

My MasterPiece project was implemented in three separate programs, `read.cc`, `MasterPiece.cc`, and `write.cc`. I also used software which I downloaded from the Internet. The software package, called Midi DisAssembler, is located at [1]. Basically, the Midi DisAssembler program takes files in MIDI format (extension `.mid`) and creates from them a text file. Likewise, the program also creates files of extension `.mid` from text files of a certain format. While this software has limitations, it was appropriate for my programming and testing purposes. One limitaion to the Midi DisAssembler is that, as I found, it will sometimes produce `.mid` files that cannot be opened with standard MIDI software, like `Finale`. This led me to wonder about the validity of the files. Were they being transfered in the traditional MIDI data structure? To date, I have no definitive answer to this. Even with its problems, the software was essential to the completion of my project. I designed my programs around its input and output. Examples of the text created by Midi DisAssembler may be viewed in Appendix A.

The diagram in Figure 1 shows the structure of my project. First, the three input pieces are fed into the Midi DisAssembler. This prepares them as text files to be translated further by the `read.cc` program. Here an intermediate text file is created for each piece. After the intermediate text files for each of the three musical works have been created, the files are given to `MasterPiece.cc` where the mathematical formulas are applied. This produces an intermediate output file representing the new composition. This composition is then translated to a usable form by `write.cc`. Adjustments for tempo may be added by hand. Then, the composition is reassembled by the Midi DisAssembler to create an output musical file.

Starting with the MIDI file in text format, as decomposed by Midi DisAssembler, my first program (in the series of three) is a translator. Its title is `read.cc`. This program scans the data file and picks up the important pieces of information. It reads the note value, the octave, and calculates the time until the note is turned off. An intermediate input file is created (see Appendix E).

The input file is then given to my second program, the one entitled `MasterPiece.cc`. Actually, three intermediate input files, each translated

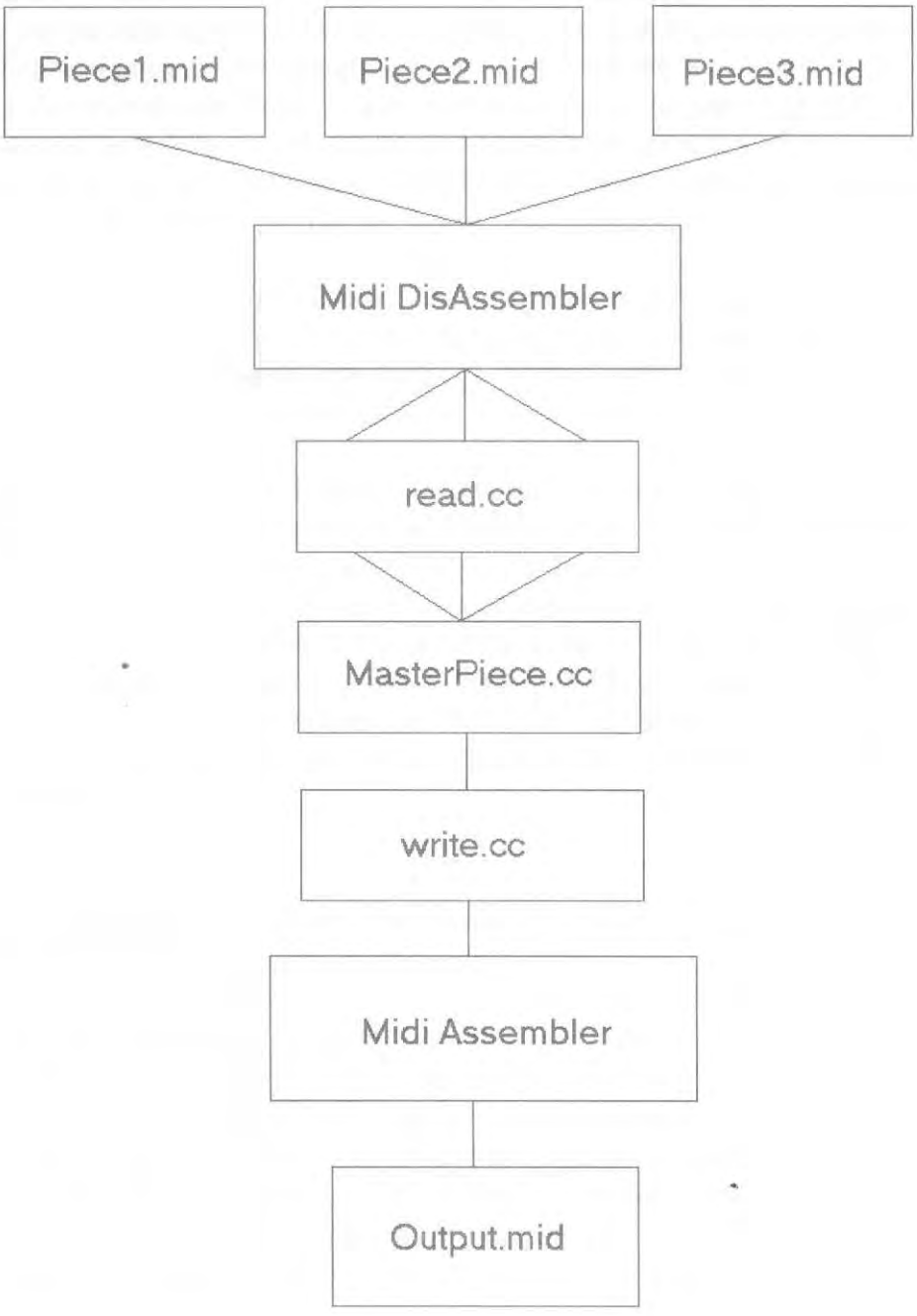


Figure 1: Overall diagram of the MasterPiece project

by `read.cc`, are given, one for each of the music pieces required to generate the motif[1]. Each piece is assigned to be a chromosome. The chromosomes are then put through a series of crossovers as seen in Figure 2. The boxes in the figure represent individual chromosomes, and the lines indicate lineage. The input pieces, themselves, will, for convenience, be short, approximately ten-second motifs from previously-composed musical works. For example, I could easily use the highly recognizable motif from Beethoven's *Fifth Symphony*. So, from the three pieces, a new motif is born.

It is worth noting at this point that a coherent piece can be created from any motif, consonant or dissonant. All this to say that one need not worry too much if the generated line does not "sound right." This line must then be formatted so it can be put into the fractal code.

Basically, I have four integers to put into the fractal formula. One represents note value. For example, `A` corresponds to 0, and `B` corresponds to 1. There is also an integer which is set to one if the note value is sharp. Since "♭" is not a character on the keyboard and since the Midi DisAssembler program only specifies sharp or natural, there is no need to specify this parameter. Also, any flat note can be represented as a sharp note one value lower. The third integer represents the offset value until the note is turned off (i.e. its duration). The octave in which the note resides is stored as well. So, the line

0 0 8 2

would correspond to `A♮` which has a duration of 8 and is in octave number 2. (The octaves start from zero.)

However, the fractal encoding scheme I have chosen to implement calls for a 2×1 matrix of numbers. I must use a linear transformation to crunch two numbers into one, thereby reducing my number of integers from four to two. Then, the matrix is put into the fractal formula. The fractal formula can be changed for different test cases. Once the new values have been generated, modular arithmetic, arithmetic on remainders, is used to separate one value back into two integers, resulting in a return to four values.

[1]This motif serves not only as a musical motif but as a fractal motif, as well.

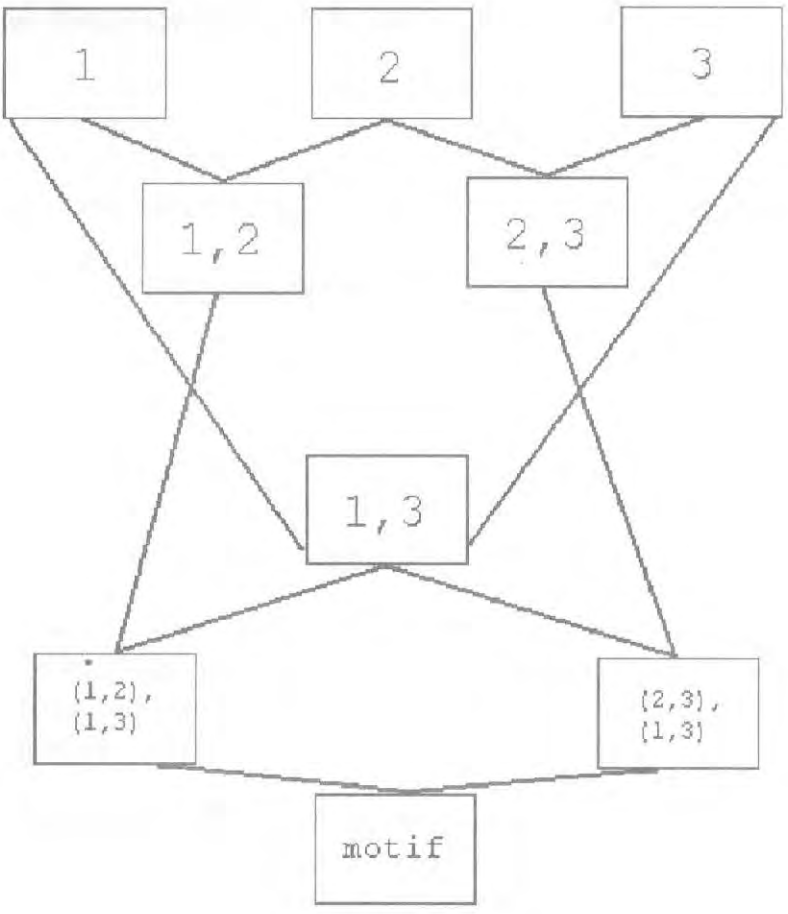


Figure 2: Crossovers used to create the motif

For example, I might start out with n_1, n_2, n_3, and n_4. Suppose n_1 is in the range [0..6], n_2 is in the range [0..2], n_3 is in the range [0..3], and n_4 is in the range [0..4]. My transform might look like

$$m_1 = 7n_1 + n_2$$
$$m_2 = 4n_3 + n_4$$

This would give the matrix $\begin{bmatrix} m_1 \\ m_2 \end{bmatrix}$. The output from the fractal might be $\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, which would be transformed into the new n_1, n_2, n_3, and n_4 as follows.

$$n_1 = f_1 \bmod 7$$
$$n_2 = (f_1 \bmod 7) \bmod 3$$
$$n_3 = f_2 \bmod 4$$
$$n_4 = (f_2 \bmod 4) \bmod 5$$

$n_1 = f_1 \bmod 7$ means that n_1 holds the remainder after f_1 is divided by 7.

The new note specified by the four resulting integers is then added on the end of the work, and the process continues. The lines are created one by one until a maximum size has been achieved for the piece. Thus, I have created an intermediate output file (see Appendix F).

All that is left is to translate the generated file into a format that can be reassembled by the Midi DisAssembler. The third program is the series, therefore, is `write.cc`. This program performs the opposite task as `read.cc`. Example output from this program is shown in Appendix G. Some adjustments by hand may be required before the file is loaded into the Midi DisAssembler. This is largely due to the fact that the Midi DisAssembler software attempts to adjust for tempo. Most MIDI software pagkages leave this detail to the individual user. Therefore, some corrections for meter must be added before the file will be accepted. While the code could be modified to print different meter and tempo markings for each trial, I found it easier to copy and paste the information from a similar file each time it was needed. After the text file is input into Midi DisAssembler, a music file with extension `.mid` is produced. In these cross-composed files lies the intrigue.

6 Analysis

6.1 Analysis of Purpose

In my experience, computer-generated music can be thought of as having three main purposes. First, it can be conducted as a thought experiment. Different formulas and techniques can be applied and manipulated, limited only by the person's imagination and ability. Second, the music can be generated to determine what can be learned by these methods. Such is the case of the AIDS RNA music. Of course, a third purpose is for entertainment. I believe that my work can be thought of in all these ways. The idea originated as a thought experiment: Could I compose a program to produce music through fractals and genetic theory? As my work continued, I discovered that it would be interesting to see what would be revealed by a study of such pieces. The most obvious initial discovery is whether or not fractals used in this manner have the potential to generate meaningful compositions. I think that they do. Also, there is a certain quality of entertainment and enjoyment about the endeavor.

6.2 Analysis of a Piece

The piece I chose to analyze was based on a Bach prelude, *It is Well with My Soul*, and *Hymn of Promise*. I chose these pieces for their unity of sound and message. These motifs were then presented as chromosomes to my MasterPiece program and expanded through a fractal formula. (see Appendix H)

I chose to analyze this piece by hand instead of converting it to a MIDI file. Therefore, I printed out the text and played the piece on a piano to listen for quality and patterns. The first half of the piece sounded somewhat melodious. I believe that this can be enhanced by MIDI software that will allow me to separate the music into instruments, thus putting the *melody* in one instrument and allowing other instruments to come in and out playing the supporting material. While this suggests more human involvement and less emphasis on chance, I believe that it does not defeat the purpose of my project. The musical patterns remain whether the music is performed on the computer in one instrument or separated and performed by human musicians.

During the first stage of my analysis, I played the sequence of notes in

order but limited them to a single octave. In other words, I did not take the octave number into consideration. I merely assumed the number to be constant. Once I became more familiar with the arrangement, I tried to move it into the correct octaves. I discovered that the piece actually sounded better when played in the respective octaves. I had supposed that a pattern of melody would be more apparent by limiting it to one octave. This was not the case.

In the second half of the composition, it seems that the music effectively makes a key change. The key change occurs in the section that is generated by the fractal pattern. One particularly interesting section contains eighteen F♯ notes in a row with varying octaves and durations. Perhaps the fractal pattern became a little too self-similar! Actually, I found this part intriguing. Quite a bit of emotion is conveyed by the F♯ notes in the variety of octaves. That section may also serve as a bridge between the melodious first half and the ending. Several times before the ending, smaller series of F♯ notes appear. The work ends on a single high A note, an appropriate ending, I feel. Interestingly, the only B that is included in the piece is a B♯, which translates to C on the keyboard.

I have successfully produced music through fractals and genetic theory. They are intriguing pieces due to their design. It is fair to admit that patterns can be found in almost anything, and, while I will argue that this is a coherent, legitimate way to compose music, ultimately, it is up to the individual listener to decide. However, is that not true with any piece of music?

7 Improvements

One aspect of my program dealing with the fractal generation pattern could be viewed as a limitation. The program plugs each note into the fractal in order, adding each new note onto the end. Instead, the notes could be thought of as a set of numbers from which the fractal could choose. This might lead to a better fractal representation, but it might also decrease the importance of the genetically created motif. Therefore, I chose not to implement this idea.

However, there is one improvement that would undoubtedly make the compositions more coherent and pleasing to the average listener. Instead of letting the program select any octave in which to place the note, the octave could be weighted. For example, the octave values commonly used in the input pieces could be given higher weights. The other values would be given lower weights. Therefore, it would be more likely that the notes would fall in a common range. This could also be accomplished by using modular arithmetic, as discussed earlier, to limit the octave range. However, the drawback here is that notes outside the range would be eliminated totally.

8 Conclusion

MasterPiece is one example of a program that will generate music. This is accomplished by using genetic theory and fractals. A motif is created by combining three input pieces using the genetic theory concepts of crossovers and mutations. This motif serves as both a musical and fractal motif. Then the fractal code is applied to the motif, producing an entire composition.

The example piece created in this manner contains coherent patterns and is a unique composition. It is a thought experiment that also demonstrates MasterPiece's potential. The piece can simply be enjoyed as entertainment, as well. In the end, it is not I who holds the final judgement on the validity and aesthic quality of my composition. Each person must decide for themselves whether to like or dislike my computer-generated music through fractals and genetic theory.

A Prelude2.txt

```
MThd | Format=1 | # of Tracks=2 | Division=1024

Track #0 ****************************************
    Time       Event
   5: 1:  1 |End of track|

Track #1 ****************************************
    Time       Event
   1: 1:  0 |On Note   | chan= 1   | pitch=C 3  | vol=64
        256 |On Note   | chan= 1   | pitch=E 3  | vol=64
        512 |On Note   | chan= 1   | pitch=G 3  | vol=64
        768 |Off Note  | chan= 1   | pitch=g 3  | vol=0
            |On Note   | chan= 1   | pitch=C 4  | vol=64
     2:  0 |Off Note  | chan= 1   | pitch=c 4  | vol=0
            |On Note   | chan= 1   | pitch=E 4  | vol=64
        256 |Off Note  | chan= 1   | pitch=e 4  | vol=0
            |On Note   | chan= 1   | pitch=G 3  | vol=64
        512 |Off Note  | chan= 1   | pitch=g 3  | vol=0
            |On Note   | chan= 1   | pitch=C 4  | vol=64
        768 |Off Note  | chan= 1   | pitch=c 4  | vol=0
            |On Note   | chan= 1   | pitch=E 4  | vol=64
        832 |Off Note  | chan= 1   | pitch=e 3  | vol=0
        981 |Off Note  | chan= 1   | pitch=c 3  | vol=0
     3:  0 |Off Note  | chan= 1   | pitch=e 4  | vol=0
         12 |On Note   | chan= 1   | pitch=C 3  | vol=64
        268 |On Note   | chan= 1   | pitch=E 3  | vol=64
        512 |On Note   | chan= 1   | pitch=G 3  | vol=64
        768 |Off Note  | chan= 1   | pitch=g 3  | vol=0
            |On Note   | chan= 1   | pitch=C 4  | vol=64
     4:  0 |Off Note  | chan= 1   | pitch=c 4  | vol=0
            |On Note   | chan= 1   | pitch=E 4  | vol=64
        256 |Off Note  | chan= 1   | pitch=e 4  | vol=0
            |On Note   | chan= 1   | pitch=G 3  | vol=64
        512 |Off Note  | chan= 1   | pitch=g 3  | vol=0
            |On Note   | chan= 1   | pitch=C 4  | vol=64
        768 |Off Note  | chan= 1   | pitch=c 4  | vol=0
```

```
          |On Note   | chan= 1   | pitch=E 4  | vol=64
      942 |Off Note  | chan= 1   | pitch=e 3  | vol=0
      994 |Off Note  | chan= 1   | pitch=c 3  | vol=0
2: 1:   0 |Off Note  | chan= 1   | pitch=e 4  | vol=0
       25 |On Note   | chan= 1   | pitch=C 3  | vol=64
      281 |On Note   | chan= 1   | pitch=D 3  | vol=64
      512 |On Note   | chan= 1   | pitch=A 3  | vol=64
      768 |Off Note  | chan= 1   | pitch=a 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
    2:  0 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      256 |Off Note  | chan= 1   | pitch=f 4  | vol=0
          |On Note   | chan= 1   | pitch=A 3  | vol=64
      512 |Off Note  | chan= 1   | pitch=a 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
      768 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      917 |Off Note  | chan= 1   | pitch=d 3  | vol=0
     1006 |Off Note  | chan= 1   | pitch=c 3  | vol=0
    3:  0 |Off Note  | chan= 1   | pitch=f 4  | vol=0
       38 |On Note   | chan= 1   | pitch=C 3  | vol=64
      294 |On Note   | chan= 1   | pitch=D 3  | vol=64
      512 |On Note   | chan= 1   | pitch=A 3  | vol=64
      768 |Off Note  | chan= 1   | pitch=a 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
    4:  0 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      256 |Off Note  | chan= 1   | pitch=f 4  | vol=0
          |On Note   | chan= 1   | pitch=A 3  | vol=64
      512 |Off Note  | chan= 1   | pitch=a 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
      768 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      827 |Off Note  | chan= 1   | pitch=d 3  | vol=0
     1019 |Off Note  | chan= 1   | pitch=c 3  | vol=0
3: 1:   0 |Off Note  | chan= 1   | pitch=f 4  | vol=0
       51 |On Note   | chan= 1   | pitch=B 2  | vol=64
      307 |On Note   | chan= 1   | pitch=D 3  | vol=64
```

```
      512 |On Note   | chan= 1   | pitch=G 3  | vol=64
      768 |Off Note  | chan= 1   | pitch=g 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
  2:    0 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      256 |Off Note  | chan= 1   | pitch=f 4  | vol=0
          |On Note   | chan= 1   | pitch=G 3  | vol=64
      512 |Off Note  | chan= 1   | pitch=g 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
      768 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      947 |Off Note  | chan= 1   | pitch=d 3  | vol=0
  3:    0 |Off Note  | chan= 1   | pitch=f 4  | vol=0
        8 |Off Note  | chan= 1   | pitch=b 2  | vol=0
       64 |On Note   | chan= 1   | pitch=B 2  | vol=64
      320 |On Note   | chan= 1   | pitch=D 3  | vol=64
      512 |On Note   | chan= 1   | pitch=G 3  | vol=64
      768 |Off Note  | chan= 1   | pitch=g 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
  4:    0 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      256 |Off Note  | chan= 1   | pitch=f 4  | vol=0
          |On Note   | chan= 1   | pitch=G 3  | vol=64
      512 |Off Note  | chan= 1   | pitch=g 3  | vol=0
          |On Note   | chan= 1   | pitch=D 4  | vol=64
      768 |Off Note  | chan= 1   | pitch=d 4  | vol=0
          |On Note   | chan= 1   | pitch=F 4  | vol=64
      904 |Off Note  | chan= 1   | pitch=d 3  | vol=0
4: 1:   0 |Off Note  | chan= 1   | pitch=f 4  | vol=0
          |Off Note  | chan= 1   | pitch=b 2  | vol=0
       76 |On Note   | chan= 1   | pitch=C 3  | vol=64
      332 |On Note   | chan= 1   | pitch=E 3  | vol=64
      512 |On Note   | chan= 1   | pitch=G 3  | vol=64
      768 |Off Note  | chan= 1   | pitch=g 3  | vol=0
          |On Note   | chan= 1   | pitch=C 4  | vol=64
  2:    0 |Off Note  | chan= 1   | pitch=c 4  | vol=0
          |On Note   | chan= 1   | pitch=E 4  | vol=64
      256 |Off Note  | chan= 1   | pitch=e 4  | vol=0
```

```
        |On Note    | chan= 1    | pitch=G 3   | vol=64
    512 |Off Note   | chan= 1    | pitch=g 3   | vol=0
        |On Note    | chan= 1    | pitch=C 4   | vol=64
    768 |Off Note   | chan= 1    | pitch=c 4   | vol=0
        |On Note    | chan= 1    | pitch=E 4   | vol=64
    968 |Off Note   | chan= 1    | pitch=e 3   | vol=0
3:    0 |Off Note   | chan= 1    | pitch=e 4   | vol=0
     34 |Off Note   | chan= 1    | pitch=c 3   | vol=0
     89 |On Note    | chan= 1    | pitch=C 3   | vol=64
    345 |On Note    | chan= 1    | pitch=E 3   | vol=64
    512 |On Note    | chan= 1    | pitch=G 3   | vol=64
    768 |Off Note   | chan= 1    | pitch=g 3   | vol=0
        |On Note    | chan= 1    | pitch=C 4   | vol=64
4:    0 |Off Note   | chan= 1    | pitch=c 4   | vol=0
        |On Note    | chan= 1    | pitch=E 4   | vol=64
    256 |Off Note   | chan= 1    | pitch=e 4   | vol=0
        |On Note    | chan= 1    | pitch=G 3   | vol=64
    512 |Off Note   | chan= 1    | pitch=g 3   | vol=0
        |On Note    | chan= 1    | pitch=C 4   | vol=64
    768 |Off Note   | chan= 1    | pitch=c 4   | vol=0
        |On Note    | chan= 1    | pitch=E 4   | vol=64
    947 |Off Note   | chan= 1    | pitch=e 3   | vol=0
5: 1:   0 |Off Note   | chan= 1    | pitch=e 4   | vol=0
        |Off Note   | chan= 1    | pitch=c 3   | vol=0
      1 |End of track|
```

B Read.cc

```
//Programmer:  Amanda Broyles
//read.cc

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void position(char pos[], FILE *f);
typedef struct notestruct noteInfo;
struct notestruct
  {
    char note[2];
    int  offset;
    int  octave;
   };


void main ()
{
  FILE *f, *fout;
  int i=0, j=0, k=0, n=0, maxnum=150;
  noteInfo array[maxnum];
  char buffer[80];
  char pos[4];
  int flag;

  f = fopen("it is well with my soul.txt", "r");
  fout = fopen("itiswell.txt", "w");
  for (i=0; i<maxnum; i++)    //initialize offset to zero
    array[i].offset = 0;

  for (i=0; i<8; i++)         //reads over first 8 lines
    fgets(buffer, 80, f);

  i = 0;                      //reset i
```

```
  for(j=0; !(feof(f)); j++) {    //reads from doc
    position(pos, f);            //and prints to out file
    if (pos[0] == 'n') {         //if on note, write to the array
      array[i].note[0] = pos[1];
      array[i].note[1] = pos[2];
      array[i].octave = (int)pos[3];
      i++;                       //increment array index number
    }

    if (pos[0] == 'f') {         //if off note
      pos[1] = pos[1] + 'A' -'a';     //capitalize note value
      flag =0;
      for (k=0; k<j && flag==0; k++) {   //and compare from start of array
        if (array[k].note[0] == pos[1] && array[k].offset == 0) {
          if (array[k].note[1] == pos[2]) {
            if (array[k].octave == (int)pos[3]) {
              array[k].offset = i - k;      //calculate correct offset
              flag = 1;                     //mark found
            }
          }
        }
      }
    }
  }

  for (n=0; n<i; n++) {
    fprintf (fout, "%c%c %d %c\n", array[n].note[0], array[n].note[1],
array[n].offset, array[n].octave);
  }

  fclose(f);
  fclose(fout);

printf ("done\n");

}

//-------------------------------------------------
```

```
void position(char pos[], FILE *f)
//returns the characters for note, octave, and on/off
{

  char buffer[80];

  fgets (buffer, 80, f);          //read a line

  pos[0] = buffer[14];            //get pertinent positions
  pos[1] = buffer[42];
  pos[2] = buffer[43];
  pos[3] = buffer[44];
  if (buffer[42] == 'v') {        //correct for tabs
    pos[0] = buffer[7];
    pos[1] = buffer[35];
    pos[2] = buffer[36];
    pos[3] = buffer[37];
  }
  if (buffer[42] == '=') {
    pos[0] = buffer[15];
    pos[1] = buffer[43];
    pos[2] = buffer[44];
    pos[3] = buffer[45];
  }
  if (buffer[42] == ' ') {
    pos[0] = buffer[8];
    pos[1] = buffer[36];
    pos[2] = buffer[37];
    pos[3] = buffer[38];
  }
}
```

C MasterPiece.cc

```
///Programmer:  Amanda Broyles

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct notestruct noteInfo;
struct notestruct
  {
    char note[2];
    int  offset;
    int  octave;
   };

void Mutate(noteInfo c1[], noteInfo c2[], noteInfo c3[], int n1, int n2,
            int n3);
int print(int n, int arr[]);
int Value(char note);
int Typed(char sharp);
char DeValue(int n);
char UnTyped(int n);
void Generate(int n1, int n2, int n3, int n4, int out[4]);


void main ()
{
  FILE *f1, *f2, *f3, *fout, *f;
  int i=0, j=0, n=0, k=0, maxnum=150;
  noteInfo array[maxnum];
  noteInfo chrome1[maxnum], chrome2[maxnum], chrome3[maxnum];
  int check[maxnum];                    //to check if position clear
  int out[4];
  int val, sharp;
  char valu, sh;
  int n1, n2, n3;
```

```
f1 = fopen ("itiswell.txt", "r");      //open files to read and write
f2 = fopen ("Prelude2out.txt", "r");
f3 = fopen("hymnofpromiseout.txt", "r");
fout = fopen("newout.txt", "w");


for (i=0; !(feof(f1)); i++) {      //scan files and assign chromosomes
  fscanf (f1, "%s%d%d", chrome1[i].note, &(chrome1[i].offset),
     &(chrome1[i].octave));
}
n1 = i;

for (i=0; !(feof(f2)); i++) {
  fscanf (f2, "%s%d%d", chrome2[i].note, &(chrome2[i].offset),
     &(chrome2[i].octave));
}
n2 = i;

for (i=0; !(feof(f3)); i++) {
  fscanf (f3, "%s%d%d", chrome3[i].note, &(chrome3[i].offset),
     &(chrome3[i].octave));
}
n3 = i;

Mutate(chrome1, chrome2, chrome3, n1, n2, n3);

f = fopen("out.txt", "r");

for (i=0; !(feof(f)); i++) {
    fscanf (f, "%s%d%d\n", array[i].note, &(array[i].offset),
       &(array[i].octave));
}

for (k=0; k<maxnum-i; k++) {
  val = Value(array[k].note[0]);
  sharp = Typed(array[k].note[1]);
```

```
    Generate(val, sharp, array[k].offset, array[k].octave, out);

    valu = DeValue(out[0]);
    sh = UnTyped(out[1]);
    array[i+k].note[0] = valu;
    array[i+k].note[1] = sh;
    array[i+k].offset = out[2];
    array[i+k].octave = out[3];
  }

  i = i + k;

  for (n=0; n<i; n++) {
      fprintf (fout, "%c%c %d %d\n", array[n].note[0], array[n].note[1],
          array[n].offset, array[n].octave);
  }


printf ("done\n");

}

// -------------------Mutate-----------------------------

void Mutate(noteInfo c1[], noteInfo c2[], noteInfo c3[], int n1,int n2,int n3)
{
  int maxnum=150;
  noteInfo temp1[maxnum], temp2[maxnum], temp3[maxnum];
  int i, j, k, n;
  int t1, t2, t3, t4, t5;
  FILE *f4;

  for(i=0; i<4; i++) {
    temp1[i].note = c1[i].note;
    temp1[i].offset = c1[i].offset;
    temp1[i].octave = c1[i].octave;
  }
  for (j=i-1; j<2*4; j++) {
```

```
      temp1[j].note = c2[j].note;
      temp1[j].offset = c2[j].offset;
      temp1[j].octave = c2[j].octave;
   }
   for (k=j-1; k<3*4; k++) {
      temp1[k].note = c1[k].note;
      temp1[k].offset = c1[k].offset;
      temp1[k].octave = c1[k].octave;
   }
   for (i=k-1; i<n2; i++) {
      temp1[i].note = c2[i].note;
      temp1[i].offset = c2[i].offset;
      temp1[i].octave = c2[i].octave;
   }
   t1 = i;
//-----------------------------------------

   for(i=0; i<6; i++) {
     temp2[i].note = c2[i].note;
     temp2[i].offset = c2[i].offset;
     temp2[i].octave = c2[i].octave;
   }
   for (j=i-1; j<2*6; j++) {
     temp2[j].note = c3[j].note;
     temp2[j].offset = c3[j].offset;
     temp2[j].octave = c3[j].octave;
   }
   for (k=j-1; k<3*6; k++) {
     temp2[k].note = c2[k].note;
     temp2[k].offset = c2[k].offset;
     temp2[k].octave = c2[k].octave;
   }
   for (i=k-1; i<n3; i++) {
     temp2[i].note = c3[i].note;
     temp2[i].offset = c3[i].offset;
     temp2[i].octave = c3[i].octave;
   }
   t2 = i;
```

```
//-----------------------------------------

  for(i=0; i<5; i++) {
    temp3[i].note = c1[i].note;
    temp3[i].offset = c1[i].offset;
    temp3[i].octave = c1[i].octave;
  }
  for (j=i-1; j<2*5; j++) {
    temp3[j].note = c3[j].note;
    temp3[j].offset = c3[j].offset;
    temp3[j].octave = c3[j].octave;
  }
  for (k=j-1; k<3*5; k++) {
    temp3[k].note = c1[k].note;
    temp3[k].offset = c1[k].offset;
    temp3[k].octave = c1[k].octave;
  }
  for (i=k-1; i<n2; i++) {
    temp3[i].note = c3[i].note;
    temp3[i].offset = c3[i].offset;
    temp3[i].octave = c3[i].octave;
  }
  t3 = i;
//-----------------------------------------

  for (i=12-1; i<2*12; i++) {
    temp1[i].note = temp3[i].note;
    temp1[i].offset = temp3[i].offset;
    temp1[i].octave = temp3[i].octave;
  }

//-----------------------------------------

  for (i=12-1; i<2*12; i++) {
    temp2[i].note = temp3[i].note;
    temp2[i].offset = temp3[i].offset;
    temp2[i].octave = temp3[i].octave;
  }
```

```
//-----------------------------------------

  for (i=0; i<25; i++) {
    temp3[i].note = temp1[i].note;
    temp3[i].offset = temp1[i].offset;
    temp3[i].octave = temp1[i].octave;
  }

  for (i=25-1; i<t2; i++) {
    temp3[i].note = temp2[i].note;
    temp3[i].offset = temp2[i].offset;
    temp3[i].octave = temp2[i].octave;
  }

  f4= fopen("out.txt", "w");

  for (n=0; n<t2-1; n++) {
      fprintf (f4, "%c%c %d %d\n", temp3[n].note[0], temp3[n].note[1],
          temp3[n].offset, temp3[n].octave);
  }

  fclose(f4);

}

// -------------------Print-----------------------------

int print(int n, int arr[])
{
  if (arr[n] == 0)
    return 1;
  else
    return 0;
}

// -------------------Value-----------------------------
```

```
int Value(char note)
{
  if (note == 'A')
    return 0;
  if (note == 'B')
    return 1;
  if (note == 'C')
    return 2;
  if (note == 'D')
    return 3;
  if (note == 'E')
    return 4;
  if (note == 'F')
    return 5;
  if (note == 'G')
    return 6;
}

// --------------------DeValue-----------------------------

char DeValue(int n)
{
  if (n == 0)
    return 'A';
  if (n == 1)
    return 'B';
  if (n == 2)
    return 'C';
  if (n == 3)
    return 'D';
  if (n == 4)
    return 'E';
  if (n == 5)
    return 'F';
  if (n == 6)
    return 'G';
}
```

```
// -------------------Typed------------------------------

int Typed(char sharp)
{
  if (sharp == ' ')
    return 0;
  if (sharp == '#')
    return 1;
}

// -------------------UnTyped----------------------------

char UnTyped(int n)
{
  if (n == 0)
    return ' ';
  if (n == 1)
    return '#';
}

// -------------------Generate---------------------------

void Generate(int n1, int n2, int n3, int n4, int out[4])
{

  int f[2];
  int x[2];                    //input matrix x

  x[0] = 7*n1 + n2;            //linear transformations
  x[1] = 10*n3 + n4;
  f[0] = x[0] * 1 + 2 * x[1];  //fractal formulas
  f[1] = x[0] * -1 + x[1] * 1;

  out[0] = f[0] % 7;      //new note
  out[1] = (f[0] % 7) % 2;      //new type
  out[2] = (f[1] + 30) % 10;     //new offset
  out[3] = ((f[1] + 30) % 10) % 5;      //new octave
}
```

D Write.cc

```
//Programmer:  Amanda Broyles
//write.cc

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

void position(char pos[], FILE *f);
typedef struct notestruct noteInfo;
struct notestruct
  {
    char note[2];
    int  offset;
    int  octave;
   };

          .
void main ()
{
  FILE *f, *fout;
  int i=0, j=0, k=0, n=0, m=0, maxnum=150;
  noteInfo array[maxnum], out[maxnum];
  char c;
  int check[maxnum];                    //to check if position clear
  int  flag1=0, flag2=0;
  char buffer[80];

  f = fopen("tempout.txt", "r");
  fout = fopen("tempoutmid.txt", "w");

//write first 8 lines
//read from file

  for (i=0; !(feof(f)); i++) {
//    fscanf(f, "%s %d %d", array[i].note, array[i].offset, array[i].octave);
```

```
    fgets(buffer, 80, f);
    array[i].note[0] = buffer[0];
    array[i].note[1] = buffer[1];
    array[i].offset = (int)buffer[3] - 48;
    array[i].octave = (int)buffer[5] - 48;
  }

  for (j=0; j<maxnum; j++) {
    check[j] = 0;                    //initialize check
  }

  for (j=0; j<i; j++) {              //create output array
    flag1 = 0;
    k = j;
    while (flag1 == 0) {             //while not found and position clear
      if (check[k] == 0) {
        out[k].note[0] = array[j].note[0];      //assign on note values into
array
        out[k].note[1] = array[j].note[1];
        out[k].octave = array[j].octave;
        out[k].offset = 0;
        check[k] = 1;                //mark as used
        flag1 = 1;                   //set exit condition
      }
      else
        k++;                         //otherwise, increment array index
    }
    flag2 = 0;
    n = array[j].offset;             //assign off note values into array
    while (flag2 == 0) {
      if (check[k + n] == 0) {
        out[k+n].note[0] = tolower(array[j].note[0]);
        out[k+n].note[1] = array[j].note[1];
        out[k+n].octave = array[j].octave;
        out[k+n].offset = 0;
        check[k+n] = 1;              //mark as used
        flag2 = 1;                   //set exit condition
      }
```

```
      else
        n++;                          //otherwise, increment array index
    }
//printf ("%c %c  ", out[k].note[0], out[k+n].note[0]);
  }

  m = 4;
  n = 0;

  for (k=0; k<(2*i); k++) {
    if (m == 4) {
      fprintf(fout, "   %d: ", n%4+1);
      m = 0;
      n++;
    }
    if (isupper(out[k].note[0])) {
      fprintf(fout, "\t%cOn Note\t", toascii(124));
      m++;
    }
    else
      fprintf(fout, "\t%cOff Note\t", toascii(124));
    fprintf(fout, "%c chan= 1\t", toascii(124));
    fprintf(fout, "%c pitch=%c", toascii(124), out[k].note[0]);
    fprintf(fout, "%c%d\t", out[k].note[1], out[k].octave);
    fprintf(fout, "%c vol=64\n", toascii(124));
  }

}
```

E Temp.txt

```
A  3 4
B  2 3
C# 5 1
A  4 4
D  5 4
```

F Tempout.txt

```
A  3 4
B  2 3
C# 5 1
A  4 4
D  5 4
F# 4 4
E  6 1
F# 6 1
E  4 4
D# 3 3
F# 8 3
D# 3 3
E  5 0
E  6 1
E  1 1
G  7 2
E  1 1
C  2 2
D# 3 3
B# 3 3
E  0 0
B# 3 3
C  8 3
E  1 1
E  5 0
```

G Tempoutmid.txt

```
1: |On Note  | chan= 1 | pitch=A 4 | vol=64
   |On Note  | chan= 1 | pitch=B 3 | vol=64
   |On Note  | chan= 1 | pitch=C#1 | vol=64
   |Off Note | chan= 1 | pitch=a 4 | vol=64
   |Off Note | chan= 1 | pitch=b 3 | vol=64
   |On Note  | chan= 1 | pitch=A 4 | vol=64
2: |On Note  | chan= 1 | pitch=D 4 | vol=64
   |Off Note | chan= 1 | pitch=c#1 | vol=64
   |On Note  | chan= 1 | pitch=F#4 | vol=64
   |Off Note | chan= 1 | pitch=a 4 | vol=64
   |On Note  | chan= 1 | pitch=E 1 | vol=64
   |Off Note | chan= 1 | pitch=d 4 | vol=64
   |Off Note | chan= 1 | pitch=f#4 | vol=64
   |On Note  | chan= 1 | pitch=F#1 | vol=64
3: |On Note  | chan= 1 | pitch=E 4 | vol=64
   |On Note  | chan= 1 | pitch=D#3 | vol=64
   |Off Note | chan= 1 | pitch=e 1 | vol=64
   |On Note  | chan= 1 | pitch=F#3 | vol=64
   |Off Note | chan= 1 | pitch=e 4 | vol=64
   |Off Note | chan= 1 | pitch=f#1 | vol=64
   |Off Note | chan= 1 | pitch=d#3 | vol=64
   |On Note  | chan= 1 | pitch=D#3 | vol=64
4: |On Note  | chan= 1 | pitch=E 0 | vol=64
   |On Note  | chan= 1 | pitch=E 1 | vol=64
   |Off Note | chan= 1 | pitch=d#3 | vol=64
   |Off Note | chan= 1 | pitch=f#3 | vol=64
   |On Note  | chan= 1 | pitch=E 1 | vol=64
   |Off Note | chan= 1 | pitch=e 0 | vol=64
   |Off Note | chan= 1 | pitch=e 1 | vol=64
   |Off Note | chan= 1 | pitch=e 1 | vol=64
   |On Note  | chan= 1 | pitch=G 2 | vol=64
1: |On Note  | chan= 1 | pitch=E 1 | vol=64
   |Off Note | chan= 1 | pitch=e 1 | vol=64
   |On Note  | chan= 1 | pitch=C 2 | vol=64
   |On Note  | chan= 1 | pitch=D#3 | vol=64
   |Off Note | chan= 1 | pitch=c 2 | vol=64
```

```
   |On Note  | chan= 1 | pitch=B#3 | vol=64
2: |Off Note | chan= 1 | pitch=g 2 | vol=64
   |Off Note | chan= 1 | pitch=d#3 | vol=64
   |Off Note | chan= 1 | pitch=b#3 | vol=64
   |On Note  | chan= 1 | pitch=E 0 | vol=64
   |Off Note | chan= 1 | pitch=e 0 | vol=64
   |On Note  | chan= 1 | pitch=B#3 | vol=64
   |On Note  | chan= 1 | pitch=C 3 | vol=64
   |On Note  | chan= 1 | pitch=E 1 | vol=64
3: |Off Note | chan= 1 | pitch=b#3 | vol=64
   |Off Note | chan= 1 | pitch=e 1 | vol=64
   |On Note  | chan= 1 | pitch=E 0 | vol=64
   |On Note  | chan= 1 | pitch=E 0 | vol=64
   |Off Note | chan= 1 | pitch=  0 | vol=64
   |Off Note | chan= 1 | pitch=  0 | vol=64
   |Off Note | chan= 1 | pitch=c 3 | vol=64
```

H Untitled1

```
A  1 3
A  1 3
A  7 3
C  1 4
E  1 4
G  1 3
C  1 4
A  3 2
C# 2 3
A  1 3
A  1 3
A  7 3
A  2 2
C# 1 3
G  1 3
C  1 3
C  1 3
E  1 3
F  1 3
G  1 3
F  1 3
E  1 3
C  1 3
C  1 3
C  1 3
F  1 3
C  1 3
F  1 3
G  1 3
A  1 3
F  1 3
F  1 3
A# 1 3
C  1 4
D  1 4
A# 1 3
```

```
C  1 4
F  1 3
F  1 3
F  1 3
A# 1 3
A# 1 3
A  1 3
G  1 3
F  1 3
F  1 3
E  1 3
D  1 3
F  1 3
A# 1 3
D  1 4
C  1 4
F  1 3
G  1 3
A  1 3
A# 1 3
A  1 3
G  1 3
G  1 3
F  1 3
C  1 3
C  1 3
A  1 2
G  1 3
C  1 3
A  1 2
C  1 3
C  1 3
A# 1 2
A  1 3
D  1 3
A# 1 2
C  1 3
F# 3 3
```

```
F# 3 3
G  3 3
A  0 0
A  6 1
F# 1 1
A  0 0
B# 2 2
F# 8 3
F# 3 3
F# 3 3
G  3 3
C  2 2
G  8 3
F# 1 1
F# 9 4
F# 9 4
F# 5 0
F# 8 3
F# 1 1
F# 8 3
F# 5 0
F# 9 4
F# 9 4
F# 9 4
F# 8 3
F# 9 4
F# 8 3
F# 1 1
F# 3 3
F# 8 3
F# 8 3
G  2 2
A  0 0
A  3 3
G  2 2
A  0 0
F# 8 3
F# 8 3
```

```
F# 8 3
G  2 2
G  2 2
F# 3 3
F# 1 1
F# 8 3
F# 8 3
F# 5 0
F# 2 2
F# 8 3
G  2 2
A  3 3
A  0 0
F# 8 3
F# 1 1
F# 3 3
G  2 2
F# 3 3
F# 1 1
F# 1 1
F# 8 3
F# 9 4
F# 9 4
D# 2 2
F# 1 1
F# 9 4
D# 2 2
F# 9 4
F# 9 4
E  1 1
F# 3 3
F# 2 2
E  1 1
F# 9 4
E  7 2
E  7 2
D# 1 1
A  0 0
```

References

[1] [http://www.borg.com/~jglatt/progs/software.htm].

[2] Thomas Benjamin, Michael Horvit, and Robert Nelson. *Techniques and Materials of Tonal Music with an Introduction to Twentieth-century Techniques.* Wadsworth Publishing Company, Belmont, CA, fourth edition, 1992.

[3] Amanda Broyles. Implementation of a genetic algorithm: Gaasp. Ouachita Baptist University, Department of Math and Computer Science, July 1998.

[4] Dennis Courtney. Math and real life: a brief introduction to fractional dimensions. [http://www.imho.com/grae/chaos/fraction.html].

[5] Richard Darst, Judith Palagallo, and Thomas Price. Fractal tilings in the plane. *Mathematics Magazine*, 71(1):12–23, February 1998.

[6] Bruce L. Jacob. Composing with genetic algorithms. [http://www.ee.umd.edu/~blj/algorithmic_composition/icmc.95], September 1995. International Computer Music Conference.

[7] Al Kelley and Ira Pohl. *A Book on C.* Addison-Wesley, fourth edition, 1998.

[8] Hans Lauwerier. *Fractals.* Princeton University Press, Princeton, NJ, 1991.

[9] Steven J. Leon. *Linear Algebra with Applications.* Prentice Hall, Upper Saddle River, NJ, fifth edition, 1998.

[10] Peter Stone. posted email from psto@xs4all.nl.

[11] Chong (John) Yu. Computer generated music composition. Department of electrical engineering and computer science, Massachusetts Institute of Technology, Cambridge, MA, 1996.