Music Generation in ArtToMusic

Rafael De Smet

April 18, 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 compare several methods, some of which I will use in the ArtToMusic project while others not. Since music is different for every person, my reasoning can be subjective at times and you may prefer other methods.

I provided a standalone application which contains several demos of the techniques that are discussed in the text. To hear the demos, just click on the corresponding button.

2 Music theory

2.1 Tempo and harmony

When we listen to music, unconsciously we focus on a couple of things to process what we are hearing. Music is the result of the interaction of rhythm and melody.

2.1.1 Tempo

The discussion of how rhythm works is beyond the scope of this paper. I will only focus on the tempo of a song. 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 determines how many quarter notes (beats) are played in one minute, which is a measure of how much time passes between two quarter notes.

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. This means the quarter notes are played at intervals of three seconds (60000/20). *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, which contains the rules to make melodies. 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.

To hear the different types of chords, click the **Major Chord**, **Minor Chord** or **Diminished Chord** button in the demo application.

2.2 Chords and progressions

Chords are grouped into chord progressions. The most common example of this is a twelve bar blues. This type of music is characterized 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.

The roman numbers denote which chord is played, relative 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 major chord. The IV is the fourth chord of the key, F major. The V is the fifth of the key, 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.

To hear some progressions, click on the buttons **Blues Form** or **I-II-V-I** buttons in the demo application.

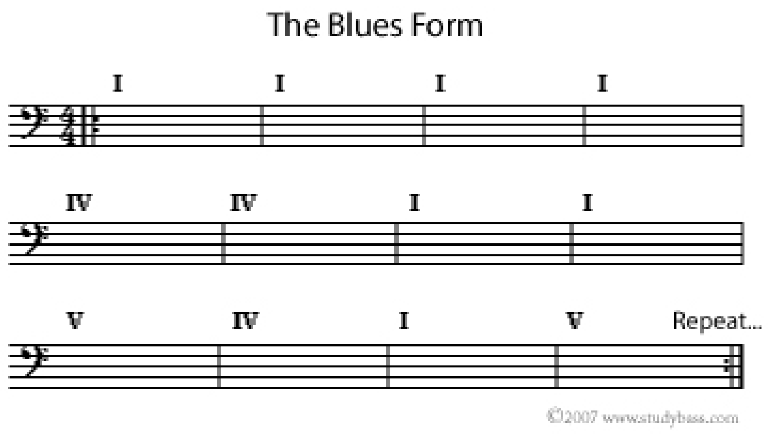

Figure 1: Twelve bar blues

3 Music generation technologies

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

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 [1]. 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 and sounds.

The Beads library has many of these UGs, such as an envelope filter, a gain, a waveplayer, etc. The AudioContext is the main UG into which every other UG plugs in. 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 music comes from other UGs.

Beads has two main ways of creating music, samples and waves.

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 to 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, called Buffers. Each Buffer stores an array of floats in the range [-1,1] that are generated by 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

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. To hear the results, click on the **Sine Wave**, **Saw Wave**, **Square Wave** or **Triangle Wave** buttons.

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

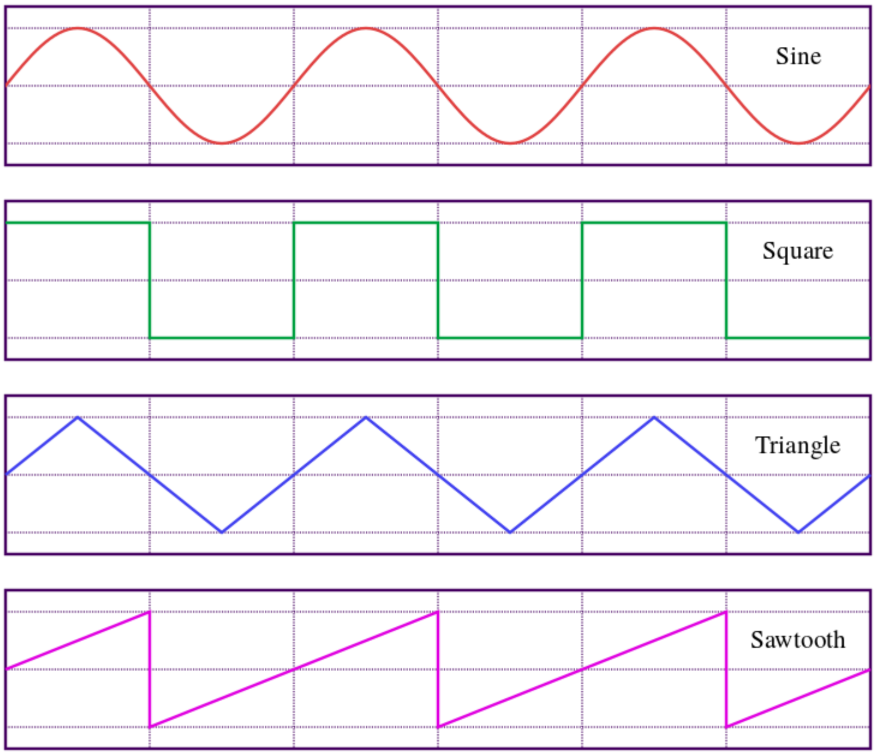

Figure 2: Buffer functions

3.1.1 Examples

The Beads project is open source. When you go to the library's website [1], you can download the latest version of the source code. This code contains several tutorial projects that showcase how the library works. There are eight different tutorials or lessons provided, increasing in complexity. It starts off easy with an explanation off how to use the AudioContext UG in code (Lesson01_AudioContext) and ends with a complete runnable program that generates random music based on some input UGs (Lesson07_Music).

I will discuss the code and results of Lesson07_Music to give the reader an idea of the capabilities of the Beads library. To hear the results of Lesson 07, click on **Music Demo** in the Beads area of the demo application.

Code of main melody part An AudioContext is mandatory. Its main UGs are a Clock and a Gain. The Clock is a special type of UG. It works just like a regular clock, and runs for ever (till you terminate the program), while other UGs are triggered when necessary. A Gain controls the volume of the audio source that is passed to it.

The Clock receives messages on-the-fly that determine which notes when to play. There are a few advantages to using a Clock. First, it has the notion of beats per minute (BPM). We add a new note to the Gain for every beat we encounter. This note is generated from a musical scale (e.g. C Major) which was chosen by the programmer. In this case we are working with a mixolydian scale based on a random chosen pitch. From this scale we generate a random note which is then added to the Gain to be played by the AudioContext.

In the code you can use multiple Buffers. To generate the main melody of the song the Sine wave is used, which is the most enjoyable of all to listen to.

Code of bass part and rhythm After the generation of the melody part, the bass and the rhythm parts are generated. The code used to generate a bass part and a rhythm part is very similar. For the bass part we again choose between multiple scales and buffers. In this case we use a pentatonic scale based on a randomly chosen pitch. To make it sound more simplistic and not too dominant with regards to the main melody, we use a Square wave for the Buffer. This produces slightly harsher sounds than the Sine wave.

The rhythm is generated using the concept of Noise. Noise in audio is anything that's not a note, for example the noise of an old TV when the channel was off the air (what was called snow). This sound is not pleasant to hear but when we strategically play small bits of it, it sounds like a snare drum playing a rhythm. Again, the code determines randomly when to play this noise.

Sample Lesson The previous lesson shows what Beads can do by generating music autonomously. Lesson04_SamplePlayer is a tutorial to use the SamplePlayer UG. This loads a sample and plays it. You can again run this sample through different UGs to modify the sound as you want. To hear the results of this lesson, click on **Sample Demo** in the Beads area.

3.1.2 Conclusion

In conclusion we can say that by using the Beads library, music can be randomly generated. The benefit of this library is that you can easily integrate it in your own application and that it handles all of the communication with your system's audio device. The disadvantage of it is that it takes quite a bit of experimenting and work to fully understand how the code works and how to use it yourself.

3.2 Wolfram Tones

Designed by the mathematician Stephen Wolfram, WolframTones is an online application to generate music. The user can choose from multiple styles of music resulting in new pieces of music. The way this works is based on an earlier discovery of Stephen Wolfram. 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 algorithm starts with one line 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. When the pattern in the top row of the rule is encountered in the current line, the color of the middle cell on the next line is determined by the bottom row of the rule. Each rule gives one of the possible combinations of colors for a cell on a line and its immediate neighbors. The bottom row then specifies what color the center cell should be on the next line [3]. E.g. in figure 3 a particular cell will be black if either of its neighbors was black on the line above. A cell will be white if both its neighbors were white on the line above.

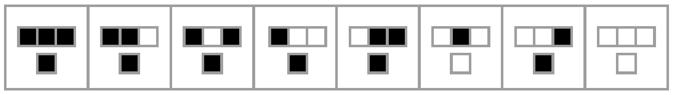

Figure 3: Rules for automata

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.

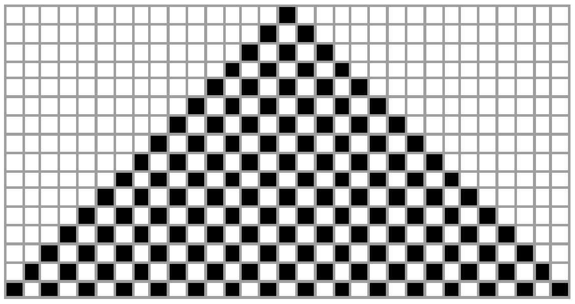

Figure 4: Result automata

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.

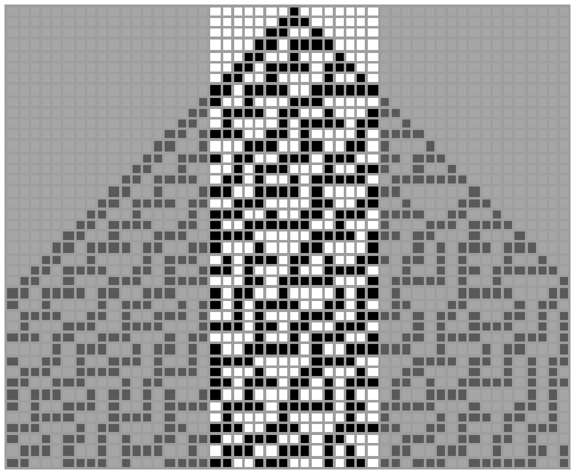

Figure 5: Partition through pattern

We can assume time runs across the score from left to right and every black cell represents a note played. Wolfram has developed his own programming language [5] to process these cellular automaton patterns.

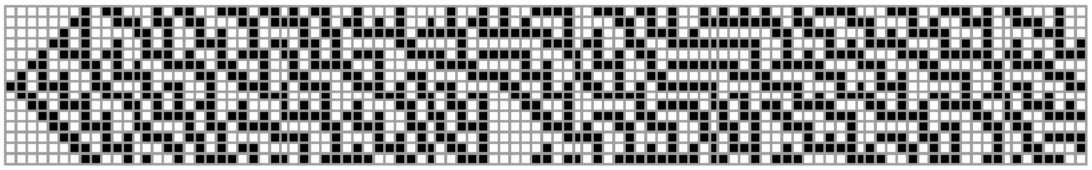

Figure 6: Musical score

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 neighbors, 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.

3.2.1 Examples

WolframTones is a web application [4], mostly black-box and closed-source. Every time you open the website, it generates a random piece of music on its own. There are multiple input parameters you can change to create music that fits your requirements.

Music Style You can choose from a wide variety of styles, such as Classical, Dance and Country.

Algorithm Control Here you can choose the automaton to be used to generate the composition. These inputs determine the rule type and the specific rule (see above) used to generate music. You can set other parameters such as the random selection seed and the height of the rule.

Instrumentation In this section you can choose what instruments you want to hear in the piece. You can choose from 117 of the 128 General MIDI instruments. The sound effects (MIDI 121 - 128) and a few other weird sounds such as a whistle are not included in the list of choices. Per instrument you can choose the role of this instrument in the song. There are 119 roles to choose from, varying from playing chords in a swing feel to playing a hip-hop inspired bass part. For every song you can select up to five instruments, each with its own individual role. This means we have in total $117 * 119 * 5 = 69615$ different combinations of five instruments playing together in different roles.

To make things even more complicated you can also add a percussion part where you can choose from 111 types of percussion, ranging from a reggae beat to a jazz beat.

Pitch Mapping In this section you can choose a musical scale and a pitch where this scale is based on. WolframTones has a massive collection of scales to choose from, to be exact 404 different scales. These range from the common Western scales such as the Harmonic Major and the Blues scale to the more unknown ones, such as the Indian and Japanese scales.

There are 48 different pitches for your scale of choice, these range from C1 (number 24, a low C) to C5 (number 72, a relatively high C). Again many choices to form a unique piece of music.

Time Control The last section is the Time Control where you can determine the BPM, the notes per beat and the duration of the song. The BPM is straightforward (see above for more information about this) and ranges from 24 (very slow, *larghissimo*) to 288 (very fast, even faster than *prestissimo*). The notes per beat range from 1 to 16.

In the demo application you can find examples of a few different combinations I tried on the website. Below the choices that were made for each result are discussed.

1. Rock/Pop style, based on rule type 31 (r=2) and rule number 3000041762. Random seed is set to 6800 and height of the score is 12. There are three instruments playing, each with another role.

   - Electric Bass (Pick) with role: Bass - Driving
   - Guitar (Jazz) with role: Lead - Upper Part
   - Rock Organ with role: Lead - Lower Part - Legato

   A Funk 3 percussion track is also added. The scale is a Minor scale based on the F2 pitch, number 53. BPM is 116 with two notes per beat and the piece is 15 seconds long, 60 steps. Click on **Wolfram Demo 1** in the WolframTones area.

2. R&B style, based on rule type 7 (r=1) and rule number 168230930. Random seed is set to 123721 and height of the score is 16. There are three instruments.

    - Electric Bass (Finger) with role: Bass - Driving
    - Trumpet with role: Lead - Upper Part - 4x4 Loop
    - Grand Piano with role: Lead - Moving - Shorter - Legato

   A Hip Hop 1 percussion track is added. The scale is a Blues With Leading Tone scale based on the B1 pitch, number 47. BPM is 88 with 4 notes per beat and the piece is 15 seconds long, 88 steps. Click on **Wolfram Demo 2** in the WolframTones area.

3. Ambient style, based on rule type 1585 and rule number 338039596. Random seed is set to 10839 and height of the score is 13. There are four instruments.

    - Electric Piano 2 with role: Polyphonic - Length 4+
    - Electric Bass (Fretless) with role: Bass - Moving - Legato
    - Harp with role: Lead - Moving - Shorter - Medium
    - Bright Piano with role: Chords - Comp

   A Jazz 1 percussion track is added. The scale is a Harmonic Minor scale based on the G#2 pitch, number 56. BPM is 38 with 3 notes per beat and the piece is 14 seconds long, 28 steps. Click on **Wolfram Demo 3** in the WolframTones area.

To show what happens when we alter only one parameter, I changed the scale, number of notes per beat and the random seed of Demo 3. To hear the version of Demo 3 with the Major Pentachord scale, click on **Other Scale**. To hear the version of Demo 3 with 5 notes per beat, and again the Harmonic Minor scale, click on **More NPB** (NPB is Notes Per Beat). To hear the version of Demo 3 with the random seed set to 15159, click on **Random Seed**.

3.2.2 Conclusion

WolframTones is a very powerful tool to create music. The way to define the characteristics of your piece of music is very easy. WolframTones provides you with the notion of musical styles and scales, rhythm etc. When we look at Beads these concepts were not so clear when providing the input parameters. This makes WolframTones a very easy tool to use. The only disadvantage is the fact that it is mosty black-box, closed-source and restricted to a website.

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 [6].

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 individual 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. Such a compostion is represented as a musical score. 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 must 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 less fit 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 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 [6].

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

There are a couple of ways to define the criteria. You can say that for different i, f_i may be a ratio between the number of tones out of a given scale 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 [6].

The GA in the paper of Matić uses a different approach. This time we calculate the fitness for each bar of the score, 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, an existing composition. 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.

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

Figure 7: Table of intervals

See figure 7 for a proposal of the values assigned to the predefined intervals [6]. 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.

3.3.1 Examples

TODO

3.3.2 Conclusion

TODO

3.4 Computoser

Computoser is another web application that generates random music [7]. It is a rule-based, probability-driven algorithm. This means that it composes music 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 (analogous to the intervals in the GA section) and to each note length. The next note of a score is determined by this probability. 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 rhythmic 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 its own probabilty.

The components of the Computoser algorithm are called manipulators. They each fill in a part of information of the score. Analogous to the UGs 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 is used in the piece? 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 tonal direction. Thirdly, a predefined sequence of notes may be used if they are from the circle of fifths or part of a chord used in the song.

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.

3.4.1 Examples

On the web application of Computoser, each track you hear is algorithmically generated. This means that there are inputs to be defined for the algorithm. Where WolframTones was black-box and closed source, the source for this application can be found on Github [9].

I will again explain the different input parameters we can choose and the results that this gives. In contrast to WolframTones, the range of inputs is considerably simpler.

Mood There are three options to choose from to set the mood of the piece of music: Major (happy mood), Minor (sad) or a random choice made by the algorithm.

Tempo For the tempo there are six choices, ranging from very slow to very fast, again with the option to choose the tempo randomly. In contrast to WolframTones there is no clear distinction between the different choices of tempo. The notion of 'very slow' is different for every person, whereas the notion of a BPM of 30 is clear to everyone.

Accompaniment (chords) Here you can choose if you want to hear chords accompanying the track or not.

Instrument Here you can choose the instrument that will be playing the piece of music. Again, in contrast to WolframTones, there is a limited choice of instruments, seven as opposed to 117 instruments at WolframTones.

Scale This parameter also has a limited range of choices. There are 'only' 13 different scales, while WolframTones had 404 unique scales.

Classical, Electronic-like, Drums, More dissonant With the last few parameters you can choose if you want a classical sound or not, an electronic-like sound or not, drums, or a more dissonant sounds.

In the demo application you can find examples of a few different combinations I tried on the website. Below is explained what choices were made for each result. Click in the application on the respective button to hear what the results sound like.

	Computoser 1	Computoser 2	Computoser 3
Mood	Minor	Major	Any
Tempo	Medium	Fast	Very Slow
Chords	Yes	Yes	No Chords
Instrument	Guitar	String Ensemble	French Horn
Scale	Minor	Major Pentatonic	Indian
Classical	Optional	Yes	Optional
Electronic-like	Optional	No	Yes
Drums	Yes	No Drums	Yes
More Dissonant	Optional	Optional	Yes

Figure 8: Input parameters of Computoser demos

3.4.2 Conclusion

The results sound more like actual music than WolframTones. It is easier to listen to because you recognize many musical patterns from 'real' music. The building blocks of Computoser are adapted to create music that sounds very familiar.

The downside of this is the limited range of possibilities, while WolframTones offers much more. The results of Computoser are not pleasant to listen to, but they are very recognizable. In the case of WolframTones, it is the other way around.

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] http://tones.wolfram.com/

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

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

[7] http://www.computoser.com/

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

[9] https://github.com/glamdring/computoser