New Methods in Algorithmic Music

By: Christian N. Cortner
Mentor: Dr. James S. Walker

Sponsored By: Office of Research and Sponsored Programs, UW-Eau Claire

Introduction

Music is inherently mathematical. The intervals between notes in a scale, a chord, the timing between notes in a rhythm, repetitions and variations of musical themes and the organization of phrases in a piece can all be described mathematically. It is this underlying structure that catches our minds and effects our emotions. There were several questions that I wanted to explore: Is there anything inherently musical in a number itself? Does a musical number only sound pleasing if it produces music that adheres to our preconceived notions? How can a musical structure be extracted from a number?

Generating Music Algorithmically

The immediately obvious method for extracting musical structure from a number is to have each digit represent a specific note in an octave. The number one, for instance represents the tonic, three a third, five a fifth etc. There are however several problems with this approach. First, The music generated would be limited to ten notes starting with the tonic and reaching up to the third note in the second octave. This limitation in range would produce rather boring music. A desirable algorithm must allow movement over an arbitrary range of octaves. Second, numbers with seemingly random sequences of digits, such as pi, would produce music that is also very random. Real melodies climb and descend their scales to produce movement that this method could not duplicate without very specific sequences of numbers. The algorithm must therefore be able to create melodic movement, even with numbers such as pi. Finally, this method does not address the musical requirement of rhythm. A more elaborate way to interpret the sequence is necessary to produce meaningful melodies.

The algorithm described here uses as its basis the fact that melodies are constructed from groups of notes ascending and descending in a scale. Instead of digits representing specific notes in an scale, this method uses them to represent intervals from the previous note in the melody in a direction determined by the current mode of the algorithm. After each note the current mode has a chance, determined by the current digit, of reversing direction. The immediately obvious problem with this is that the melody might quickly climb or descend into inaudible or unpleasant frequencies. To work around this, this algorithm introduces floor and ceiling octaves. If a generated note would exceed either of these, the algorithm forces the mode to flip in the opposite direction. Note, however, that the interval specified by the digit and therefore hopefully the musical "feel" of the entire number, is preserved.

To generate rhythm, it seems natural to use the digits as multiples of a base note length such a sixteenth or an eighth note. For instance, if the base duration is a sixteenth note, the number four represents a half note. The downside of this is that using the same number for both the melody and rhythm is putting a lot of demand on a single sequence. Certain melodic intervals always correspond to specific durations; a number that is a fifth from the previous note always has a length of five base units. This may sound interesting but seemed a limitation. To work around this, the algorithm described here allows separate numbers (or a single number) to be used for the melody and rhythm.

Research Objective

A new method in algorithmic music was investigated. A method for automatically generating melodies was developed. It uses a combination of random selection based on mathematical number sequences. Initial work was done on creating software that implements this automatic method.

The Algorithm (As implemented in software)

1. Determine base parameters for the melody to be generated
   - **a)** **Bo** = Base octave
   - **b)** **Co** = Ceiling octave
   - **c)** **Fo** = floor octave
   - **d)** **St** = Scale type (Major or minor)
   - **e)** **K** = Key to be used (Any note in the chromatic scale)
   - **f)** **Bl** = Base note length (Entered as divisions of a quarter note)
   - **g)** **Ms** = Melody seed (Number or mathematical expression). Converted to a set consisting of the digits.
   - **h)** **Rs** = Rhythm seed (Number or mathematical expression, or use **Ms** for both rhythm and melody). Converted to a set consisting of the digits.
2. If **|Rs| < |Ms|,** resize the set **Rs** to equal **|Ms|.** Set all extra digits to **1**. We are not concerned if **|Rs|>|Ms|**, because we will only ever retrieve from **Rs**, **|Ms|** digits.
3. Generate and store all notes from the bottom of **Fo** to the top of **Co** in the desired key using the correct scale type, major or minor. These numbers are stored numerically as semitones from the lower pitch limit of C-1 having a value of 0 in an ordered set **Bs**, big scale.
4. Choose an initial mode of Up or Down randomly.
5. Create the first note at the frequency of the tonic in **Bo**.
6. Generate |Ms| more notes using the following rules.
   - a) Extract the first/next digit **d** from **Ms.**
   - b) If **d** is 1-8, extract from **Bs** the note at an interval of **d** from the previous note in the direction of the current mode.
   - c) If **d** is a 0 or 9, generate a rest note.
   - d) If the generated note exceeds the limits of **Co** or **Fo**, switch the current mode and extract from **Bs**, the corresponding note in the opposite direction.
   - e) If the **Bo = Co** = **Fo**, there is chance that the generated note will exceed **Co** in the positive direction and **Fo** in the negative. If this is the case, generate a new note by counting notes in the current direction until the upper or lower limit of the range is reached and then wrapping around from the bottom of **Bo** to the top of **Co**, or vice versa, until **d** is reached. Extract the note corresponding to this position from **Bs**.
   - f) Store the new note in set **N**.
   - g) Determine if the mode will change as follows (this produces a higher probability of maintaining the current mode, resulting in more stable melodies):
     - i. Generate a random number **r** from 0 to 9
     - ii. Subtract **d** from **9**. If **(9 – d)** is greater than **d**, then set our chance **c** to **(9 - d)**, otherwise set **c** to **d**.
     - iii. If **r>=c**, randomly choose a new mode of up or down, otherwise maintain the current mode for the next note.
   - h) If **d** is not the last digit in **Ms**, return to step a).
7. Generate the rhythm for the **|Ms|+1** notes as follows.
   - a) Extract the first/next digit **d** from **Rs.**
   - b) Set the length of corresponding note **n** in **N**, to **d**.
   - c) If **d** is not the last digit in **Rs**, return to step a).

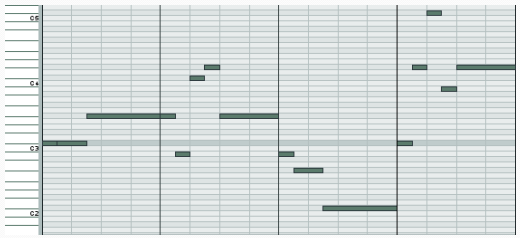
The Software

The software developed was written in Java and uses an open source version of Java Expression Parser by Singular Systems to evaluate mathematical expressions. The program, called *musicgen.exe* has a simple command line interface and when run, produces a MIDI file that can be loaded into a piece of software called a sequencer in order to play back the melody. When the program is run, all of the base options specified in the algorithm can be supplied or omitted (except for the seeds), in which case default values are used. The following shows usage information generated by the program and two sample runs.

```
C:\>musicgen.exe -h
 -c N   : Ceiling octave. Default is 4
 -f N   : Floor octave. Default is 4
 -k VAL : Choose key of song. Ex. A#. Default is C
 -o N   : Octave. Default is 4
 -r N   : resolution, ticks per quarter note. Default is 4
 -s VAL : Scale type, major or minor. Default is major

C:\>musicgen.exe -c 4 -f 2 -o 3 -k C# -r 8 -s minor
Enter a number or expression to use a melody basis : 3.141592653589793
Enter a number or expression to use a rhythm basis (press enter to use melody se
ed): 1251111412511114
Enter a name for the resulting midi file: test.mid
Done! The midi file has been written to: test.mid


C:\>musicgen.exe -k G -s major
Enter a number or expression to use a melody basis : e^2
Enter a number or expression to use a rhythm basis (press enter to use melody se
ed): sqrt(pi)
Enter a name for the resulting midi file: c:\test2.mid
Done! The midi file has been written to: c:\test2.mid
```



Example: Melody produced by running "musicgen.exe -c 4 -f 2 -o 3 -k C# -r 8 -s minor ". The output is displayed in Reaper, a free audio/midi sequencer produced by Cockos Incorporated.

Results

The quality of the melodies produced by the algorithm vary greatly with the input. Because it only produces notes in a given scale, even random seeds can result in melodies that are pleasing. It is interesting to note that the melodies generated always seem to have a feel unique to the seed numbers. For instance, a piece generated with pi, sounds similar to other melodies generated with pi, even though no two have exactly the same notes. However, it is unfortunate to note, that pi doesn't sound very good! With numbers such as pi and e that have a very random sequence of digits, the algorithm produces better results if shorter sub-sequences are used as a seed. Otherwise the melody produced lacks a general theme that a human produced one would have.

The rhythm portion of this system is definitely the weakest point. Unless subsequent groups of digits in the rhythm seed consistently add to multiples of the same number, corresponding to time signatures that we are used to hearing, such as 3/4 or 4/4, the algorithm produces non-coherent rhythms with constantly changing, or non-existent perhaps, time signatures.

Overall I am pleased with the outcome of this project. The program does produce musical results. While I didn't find any long naturally occurring number sequences that are truly musical, if they exist, then I feel this algorithm stands a good chance of producing interesting melodies from them. In addition to this, the software can also work well as an inspirational tool by producing melodies that a human might not think of. Several times during my work on this, I found myself building songs around the results.

Acknowlegements:
Intramural funding source: UW-Eau Claire Center of Excellence for Faculty and Undergraduate Student Research Collaboration
Contact Information:
Chris Cortner, cortnecn@uwec.edu, UW-Eau Claire