Cellular Automata and Music: A New Representation

Rich French

March 22, 2013

1 Introduction

Due to the temporal and complex behavior of Cellular Automata systems, researchers have looked for a link between these systems and musical composition [8]. However, there hasn't been a consensus on what the best way to translate these complex systems into music would be. If a musical representation could be created that allows for all of the components of musical composition the larger work of evaluating which CA rules are musical could be accomplished. We will look at a Cellular Automata system that uses layers to dictate the different aspects of a musical piece. We wish to show that this representation provides more musical elements than other systems, and it will do it on its own without human input except with a few initial criteria. My motivation for this project involves looking at the work done by the likes of Iannis Xenakis, Mozart and John Cage in the field of algorithmic and experimental music [13][9]. We think of musical composition as a high-level human exclusive activity. If our system can be shown to produce music without much human input, it might shed light on the possibilities of computers to perform high level tasks. It also provides an opportunity for the musical layman to delve into musical composition.

2 Cellular Automata and Music: Previous Work

2.1 Cellular Automata

Cellular Automata are matrix systems where the spaces, or cells can be one of a finite number of states. The most intuitive way of thinking about this is with the concept of life and death, a cell is either filled in or it is empty. When multiple states are used, essentially it means that there are multiple ways of being "alive". A rule is applied to the system, these rules take the current state of a cell and then checks the states of the cell's neighbors to determine what the state of the cell with be for the next iteration. This rule is applied to every cell in the automaton, and the result is a completely different configuration of cells [12]. These simple systems can create very complex behavior and interesting patterns. For this project we will be looking at mainly 2-dimensional Cellular Automata but rules are technically possible for any amount of dimensions. The most famous of 2-dimensional rules is Conway's Game of Life, which creates very complex

behavior from a fairly simple and intuitive rule. First, we need to understand how a "neighborhood" is determined in a cellular automaton. A neighborhood is essentially a group of cells that surrounds the current cell that is being changed. When the neighborhood radius is 1, this will be the 8 cells that surround any given cell in a Cellular Automata system [12]. A rule is determined by stating what amounts of each states in this neighborhood will lead to a change of state of the selected cell. For Conway's Game of Life there are two cell states, alive or dead. A cell is "born", that is it becomes filled in during the next iteration, if it is surrounded by exactly 3 neighbors. An alive cell will survive only if it is surrounded by two or three neighbors. In all other instances a cell will be dead on the next iteration. In order to create different behaviors, one only has to change how many cells are required to be born and how many neighbors are required to survive to the next iteration [6]. These are the components needed to create a cellular automata system, an idea of where to look for neighbor information and a rule that determines how the state of the current cell will change in respect to this neighborhood information.

To show the vastly different possibilities available with cellular automata and because it will become useful later I will also explain the family of CA rules called excitable medium CAs. Excitable Medium CAs have multiple states and are cyclic meaning that when a cell reaches the highest possible state, it then will change to the 0 state[11]. A cell has two options for behavior, either it stays in its current state or it changes to the state one more than its state [11]. The result of these rules is that from a random initial configuration of cells will iterate towards more stable groups of states [11]. This differs from life automata in that the goal isn't necessarily generating complex behavior, but instead the goal is to generate order from chaos.

2.2 Music

If our CA music system is successful, it will have the many features of a musical piece called for by music theory. Of course, because music is an artistic medium composers can break the mold with some of these concepts and ignore them but a brainless system needs these basic features to create a musical result from the near-chaos of our system. Our goal is not to create best possible music, our goal is to create results that are musical, results that whether or not a person cares for the style of the piece, they would still agree that it is music as they understand it.

The first aspect of music that our system needs to contain is melody or harmony. This designates which notes will be played in a musical piece, and the relationship between these notes [1]. Most harmonies are created by matching notes in a particular musical scale, chords are also formed from notes of the same scale. Scales are groups of notes that are related by the pattern of intervals used to create the scale. An interval is simply the distance between one note and another note[1]. If you are looking at a piano, a half step interval would be moving to the piano key that is touching the one you are currently on. A whole step would involve moving two piano keys over from the original key. One of the most common scales in western music is the major scale which is designated by the intervals you move from the root note (where W is a whole step and H is a half

step), WWHWWWH [1]. The root note of the musical scale determines the key of the musical piece. Notes of different scales will typically clash and by consequence not sound musical to a listener [1]. In order for a CA music system to sound good to a typical listener, notes of a melody should fall into an established scale.

The next aspect of a musical piece that needs to be accounted for is rhythm and note placement. Rhythm designates the timing of a musical piece. Rhythm is typically a regular pattern of beats designated by a time signature [1]. A time signature is a ratio where the numerator designates how many beats make up a single measure, and the denominator determines what note represents a single beat [1]. So, a time signature of $\frac{4}{4}$ means that a quarter note is a single beat and four of these quarter notes makes a measure. Then, the rhythmic pattern is designated by its speed, or beats per minute. In other words, you can play a piece with the same rhythmic pattern but faster or slower depending on the BPM of the musical piece. Related to this is note duration. Essentially, notes lengths are some fraction of the length of measure long note. A whole note lasts for an entire measure, a half note is half of that length, a quarter note is half of the half note's length, etc. This is tied to rhythm because how long these notes actually last is determined by the rhythm of the piece. If the time signature was $\frac{2}{4}$ then a whole note would be the same thing as a half note in a song with a $\frac{4}{4}$ time signature.

Another aspect of music is the structure of the musical piece. Typically a piece of music is split into sections where each section has a different role in the overall musical piece [1]. For example, a common musical form is the Rondo form which takes the form ABACAD... for as many unique sections as is appropriate. Another common form is the verse bridge chorus form of much popular music. A successful CA music system would be able to impose some structure on the musical piece, or derive the different sections separately to create an overall song in a structured form.

2.3 Cellular Automata and Music

There have been many attempts to generate music over the years, including applications using Cellular Automata. For centuries people have played with the idea of using mathematical laws to generate music. Wolfgang Amadeus Mozart created a game called "Musickalishes Wurfelspiel" which produced fugues from already established bars of music which would be ordered by a dice roll [13]. The first artist to use Cellular Automata for music composition was the Greek composer Iannis Xanakis. The composer used a 5 state, 3 neighborhood one dimensional automata to determine which instruments would occur when and the chord progression of a few bars in his composition Horos [9]. Researchers have attempted to create systems more academically sound that could create music without a composer's input. Three main concerns emerge when creating a CA music system: What Cellular Auotomata rules to use in the system, how to represent a CA in a musical manner, and how to interact with this data once iterated. Since our project is mainly focused on creating a good representation for CA music, I will mostly focus on previous work that focuses on that aspect.

The decision of what CA rules to use is a large one. In Cheyrons' work with creating an algorithm for iterating a sound wave using an automata (called the Linear Automata

Synthesis algorithm), the algorithm was capable of creating 10^4500 distinct automata [5]. This is not even the total number of automata available but only the amount available to the system due to memory constraints, so a brute force approach of checking every rule and evaluating its musicality is essentially impossible [5]. Some rules examined by researchers have included Conway's Game of Life, elementary rules including rule 54 and 110, and voter models like the multi-type voter method [6][3][8]. The problem with evaluating the results of these rules is that the success of the rule is entirely dependant on the representation chosen. In order to solve this problem, a standard CA music system would be needed to evaluate these rules.

The next major problem associated with cellular automata and music is how to best represent a piece of music or a sound as a cellular automata. One of the simpler representations is the Cartesian representation of Miranda [6]. This entails plotting notes on a 2-dimensional grid where each space represents a group of three notes [6]. The x coordinate designates how many half-steps from some arbitrary base note the second note is and y designates how many half-steps up the third note is from the middle note [6]. The placement of these notes is designated by a complex grammar as follows: Tgg = abcd—dcba Dur = mnopjponm Where abcd are the states of the north, south, east and west neighbors and mnop are the neighbors at the corner of the current cell. Depending on the four bit binary expression that results from this grammar, the notes are then organized into chords or single notes [6]. The system also uses a second layer with a Cyclic CA to determine the instrumentation of the musical piece, where each state determines a different instrument that the top layer will play. While this is a simple approach, there are some concerns. The first is that this method arbitrarily groups three tones together. Unless there was a way of linking three notes together in such a way that their musical relationship is preserved, I don't see a reason to do this since the notes in a piece should be able to evolve how they see fit instead of being arbitrarily linked to other notes in the piece. Another concern is that even with this complex grammar for note placement, rhythm isn't evolved or iterated. If a piece of music starts out in $\frac{4}{4}$ time, it will end in that same time signature [6].

Other methods, such as the ones utilized by Serquera and Chareyron generate sounds based on their waveform instead of their particular note value [8][5]. The work of Serquera uses a method which converts a histogram representation of a cellular automaton into a spectrograph, which can then be used to play the sound. The histogram uses the grey-scale values of an image of a cellular automata to graph the different probabilities of each gray-scale value occurred, calculated by the number of pixels with that value divided by the total number of pixels [8]. Spectograms are a common way of graphically repre- senting sounds, and the histogram values are converted into a spectogram by considering each bin of the histogram a bin of the spectogram [8]. When using viable automata, this histogram representation has been shown to have sound-like properties such as sine wave patterns [8]. The work of Cheyron also uses a wave-form representation to evolve sounds [5]. Here, for each sound to be iterated, it is first recorded and placed inside of a wavetable so every element, or sample, of the wavetable represents a cell in the cellular automata [5]. The value of an element in the wavetable is then stated to

be the state of the corresponding cell in the cellular automata [5]. While both of these methods are very interesting due to their high level their major draw-back is that both represent specific wave-forms which might not translate to a specific musical note, and so these methods would be better suited to possible real-world applications with sounds outside the realm of music.

Another representation is the one used by the WolframTones music system. This system creates music by taking an elementary one dimensional automata of many iterations, taking a rectangular section of the automata from top to bottom and then turning this rectangle on its side [7]. A note's pitch is determined by its vertical location, which is then mapped to a certain scale that is chosen by the user. One automata is used to create a musical piece through the use of algorithms called "roles" that choose notes to be played by a group of instruments [7]. The notes are then played from left to right. This system will be very similar to the system used by us but there are still some key differences. For one, with the WolframTones system the music isn't generated exclusively by the CA. For one, an algorithm is needed to pick what notes will be played and which wont be. While the pitches are determined by a CA, this seems to obscure the CAs total behavior. We aren't trying to see if parts of a CA can be turned into music, but if an entire CA system can be analogous to a musical piece. Also, many aspects of the musical piece are imposed by the human user, like the tempo and current scale to be used [7]. I would like to see a system where these aspects of the piece are generated just like the melody of the piece is generated.

3 The New Representation

Our representation seeks to use basic music theory to create a CA representation that will produce a suitable musical output. Most CA music systems attempt a one-size-fits-all approach, looking for a single automata to contain all of the information for a musical piece. However it doesn't make sense to think that the different components of music should be evolved with similar behaviors. The melody of a piece of music calls for more complex behavior than say rhythm which requires more cyclic behavior. Also, trying to fit all of the musical components into one CA might result in overfitting the system, or like in WolframTones, might require algorithms beyond CAs to make the system fit in a musical way. I wish to create a system that exclusively uses CAs to create music, and also uses the correct types of CAs for different aspects of a musical piece.

The basic layer of the musical composition will be the melodic layer. This layer will contain information on what notes should be played, in what order, and what their highest possible duration will be. Notes are taken from left to right, providing a strict idea of note ordering in the system. For the melody, cells are either alive or dead. Too many states would overcrowd the system, leaving little room for rests and generating too many chords. In order to determine what pitch an alive note represents, one must look at its vertical location. The pitches are determined by a musical scale. For a scale an octave long, 8 vertical spaces would be needed with the top space being the root note of the scale (also known as the key of the scale), with the higher note of the scale

being placed in order as you move down the system. While the system needs at least an octave of a scale, the CA can be made taller. In this case, the pitches of the scale would be repeated an octave higher until there was no longer room. Some chords like 9 and 13 chords require these extra notes so allowing for their inclusion will further allow the automata to create the interesting sounds available to human composers. When multiple notes are alive in the same column, this designates a chord. It was important to the system that whether to play multiple notes or single notes wasn't imposed in the behavior of the system. The shape of the final CA iteration is the only thing that determines this. A rest occurs when there are no alive cells in a column. A good rule for this system will be one that is sparse or open enough to allow for this. Now, if we imposed a single note duration like each note is a quarter note, this would be enough. Simply move from left to right checking each column for notes, if there are some play them for a quarter note duration, if not then there will be a quarter rest. However, music requires different note durations to allow for the many different compositions that are possible. In fact two songs with the same notes, but different choices for note durations will sound much different. We allow for longer notes by stating that horizontal groups of notes with the same pitch have the opportunity to have longer durations. We could end here, content with the fact that we allowed for higher note durations but we need to recognize that the system now doesn't allow for two notes of the same pitch to be played successively. A song as simple as twinkle twinkle little star would be impossible to play in our system. However, we would prefer to not impose more constraints into our melody CA. The solution seems to be to create a new layer, with a new CA rule to determine the final note durations. The smallest length a note can be is a 1/16 note, so in other words 16 horizontal notes would be a whole note, 8 would be a half note, 4 a quarter note and so on. So, each measure is 16 horizontal squares. When determining the total horizontal length, this will be determined by multiplying 16 and the number of bars or measures in the musical piece. For this layer, the family of Cellular Automata that we will use will be the two-dimensional life automata. This family of automata can provide very complex behavior, but their rules only deal with two states alive or dead which allows for an appropriate level of "sparseness" for intelligible chords and the occurrence of rests.

Note duration would require a multistate automata which will indicate where to break the horizontal groups of notes obtained by the melody layer of the CA music system. We will need 16 states since this will allow for a whole note to be broken into 16 different notes if need be. This "note duration" layer would be the same size as the melody layer. When looking to break a horizontal group in the melody layer, the system will look at the same location in the note duration layer. The only information we are interested in is a change of state from left to right. If the entire note is one state or color, then the system will play a single note of that length. For example if a horizontal group is 8 cells long, and in the note duration CA the same location contains one state then the system will play a half note. However, if there is a change of state at this location, say the first two cells are "yellow" then the next four are "blue" and then the last four are "red". This will indicate to the system that we need to play one eighth

note stop, then play a quarter note of the same pitch, stop then play another quarter note of the same pitch. A separate consideration needs to be made to handle chords. Since it will be rare for the system to generate three or more horizontal groups, it will be more likely that a chord will contain a note that ends up having a longer duration than the other notes of the chord. I believe this will have the issue of creating only 1/16 length chords followed by long holds of a single note of that chord. This is certainly possible in music, but longer chords where all the notes last for the same duration seems to be much more common. This is why for chords, the note with the longest horizontal length will determine the total length of the chord. Note breaking will only be looked at for the "longest" note of the chord. If other notes besides those contained in the chord occur they will be played normally, but new notes of the chord will be ignored until the chord is finished playing. Excitable medium CAs take a random initial condition of these states and results in more stable groups of these states. This will be the family of CA rules that we will use for this layer.

Another layer will be implemented which will choose the current scale that the pitches fall into. This layer will be a multi-state automata where the different states indicate different possible scales that the piece can use. Again, this layer will be the same size as the melody layer, where a given column determines the scale of the column at the same location on the melody layer. What scale will be played will be determined by which state is most represented in each column. If there is a tie, then the scale with the higher state value will be chosen. This layer allows for the key to be chosen by the CA system instead of being imposed by the user. A large goal of this project is to create songs with a CA system without alteration from a human user. The musical piece should be completely generated. Again, an excitable medium CA will be used as the rule for this layer. A note about the interaction of these layers would be that when there is a change of scale, there is also a note break if there are any notes in the middle of being played while the scale changes. This is because if a human player was pressing these notes, he would have to change his finger position to play the new chord or note.

One of the reason why I'm for this layered representation is because any other aspect of a musical piece only requires a new layer to be created. For example, if we wanted to have our system play multiple instruments we could have another layer with multiple states, where these states would each represent a different instrument. A note in the melody layer would be played by the instrument whose state is at the same location as the note in the melody layer. Another option would be to have multiple systems for multiple instruments, and the instrumentation would be determined by the state with the highest representation through the entire layer, instead of looking cell by cell or column by column. The point is the other layers can stay the same with the same functionality when new features are added. This means that a new feature won't overfit the system or cause an entire rewrite of how the system works, which would occur with a one-size-fits-all approach to the problem. Another reason why this representation is intriguing is that while we will be mainly looking at each CA system as an entire song, this isn't imposed on the system. A user can look at the song measure by measure, taking each iteration as the next measure of the song. Also, one can look at the current

system as being a single section of a song. You could iterate a system as the verse of a song and another system could be thought of as the chorus of a song.

3.1 Work Done First Term

The first step during this time was putting together the details to this representation. Once that was completed the job was realizing this representation in an algorithm that takes these layers of cellular automata and converts them to an actual music file. This requires a suitable environment with which to work with CAs and some way to interact and extract data from this environment. We discovered a great open source program called Golly which is used to visualize and interact with these CA systems. Golly provides built in functionality to deal with one or two dimensional cellular automata systems [10]. It provides many different neighborhood configurations, provides built in functionality for life rules and provides the materials for creating other families of cellular automata rules, it provides layers, and allows for as many states and whatever size you want to impose [10]. The most helpful feature of Golly however is its ability to take Perl and Python scripts and modules for interacting with and extracting data from these CA systems. This allows for our translation from CA to music to be a script within Golly. The only extra component needed is a module for creating MIDI files. For this, I chose MIDI Perl 0.82 which provides a simple environment for creating MIDI files, but it still provides enough functionality to create complex musical pieces [4]. Essentially, while the script is searching the CA system for the information it wants, whether a cell is filled in or not for example, this information is translated into note information that can be fed into the MIDI song being built. When the program finishes, the song will be published.

I was able to get the translator script to work in the sense that it took CA information and made a music file out of it, but it hasn't yet been built to the full extent of our representation. Essentially, I got the translator to work by imposing that every filled in note or chord is a quarter note in length and by using the same scale for the entire song. What is gained from this is that the current choice of tools is adequate for our design, MIDI Perl still works well when it is interacting with Golly and it also shows that MIDI Perl can build a song incrementally with rules instead of a single script that simply states the notes to be played and in what order. I hit a snag when trying to make this script more modular and readable though. The array that contains the note events of the song seems to have trouble maintaining its information after being passed from submodule to submodule in Perl. This is probably due to my inexperience with writng Perl scripts, but this was the point that I currently am stuck on going into next term.

Another aspect of the project was determining which CA rules are interesting for the different layers of our system. For the melody layer, we are looking for rules that provide a large amount of complexity but also are adequately spacious so chords make sense and rests can occur. We will look at Conway's Game of Life because it has been used by many other CA music systems like Miranda's CAMUS system and it since it is the most well known CAs, it will provide a good baseline for our system. Another rule that we will look at is Seeds which is a rule that states that a cell is born if it has two neighbors and doesn't survive in other cases. This is an interesting rule because since a large amount

of cells die each iteration, we get the space we are looking for and on top of that the rule provides complex behavior. Other possible rules include Antilife (B0123478/S01234678) and Highlife(B36/S23). These rules are certainly complex enough to be interesting, but they might result in some odd configurations of chords. These rules will be used to create a corpus of songs for evaluation. One thing to evaluate will be if any of these rules produce more musical results than the others. Perhaps some rules are better for different roles. For instance, there might be a rule that is more cyclic which would be good for an instrument providing rhythm. A more complex rule should be good for lead melody. For the other layers of the CA system, an excitable medium automaton will be needed. One rule is the Greenberg Hastings model which is the simplest form of excitable medium automata [11]. The rule states that in order to change from the 0 state to the 1 state, there must be a certain number of "1" states surrounding the current cell, this threshold produces different behavior depending on its value. After this, states 1 and greater automatically change to the state one greater than it. We will also look at the "Perfect" and "Imperfect" excitable medium CAs. For Perfect, if a cell of a certain state is surrounded by 3 or more cells with state+1, then during the next iteration the cell will have state+1. Imperfect is similar except this threshold is 2 [11]. These provide both a consensus and grouping of similar states while still containing some complex behavior, which might be useful for the note duration layer.

3.2 Work Done Second Term

The first half of the term was focused on finishing the translator, and implementing the duration and scale layers. During this part of the project I made the program much more modular. The addition of the other layers to the design was the major factor for slowing down the program since changing layers takes much more time than performing CA commands, which Golly does very well. To implement the new features, I changed to a new song format in MIDI Perl called score, which allows you to input note events by their start and stop times rather than their duration and order. The nature of our algorithm made it necessary to go through each row left to right, checking the duration everytime. This caused a major slow down due to much shifting between layers, this would be a good place to improve our algorithm if future editions were possible. Our other task was to implement a Cyclic excitable medium CA into Golly for our scale and duration layers. Unfortunately, Golly does not have the ability to input these rules easily so a rule table must be used. In a rule table, all transformations for all available states are inputed manually in a format similar to the following: C,N,S,E,W,C+1, where C is the current cell, C+1 is the current cell after the next iteration, and the rest are the neighbors of the current state. Unfortunately due to time constraints we were only able to implement the CyclicCA rule in this way. CyclicCA had it's limitations for our purposes. I was hoping for behavior that would provide definite patches of order to our random system, but CyclicCA can only change so many of these cells and when it does all of them change to a single state. This ability to create order also seemed to differ depending on the number of states; the scale layer tends to move towards order more easily than the duration layer, which has more states.

4 Evaluation/Results

4.1 Evaluation

The major judgment of success will be an experiment that uses the concept of magnitude estimation to evaluate the musical pieces generated by our system. Magnitude Estimation is a method of experimental evaluation that has been used extensively in psychiatric and linguistic tests [2]. It is good at providing concrete results for fairly subjective phenomena. For instance, when used with grammar it can determine not just that a sentence is ungrammatical, but exactly how ungrammatical people think the sentence is [2]. The process of magnitude estimation works by first showing to the evaluator a piece of data and having them rate the data by a certain criteria. Then more pieces of data are looked at and the evaluator rates them based on their original rating [2]. So, for our evaluation we would first play an established musical song (in MIDI so people aren't rating it high because it is played with real instruments) and have the evaluator rate how musical the piece is based on how they understand music. Then other songs are played and the evaluator is asked to rate these songs compared to their original rating. These songs will be randomly generated songs in our system, and songs generated by rules in our system. The desired output is for the songs we generate to score higher than the random tracks while approaching the score of established songs. On top of this evaluation, we will also have an expert listen to the musical piece and review it. This will reveal what the musical pieces are missing, what is interesting about their construction, what musical aspects they feel are represented well, etc. Both of these evaluations together should give a total picture of whether or not our system produces a musical output. We created this experiment using the Wextor web-experiment service; we used a song list of 3 songs each of 4 rules: Life, Antilife, Highlife, Seeds, 3 random songs made with our system, and 3 songs which are MIDI songs created by humans. The base song is a MIDI version of Row Row Row Your Boat. There is a Turing Test element to this experiment, meaning that the participants are told that the tracks they are listening to were created by children after using a music-training software.

4.2 Results

Rule Type	Avg. Ranking	Avg. Original
Life	11.05	14
AntiLife	9.8	14
HighLife	13.056	14
Seeds	12.833	14
Real Song	19.789	14
Random	10.4762	14

Overall, our results showed no conclusive evidence of performing better than real MIDI tracks. However, Seeds and Highlife did perform significantly better than the random tracks made by our system. This implies that there are moments of musical clarity within these tracks even if it doesn't result in a "song" the way Row Row Row Your Boat is

a song. One of the major weird effects of our current CA translator is that it results in long pauses, that from the comments I have personally recieved sound alien to the user. Another issue was the constant short duration of the notes played. The duration layer tended to stay in random patterns due to the limitations of the CyclicCA rule and this caused patterns that would normally produce longer notes to be chopped up into sixteenth notes. While these results are disappointing, I believe they are promising in some regards. The songs produced by our system do have moments of musical clarity, but they are fleeting and overall their impact is lessened by the strangeness of the rest of the song. It is of note that the real songs were always rated much higher than the rest of the songs presented; this implies that the change between these songs was very noticable to the participants.

5 Conclusions/Future Work

From the results of our experiment, my conclusions are that human-composed music is incredibly difficult to replicate, even with a system that has many musical qualities. The fact that human-composed music was so easily pointed out by the participants shows that there is still a large gap between the songs created by our system and songs made by humans. Two qualities were definitely missing in our system that may have attributed to this. For one, most of the midi tracks presented had multiple instruments which might have made a more cohesive song. An instrumentation layer could be implemented in our system in a similar manner to the scale and duration layers. The next attribute that our songs were missing was repetition. With Row Row Row Your Boat, each verse has a definite beginning and end that you can follow, and these parts repeat. Our songs lacked any sort of structure, meaning that it is one stream of consciousness of notes. Future editions of this system would include song structure as an element. I have a theory that one problem of our system is that it may have been too open-ended, because we were trying to provide a way for the computer to produce all the sounds a human can produce. However, this provides more and more opportunity for the system to produce mistakes and strange behavior like the long pauses discussed above. Two options for dealing with this could be to confine the system a little more; make the shortest note an eighth note, or use only penatonic scales for example, so there is less room for errror. Another option would be to implement a kind of machine learning to determine what patterns of CAs are "good" for musical output and which ones are strange. I believe that this system could be improved to make better sounds, but I'm now more confident working on this project that there might be something very special about how we think of creative activities like composing music.

References

[1] J. R. Alvira". "tutorials: teoria.com". http://www.teoria.com/tutorials/index.php, 2011.

[2] E. G. Bard, D. Robertson, and A. Sorace.

[3] E. Bilotta, P. Pantano, and V. Talarico. Music generation through cellular automata: How to give life to strange creatures. In *Proceedings of Generative Art GA 2000*, 2000.

[4] S. M. Burke. Midi perl. http://interglacial.com/ sburke/midi-perl/, 2012.

[5] J. Chareyron. Digital synthesis of self-modifying waveforms by means of linear automata. *Computer Music Journal*, (4):25–41.

[6] E. R. Miranda and A. Kirke. Game of life music. In *Game of Life Cellular Automata*, chapter 24, pages 489–501. Springer, 2010.

[7] W. Research. Wolfram tones. tones.wolfram.com, 2005.

[8] J. Serquera and E. R. Miranda. Cellular automata sound synthesis with an extended version of the multitype voter model. In *Audio Engineering Society Convention 128*, 5 2010.

[9] M. Solomos. Cellular automata in xenakis' music. theory and practice. In *International Symposium Iannis Xenakis. Conference Proceedings*, pages 120–138, 2005.

[10] A. Trevorrow and T. Rokicki. Wolfram tones. http://golly.sourceforge.net/, 2012.

[11] M. Wojtowicz". "cellular automata rules lexicon, family: Cyclic ca". *http* : $//psoup.math.wisc.edu/mcell/rullex_cycl.html$, 2001.

[12] S. Wolfram. *Cellular Automata and Complexity.* Addison-Wesley, 1994.

[13] M. H. Yamaguchi. An extensible tool for automated music generation. Master's thesis, Lafayette College, 2011.