Music Generation in ArtToMusic

Rafael De Smet

March 27, 2017

Contents

1 Introduction

In this paper I will discuss several methods to generate music with software. Analogous to the paper in graphical analysis, I will discuss methods I use in the ArtToMusic project and methods I did not use and compare them to each other. Since music is different for every person, my reasoning can be subjective at times and you may prefer other methods.

2 Music generation - a discussion

2.1 Tempo and harmony

When we listen to music, unconsciously we focus on a couple of things to process what we are hearing.

2.1.1 Tempo

One of these things is the tempo. A very simple way to talk about music and to distinguish between different pieces of music is to determine whether it's a fast or a slow piece. This discussion can be subjective as different people have different notions of speed. Luckily we can objectively determine the tempo of a piece, using the BPM (beats per minute). This tells us how many quarter notes are played in one minute, or how fast/slow the quarter notes are played after each other.

In music theory there are a couple of terms to indicate certain intervals of speed, based on the BPM. For example, the term *larghissimo* means that the piece of music has a BPM in the range]0, 20], which is very slow. *Moderato* means moderate, a piece that has a moderate speed usually has a BPM in the range [108, 120]. Most of the popular songs you hear on the radio are *moderato*.

2.1.2 Melody

A second crucial part of music is the melody. All melodies of songs are based on the rules of harmony. We can formulate a couple of questions that will help us understand the purpose of harmony, e.g. which notes sound good when played together, and which don't? Which notes make up a chord?

The notion of chords is very important. A chord is a group of notes played at the same time. There are major chords, which sound happy, and there are minor chords which sound sad or melancholic. These two types of chords are the most commonly used. There are many other types of chords such as the diminished, the sus4, add9 etc. But for this project I will stick to the two most important types and one special chord, the diminished.

2.2 Chords and progressions

A crucial element of music is the use of chord progressions. The most common example of this is a twelve bar blues. This type of music is characterised by a strict pattern. Every song that uses a twelve bar blues schema follows the same chord progression and thus sounds very similar and recognizable, see figure 1.

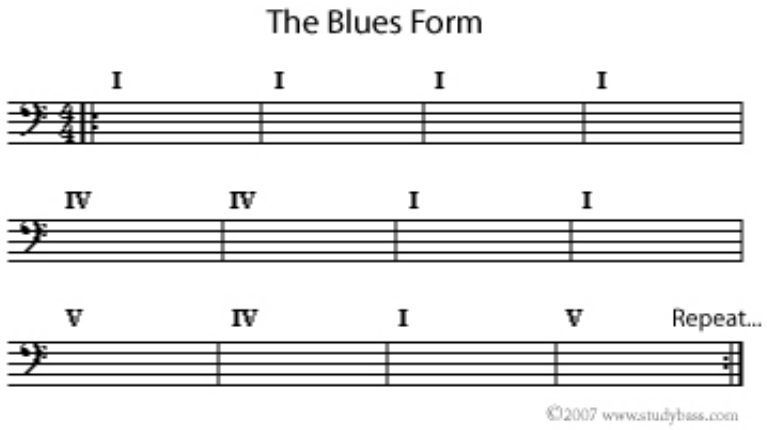

Figure 1: Twelve bar blues

The roman numbers denote which chord is played, relatively to the key of the song. For example if the piece is in the key of C major, the I denotes the first chord of the key, the C chord. The IV is the fourth chord of the key, the F major. The V is the fifth of the key, the G major.

There are many other kinds of progressions, e.g. I-II-V-I, or any other combination. Most popular songs use multiple chord progressions, one for the verse, one for the chorus and so on. It's the concept of combining different chords in one progression that allows us to create new original pieces of music.

2.3 Comparison of music generation technologies

In this section I will be comparing multiple music generation technologies. I will discuss libraries as well as online applications that have the same goal, creating enjoyable music based on software.

2.3.1 Beads

The Beads library, developed by Ollie Bown with the support of the university of Melbourne, is a very handy tool to generate music. Beads uses the concept of unit generators (UGs) as the core of the software. As described by Evan Merz, the author of *Sonifying Processing: The Beads Tutorial* [2], a unit generator is a "building block of the audio signal". One unit generator takes care of one function in the generation of music.

A simple example of a unit generator is a guitar player's distortion pedal. The clean signal of his guitar enters the unit generator and a distorted version of the signal is produced. Beads works as a series of unit generators (or guitar pedals). You can plug one UG into another to create a chain of effects, sounds etc...

The Beads library has many of these UGs, such as an envelope filter, a gain, a waveplayer, etc. The AudioContext is the main UG where every other UG plugs into. Without this there is no music. It doesn't create any sound by itself, it just acts as an interface to the computer's hardware. The audio comes from other UGs.

Beads has two main ways of creating audio.

Samples You can load multiple samples for later use. Once you have found the file and loaded it via the SamplePlayer, you can use it as any other UG, connect it to the Gain UG to determine the volume or apply multiple effects on it, such as pass it through an EnvelopeFilter, etc.

Waves Another technique is to use audio waves that are generated in real time. The most simple form of audio is a sine wave that produces one tone. Beads has implemented five different kind of waves, each called a Buffer. Each Buffer stores an array of floats in the range [-1,1] that represent the

function it is based on. In the following list you can see the different buffers and their functions.

- SINE: based on the sine function $y(t) = Asin(\omega t + \phi)$
- SQUARE: based on the square function $y(t) = sgn(sin(t))$
- TRIANGLE: based on the triangle function, with range from -1 to 1 and period 2a, $y(t) = \frac{2}{a}(t - a\lfloor\frac{t}{a} + \frac{1}{2}\rfloor)(-1)^{\lfloor\frac{t}{a}+\frac{1}{2}\rfloor}$
- SAW: based on the saw function $y(t) = t - \lfloor t \rfloor$
- NOISE: based on a random variable to determine each value in the buffer

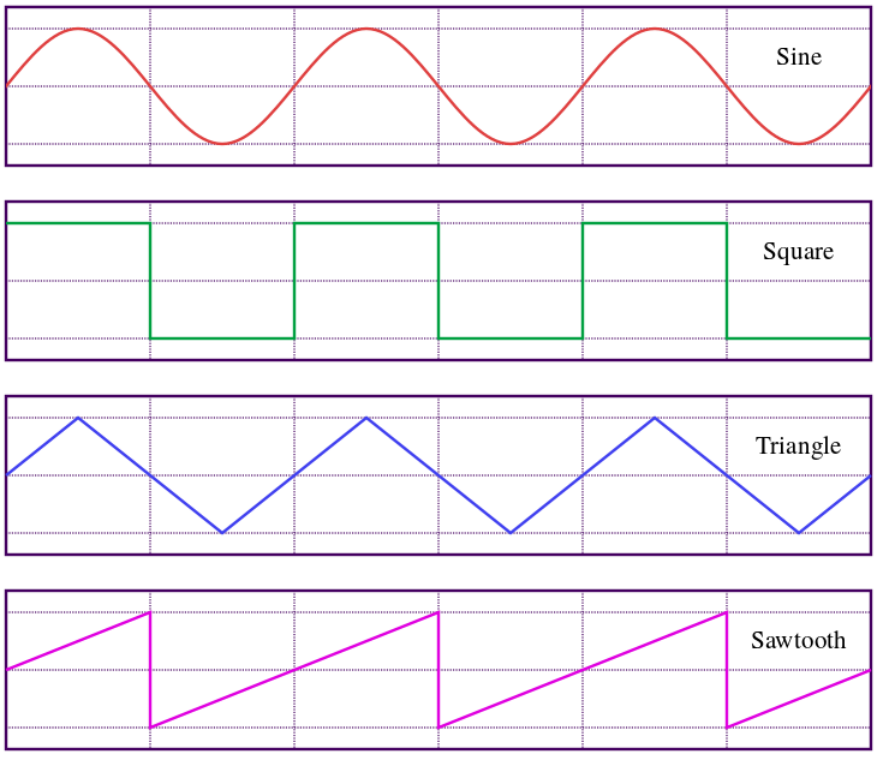

Figure 2: Buffer functions

Every Buffer has other (mathematical) characteristics and thus sounds differently. A sine wave produces a gentle tone, while a saw wave is less enjoyable to listen to. With the Online Tone Generator `http://onlinetonegenerator.com/`, we can hear the difference between the various functions.

Using the Beads library we can create our own music simply by passing the right input parameters to the different UG's to get the desired music.

2.3.2 Wolfram Tones

Desgined by the mathematician Stephen Wolfram, WolframTones is an online application to generate music. The user can choose from multiple styles of music and the application generates a new piece of music. The way this works is based on a discovery of Stephen Wolfram himself. In the early 1980s Wolfram was researching "one-dimensional cellular automata". He discovered that a set of very simple rules can create a very complex situation.

His experiments start with a row of cells, each cell black or white. The set of rules, determined before the execution, decide which colour every cell on the next line will get and so on. Figure 3 shows a possible set of rules. The top row in each box gives one of the possible combinations of colors for a cell and its immediate neighbours. The bottom row then specifies what color the center cell should be on the next step in each of these cases [1]. The rules of figure 2 can be described as follows. A particular cell will be black if either of its neighbours was black on the step before. A cell will be white if both its neighbours were white on the step before.

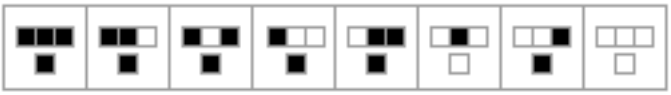

Figure 3: Rules for automata

This gives the result in figure 4. In this case the result is very well structured and organized. Wolfram discovered that there are 256 of these simple sets of rules, based on eight individual rules. Not every one of these 256 sets gives nicely structured results.

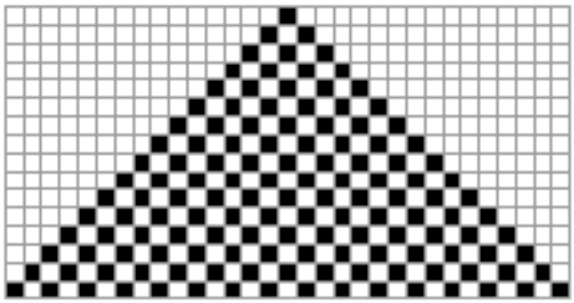

Figure 4: Result automata

Music Wolfram used his automata to create music. Let's say we used one of the 256 sets of rules to create a result pattern. We can take a partition through this pattern of 15 cells wide. When we flip this partition on its side we can treat it as a musical score. Figure 5 shows the partition from a resulting pattern and figure 6 shows it as a musical score.

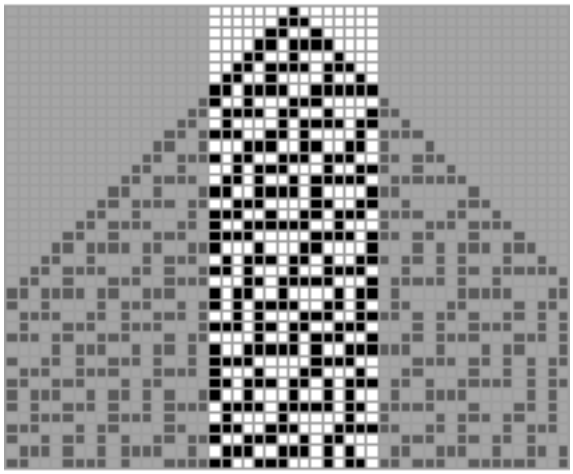

Figure 5: Partition through pattern

Now we can assume that time runs across the page from left to right and every black cell represents a note played. Wolfram has developed his own programming language [4] to process these cellular automaton patterns. The music generation process is black-box.

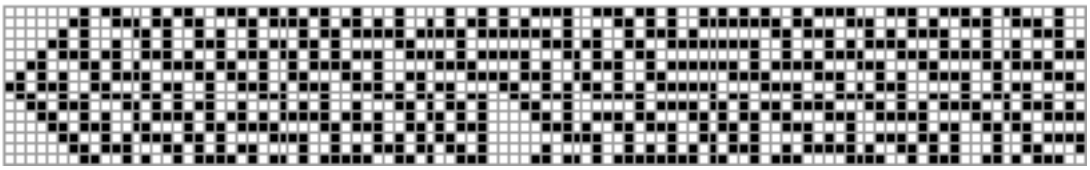

Figure 6: Musical score

The number of rules isn't limited by eight individual rules. We could also determine the color of the next cell by looking at his five upper row neighbours, instead of three. This gives a much bigger number of results (around 4 billion) and much more interesting results to use in the music creation.

2.3.3 Genetic algorithms

Another interesting case to focus on is the use of genetic algorithms to produce music, as explained in the paper by Dragan Matić [2].

A genetic algorithm (GA) is based on the idea of natural selection. The execution of a genetic algorithm starts with an initial population of individuals. During the execution of the algorithm, it will try and find an optimal solution based on predetermined rules. Each indiviual of the population has an attribute 'fitness'. This attribute denotes the quality of each individual of the population.

The particular GA used in the paper of Dragan Matić uses complete musical compositions as the individuals of the population. A composition is any combination of rhythm and notes, while the concept of good and bad music is represented in the fitness attribute. Because the notion of good and bad music is very subjective and different for each person, Matić uses a reference individual (another composition) for the evaluation of the resulting composition.

We start with the initial population, containing n individuals. After this, an iterative process begins. The fitness of each individual is calculated in each iteration. Based on the best individual (best fitness), we test if the stop

condition is met. This stop condition can be set by the user before execution. If the algorithm decides to stop, the best individual is played.

If the algorithm continues, we remove two-thirds of the individuals in the population. Mutation operators are applied to the remaining individuals to generate new different individuals. Now the next iteration starts and we calculate the fitness of the new individuals and test if we have found a good composition. This continues till the stop condition is met.

Determining the fitness This GA uses a function to compute the total fitness of an individual based on different criteria. Since there is an extra subjective aspect to music, the end result of the algorithm may comply to the criteria of the fitness function but still seem a bad choice (bad piece of music) to the listener.

Equation 1 gives the total fitness of one individual. λ_i represents the weight of the value f_i to the total fitness and n is the number of criteria [5].

$$f = \sum_{i=1}^{n} \lambda_i f_i \tag{1}$$

There are a couple of ways to calculate this. You can say that for different i, f_i may be a ratio between the number of tones out of a given tonality and the total number of tones, the ratio between the number of dissonant intervals and all intervals, the ratio between the number of appearances of some pattern in relation to the total number of notes, the density of notes, etc [2].

Categories of intervals	Intervals	Values Proposal
Perfect Consonants	unison, perfect fourth, perfect fifth, octave	1
Imperfect Consonants	minor and major thirds and sixths	2
Seconds	minor and major seconds	3
Sevenths	minor and major sevenths	3
Intervals greater than octave	all intervals greater than octave	5

The approach the GA in the paper of Matić uses a more general approach. You calculate the fitness for each bar and the and the total fitness is the sum of those values. Now, we have a new definition for f.

$$f = \sum_{j=1}^{k} \sum_{i=1}^{n} \lambda_{ij} f_i \tag{2}$$

where λ_{ij} is the weight of value f_i in the j^{th} bar, n is the total number of criteria and k is the total number of bars.

Comparison to the reference individual To know if the GA has chosen a good piece of music, we must compare it with the predefined reference individual. For this comparison we need a measure of similarity. This is determined based on the dissimilarity between them. To calculate the dissimilarity we use the note distance and the number and type of "good" intervals.

These intervals are predefined. See table 1 at the top of this page for a proposal of the values assigned to these intervals [2]. An interval here means two consecutive notes in a bar and their relation to each other. Are they in unison, a perfect fourth etc? Based on these values given to every note combination we can compute the fitness level of a bar of music.

2.3.4 Computoser

Computoser is another web application that generates random music. It is a rule-based, probability-driven algorithm. This means that it composes by following a set of rules and makes decisions between several (musical) alternatives based on predefined probabilities.

Computoser gives probabilities to every type of note interval (analoguos to the intervals in the GA section) and to each note length. This means that for example a fifth (a distance of five notes in the scale) gets a probability of 25%. This is a fairly high chance because a fifth is very commonly used in composing.

Computoser uses seven groups of rules.

1. Structure of the piece of music
2. Rhythm: notes must conform to a rhythm scheme
3. Repetition: each component of the piece (a group of notes) is repeated several times for memorability
4. Variations: slight changes are made to each component to make it more interesting
5. Dissonance and syncopation: unexpected elements in music are what makes it interesting to listen to
6. Endings: there are several predefined ways to end a piece of music
7. Effects: different kind of sounds, such as distortion, tremolo etc

Analogous to the probabilities given to intervals, each of the rules used in the algorithm has a certain percentage that decides which rule to use.

The components of the Computoser algorithm are called manipulators. They each fill in a part of information of the score. Analogous to the Unit Generators of the Beads library, these work as a chain. Subsequent manipulators depend on previous ones. There are four main manipulators.

1. Part configurer: what parts are there in the piece? Is there a bass line, a piano part?

2. Scale configurer: what scale/key is the piece in? The most likely scales are the major and minor scales. Scales like the Dorian are less likely.

3. Meter configurer: what meter is the piece in? 4/4 is the most common meter, but you can have a piece in 6/8, 3/4, etc.

4. Part generator: each part has its own generator and decides for each part what notes to play.

The part generator is obviously the most important one of the algorithm, because it decides what notes to play. There are three main aspects to this generator, pitch, length and variation. Again, the decisions are all based on probabilities.

Pitch Apart from using the probabilities there are a few other constraints used to determine what note to play next. First, there should not be more than two subsequent unstable notes. Unstable notes are notes that are quite dissonant. Secondly, long jumps between notes (more than 7 steps/notes) require a step in the opposition direction. Thirdly, a predefined sequence of notes may be used, if they are from the circle of fifths or part of an extended chord.

Length Both probabilities and some other constraints are used. The first is the measure size, all measures should have the same length. The second constraint is rhythm. Each measure is either simple (only one down-beat, e.g. 2/4) or compound (more than one down-beat, e.g. 4/4).

Variation Computoser uses motifs to construct the parts. There are a few simple motifs that we use as a basis. These motifs can undergo several musical variations to create new ones. A few of these are transposition (all notes go higher or lower within the same scale), inversion (turning the melody 'upside-down'), retrograding (playing the motif backwards), etc.

3 ArtToMusic's way to generate music

The most challenging concept of this project is the translation of graphical concepts, things we see on the image, into musical concepts, things we hear. I had to make choices regarding which graphical elements I wanted to translate into music.

3.1 Tempo

I decided to use the edges of objects in the image to determine the tempo of the music. When you have a picture with a lot of shapes in it, you are likely to get a piece of music with a faster tempo. I used edge detection algorithms, discussed in the paper on graphical analysis, to get a representation of the image where only the edges were shown. Depending on how many edges there are in the image and predefined ranges (based on musical theory), I randomly generated a BPM. If the picture has very few edges, the number would be generated in the *lento* range, [40-60].

3.2 Melody

Based on the rules of harmony, the program works with premade chord progressions. These are enumerations of a number of chords in a certain order which creates a melody. For example, the chord progression I-II-V-I is very commonly used. This means we play the first chord of the key we are in in the first bar, then the second in the second bar and so on. All of the chords used are major, minor or diminished.

Based on how much of certain color there is in the image we are analysing, we choose a different chord progression to work with. If there is a lot of red in the image, the program chooses the I-II-V-I progression, for instance. Other dominant colors lead to other chord progressions.

Using the technique of image segmentation, I will determine how many dominant shapes of colors there are. If this number is high, I generate an extra bassline to accompany the main melody.

3.2.1 Chord Progressions

I will be using some standard chord progressions and some more unusual ones. Several progressions are known to the ArtToMusic program and can be played easily, given the key of the song.

A few of these standard chord progressions are the following:

- I-II-V-I: a happy, vibrant feel
- I-V-VI-IV: a neutral feel
- I-IV-I-V: a very upbeat feel

To make matters more interesting, I use some famous numbers from mathematics, such as π, ϕ and e. These irrational numbers can be seen as a series of digits. These digits can be seen as chords in a progression. For example, the first five digits of π are 3, 1, 4, 5, and 1. When we see them as chords we get the following chord progression: III-I-IV-V-I, which seems a very interesting progression.

References

[1] Ollie Bown (http://www.beadsproject.net/) 2008

[2] Evan X. Merz *Sonifying Processing: The Beads Tutorial* 2011

[3] Stephen Wolfram *A new kind of science* 2002

[4] The Wolfram Language Documentation Center *http://reference.wolfram.com/language/guide/LanguageOverview.html*

[5] Dragan Matić *A genetic algorithm for composing music* 2010, Faculty of Natural Sciences, University of Banjaluka, Bosnia and Herzegovina

[6] Bozhidar Bozhanov *Computoser - rule-based, probability-driven algorithmic music composition* 2014, independent researcher