Programming a computer to play the 12-bar blues.

Paul Morris

BSc (Hons) in Computer Science

2005

PROGRAMMING A COMPUTER TO PLAY THE 12-BAR BLUES
submitted by Paul Morris

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

There have been many attempts in the past of varying success, to try and write programmes that compose music. In this project chaotic systems are used to compose music in the style of the 12-bar blues. Chaotic systems are used to research mapping styles, the programs sense of rhythm, harmonies, and integration of the two parts of music. Song structure is also identified as being of particular interest in this research field and two methods of creating song structure are investigated and shown to be valid. Music is produced consisting of a melody and of a bass line, and is tested on a sample of people and found to be of respectable compositional quality. It is concluded that chaotic systems are a viable compositional tool, with interesting properties that can be exploited for the composition process.

Acknowledgements

Psalm 121:2 "My help comes from the Lord, the Maker of heaven and of earth" - Thankyou. I would also like to thank my wife Hazel, for all her love and support. Many thanks also go to my supervisor, Professor John Fitch, for his help and guidance.

Contents

Chapter 1

Introduction

The aim of this project is to construct a program to compose music in the style of the blues, with an authentic blues sound.

The blues is a style of music that can take many forms, but originally artists used it to tell emotion filled tales of hardship. Through time far more upbeat and uplifting styles have evolved along side the original way of playing.

The 12-bar blues is normally played in 4/4, and is commonly agreed to take the chord progression of the form I,I,I,I,IV,IV,I,I,V,V,I,I, or I,I,I,I,IV,IV,I,I,V,IV,I,I. I prefer the second of the two and so will be using this chord progression.

The 12-bar blues progression in C then is:

C-C-C-C-F-F-C-C-G-F-C-C

The blues scale in relation to a major scale is as follows:

1 - 3b - 4 - 5b - 5 - 7b - 1

So the C blues scale is:

C - Eb - F - Gb - G - Bb - C

With this project we will focus on blues improvisation. This is often purely instrumental and is very closely linked to Jazz, improvisation being a defining property of Jazz. The 12-bar blues form will be used as a base for the improvisation, and the blues scale to form the melody and bass line. Any composition will include at least 2 instruments, one playing a bass line or another form of backing, with another playing a melody over the top.

Due to the nature of improvisation, this does give this project a certain freedom, as improvised music, very bluntly is made up on the spot. A musician would tell you that their improvisation is based on certain patterns they have developed and so is not just a set of random notes strung together, improvisation is a personal way of playing, and no two musicians will play the same. Improvisation can focus around a similar theme/feel for that piece of music, and/or engage in

a call and response style of playing. The style of music depends greatly on the mood of the musician at the time. The challenge of this project will be to try and take what in real life is a very personal way of playing music, often based on emotion and feeling, and try to somehow generate that music from a machine.

The blues is often played as part of a band, and is not only played by solo artists. Musicians play off and around one another, maybe taking it in turns to improvise a melody while the rest back them. This project could extend to include multiple instruments in a composition, which play with each other.

Another extension to the project is to add structure to the music. This could mean the addition of verses and a repeated chorus. This adds further challenge as it would move away from straight improvisation, imposing the need to flow smoothly from one part of the song to another. Alternatively an overall structure could be added to the improvisation, with it seeming to keep a theme, rather than wandering aimlessly.

The flat notes in the blues scale are said to be the "blue" notes. A true blue note does not actually appear on a normal piano. On the C blues scale the first blue note, Eb, would as a true blue note be somewhere between Eb and E. Musicians on a piano would try to re-create this note by slurring the two notes Eb and E (playing Eb closely followed by E). As this is a defining sound in the blues, it will need to be a consideration in the project, as to how the sound is created.

There are several methods for the actual generation of the music. The first, and slightly trivial method would be to randomly select notes of the blues scale, of random duration, to construct a melody. This would not be of any real value, and we need to consider more complex methods of algorithmic composition.

There has been in the past varying attempts to program computers to compose music, two examples are that of a mathematical statistical approach and one based on chaotic systems.

Iannis Xenakis used computers to generate music using a stochastic process [16]. This used statistical and probabilistic methods to deduce music from a list of note densities and probabilistic weights, which used a random number generator for specific decisions.

Non-linear dynamical equations that are iterated can be used to generate music [1]. The system of equations is put through an iteration, a solution calculated, and that solution is input back into the system for the next iteration. There are three categories of behaviour of these systems; constant - where all points are the same, oscillatory - where all points stay within a set of points, and chaotic - where all the points are seemingly random, not returning to the same point twice, but often back to similar ones [7]. This behaviour has the most musical interest, as the data produced has quite a high similarity with its past, but will not create exactly the same thing again, which has the possibility to create interesting music.

This project shall research methods of algorithmic composition and then select the one deemed most suitable to compose the blues. That method shall then be explored further and the knowledge gained will be used to complete the design process. Next the composer shall be implemented

and experiments shall be carried out to try and develop the best sound. The composer, once written, will be evaluated by a sample of people to ascertain how good the music produced is.

Chapter 2

Literature Survey

2.1 Introduction

The purpose of this review is to research the current methods and technology already developed for algorithmic composition. It is important to get a broad understanding, as well as specific research on composition in the style of the blues, so lessons may be learnt from others and adapted for use in this project.

2.2 Compositional Methods

The early programs written to compose music were based on random number generators. Possibly the most famous of these early compositions was the "Illiac Suite for String Quartet" by Lejaren Hiller and Leonard Isaacson. These early pieces used the Markov process, which works by using past events to predict future ones. This process was plagued by either having too much or too little knowledge of the past. If past knowledge was too small, "the result sounded as random as if the notes had simply been selected from an arbitrary set of weighted probabilities" ([14] p54). If past knowledge was too great "what was "predicted" was nothing more than a replication of the original data from which the probabilities were computed" ([14] p.54).

Many researchers have attempted to improve this situation with their own methods, which range from probabilities and heuristics, to neural networks and learning algorithms, genetic algorithms and also chaotic systems. All of these are of possible use for a blues composer.

2.2.1 Neural Networks

One particularly interesting article by Eck and Schmidhuber [9], was on generating the blues using neural networks. This paper proposed using a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) to add global structure to the music generated by the network. Previous work had been done to generate music using RNNs. Though these attempts had been partially successful, Mozer after his attempts wrote "While the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organization"(Mozer 1994, cited by Eck and Schmidhuber [9] p.1).

Eck and Schmidhuber used a form of 12-bar blues as a basis for all of the music generated. The pentatonic scale was used to build the melody and a limit of 8 notes per bar was introduced. Training melodies were written by Eck to fit musically with each chord.

Two experiments were carried out. First the network was only presented with the chord structure, to ensure that the LSTM was not using structure in the melody to predict the global chord structure. Once the network had learned to generate one chord cycle, it could then generate any number of cycles. Secondly, both chords and melody were presented to the network. After learning, the network was given a seed note, or a series of seed notes, and then allowed to improvise. The resulting music is a reasonable approximation of blues style music. The network learned the chord structure and constrained the melody to that structure.

With the author's own admission, the experimental set up made the compositional task very simple. After listening to several outputs from this system, it was difficult to tell the outputs apart as they were all very similar. Though an element of structure has been added to the music, there is nothing defining about any piece produced. Also the notes are constrained to a single octave and no chords are used within the melody. That said, the work described could provide a very good starting point for this project, offering LSTM RNNs as a viable way of implementing my project. For this project it could be taken onto a more complex model, or used to measure up against if a different model is used.

Another interesting paper which proposed the use of hierarchical RNNs for learning musical structure was one by Burr [3]. They too identified the problem of adding structure to the music, and proposed the idea of using layers of interacting RNNs to form a hierarchy that would give the music structure. "A hierarchical recurrent network consists of a collection of subnetworks each of which operates on a different time scale" ([3] p.218). The time scales proposed in this paper are every beat, every 3 beats, 12 beats and 24 beats, but the authors only implemented the first two levels. The use of networks on differing granularities could capture local and global regularities within the music, therefore these networks could be used to add an overall structure to the music.

The authors used this network to generate waltzes and were able to demonstrate its ability to organize melodic patterns in a structural manner. Unfortunately it suffered from including whole segments of melody from the training data. An extension of this work would be to add the higher level subnetworks to the network to try to add greater structure to the music. It would also be good to eliminate the problem of including segments of training data in the output.

Both of these works are aimed more at improving the overall structure of the music rather than struggling with the problem of generating music in the first place. If a contribution to musical composition using neural networks were to be made in this project, we would need to address this problem. Also chords and/or an inter-related bass line and melody could be introduced, something which neither of these works address. One potential problem of using neural networks is that the quality of the output will be highly dependent on the training data. Further research would be required to see what the best ways of training these networks are.

2.2.2 Genetic Algorithms

Genetic algorithms appear to be a popular approach to musical composition. Very basically, genetic algorithms work by producing small sections of data called genomes (in this case a short

series of notes), which are then evaluated by some function. The best ones are selected and then combined together to produce a new generation. One approach by Gibson and Byrne [10] incorporates genetic algorithms with neural networks by using a neural network as the evaluating function. This implementation was built to produce music similar to traditional hymns. It simplified the composition process by restricting the composition to C major and by producing harmonies that are restricted to only three chords. Composition is broken down into three parts; rhythm, primary melody and harmonisation, and composed in that order. This method also has the interesting feature that two separate neural networks are used to evaluate the melody, one focusing on intervals between pitches and another on the overall structure.

The prototype produced for this system could only produce four bars of music. It is also highly dependent on the users input. This method is obviously in the early stages of development but could be expanded to serve as a possible method for our music composition.

Another genetic algorithm for music composition was written by Marques et al [12] (the was for the composer to be good enough to be used to create ambient music). The fitness function in this system is different from the above as it just uses some simple rules. There are three parts to the evaluation; harmony - which evaluates the intervals between notes played at the same time, tone - which evaluates how suitable a note is for the chosen tone and scale, and finally melody - which evaluates intervals between consecutive notes. This is a fairly complex system compared to others, allowing 8 octaves of notes, pauses, chords and multiple voices to make up the music generated. The results of the generation are said to sound pleasant to the authors of the work.

A slightly different approach to genetic algorithms is taken by Puente et al [6], who composed music using grammatical evolution. This differs to normal genetic algorithms by grammatically translating genomes, (or genotypes as they are called in this article) to phenotypes to then evaluate them. Operations to create each new generation are still applied to the genotype. The phenotype is what is used at the end of the process to define the melody. There are 83 different notes with 7 different lengths. The program was written to consider pitch, differences between consecutive notes and lengths of notes. The authors intend to develop the program to compose music in the style of well-known composers, which means it should be flexible enough to be adapted to play the blues.

Though these methods all generate music, they appear to be a step behind neural networks in the sense that most have no real mention of adding structure to music, but are concerned with generation of something that is pleasing to the ear. They also fail to mention the question of; as these programme's fitness functions all look at very small segments of code, how is any kind of global structure and interest going to be introduced at that low level of granularity? If this blues composer were to use one of these methods, this is question could also be answered. That said Gibson, and Byrne [10] and Marques et al [12], do use harmonies that the two neural networks above did not attempt to implement.

2.2.3 Chaotic Systems

Chaotic systems are dynamic equations. They are made using an iterated equation, and each iteration is called an orbit. Every output of the equation is fed back in as the input for the next orbit. Though this is simple enough, these systems can give very interesting data for us in algorithmic composition. Bidlack [1] carried out some experiments using these systems. He shows how systems in multi-dimensions can be used to produce, for example, values for pitch,

amplitude, and length if using a 3-dimensional system. He also produces figures showing regular patterns produced that could be used for a bass line, as well as more random systems that could be used to produce a melody. Bidlack points out that the start values given to these systems will vary the output dramatically, with some inputs causing the systems to quickly go to infinity. Bidlack warns that precision in calculating values for these systems is very important. Vastly different outcomes are inevitable with a system computed with single precision numbers, compared to one with double precision. He also points out that systems compiled and run on different processors are likely to give different results. These problems aside, Bidlack shows chaotic systems to be a very definite possibility for use in algorithmic composition.

Leach and Fitch [11] are others to use chaotic systems to generate music. The aim of this paper was to use dynamic equations to model and produce a musical interpretation of events in nature. One comment when speaking about chaotic systems says that "each iterated point is a result of a precise formula which contains no random elements, the result is a highly structured orbit which has elements of near repetition within it everywhere" ([11] p.14). This is very interesting, because as a result of this, these systems could have an in-built overall structure to them. This was something those using neural networks were searching for. Using these systems we effectively get a lot of structure for free! The authors also introduce the idea of using the output of one system to affect the other. This idea is also interesting as it could be a possible means of adding a higher level of granularity to the structure of the music. For example, if you imagine a sin wave, you could input into the system generating the wave a parameter. That parameter could simply add a number to the systems output. If controlled this parameter could raise and lower the sin wave. It seems to that this idea could be applied to chaotic systems (or maybe others) as a way of controlling the structure of the music further.

The authors use the data from the systems slightly differently to Bidlack, using them to form a tree structure instead of mapping them straight to a musical form. Composition in this system is controlled to quite an extent by the user, though a lot of the ideas behind the process could be used toward the creation of a compositional system.

Although the use of chaotic systems in music composition does not appear to be wide spread, they do appear to offer a lot of good prospects for use in this area. Because of this, it could be a good area to contribute to current research by using chaotic systems to produce a specific musical style.

2.2.4 Others

Moorer [13] did a study on music composition using heuristics. He used an approach of giving the music structure by creating what he calls major and minor groups. These groups are chosen using probabilities. For each section he decides whether to repeat a group or create a new one. Each major group is made of two minor groups, so a composition could take the form ABAC, which when broken into minor groups has the form abacabbc. This overall structure chosen, probabilities are used to generate rhythm, chords and melody, in that order, for each minor group. The order is not accidental, and is used so that the chords fit around the rhythm, and in turn the melody fits around the chords and rhythm. In conclusion the author says the pieces generated are "strangely alien sounding" ([13] p.112). The reason offered for this is that the method he used to structure and generate the music is not the one that humans use. However this is not proven and his method is an interesting idea.

Dubnov et al [8] did some research "to capture some of the regularity apparent in the composition process by using statistical and information-theoretic tools to analyse musical pieces" ([8] p.73). Their method attempts to automatically discover patterns or phrases in the music and then assign stochastic generation rules to re-create them. They use two methods, the first of which is incremental parsing. This method scans a musical input from left to right and builds a dictionary of motifs in the music. It adds to the dictionary every new phrase that differs by a single last character from the longest match that already exists. The other method uses prediction suffix trees. They work in a similar way to incremental parsing but add to the dictionary only motifs that occur a significant number of times in the music, and are meaningful to predicting the future. Experiments show that incremental parsing captured the most musically meaningful motifs, but would tend to copy the training data. Prediction suffix trees are much more inventive and add good interest to the music, but have a tendency to include "false notes" ([8] p.79).

Cope [5] has completed some rather successful research into musical intelligence. His program, when given a database of music in a certain style, analyses the data to then produce a piece of original music from it. He has had particular success in producing classical music and in his own words has "delighted, angered, provoked and terrified" ([5] p.25) audiences who have heard it. Overall his program worked by analysing a database of music, breaking it down and then recombining it in a new way to create original music. First the music is analysed to find recombination rules, which are rules or instructions that govern the way in which notes are combined together to form music (initial work relying solely on rules, provided correct but lifeless music). He continued the analysis by breaking the music into beats and saving them as objects, storing the pitch, as well as what pitch the beat moved onto in the next beat. To add structure to the music he added context sensitive data to what was stored for each beat, like its position within a phrase of music. He also analysed the music for what he called signatures, which are repeating patterns of notes that give the piece of music its style. It was also required to store information on a section of music's character, which he defines as the rhythmic configuration and number of pitches, so that he could combine bits of similar character to maintain continuity.

This approach looks very interesting as the program itself has no internally written rules about the style of the piece of music it is writing, but through analysis of music libraries was able to very accurately re-create a specific style. The success of this method makes it stand out from the others mentioned, but unfortunately Cope only seems to have tested the system on classical music, so it remains to be seen how successful his program is at differing styles. A good continuation of this work would be to see if it could be adapted to something very different to classical music such as the blues, but this would require a lot of time learning in detail about his system before trying to use it, which is something that is not possible.

2.3 Evaluation and Method Selection

Most of the methods researched have overcome the problems faced by early compositional methods of either random music or repetition of training data. A lot of these methods seem to be wrestling with the problem of adding structure to the music. Of the nine methods mentioned, five of them include this problem in their work. As mentioned above, interestingly the papers read on genetic algorithms do not seem to have yet examined this problem.

Neural networks and other forms of machine learning appear to be a popular method for algorithmic composition. There has already been some success in developing them to play the

blues using LSTM RNNs by Eck and Schmidhuber [9]. Other projects include BoB, which is an agent written by Thom [15] that after some initial background training, will listen to a musician playing the blues and then play along with them in real time, depending on the musicians style. GenJam (Biles 1994, cited by Burton and Vladimrova [4]) is another Jazz improvisation program. GenJam uses humans as the evaluator to train the program, and skips the difficulty of writing a good evaluation function for its genetic algorithm. The author notes this is very time consuming and there is not really time to spend hours training a system in this project. Machine learning is also attractive as it is easy to change the style of music produced by changing the training the program receives. With the success already achieved in algorithmic composition using neural networks, it would seem to be a good choice of compositional method for my project.

That said, chaotic systems are intended to be used for this compositional system. They have good properties for use in this project, firstly the mechanics of implementing a dynamical system in a programming language is relatively simple. After all, these systems are just an iterated equation, which means that it should be relatively simple to get going. This also means that we should be able to try out several systems of equations in the course of the project with relative ease, by just changing the part of the code that generates each orbit. Furthermore the properties of these equations that could potentially add structure to the music generated are very interesting. These systems can also be used to give unique but regular output, which is almost begging to be interpreted as a bass line, as well as more chaotic outputs which fit well with the idea of improvisation. No research has been found that attempts to use these systems of equations to generate blues music, which makes this a valid area for this research to focus on.

2.4 A Closer Look at Chaotic Systems

Devany [7] tells us dynamic systems have three overall types of behaviour. The first is fixed point, where each orbit of this system produces the same output. The next is a periodic orbit or a cycle, where each orbit will lie on a set of repeating points. This last one is chaotic, where the orbits seem to wander randomly. The behaviour we are most interested in is the chaotic orbits. Bidlack [1] breaks down the chaotic behaviour into two further categories, dissipative and conservative. Dissipative systems correspond to phenomena in which friction takes place, whilst conservative systems are those in which energy is conserved. More specifically, the phase space (the area or volume in which the dynamics of the system takes place) shrinks with time in a dissipative system. After an initial phase (called the transition phase), the system will shrink towards an attractor, which itself can be a single-point, periodic, or a chaotic attractor. The phase space of a conservative system remains constant. Bidlack notes that chaotic systems are either formulated as iterated maps in one or two dimensions (which are notated mathematically with difference equations), or as continuous flows in three or more dimensions (which are notated mathematically as differential equations).

After completing his research, Bidlack makes the following observations on the types of chaotic systems; "A dissipative system, once it has settled onto its attractor, maintains a relatively constant behaviour....Conservative sequences, on the other hand, can exhibit much less internal consistency, and are marked by sudden changes in texture and range of values" ([1] p.46) This implies that dissipative systems stand as a better prospect for producing a good bass line and conservative systems would be better to produce the improvised melody. If we were to use multiple lead instruments in a piece, dissipative systems could be used to produce music for instruments to play in the background while another instrument takes the lead.

Bidlack is also kind enough to provide the equations for the systems he used and example C code to go with them. These would provide an excellent starting place for this research as Bidlack also gives some insight into how these particular systems work. They are:

- The Henon Map: a dissipative chaotic system in two dimensions.

$$x_{n+1} = y_n + 1 - Ax_n^2$$

$$Y_{n+1} = Bx_{n'}$$

- The Standard Map: a conservative two-dimensional map, whose phase space is the surface the unit torus.

$$I_{n+1} = I_n + K sin\Theta_{n'}$$

$$\Theta_{n+1} = \Theta_n + I_{n+1'}$$

- The Lorenz System: a dissipative system in three dimensions. Where $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the first derivatives of x y and z with respect to time.

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = Rx - y - xz$$

$$\dot{z} = xy - Bz$$

- The Henon-Heiles System: a conservative system in four dimensions. Where $\ddot{x}$ and $\ddot{y}$ are the second derivatives of x and y with respect to time.

$$\ddot{x} = -x = 2xy$$

$$\ddot{y} = -y = x^2 + y^2$$

2.5 Development Environment

The aim of this project is to produce a program that outputs data that could be interpreted as blues music by some means, not to spend much, if any time writing code to cause a computer to make sounds. With this in mind a program or a language that will take an input of data and convert it into sound is required.

2.5.1 Csound

This is a system for producing and manipulating sound and music. It has two languages, the orchestra language, and the score language. The orchestra is used to specify the instruments playing in a piece of music, whilst the score is used to specify when and with what properties (i.e. pitch, amplitude, duration) each instrument will play [2]. This seems perfect for this system, which could output a Csound score file, which could then be put through the Csound compiler to produce the final output. For the orchestra file a few already developed instruments could be used.

There are resources on the Internet for Csound and there is the Csound textbook on how to use this system. There is also the added bonus of the project supervisor, John Fitch who is a bit of an expert on Csound, who could help if there are any difficult problems. These things make Csound the obvious choice to generate the music from this project.

2.5.2 The C Programming Language

The blues composer shall be programmed in C. This is mainly because it is a good powerful language that the author is already familiar with. There is no need to spend any time learning a whole new language. C is very widely used and should be quite adequate for this system.

2.6 Note Representation

One issue faced by many of the researchers in the papers read was note representation. Several of the research projects used a quite constrained set of notes. For example Eck and Schmidhuber [9] limited their selves to an octave and Gibson and Byrne [10] limit their selves to C Major. For this program it is not desired to be limited to a particularly small range of notes or a single key. Multiple durations of notes, a good pitch range over several octaves, as well as amplitude changes and changes between 3 blues scales are desired. The Csound score file basically consists of a note list that for each note, an instrument number, start time, duration, as well as other parameters for that instrument. These parameters often consist of amplitude and pitch and possibly a few others [2]. Csound allows floating point entries for note start time and duration, which will allow for quarter, or half notes, or whatever duration required. Also Csound can produce a wide range of pitch values. All are given as simple numeric values, similarly then notes in this system will be represented numerically to co-inside with Csound for simplicity of use.

2.7 Conclusion

In conclusion it is decided to use chaotic systems for the generation of music. This is due to their very promising structural properties and ease of implementation. It is also intended to program the composer in C and output the musical data into Csound format for compilation into a playable form.

Chapter 3

Requirements and Design

This section describes what the program needs to do, discusses some of the problems and includes some initial program design. As this is a piece of research the detail of how the program actually works is in the implementation section.

3.1 Output Form

There are two main ways in which this program could function:

1. To produce a program that when started will begin a blues improvisation and will continue to play until halted or for a set piece length. Each time the program is run the improvisation produced is different.

2. To produce 1 or more blues pieces that use an algorithmic method to produce the music. The parameters for the code will be fixed and chosen by the author, the output of the program will not change without any intervention.

If we consider the first option, in order to get a different improvisation each time, the starting values used by the equations in the program will have to be varied with each execution. It is not feasible for this project to generate different dynamic equations for use in each composition, though it may be possible to pick some from a selection of pre-programmed options.

The starting values an equation is given will greatly affect the way the system behaves. This system could show fixed point, cyclic, chaotic behaviour, or escape off to infinity. As part of my research for such a system, bounds would have to be found in which we can vary the starting data but still get the desired behaviour. Also it would need to be decided how to determine the starting values. This would most likely be a random decision.

Very small changes in dynamic systems can lead to quite different outputs. This could be either this is a good thing that a different output can easily be generated, or a small change may cause the system to no longer produce good music. The data produced by these equations is only a small part of the compositional process, if the mapping of the data remains constant then each production will still retain a similar feel to it.

The other big issue is that of either allowing the program to run indefinitely or producing a piece of a fixed length. If the piece was of indefinite extent and played as the program was running the music will have to be written in real time. This is difficult. Working out of real time would allows us to fill in different parts of the music in any order required. For example one part of the music could be produced, replicated, and then the other sections filled. Also one part to the music could be produced, like the bass line and then melody can be worked around that. Real time production would mean we would need to be simultaneously be producing all the parts of the music. This is likely to make it harder to fit the parts around each other, the only information available will be what has happened in the past.

If the program output were varied with each execution, fixed length improvisations would be produced as to avoid the difficulties of producing the music in real time.

The second option appears to be the easier of the two. It is also the way in which the program will work during development, as some sort of consistency in the data being produced will make it easier to analyse. This option also has more opportunity for the best sounding outcome, as it will be feasible to choose which starting values produce a good pattern. The possibility of creating a program with a varied output will be viewed as a possible extension, depending on the success of its development in earlier stages.

3.2 Representation of Notes

The decision as to how to store the notes generated is an important one. This will have a knock on effect throughout the program, especially as notes that have already been generated can be processed to help generate others. The details of each note will be stored in some kind of data structure, but the big question is how to store these data structures. There are two possibilities to consider:

1. To use an array with a space for every single possible time step in the piece, so each note is stored at the point it begins in the music.
2. To store the notes in some sort of linked list.

At first inspection option one appeared to be flawed from the start as it would require me to use arrays of ridiculous sizes to have space for every time step. But on closer inspection the size needed actually isn't that great. To allow notes of a 32th of a bar long, called a Demisemiquaver, would only require 384 cells for 12 bars. Using notes this short will give a high possible degree of complexity and also a reasonable array size.

This option also has a great advantage that sections of music are easily located. To copy the first 12 bars we can just take the first 12 bars pf cells and do so. It also means that notes will all appear in order they are played. This poses a great advantage over a linked list as there is no guarantee where in the list a section of music will be, or if the notes will be jumbled up or not. This will waste memory, as many time steps will not be used, but this is not of great concern.

Although the second option will be a far more efficient use of memory, it does make it harder to store the notes in an organised manner. Rules can be imposed to only add notes onto the list in time order and to keep sections of music strung together. Also this representation has no complication if more than one note is to be played at the same instant, unlike an array of notes. This said, the first option appears to be the simplest one and so shall be used this my program.

Now that it is decided notes shall be stored in an array, we need to consider what happens when more than one note needs to be played at the same instant. This could happen for one of two reasons, either notes from two or more parts are being played, or a single part is playing more than one note.

To overcome the problem of having more than one part, an obvious solution is to have a separate array for each part. This means that it will always be clear which part is playing a particular note, as all the parts are kept separately. Otherwise all the parts would be jumbled together which could cause complications if ever only notes from a single part are wanted.

To overcome the problem of a part playing more than one note, the structure stored in the array will point to another note structure, this gives the option of having more than one note occupy a cell in an array, allowing chords.

For ease of use each array will be kept inside a data structure, a pointer to another note array will be added to it, which is convenient if we want to carry out a process on every note in the program, like outputting the data. This is because it easily allows the program to traverse through all of the parts stored.

So in summary to represent the notes of a piece there is

- A structure which holds an array for each part/instrument, which has 32 cells per bar. It also holds a link to another note array. This will be called the *part* structure.
- A structure to hold each note, with a pointer to link notes together to form chords. This will be called the *note* structure.

3.2.1 Data Structures

The following is the data structure for notes:

```
typedef struct NOTE
{
 int instrument;
 double start;
 double duration;
 double pitch;
 double amplitude;
 struct NOTE* chord;
} NOTE;
```

The essential data for each note is stored to output sufficient information to Csound. The chord item is the pointer described above to allow a part to play more than one note at a given instant.

The following is the data structure for parts:

```
typedef struct PART
{
 NOTE *partArray[SONG_LENGTH];
 struct *PART next;
} PART;
```

The items stored are as described above, an array to store a note at every time step as well as a link to another part structure.

3.3 Scales and Octaves

To generate notes on a scale rather than an arbitrary frequency we have to follow the Csound way of representing them. 1.00 is C in octave one, 1.01 is Db, 1.02 is D, 1.03 is Eb, up to 1.11 is B and then 2.00 is C in octave 2 etc. To successfully map to notes to scales it was decided to store in arrays the notes in each blues scale:

```
double cScale[6] = {0, 0.03, 0.05, 0.06, 0.07, 0.10};
double fScale[6] = {0.05, 0.08, 0.10, 0.11, 1, 1.03};
double gScale[6] = {0.07, 0.10, 1, 1.01, 1.02, 1.05};
double *currentScale;
double octave;
```

Two further variables are included here, a pointer named currentScale and a double octave. currentScale will be used to point to the start of the array of the current scale in use. As the 12-bar blues format is being used, these will be one of 3 scales: C, F and G in this case. This will allow access to notes in every scale through the same variable, which can be switched to the correct scale at the correct time in the 12-bar progression. The octave pointer is used to set which octave the notes will be played in. In the array they all notes are in octave 0, by adding on to their value the number stored in octave, it will allow the octave of the note to be easily adjusted.

Blues musicians do not restrict their selves merely to the pentatonic scale, which is what has been modelled here. Although a musician's music may be based around these scales other notes are not ruled out. So this representation is slightly restrictive but if these notes are used it is more likely to produce a blues sound. An extension from this point would be to include other notes from outside the pentatonic scale without losing the blues sound of the music.

3.4 Dynamic Equations Implementation

The equations their selves are pretty simple and just require the arithmetic typing in correctly. The issues to be faced are how to collect the data generated by the equations and how to supply starting values.

To collect the data it was decided to use a global variable for each dimension of the equation's phase space. A call to the equation's function would execute one orbit and the global variables updated so that the data is ready to be collected. There is the possibility of outputting the data in batches to an array, but the above method is the simplest and so most preferable.

Although the issue of providing starting values seems trivial, it is worth commenting that these values will be passed to the functions as parameters. It was decided to do this rather than hard coding them at the top of the function to fixed values as it allows them to be changed. They are then also easily passed to the function if it is decided to vary the starting values for different program outputs each time it is run.

3.5 The Note Creation Process

This is the most complex and difficult part of the project. To produce data from dynamic equations is relatively simple, it is the mapping of this data to notes that will determine what that data sounds like. The methods chosen will have a huge effect on the sound generated. An extreme example of a simple mapping is that the data is multiplied by 0 and then added to the current octave. Now any data produced will simply map to C in the current octave being played, which is not particularly interesting. More interestingly the data could be mapped to notes via ranges, so that data within a certain range will be mapped to a note, or the next data item could be used to offset from the previous note played.

The method used to map the data is subject to a lot of experimentation. The mapping is what tells the computer how to interpret the data in a way that sounds good. There are two main but rather different parts that will be implemented, the bass line and the melody. Both are very different and require separate consideration.

3.5.1 The Bass Line

The bass line is usually much simpler than the melody. It is not the main interest to the listener and is just there to support the melody in the background. The bass line can take on a complex form and wander around the notes and can be improvised in the same way the melody is, but often just takes on a repeating form. Initially the program will implement the latter, but if there is time it could be extended to the former.

As mentioned earlier Bidlack [1] notes that there are two types of dynamic systems, dissipative and conservative. Furthermore he notes that a dissipative system after a transition phase will settle toward an attractor and exhibit a fairly constant behaviour. This type of behaviour is ideal to be mapped to a bass line as we are trying to create a bass line that repeats or behaves in a constant manner.

The biggest difficulty will be fitting the repeating structure of the data to the structure of the music (4 beats in a bar and 12 bars). If the data were to repeat over 5 beats for example, it would not sound very good.

So the main tasks for the bass line are:

- To implement a dissipative system to produce data to be mapped.
- To map those values to notes in the correct scale.
- To ensure the repeating structure of the data fits the music.

Implementing the dissipative system should not be hard, but different systems will behave differently and so a certain amount of experimentation as to which system to use will be beneficial. Good starting places are the Henon Map and the Lorenz System detailed by Bidlack, as he has also provided C code for their implementation. The mapping of the data to pitch values is also open to experimentation. As the bass line is quite simple, a straight forward mapping of ranges of data values to notes may be all that is required.

The task is complicated by the need to fit the data to the bar structure, as the mapping to pitches will have to be interlinked with this. One option may be to ensure that on significant beats (the first beat of a bar), the pitch is forced to either the root note, or the top note (0.00 or 1.00 pitch values). But this, instead of fitting the data, would rather be masking the fact that it does not fit. Another very much more complicated option would be to analyse the data for the repeating structure and then scale that structure to the 4/4 rhythm.

3.5.2 The Melody

The melody is going to be the most complex part of the program. It is of main interest of the listener and so is the most important part to sound good. The complexity given to the melody will depend on the progress made and the difficulties that occur along the way, it is hard at this stage to know what is going to be achievable. A good blues melody would be expected to have notes of varying length, to move over several octaves, to have repeated structures, or riffs, to include chords, the list goes on.

Conservative systems for the creation of the melody are intended to be used, as noted by Bidlack [1], they do not have the same repetitive quality as dissipative systems and have changes in texture and range of values, which should provide a good interesting set of data for interpretation.

The main tasks and areas of difficulty are:

- To implement a conservative system to produce data to be mapped.
- To map from the data to notes in the correct scale.
- To allow repeated note structures, blues style riffs, or repeated segments of the piece.
- To allow a variety of length of notes and octaves
- To build chords from the data.
- To fit the notes around the already existing bass line.
- To ensure the notes flow from scale to scale within the 12-bar structure.
- To produce a melody with a blues sound.
- To maintain an overall structure in the melody.

Experimentation will start off by using the conservative systems detailed by Bidlack [1], namely the Standard Map and the Henon-Heiles System. This part of the project will need to be very experimental as to what sounds good. The simpler parts will be implemented first, such as a mapping of the data to the notes with a variety of lengths and octaves. But chords and repeated note structures or riffs will require specific and individual experimentation.

Chords also could be part of the composition and could be implemented to be more likely to appear or important beats, or a pattern of chord intervals could be developed. The chords can be constructed around a bass note that would be played at that point in the melody. Though the word chord was used here, we are not have to limited to three notes per chord, two or three notes played at the same time would both be a good addition to the program. A major chord, constructed from a bass note, will have one note played 4 semitones above the chord's bass note,

and another 7 semitones above the bass note. These notes can be moved up and down octaves and played in any order to create inversions of the same chord (e.g. move the top note down an octave). This method of creating chords is crude and takes no account of the scale being played which is a disadvantage. The notes playable in a chord could be restricted to those in the current blues scale as well as the equivalent usual scale, and this should keep the chords in tune.

When blues musicians play instruments such as the piano where there is no way to play a note in-between two notes, to try and capture a true blues note they will slur the keys. This means, for example, playing in C, the musician will play Eb immediately followed by E, creating the effect of a single note being played in between the two. This could be easily implemented and would be a very simple addition to the code that has a distinctive blues sound.

It should also be relatively simple to repeat structures of notes or segments note for note. For example, if once C,D,C was played, G,A,G could then be repeated, which has the same structure as before but is played at a different pitch. This would also be a simple addition, and is something used by musicians, but it may be difficult to decide which sections sound good to capture and repeat, but experimentation may show that an almost random selection has good effect.

We must also ensure a good flow from scale to scale. It should not be noticeable by a sudden change in pitch that is out of place with the direction of the music. A solution could be to intentionally draw the notes toward a good exchange point, where a note in one scale can neatly follow on in another when a change is approaching.

Further additions will be added as progress is made. The direction taken is largely going to be down to how the implementation is going and what can be done to make things sound better in the time available.

3.6 Project Requirements

Here are the overall requirements of the project. This is a research project so exact details are intentionally not included and things are left more open than in a traditional software engineering document. Requirements intended to be definitely included are in normal font, where optional extensions are included in italic.

1. Research must be carried out as to possible ways of using chaotic systems as a compositional method.
   (a) This must be in the style of the 12-bar blues.
   (b) The complexity of the program should be taken as far as possible in the time available.
2. The program must output a single blues piece dependent on initial variables.
3. The program must generate a bass line in a blues style.
   (a) This should be a simple repetitive bass line suitable as a backing.
   (b) It should play notes within the correct scale and change scales with the 12-bar structure.
   (c) The repeating structure should fit the 4 beats per bar structure of the music.

   (d) *The bass line should be made more complex to form a wandering bass line*

4. The program must generate a melody in a blues style.
   (a) To implement a conservative system to produce data to be mapped.
   (b) It should be of a relatively high complexity and hold the listeners interest.
   (c) It should play notes within the correct scale and move smoothly between scale changes.
   (d) It should play notes of a variety of lengths and over a variety of octaves.
   (e) It should combine well with the bass line.
   (f) It should maintain an overall structure.
   (g) *Repeated note structures and segments should be included in the composition.*
   (h) *It should play chords as part of the composition.*
5. *The program should be amended to produce a different output with each execution.*
6. New features should be added as their need becomes apparent in the development process.

Chapter 4

Program Development

In this section details are provided on what was actually done, things that were tried and failed, the decisions made and the reasons for them.

4.1 Initial Groundwork

Before beginning implementation of the actual composer the data structures and methods for controlling them had to be created. Implementation of data structures and initialisation methods went without a problem or improvement on previous planning. Work also had to be done to control the way the code moved through the scales used in the bar structure, which essentially consisted of counting bars and changing a pointer to the correct scale at appropriate times. Methods were also created to provide suitable output to Csound. This work was essentially house keeping which needed to be done before the actual work of composition, and so are not covered in great detail.

4.2 Bass Line Experimentation with the Henon Map

Once the basic program structure was finished work started to develop the bass line. Partly because it is the simpler part of the program, and also because the melody can be partly dependent on the bass line. This also gave experience of using the chaotic systems before tackling the more difficult problem of the melody.

4.2.1 Initial Tests

It was decided to implement the Henon Map because it is a dissipative system and is also very simple. This is a two dimensional equation, which gives the opportunity for a pitch value and a duration value. One dimension is simply 0.3 of the other, and as the bass line is simple anyway, the duration and loudness of the notes will all stay the same and only the pitch will vary. Initial implementation of the Henon Map provided interesting and encouraging results. The first outputs just contained the raw data mapped to pitch, as well as constant added to the pitch to raise the octave to a suitable level. The notes produced had a clear structure present and were certainly not random. This was good as it showed that the research was valid and we had some good raw material to work with to generate music. You can listen to this output in file Out1. All outputs can be found on the attached CD.

4.2.2 Mapping of the Pentatonic Scale

The next step was to implement a simple mapping of the data to the pentatonic scale. First an array was used to store enough values from the Henon Map to supply notes for the whole piece of music. The range of these values was then calculated and then split evenly into 7 sections, with each section corresponding to one note in an octave of notes. Now each value in the array can be tested to see where in the range of values it appears, and so be mapped to a note. The initial output from this, though interesting, does not sound much like a blues bass line. Although it does have a repetition to it, it does not really fit around the bar structure, and just sounds a bit haphazard. The mapping needs to be improved to take into account the bar structure, and it also needs a much better blues sound. Different start values for the map resulted in no real improvement. This output is file Out2.

Before adding in a feature to watch the bar structure, it was decided to try an idea for a more complex mapping that could produce better results. Instead of using the data ranges to map straight to notes, the data can be used to jump an offset from the previous note played. A bass line will often walk up and down a scale steadily and not make major jumps, so the data produced by the Henon Map was used to determine the size and direction of each step. The code from above was re-used, but instead of mapping a range straight to a note, it is used to jump a number of spaces up or down from the last note played. Initial attempts failed with the pitch value escaping very quickly very high. The problem is once again of not using the bar-structure of the music, as this could be used to control the bass line to walk up the scale, and then down again at specific points, so that it does not escape.

4.2.3 Adding Structure

To add structure the bass line mapping was changed back to the simpler mapping, as this actually generated something audible to work with. The solution was actually very simple. A variable was created called barCount that was used to count the half-beats in each bar (half-beats were counted as this is the granularity the bass line was working at, but the bass line was changed later to slow down and only play on the beats as it sounded better). Each time this count reached 8 it was reset. The variable barCount now can be used as an indicator to the current position within the bar. Another variable was created called direction, which was set to 1, and each time barCount was reset the direction variable was multiplied by -1. So if direction is currently 1, the bass line should be walking up, and if it is -1 then it should be walking down. Using these two bits of data, the program could tell where in the bar it was, and what direction the bass line should be walking. The mapping was changed so that if the bar count had just been reset and the direction was 1, the mapping would produce the bottom note in the scale. If the bar count had just been reset and the direction was -1, the mapping would produce the top note in the scale. This meant that the first note of each bar would always be either the scales root or top note, which had the effect of bedding down the sound of the bass line to the scale.

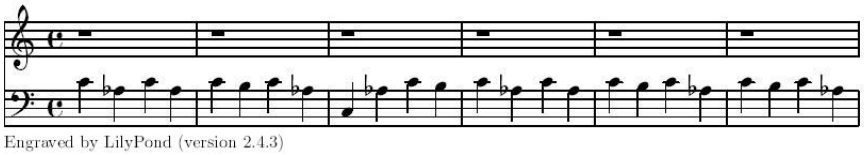

Figure 4.1: Before Added Beat Structure

It was anticipated to need to make far more complex changes to the mapping to create a good sound. The figures here show the appearance of the bass line before and after the changes were made. As we can see the effect on the music looks very minimal on paper. But this change, along with some tweaking of the ranges in the mapping (to make extreme notes less common), made a significant change to the sound from the previous output and produced quite a good repeating bass line. Figure 4.2 corresponds to Out3.

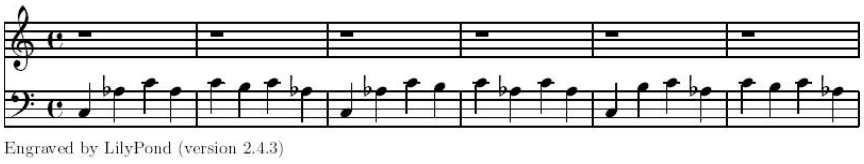

Figure 4.2: After Added Beat Structure

This mapping was also tried with the system start variables set to a more chaotic orbit, which can be heard in Out4. Again a quite reasonable bass line was produced, although it did not have much direction and the repetitive bass line was the main aim. It was decided at this point to leave the bass line, as we have a quite reasonable wandering and repeating bass line, and as the bass line is not of as much interest as the melody.

4.3 Melody Experimentation with the Standard Map

After starting work on this area it quickly became apparent that a whole dissertation could probably be written on just one area of the melody. To create a authentic blues improvisation sound there are so many different paths to be explored and tested, things to try, and there was no way that they would all be explored in this project. It was decided to spend time exploring each area a little rather than to focus the attention of the whole project on any single one.

4.3.1 The Basics

The Standard Map has two dimensions, which are used for pitch and duration of note. Once there was a simple version of this working, it was immediately clear the melody mapping needed to work in a different way to the bass line. As the duration of each note in the bass line is the same, it was possible to just generate every note required for the piece and put them all in an array to work with them. Now there are different durations of note, it is not clear how many notes will be used in a piece, so it is difficult to put them in an array. To solve this the code was changed to work on a single note at a time. A new variable was used to hold what point in time had been reached in the piece, which is incremented as each note is known.

The initial reason of putting all the notes in an array was it allowed us to scan the array to find the range of values produced. With this new method this is no longer possible. But as the phase space of this equation is the unit torus, all the values appear between 0 and 2Pi, therefore we have our range. Using a very similar mapping as the bass line, by splitting up the range of possible values into section which correspond to notes, it was possible to get some ok results, but not particularly good. First results can be heard in Out5, and better results are in Out6, which

are after swapping round which variable was used to map pitch and duration, and tweaking of the ranges used to map to different notes.

4.3.2 A More Advanced Mapping

This current mapping only allows notes over one octave. To improve the mapping it was attempted to implement the failed method of mapping for the bass line. This was the method of using the data from the equation to offset from the previous note played. As the melody can jump more than the bass line, higher offsets can be allowed, and also this means each note in the melody has a relation to the last and so will hopefully flow better.

The partially working code was used from the bass line, but amended so that the available offsets from the previous note suit the melody better. Limits were imposed on how high or low the melody could walk in order to solve the problem of it escaping too high. It was not appropriate to impose a period of walking up or down like in the bass line, as the melody doesn't behave like this, making the upper and lower limits preferable. It took some time to get the balance of offsets correct, at first the melody was a bit jumpy and larger offsets needed to be less common, but a good balance was achieved, which can be heard in Out7.

4.3.3 A Sense of Rhythm

The program needed a bit more of a sense of rhythm. The melody currently has no ties to the rhythm and anything that makes it sound like it does is purely coincidental. To add rhythm notes were placed to be played more often on the beats of the bar, rather than wandering between beats. The code used tested if the current place in the song was within 2 32th's of a beat, if so the note is moved to be on that beat. This means that notes that would just be close to the beats are actually played on the beats, creating a better rhythm to the song.

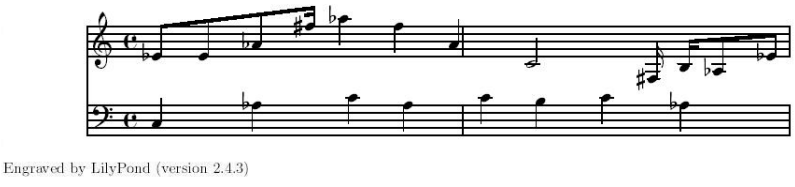

Figure 4.3: Before Rhythmic Changes

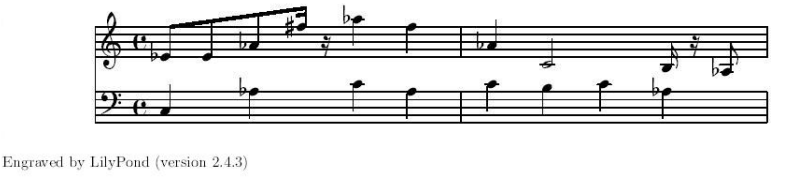

Figure 4.4: After Rhythmic Changes

Here are some examples taken from the same point in a piece. We can see that in figure 4.4 rests have been introduced to bring the music in line with the beats of the bar. In the first bar a rest is introduced to complete the second beat, allowing the next two beats to continue in step with the bass line. In figure 4.3, there is no rest and the remainder of the bar and the next bar become offbeat.

This change brings the melody more in-line with the bass line, as the bass line consists solely of notes played on each beat, so they now fit together better. This improvement has a subtle effect, but it does sound better and was certainly worthwhile.

Shortly after doing this, a consistent error was discovered within the code that involved how the program keeps track of time within the song and creates note start times. Hence the output file to go with these changes (Out8) has a slightly different sound to it as well as the change in rhythm. The error actually meant that notes were played twice as fast as they were meant to be but it was compensated for by spreading them out further than they should. This made the notes sound jerky, but now it is fixed and the music has a much better flow to it.

4.3.4 Further Links to the Bass Line

Although the melody fits rhythmically to the bass line now, there is nothing to ensure the pitch of the notes work well together. This is a complex problem. Firstly there is nothing to say that any notice of the pitch of the bass line should be taken at all, as blues musicians cannot look into the future to see what the bass player will play to ensure the melody works with it. Indeed they will be playing along to a certain rhythm and in the same key, scale, and also use their experience to ensure the music sounds good. One option chosen to implement was to harmonize the melody with the bass line in places to try and make them fit together. To do this, at any point where a note in the melody was played with one in the bass line, the pitch was adjusted, as long as the jump wasn't too great, to harmonise with the bass line. But whether this is a good idea or not is up to debate, as it has the effect of breaking the flow of the melody created by purely using the data from the Standard Map. Sometimes it would sound slightly odd and/or worse than the original notes, due to this in future outputs, no harmonisation is used, but you can hear its effects in Out9.

4.3.5 Chords and Harmonies Within the Melody

The problem here is to take a single line of notes and add in chords and harmonies to it. As harmonies can just consist of one extra note, it was decided to try and tackle this first. There are two main parts to the problem:

1. Which notes are to be given a harmony?
2. What note do we add to produce it?

The second of these questions is the easier. Apart form the fact we only have 12 notes in an octave to choose from, generally a good note to harmonise with is the one 4 semi-tones up. This is the second note in a major chord, so for C it would be E. But for the blues it is better to drop this by a semi-tone, making it the second note in the minor chord, so for C we now get Eb, which has much better blues sound.

So which notes do we add a harmony to? This proves to be a much more difficult question. To harmonise every single note would be simple and add no real interest to the music, but then which ones do we pick? This was the major downfall to any method attempted. The aim was to produce harmonies on notes that occur on some beats of the bar to make it sound rhythmical, and although this sounded ok some of the time, it often also often sounded quite terrible. The effects can be heard in Out10. The note added needs to be more carefully selected, also analysis of the rhythm and structure of the melody needs to be done to select good points for adding chords and harmonies to make it sound good. A musician would use the rhythmic structure to decide, rather than just adding them in only on beats.

There is another way of doing this that was stumbled across somewhat by accident. A consistent error to do with timings was briefly discussed earlier, where some of the maths in the program was wrong and it was counting 2 beats whenever it should have been counting one. This had the effect of stretching the notes out over double the space in time they should, but the tempo of the song has just been sped up to compensate. Some of the incorrect maths was noticed and corrected, which had the effect of squashing all the notes up into each other. Where before, for example, the notes had occupied 8 bars, they now occupied 4. If you imagine 8 bars of music all full of notes, you squash them together and they all bunch up. But because of the time granularity the program was working at, some of the notes were squashed so that they played at the same point in time. This created sets of usually 2 notes, played at the same time. You can hear this effect in Out 11. It is up to the listener as to whether they prefer this output or other earlier outputs. This one is much more fast paced as the notes are all close together, but still sounds good.

There was difficulty re-creating this effect once the error within the code was fixed, although the incorrect code could still be used. What was actually happening is that subsequent notes generated by the Standard Map are in fact being played at the same time as the previous one if the delay between them was short enough. The problem with solely using this technique in composition is that it would be hard not to have fast paced music with all the notes following quickly on from each other, slow sections would not occur. This is why no great efforts were made to re-create the effect and use it. It is something that could be used along side other methods to generate sections of music, or possibly modified to allow slow and fast paced sections.

4.3.6 Adding Song Structure

After completing the literature review, this was one of the main areas of research identified as being something that had been looked at, but not yet solved adequately. The inherent structure of chaotic systems was one of the things that drew us to them, making this section one of the more important parts of the program. To add global structure to the program, two options were considered:

1. Use the property of chaotic systems that very small changes in starting variables, will create at first similar, but then very different outputs. This could be used to create different sounding parts of music, which still had properties from other parts. For instance the starting variable for one verse could be x, and the next verse it could be x + 0.001.

2. To use the repeating structure of chaotic systems that they often pass through very similar parts of the music, but never do exactly the same thing twice.

Now we also actually hit a slightly more philosophical argument here. The first option involves intervention from the programmer. The programmer needs to tell the program when to make amendments to the variables, and what amendments to make. This will in effect take the decision about the music's structure away from the data produced by the chaotic system, as any variation will have to be inbuilt into the rules. On the other hand, the second option depends completely on the data produced by the system and will be variable with each composition. The problem is that the second option is difficult because we will need to find out what time scale the program repeats over. If it takes 3 hours to get any sort of repetition of similar results, not only will it be very tedious waiting for it, but you are also unlikely to notice as you will not have any idea of what the music sounded like 3 hours ago. So do we follow the argument that we should be letting the computer do as much of the composing as possible and try to extract inherent structure from the data, or use the properties of the system to put in the structure manually?

There is nothing inherently bad about the first option. It is similar to teaching a musician that there is a set of structures they can use that sound good, rather than telling them how to invent their own. Should we actually be worrying too much about this at all, as we're playing the blues here not writing hymns? Blues improvisation is often free and has no verses or choruses. Should a repeated structure be added at all? The answer is yes as blues musicians will focus around themes and repeat similar sounding bits within an improvisation, therefore adding in a element of repetition and structure to the music is important, but just not the traditional verse-chorus structure.

It is also worth noting that the music generated does appear to have some structure already. The music is not random, and does appear to follow a general theme through the improvisation (though it will on occasion go of on a bit of a wild tangent). But here we are concerned about bringing a global structure to the music, which is different.

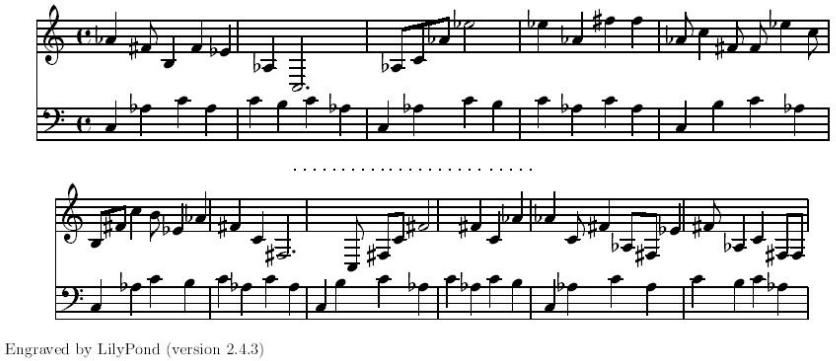

Figure 4.5:

So using the first option, it was easy to give the music structure. Exactly what structure we put in is up to taste. The original starting values of the variables of the system are stored, and at certain points within the program, either the current working values can be replaced with the originals to get a exact repetition, or the original values with a very small offset can be used. The point in time this is done is also up for discussion. Having the 12 bar structure could lead us to decide to treat each 12 bars as a block and make manipulations at the end of each

block, but there is nothing to tell us not to make changes within blocks. Figure 4.5 shows this method's effect with two small sections taken from the output. We can see that from the third note onwards of the second section, the structure of notes is repeated from the first section but at a different pitch. The repetition is exact for a stage and then begins to change and takes its own course. This is an example of *nearly* the same values being entered into the Standard Map's variables later in the piece. Output Out12 uses this technique, it plays 48 bars and so is much longer than previous outputs.

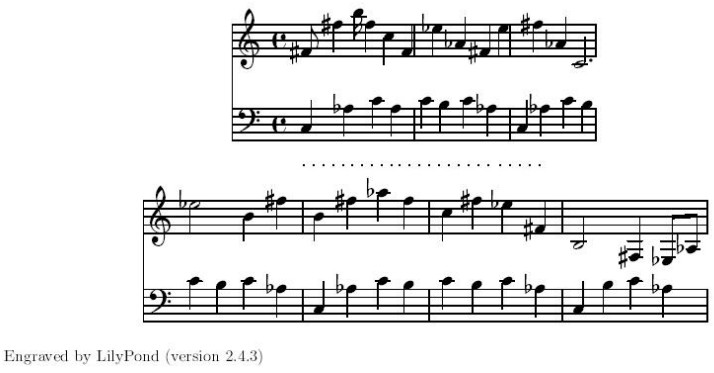

Figure 4.6:

As mentioned above the second option does present some difficulties. At the beginning of implementation it was unknown to what extent the Standard Map produces an overall repeating structure, and if it does, what kind of time scale it will do so. To add a good structure to a piece of music, it will need to be found and scaled appropriately to the length of the song. To try and discover if this method can be used to get a overall song structure the start variables were varied to get different results, and the Standard Map was sampled every second, third, fifth or sixth orbit, to get a wider sampling range. After several attempts with no apparent structure, several outputs with structures were found. Above in figure 4.6 we can see two sections of music from different parts of the piece, but there is a very evident repetition of the style of music being played. A series of crotchets are played in both sections, with both having a similar pitch pattern as well as a longer note at the end. Out13 is also an example of structure formed in this way. The audio output in particular has very clear phases that the music moves through which make the structure easy to see. In pieces that use different starting variables, subtler repetitions may have been present and just overlooked. To produce this effect with any kind of consistency, some sort of analysis of data produced by the Standard Map would need to be carried out within the composition process. The analysis would be required to find the repetition and scale it accordingly to fit the length of the piece, but to do this is something which there was not time for in this project.

Chapter 5

Evaluation

Now that the development stage is completed it is important to evaluate the work that has been done and reach some conclusions.

5.1 Questionnaire

In order to get a good evaluation of the composer it was decided to ask some people to listen to two of the outputs, namely Out12 and 13 (referred to as Output 1 and Output 2 respectively, in the evaluation), and then give their opinions. This is beneficial for two reasons, firstly while developing this system the author has spent hours listening to test outputs and so is very much accustomed to anything the system can play, and secondly an unbiased, critical view is quite beneficial. Ten questions where asked to ten people and the results are as follows.

Please rate your musical knowledge/experience on the scale of
1.None
2.Beginner
3.Intermediate
4.Advanced
5.Professional

Luckily a selection of people with a good range of abilities was sampled, with two people in each category. This should provide a good balanced view.

Do you play the blues?
1.No
2.A little
3.A lot

Again a reasonable spread was found, with four people never playing, four people playing a little, and two people playing a lot.

Please rate on a scale of 1 to 10, 1 being low, 10 high, how much you enjoyed each output:

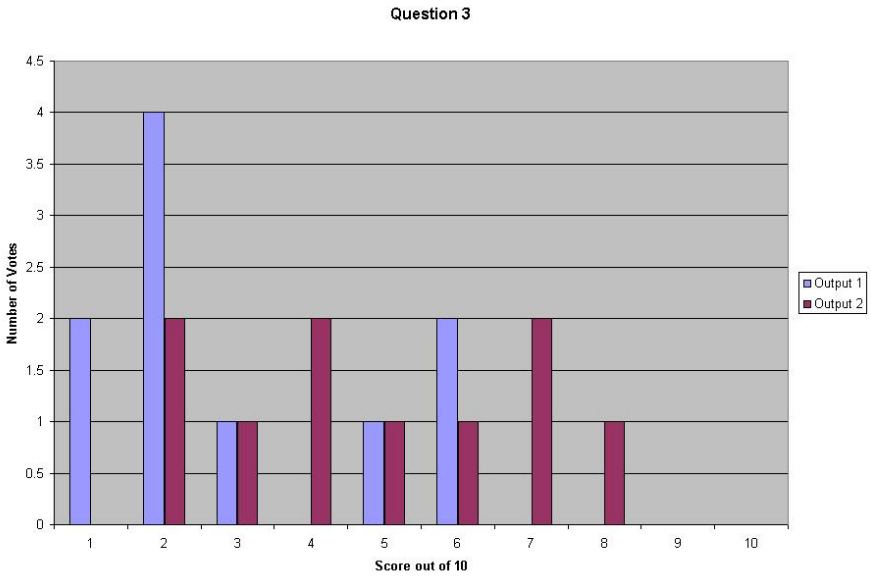

Figure 5.1:

These resulting where slightly shocking to this systems author, as the first output was always preferred. This result could be due to the fact that the second output is simpler and has less wrong notes, so to a casual listener it sounds a lot better. Output 1 on the other hand, has more tendencies to play wrong notes that have to be forgiven, but is more inventive and gives more complex and interesting melodies.

Overall these results pleasing are pleasing. Some people may have not enjoyed the music whatsoever but a reasonable number did. With 8 votes in total of 5 or above, averages of 3 for output 1 and 4.8 for output 2, this is a reasonable result.

If you heard someone play this as an off the cuff improvisation, how would you rate their ability?
1.Appalling
2.Beginner
3.Intermediate
4.Advanced
5.Professional

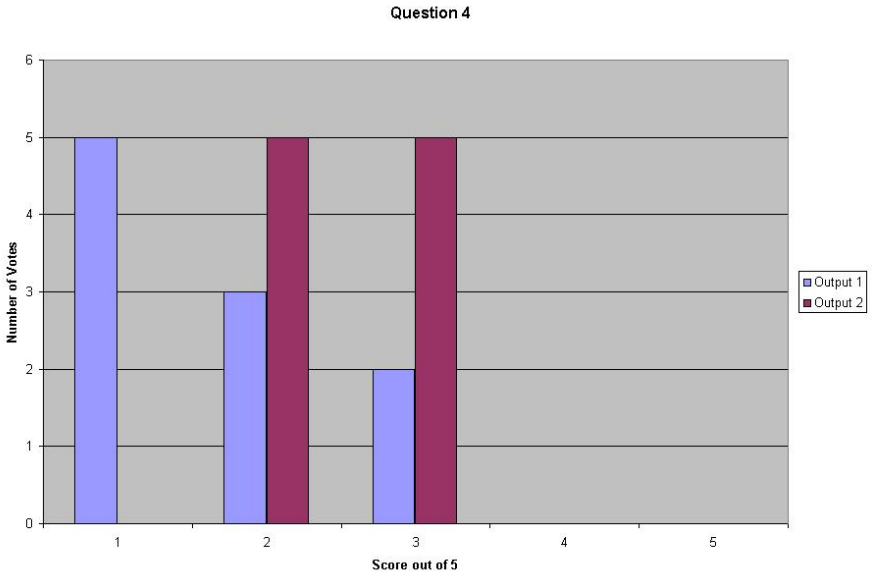

Figure 5.2:

Again output 2 easily wins over output 1. Although the results for output 1 are satisfactory at best, the results for output 2 are very pleasing. There was no expectation to get any advanced or professional votes, but the even spread over beginner and intermediate for output 2 is very good, showing that a reasonable standard has been met.

Please rate on a scale of 1 to 10, 1 being low, 10 high, how well you rate the melody:

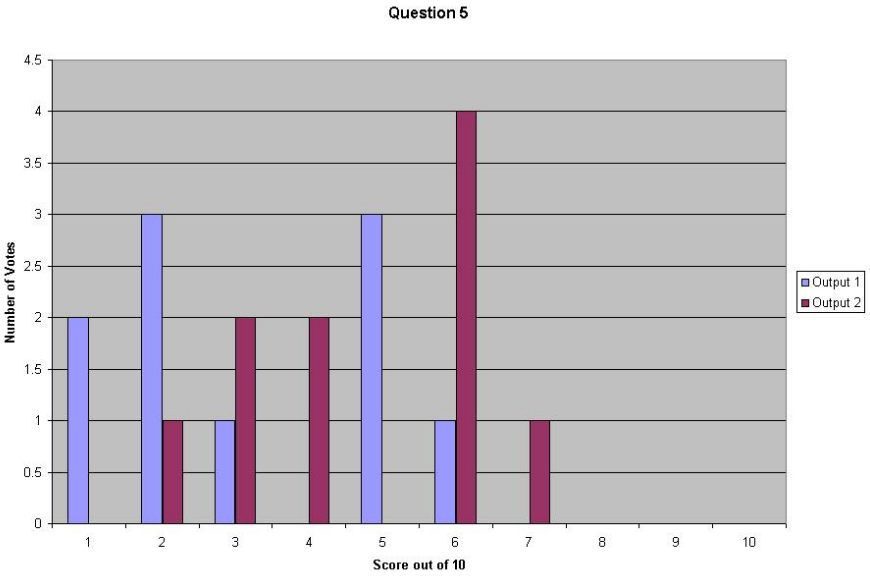

Figure 5.3:

Here we can again see a reasonable mix of results. The average for output 1 is 3.2, and the average for output 2 is 4.7. Compared to how people rated the music overall, the melody is not rated as well. Although the averages are not good, they are not desperately low, we also observe the same trend of output 2 winning over output 1.

Please rate on a scale of 1 to 10, 1 being low, 10 high, how well you rate the bass line:

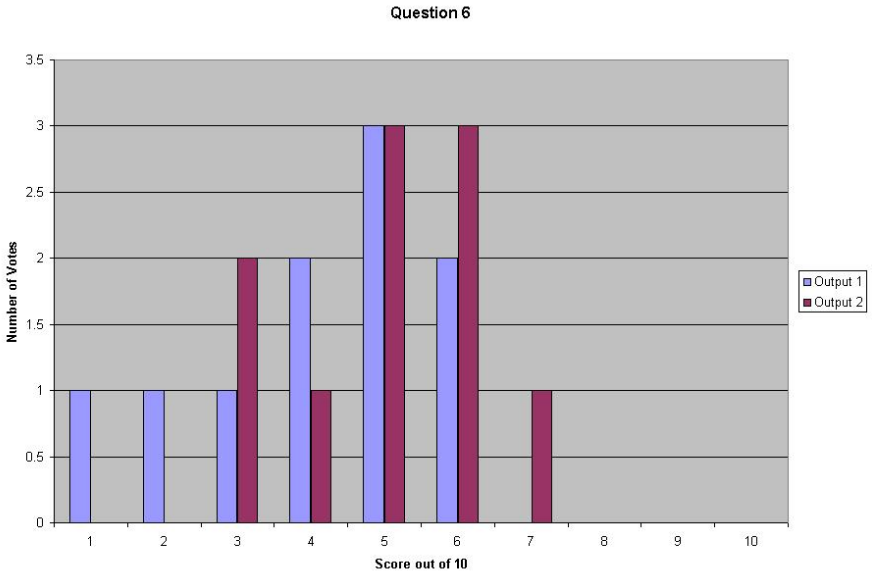

Figure 5.4:

This result is mildly amusing due to the fact that when this questionnaire was written, it was forgotten that the same bass line was used for both outputs, but again we see the same bias toward output 2. This must be due to people's general better feeling toward the output. The average for output1 is 3.7 and 5 for output 2, which marks quite a large difference, psychologists would have a few comments here! Overall this feedback is quite pleasing as the bass line is rated well.

Please rate on a scale of 1 to 10, 1 being low, 10 high, how well the melody works with the bass line:

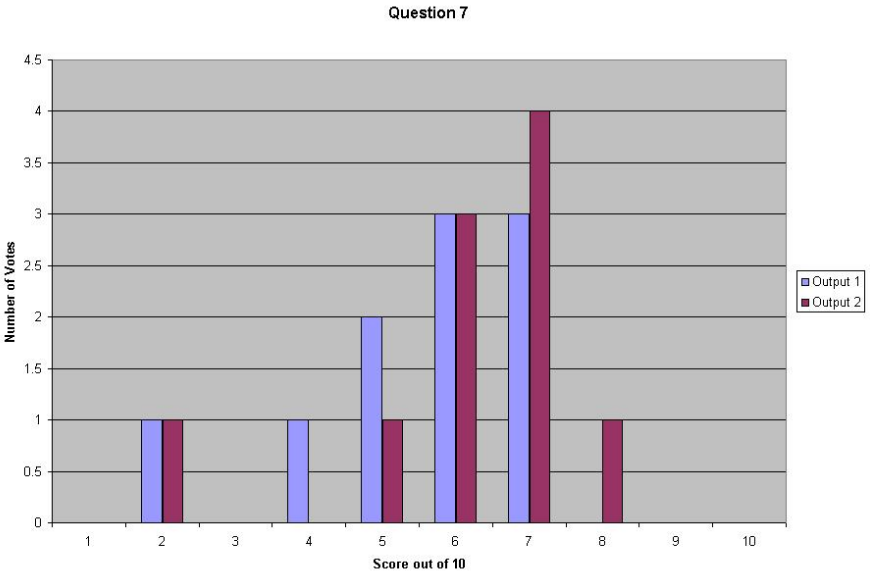

Figure 5.5:

These are interesting results. The melody was only linked to the bass line in two ways, firstly the scales they were using always matched and secondly the rhythm of the melody was designed to fit in with the bass line. Roughly matching the rhythm was not difficult and is quite a simple link, but these two things together have provided very good results. These are the best averages so far with the average of 5.5 for output 1 and 6.5 for output 2. These results are also a lot closer, which is to be expected as both outputs used the same method for this part of the program.

Please rate on a scale of 1 to 10, 1 being low, 10 high, the improvisations sense of rhythm:

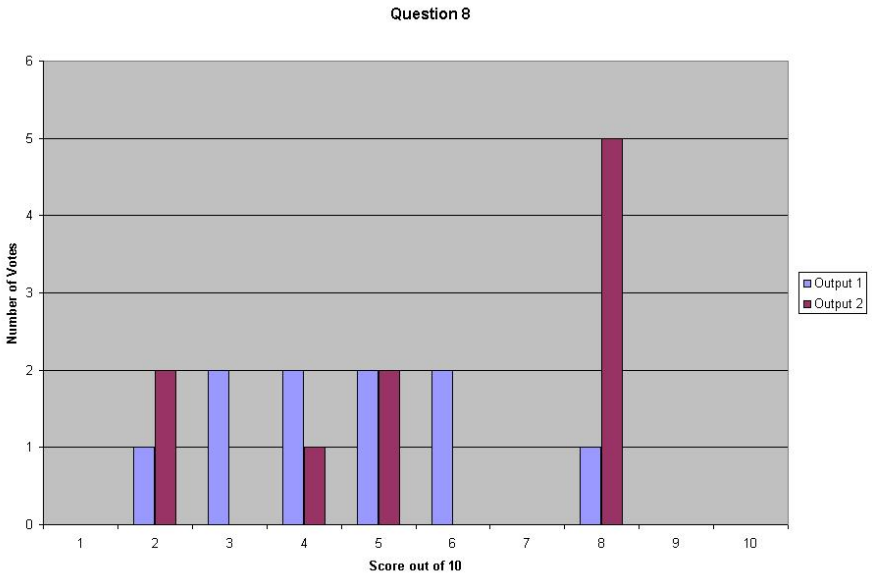

Figure 5.6:

Here we see a somewhat odd spike for output 2 and a reasonably even spread for output 1. The average for output 1 is 4.6 and 5.8 for output 2. This shows that the relatively simple steps taken in this project, already produce what people call fairly intermediate results for a composer. The spike for output 2 of very good feedback may be due to the sections in the output of same duration, fast paced notes. This is due to the chaotic system moving through a specific phase space that enabled this, and distinguished it quite a lot from the other output.

Please rate on a scale of 1 to 10, 1 being none, 10 lots, the overall structure of the melody (whether you think it wanders aimlessly or has some sort of direction):

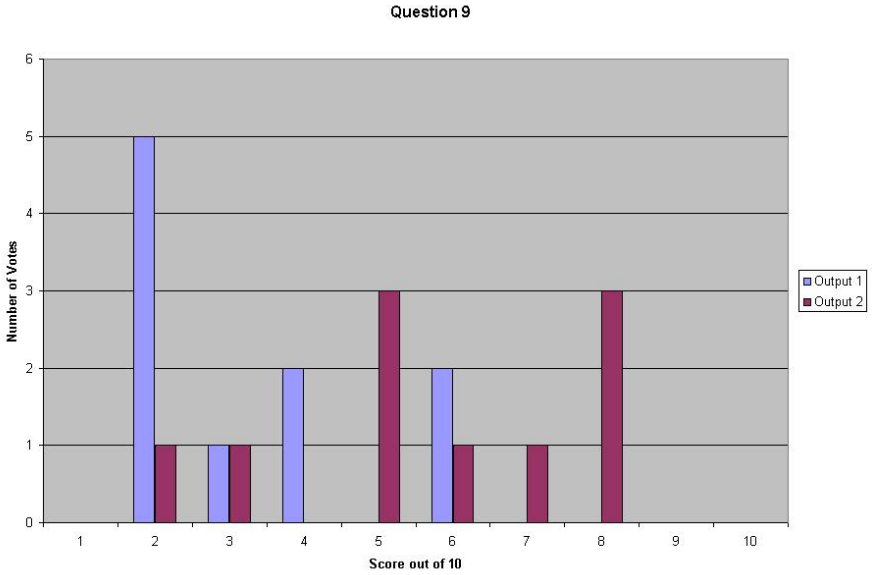

Figure 5.7:

This is possibly one of the most important bits of feedback to this project, as one of the main aims of this research was to look at how structure could be introduced into music. Two methods where used to do this and the only real difference between the outputs, apart from the starting values to the chaotic system, was the way the structure was generated. We have a clear winner over the two methods, which is output 2 which used the chaotic system itself to find structure rather than using an inbuilt structure given by the programmer.

We can easily see that there are very encouraging results for output 2, with much less encouraging results for output 1. The average of 5.7 is slightly low and possibly misleading for output 2 as most of the votes are grouped around 6 and 7. The overall structure to output 2 is very clear, it has quite distinct sections of music. One comment from the person rated as a professional regular blues player said "quite a lot of the phrases do have a melodic shape to them", which is very encouraging as to the structure on a smaller scale as well. These results show the goal to add structure to the music has been achieved, although not to a professional level, but to an intermediate to advanced one, as indicated above.

Output 1 has an average of 3.3, which is significantly lower. The problem appears to be that the structure in this output is significantly subtler. We have shown that a repetitive structure is there in figure 4.5, and repetitions are made at a number of points, though not at the same pitch. Listening to the output once this is clearly not apparent, work needs to be done to make a more apparent structure in the improvisation.

Although the sound is obviously computer generated, could you believe if played to you without any prior knowledge, this was composed by a person, rather than generated by computer?

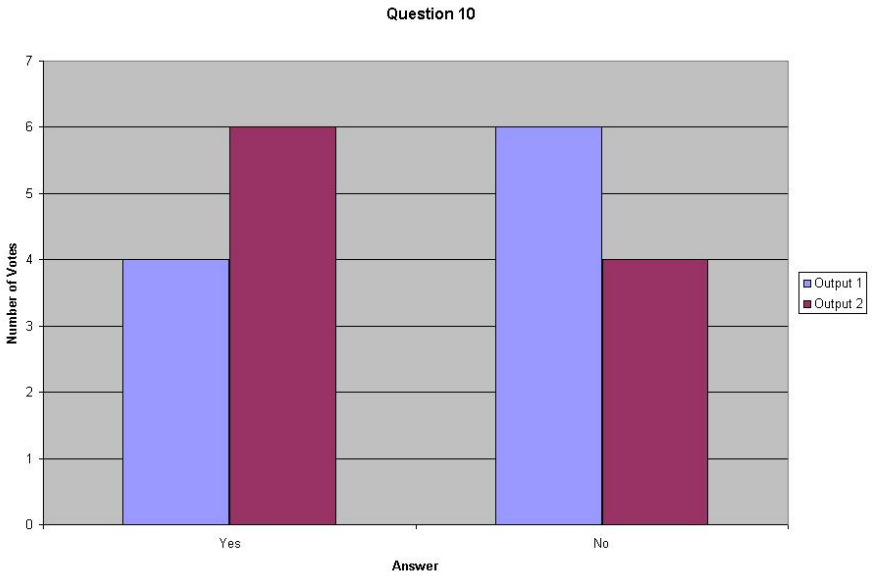

Figure 5.8:

In total here we have a complete 50/50 split. No attempt was made to hide that the music was computer generated, but it is very good that half the population think they could be fooled by my system that a person wrote the music. Although no one would be fooled into thinking a professional composed it, it is still a good result. It shows that we are at least on the way to creating a decent composer.

Any Further Comments?

An open space was left for people's comments. One person left the following in reference to output 1 "The notes are computer generated but also played by a computer; if they were played by a good human musician maybe it would sound much better than the awful, non-descript computer sound on this recording". Here possibly lies the root of the main bias toward output 2 rather than output 1. Output 1 had more tendency to be more inventive and in particular to play higher notes. Unfortunately with the instrument used, some of these higher notes tend to sound rather more screeched than played and can sound quite piercing. Unfortunately this was the best voice available at the time and may have been the downfall to an extent of this program, or at least this output. Another comment reads "The first sounds like its hitting flat notes but is more inventive", which supports this opinion about output 1. Obviously work needs to be done to eliminate bad notes from the composition, as not much work was done in this area, apart from ensuring the correct scale.

This comment was also left "There are some nice moments in 2 - the minor 3rd to root "dee-doo-dee-doo" on the 5th bar of the 1st chorus is cool, sounds almost like something a human might play at that point. There are a couple of other points like that in the second one as well, where what the computer comes up which sound almost like someone with a rudimentary/intermediate knowledge of improvisation might play". This is also quite a revealing comment. Although the system was able to give some good moments, they were few and far between. In order to raise the standard of this program to a professional level, far more effort needs to be put into teaching the program about music structure, riffs or patterns of notes that sound good. Another comment reads, "if you eliminated the notes which sound "out of tune" youd have a very acceptable basic blues improvisation in output2", which is saying a similar thing. As the program leaves these issues pretty much down to chance and the structure of the chaotic system, it gives an OK output. In order to give an excellent output this is something that must be addressed.

5.1.1 Reflection

Overall from this questionnaire we can say that system can produce music that people in general can find quite acceptable and even enjoy, but is also capable of produce music people find quite dire. There may be reasons that are not the programs fault, like the instrument used, but the main comment is that wrong notes need to be removed which is something that must be addressed. The feedback is more positive when it comes to the structure of the music and its sense of rhythm. Not only has it been shown that relying solely on the chaotic system to provide the structure to the music is possible, but also provides good results.

5.2 Overall Project Evaluation

This has been a research project, which has dictated to an extent the way that it was carried out. There were three main stages to the project, the subject research, requirements and design and the implementation.

Subject Research

The subject matter of this project is quite specific. It was quickly discovered that most of the research done in this area is not in books but in journal articles and the like. The biggest resource available for this kind of material was online databases of journals. Using these resources a good overview of the subject area was covered. Once the search was narrowed down to detail on chaotic systems, mathematical books on the area could then be consulted. The information gained from these resources gave a good ground to stand on when going into the design and implementation stages.

Though more detailed reading could have been carried out for a deeper understanding of the varying composition methods, sufficient information was gathered to make an educated decision on which path to take. Once chaotic systems were chosen, information on them was obtained to a good level, which enabled the design and development stages to progress quickly and smoothly.

Requirements and Design

This section was very important as it was an opportunity for laying out exactly what the intended implementation was and what possible ways there where of doing it. We were unable to go into much detail at this stage as a lot of the detail of what would actually be done was

unknown and subject to experimentation. This section was based upon the research that had already been completed and the ideas of the author. Its greatest value was as a point of reference when implementing the system as it kept the project moving through the sections, rather than going off and spending many hours on one area of interest, which it easily could have done.

The requirements developed were realistic for the time scale of the project and the final design of the implementation used many of the ideas developed in this initial stage.

Program Implementation

The project was developed in an incremental fashion, with each section being added on one at a time. This suited the project well as it meant attention could be paid to specific parts of the music while listening to outputs, rather than attempting to listen to several things at once. Developing the bass line before the melody was also very useful as it in effect gave the author some practice at using these systems before moving on to the more challenging part of the implementation.

The experimentation was carried out as a mixture of trial and error as work continued from the design phase. This method worked well as a lot of things just needed to be tried to see if they sounded good or not.

As this was a research project rather than software development, the final code is not what you would expect from a professional product. The code was written to be functional and not fast or efficient. Due to this none of the code has been optimised, but this does not matter for it's purpose and any sub-optimal code has no noticeable effect.

The final product of the program was by no means perfect. Both the harmonies between the bass line and the melody and harmonies within the melody had a negative effect, and were not active in the examples given to people for the questionnaires. The parts of the program that where unsuccessful where not wasted though, as they still teach us lessons about the composition process.

The composition process was successful in meeting one of its major targets, which was to add structure to the music. This was done using two separate methods both of which were successful. There is still a lot of work to be done in this area, with lots of extensions possible. Although the work done has provided some interesting results to work from.

Chapter 6

Conclusion

The initial aim of this project was to "construct a program to compose music in the style of the blues, with an authentic blues sound". This aim has certainly been met to a degree. Music has certainly been composed in the style of the blues, as to how authentic the sound is of personal opinion. The music certainly does not have an authentic professional sound, but has a blues feel to it. It was never expected to get a completely authentic sound so with this in mind, it is reasonable to say the overall objective of this project has been met.

6.1 Requirements Review

Here we shall review the requirements set for this project, which are found in section 3.6. Not all of the requirements are worth mentioning individually, so we will focus on the more interesting ones.

6.1.1 Requirements Concerning the Bass Line

This should be a simple repetitive bass line suitable as a backing: The resulting bass line can clearly be seen to have a repetitive structure in Figure 4.2. Due to it's simple nature it works quite well in the background and does not dominate the melody.

It should play notes within the correct scale and change scales with the 12-bar structure: Measures were taken to meet this requirement, which the program fulfils completely.

The repeating structure should fit the 4 beats per bar structure of the music: This requirement was also met once some effort was taken to keep track of the programmes progression through the bar structure.

The bass line should be made more complex to form a wandering bass line: This was an optional requirement, but was met as the bass line can take on a wandering style if it is changed to a chaotic orbit, rather than a periodic one.

6.1.2 Requirements Concerning the Melody

It should be of a relatively high complexity and hold the listeners interest: This is a slightly difficult requirement to evaluate, but the composition does have a reasonable complexity

with a good variety of note lengths over several octaves. From the questionnaires we can conclude that it will certainly hold some listeners interests, although not all. We can conclude that this objective has been met to a large extent.

It should play notes within the correct scale and move smoothly between scale changes: Work was done to ensure notes where in the correct scale. It was found that nothing really needed to be done to ensure a good flow of music between scales as the music seemed to flow naturally enough without any intervention.

It should play notes of a variety of lengths and over a variety of octaves: This requirement was met. Notes are played over 3 octaves with 6 different note lengths used in composition.

It should combine well with the bass line: Although efforts to harmonise with the bass line failed to sound reasonable, this requirement was still met. The melody was made to fit rhythmically with the bass line and used the same scales of notes to prevent major clashes.

It should maintain an overall structure: This requirement was met using two methods. One was to directly manipulate the variables of the chaotic system, the other was to use the inherent structure within the system.

Repeated note structures and segments should be included in the composition: This was an optional requirement and is linked closely to the one above. It was also met and uses the same methods as above.

It should play chords as part of the composition: This was also an optional requirement. An attempt was made to include harmonies within the composition, though they were included, they did not sound particularly good.

6.1.3 Overall Requirements

The complexity of the program should be taken as far as possible in the time available: The program achieved quite a reasonable complexity and achieved what where evaluated to be average results for a blues composition. This is quite a reasonable accomplishment for the time allowed for the project.

The program should be amended to produce a different output with each execution: This was an optional requirement and has not been fulfilled. It would have been very easy to simply put in a random number generator to select starting values for the variables in the chaotic systems. This was not deemed worthwhile as it would not really add any academic value, and also is quite likely to not produce consistently good results at this stage of development.

6.2 Future Developments

Rather than solving a lot of problems, this project has rather highlighted a lot of areas in which more work can be done and found out some more about them. Before a truly authentic professional blues improvisation can be made, or a general composer using this kind of method, these areas need more work, many of which are interlinked.

The Bass Line

The bass line in this implementation is very simple. There is a lot of scope for it to be improved. Firstly it currently uses notes of fixed duration, to have notes of varied duration a different chaotic system needs to be used. Because although this system has two dimensions so one that could be used for note duration, the second was simply 0.3 of the first (and so not very interesting). A more complex system therefore would be beneficial. Also more rules as to what should go in a bass line could be implemented, with the possibility of paying more attention to a free walking bass line rather than a purely repetitive one.

The Mapping of Notes and Melodic Structure

One of the main issues to come out of the questionnaires was the need for wrong note elimination and better melodic structure. This system was able to produce some melodic structure, but to produce a good sound research needs to be done into the best ways of teaching the computer to produce melodic forms of the style it is composing. Also the composer needs to be taught rules about note clashes and when a note which is in the correct scale, can still sound wrong. A potential solution to these problems would be a set of rules for building original melodic structures and/or a library of structures that could be used. But for a machine to truly compose it would be better for it to generate its own melodic structures, even if they were stored for future use, rather than using a library fed in by humans. But the use of libraries does have the advantage of easily being able to change the type of music produced, by simply changing the library. So it is up to the reader to choose between these. Either of these methods could also be used as an aid to eliminating wrong notes, but a rule-based system for this would probably be more beneficial rather than producing a library of either correct or incorrect scenarios. These rules and/or libraries could be used by the mapping function along side the data produced by the chaotic system to compose the music.

The Link Between the Bass Line and Melody

In this system it was very simple to match the rhythm of the bass line to the melody due to the fact the bass line was very simple. If the bass line where to be made more complex with a more dynamic rhythmical structure, it would be important to ensure the melody kept in step. This would need to be done by a system that used the bass line as part of the equation for the rules of the melodies rhythmic structure, and so could be linked into the rule system mentioned above.

The Music's Rhythm

Although my program was taught that it was important to try to stick to the beats of the bar there is a lot more scope for improvement. For example, as first beat in the bar is more important that others, the program could be taught to have a strong inclination to play notes on that beat. Also the rules of the system could be extended to include rules about how note durations link together, so that runs of fast notes can be played at appropriate times and that long and short notes fit into the bar structure of the music.

Chords and Harmonies

An attempt was made at this with the system but it sounded better without it. The program needs to be taught rules about chord structures and harmonies, and how to make sure that these fit in with the notes being played in the bass line. It is also important to ensure that the rhythm

that chords are played in fits in with the rest of the music, which will include also the rhythm of the bass line.

The Music's Structure

This system has a certain amount of structure within it but there is still plenty of room for improvement. Firstly the technique of finding the songs structure purely from the chaotic system used needs to be developed. The time scale at which orbits move through similar phase spaces needs to be found so that this can be scaled to the music to provide phases of music. This would need to be done early on in the compilation process so that the correct set of data can be used to produce notes. Effort also needs to be made to ensure that sections of the music are distinct so that listeners will notice they are there, which could be done by checking that the phase space occupied by each section are sufficiently different. Some manual control could be added to this by the other method used of directly manipulating the chaotic systems variables to impose any traditional song structures that may be wanted.

The effect of using the structure of the chaotic system in pieces of long duration is of yet unexplored. We do not yet know if the structure will bring the music to some kind of culmination, or just wander on within a more limited structure forever. This is another avenue that more work would be beneficial in.

The Chaotic Systems Used

Currently only used two-dimensional systems have been used. There is scope for using systems of many dimensions, for example a dimension could be used to determine the amplitude of the note. Different systems may sound a lot better than others, or be better for different styles of music. Dimensions could even be used to determine the structure of the music, such that as the dimension moves through the phase space, it affects the sound of the rest of the piece. There is a lot of room for experimentation here.

6.3 Contribution & Relation to Other Work

One area identified as currently on going research from the literature survey, was that of adding structure to music. Eck and Schmidhuber [9], Burr and Miyata [3] and Dubnov et al [8] were some of these to do so each with various amounts of success. Chaotic systems were identified as having good potential for adding structure to music and where chosen as the composition method partially for this reason. In this project chaotic systems have been used to add structure to the music with two separate methods. Direct variable manipulation provided an evident structure, but when tested on a sample of people was not particularly apparent and was perhaps a bit subtle. The method of using the structure within the chaotic system itself to provide phases of music was a lot more successful when tested and was much more apparent to the listener. By using chaotic systems in these ways, this project has made a contribution into this research area.

Also no attempts to use chaotic systems to produce music in any particular style of music were found. Though some attempts may have been made to do this, they are not particularly numerous. This project has successfully shown that chaotic systems can be used to compose music in the style of the blues to an average quality, and has given direction to those wishing to take this further.

Eck and Schmidhuber [9] completed some work on using Recurrent Neural Networks to compose blues music. The blues generated by them was found to be quite simple and although it contained a structure there was nothing defining about the structure of any one piece produced in this authors opinion. Although outputs from this project can also sound reasonably similar, often pieces will have quite different phases they will move through and have differences to them. Eck and Schmidhuber's work is much less likely to produce anything that was offensive to the listener than material from this project. Their music was generally pleasant but less interesting, where as music from this system is more interesting but also sometimes unpleasant. Work needs to be done to allow a wider variety of music, which is still kept within the rules of what sounds good. This difference between the works could also be partially because music is allowed to be more complex in this project, due to a wider range of scales and note durations available to the program.

6.4 Final Conclusion

Despite it's limitations, music has been composed in the style of the blues to a reasonable level of proficiency. Chaotic systems have been found to be a feasible method of algorithmic composition and also have been found to have structural properties that can be exploited in the compositional process. The main objectives of the project have all been met and a good contribution has been made to this area of research. Due to these reasons, this project is a success.

Chapter 7

Appendices

7.1 Program Code

```
#define SONG_LENGTH (384*4) /*12 bars = 384 32th notes*/
#define HENON_A 1.076 /*1.4 for a chaotic bass line orbit*/
#define HENON_B 0.3
#define STANDARD_K 2
#define STANDARD_I 2
#define STANDARD_T 2


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


/*The structure for holding the note*/
typedef struct NOTE
{
 int instrument;
 double start;
 double duration;
 double pitch;
 double amplitude;
 struct NOTE *chord;
} NOTE;


/*The structure for holding the part*/
typedef struct PART
{
 NOTE *partArray[SONG_LENGTH];
 struct PART *next;
```

```
} PART;


/*Variables to hold and control the parts of the piece*/
PART *bottomPart = NULL;
PART *bassLine;
PART *melody;


/*Variables to hold and use the note values for the scales*/
double cScale[6] = {0, 0.03, 0.05, 0.06, 0.07, 0.10};
double fScale[6] = {0.05, 0.08, 0.10, 0.11, 1, 1.03};
double gScale[6] = {0.07, 0.10, 1, 1.01, 1.02, 1.05};
double *currentScale;
double octave = 7;


/*Variables for the Henon Map*/
double Henon_x = 0;
double Henon_y = 0;


/*Variables for the Standard Map*/
double S_iota = STANDARD_I;
double S_theta = STANDARD_T;
double S_K = STANDARD_K;


/*This is the function to create a new note*/
NOTE *newNote()
{
 NOTE *note = (NOTE *)(malloc(sizeof(NOTE)));
 note->instrument = 0;
 note->start = 0;
 note->duration = 0;
 note->pitch = 0;
 note->amplitude = 0;
 note->chord = NULL;
 return note;
}


/*This is the function to create a new part*/
PART *newPart()
{
 int i;
 PART *part;
 PART *pointer;
 part = (PART *)(malloc(sizeof(PART)));
```

```
 for (i=0;i<SONG_LENGTH;i++)
 {
  part->partArray[i] = NULL;
 }
 part->next = NULL;

 if (bottomPart == NULL) bottomPart = part;
 else
 {
  pointer = bottomPart;
  while (pointer->next != NULL) pointer = pointer->next;
  pointer->next = part;
 }
 return part;
}


/*This is the function to print out the piece to Csound format*/
void printSong()
{
 int i,j;
 int barProgression = 1;
 NOTE *note;
 printf("%s\n", scoreFileBits);
 for (i = 0; i<(SONG_LENGTH); i++)
 {
  /*This loop adds in background chords to the piece*/
  if ((i % 32) == 0)
  {
   if ((barProgression <= 4) || (barProgression == 7) || (barProgression == 8)
   || (barProgression >= 11))
   {
    printf("i2      %d      2        1000   8.00  .2    .2     .75     .03
    5.5\n",i/8);
    printf("i2      %d      2        1000   8.04  .2    .2     .75     .03
    5.5\n",i/8);
    printf("i2      %d      2        1000   8.07  .2    .2     .75     .03
    5.5\n",i/8);
    /*printf("i2      %d      1        1000   8.00  .2    .2     .75     .03
      5.5\n",(i+2)/8);
    printf("i2      %d      1        1000   8.04  .2    .2     .75     .03
    5.5\n",(i+2)/8);
    printf("i2      %d      1        1000   8.07  .2    .2     .75     .03
    5.5\n",(i+2)/8);*/
   }
   else if ((barProgression == 5) || (barProgression  == 6) || (barProgression
   == 10))
   {
    printf("i2      %d     2        1000    8.05  .2    .2     .75     .03
```

```
   5.5\n",i/8);
   printf("i2      %d     2        1000    8.09  .2    .2     .75     .03
   5.5\n",i/8);
   printf("i2      %d     2        1000    8.12  .2    .2     .75     .03
   5.5\n",i/8);
   /*printf("i2      %d     1        1000    8.05  .2    .2     .75     .03
     5.5\n",(i+2)/8);
   printf("i2      %d     1        1000    8.09  .2    .2     .75     .03
   5.5\n",(i+2)/8);
   printf("i2      %d     1        1000    8.12  .2    .2     .75     .03
   5.5\n",(i+2)/8);*/
  }
  else
  {
   printf("i2      %d     2        1000    8.07  .2    .2     .75     .03
   5.5\n",i/8);
   printf("i2      %d     2        1000    8.11  .2    .2     .75     .03
   5.5\n",i/8);
   printf("i2      %d     2        1000    9.02  .2    .2     .75     .03
   5.5\n",i/8);
   /*printf("i2      %d     1        1000    8.07  .2    .2     .75     .03
     5.5\n",(i+2)/8);
   printf("i2      %d     1        1000    8.11  .2    .2     .75     .03
   5.5\n",(i+2)/8);
   printf("i2      %d     1        1000    9.02  .2    .2     .75     .03
   5.5\n",(i+2)/8);*/
  }

  if (barProgression == 12) barProgression = 1;
  else barProgression ++;
 }
}

/*This loop goes through the two parts and prints out all the notes*/
for(j = 0; j < 2; j++)
{
 for (i = 0;i<SONG_LENGTH;i++)
 {
  note = bottomPart->partArray[i];
  while (note != NULL)
  {
   if (note->instrument == 1)
   printf("i %d %1.2f %1.2f %1.2f %1.2f\n", note->instrument, note->start,
   note->duration, note->pitch, note->amplitude);
   else if (note->instrument == 3)
   printf("i %d %1.2f %1.2f %1.2f %1.2f\n", note->instrument, note->start,
   note->duration, note->amplitude, note->pitch);
   note = note->chord;
  }
```

```
  }
  bottomPart = bottomPart->next;
 }
 printf("e\n");
}


/*This function performs a single orbit of the Henon Map*/
void Henon(double A, double B)
{
/*Taken from Bidlack, R, Chaotic Systems as Simple (but Complex) Compositional
Algorithms, Computer Music Journal*/
 double new_x, new_y;

 new_x = Henon_y + 1.0 - (A * (Henon_x * Henon_x));
 new_y = B * Henon_x;

 Henon_x = new_x;
 Henon_y = new_y;
}


/*Thisis the function that maps the data from the henon map to notes*/
void mapHenon(PART *part, double A, double B)
{
 double notes[(SONG_LENGTH/4)]; /*This array is used to hold the notes for the
 length of the piece*/
 double min, max;               /*These are used to find the maximum and
 minimum values of the data*/
 double range;                  /*This holds the range of the values of the
 Henon Map*/
 int direction = 1;             /*This variable holds the direction the bass
 line should be walking*/
 int barCount = 0;              /*This variable is used to hold the position
 within the current bar*/
 int barProgression = 0;        /*This is used to store the progression of the
 piece through the 12 bar structure*/
 int i;                         /*Loop control variable*/
 int m = 1;                     /*Variable used to change the granularity of
 the bass line*/

 /*This function enters values from the Henon Map into the Array*/
 for (i=0; i<(SONG_LENGTH/4); i++)
 {
  Henon(A,B);
  notes[i] = Henon_x;
 }

 /*The minimum and maximum values are calculated, and the range of values found*/
```

```
min = notes[0]; max = notes[0];

for (i=0; i<(SONG_LENGTH/4); i++)
{
 if(notes[i] < min) min = notes[i];
 if(notes[i] > max) max = notes[i];
}

range = (max - min) / 7;

/*This is the main loop for processing the pitch values*/
currentScale = cScale;
for (i=0; i<(SONG_LENGTH/4); i++)
{
 if (m == 1) /*The code is only run every other iteration to make the bass
 line slower than it was*/
 {
 /*Create and set up note*/
 part->partArray[(i*4)] = newNote();
 part->partArray[(i*4)]->instrument = 1;
 part->partArray[(i*4)]->start = (i / 2);
 part->partArray[(i*4)]->duration = 2.54;
 part->partArray[(i*4)]->amplitude = 150;

 /*This controls the changes of scale throughout the 12-bar progression*/
 if(barProgression == 32) currentScale = fScale;
 else if (barProgression == 48) currentScale = cScale;
 else if (barProgression == 64) currentScale = gScale;
 else if (barProgression == 72) currentScale = fScale;
 else if (barProgression == 80) currentScale = cScale;

 /*Once the end of 12 bars is reached, reset the counter*/
 if (barProgression == 96) barProgression = 0;
 else barProgression += 2;


 /*This is what maps the pitch values to notes*/
 /*The first two clauses put in the scales root notes if it is the first beat
 of the bar*/
 if ((barCount == 0) && (direction == 1)) part->partArray[(i*4)]->pitch =
 octave + currentScale[0];
 else if ((barCount == 0) && (direction == -1)) part->partArray[(i*4)]->pitch
 = octave + currentScale[0] + 1;
 /*The rest of the clauses map the pitch value to the correct note*/
 else if (notes[i] < (min + range/2)) part->partArray[(i*4)]->pitch = octave +
 currentScale[0];
 else if (notes[i] < (min + range)) part->partArray[(i*4)]->pitch = octave +
 currentScale[1];
 else if (notes[i] < (min + 2*range)) part->partArray[(i*4)]->pitch = octave +
```

```
  currentScale[2];
  else if (notes[i] < (min + 4*range)) part->partArray[(i*4)]->pitch = octave +
  currentScale[3];
  else if (notes[i] < (min + 6*range)) part->partArray[(i*4)]->pitch = octave +
  currentScale[4];
  else if (notes[i] < (min + 6*range + range/2)) part->partArray[(i*4)]->pitch
  = octave + currentScale[5];
  else if (notes[i] < (min + 7*range)) part->partArray[(i*4)]->pitch = octave +
   currentScale[0] + 1;

  m++;
  }
  else m--;

  /*House keeping to keep track of where in the bar the program is*/
  barCount++;
  if (barCount == 8)
  {
   direction = direction * -1;
   barCount = 0;
  }
 }
}


/*This function performs a single orbit of the Standard Map*/
void Standard(double K)
{
/*Taken from Bidlack, R, Chaotic Systems as Simple (but Complex) Compositional
Algorithms, Computer Music Journal*/
 double new_iota,
 new_theta,
 TWO_PI = 6.283185;


 new_iota = S_iota + (K * sin(S_theta));
 new_theta = S_theta + new_iota;

 S_iota = new_iota;
 S_theta = new_theta;

 while (S_iota <0.0) S_iota = S_iota + TWO_PI;
 while (S_iota >= TWO_PI) S_iota = S_iota - TWO_PI;
 while (S_theta <0.0) S_theta = S_theta + TWO_PI;
 while (S_theta >= TWO_PI) S_theta = S_theta - TWO_PI;
}


/*Thisis the function that maps the data from the standard map to notes*/
```

```
void mapStandard(PART *part, double K)
{
 NOTE *note;              /*Variable to hold a note*/
 NOTE *new_Note = NULL;   /*Variable to hold a note for a harmony*/
 double min = 0,          /*Minimum value possible for Standard Map*/
 max = 6.283185;          /*Maximum value possible for Standard Map*/
 double range;   /*Range to be used for duration selection*/
 double prange;           /*Range to be used for pitch duration*/
 int currentNote = 0;     /*Variable to hold current note*/
 int barCount = 0;        /*Variable to hold the position within a bar*/
 double songPosition = 0; /*Variable to hold the position within the song*/
 double currentPosition;  /*Second variable to do the above*/
 int barProgression = 0;  /*Variable to hold where the song is in the 12-bar
 structure*/
 int progressionOffset = 0; /*Variable used to hold how many times the song has
 been through the 12-bars*/
 double original = S_K;     /*Variable to hold the systems starting values*/

 /*Calculate the range spaces*/
 range = (max - min) / 7;
 prange = (max - min) / 9;


 /*Set the current scale*/
 currentScale = cScale;

 /*Carry on for length of song*/
 while (songPosition < 1*SONG_LENGTH)
 {
  /*If 12-bars have been completed amend bar position variables*/
  if (barProgression >= (384*1))
  {
   barProgression = 0;
   progressionOffset += 384;
  }
  else barProgression = (songPosition - progressionOffset);


    /*This code will add the programmer controled structure to the music*/
    /*if (((progressionOffset / 384) == 1) && (barProgression == 256))
    {
     S_K = original;
     S_theta = original;
     S_iota  = original;
    }
    if (((progressionOffset / 384) == 2) && (barProgression == 0))
    {
     S_K = original + 0.001;
     S_theta = original + 0.001;
```

```
 S_iota  = original + 0.001;
}
else if (((progressionOffset / 384) == 3) && (barProgression == 0))
{
 S_K = original;
 S_theta = original;
 S_iota  = original;
}*/



/*Code for controlling the scale being played*/
if (barProgression >= (320*1)) currentScale = cScale;
else if (barProgression >= (288*1)) currentScale = fScale;
else if (barProgression >= (256*1)) currentScale = gScale;
else if (barProgression >= (192*1)) currentScale = cScale;
else if (barProgression >= (128*1)) currentScale = fScale;

/*Get the next orbit, do this more than once to travel through the orbits
faster*/
Standard(K); /*Standard(K); Standard(K); Standard(K);*/

/*Create new note and set standard values*/
note = newNote();
note->instrument = 3;
note->amplitude = 8000;


/*This code will being the song position onto a beat of the bar if it is
within 2 32ths of a bar of a beat*/
if (((int)songPosition % 8) >= 6) songPosition = songPosition + (8 -
((int)songPosition % 8));
else if ( ( ((int)songPosition % 8) <= 2) && ( ((int)songPosition % 8) >= 1) )
songPosition = songPosition - (8 - ((int)songPosition % 8));

/*Set the note start*/
note->start = (songPosition/8);

/*Calculate the pitch value as an offset from the last*/
if (/*S_iota*/ S_theta < (min + 1*prange))
{
 currentNote += -4;
 if (currentNote < 0)
 {
  currentNote = 6 + currentNote;
  if (octave > 6) octave--;
 }

 note->pitch = octave + currentScale[currentNote];
```

```
}
else if (/*S_iota*/ S_theta < (min + 2*prange))
{
 currentNote += -3;
 if (currentNote < 0)
 {
  currentNote = 6 + currentNote;
  if (octave > 6) octave--;
 }

 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 3*prange))
{
 currentNote += -2;
 if (currentNote < 0)
 {
  currentNote = 6 + currentNote;
  if (octave > 6) octave--;
 }

 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 4*prange))
{
 currentNote += -1;
 if (currentNote < 0)
 {
  currentNote = 6 + currentNote;
  if (octave > 6) octave--;
 }

 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 5*prange))

{
 currentNote += 1;
 if (currentNote > 5)
 {
  currentNote = -6 + currentNote;
  if (octave < 8) octave++;
 }

 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 6*prange))
{
 currentNote = currentNote;
```

```
 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 7*prange))
{
 currentNote += 2;
 if (currentNote > 5)
 {
  currentNote = -6 + currentNote;
  if (octave < 8) octave++;
 }

 note->pitch = octave + currentScale[currentNote];
}
else if (/*S_iota*/ S_theta < (min + 8*prange))
{
 currentNote += 3;
 if (currentNote > 5)
 {
  currentNote = -6 + currentNote;
  if (octave < 8) octave++;
 }

 note->pitch = octave + currentScale[currentNote];
}
else
{
 currentNote += 4;
 if (currentNote > 5)
 {
  currentNote = -6 + currentNote;
  if (octave < 8) octave++;
 }

 note->pitch = octave + currentScale[currentNote];
}


/*This is the code for harmonising with the bass line, will occur only on a
beat*/
/*if (((int)songPosition % 8) == 0)
{

 if (bassLine->partArray[(int)songPosition] != NULL)
 {
  double pitchValue;
  double bassValue;

  /*Remove the octave number from the pitch value*//*
```

```
  bassValue = bassLine->partArray[(int)songPosition]->pitch;
  bassValue = bassValue - (int)bassValue;
  pitchValue = note->pitch - (int)note->pitch;

  /*If the pitches are within range of eachother, adjust the melodies pitch*//*
  if (((bassValue - pitchValue) <= 0.05) && ((bassValue - pitchValue) >= 0.03))
   note->pitch = bassValue - 0.04;
  else if (((bassValue - pitchValue) <= 0.03) && ((bassValue - pitchValue) >=
  -0.05))
   note->pitch = bassValue + 0.04;


  /*Make sure the pitch value is legal*//*
  if (note->pitch < 1)
  {
   if (note->pitch <= -0.01) note->pitch = note->pitch + 0.12;
   else if (note->pitch >= 0.12) note->pitch = note->pitch - 0.12;

   note->pitch = note->pitch + octave;
  }
 }
}*/



 /*Calculate note duration, current position is used to hold the current
 position so songPosition can be updated*/
 currentPosition = songPosition;
 if (/*S_theta*/ S_iota < (min + range))
 {
  note->duration = 0.25;
  songPosition += 2;
 }
 else if (/*S_theta*/ S_iota < (min + 3.5*range))
 {
  note->duration = 0.5;
  songPosition += 4;
 }
 else if (/*S_theta*/ S_iota < (min + 6.33*range))
 {
  note->duration = 1;
  songPosition += 8;
 }
 else if (/*S_theta*/ S_iota < (min + 6.85*range))
 {
  note->duration = 2;
  songPosition += 16;
 }
```

```
   else
   {
    note->duration = 3;
    songPosition += 24;
   }

   /*Code to add harmonies within the melody*/
   /*if (((int)songPosition == songPosition) && (((int)songPosition % 24) == 0))
   {
    chord = 0;
    new_Note = newNote();
    new_Note->instrument = note->instrument;
    new_Note->start = note->start;
    new_Note->duration = note->duration;
    new_Note->pitch = note->pitch;
    new_Note->amplitude = note->amplitude;
    new_Note->pitch += 0.03;
   }*/




  /*Code to add the notes into the part array*/
  if (part->partArray[(int)currentPosition] == NULL) part->partArray[
  (int)currentPosition] = note;
  else
  {
   note->chord = part->partArray[(int)currentPosition];
   part->partArray[(int)currentPosition] = note;
  }


  if(new_Note != NULL)
  {
   if (part->partArray[(int)currentPosition] == NULL) part->partArray[
   (int)currentPosition] = new_Note;
   else
   {
    new_Note->chord = part->partArray[(int)currentPosition];
    part->partArray[(int)currentPosition] = new_Note;
    new_Note = NULL;
   }
  }
 }
}
```

```
int main()
{
 int i;
 bassLine = newPart();
 melody = newPart();

 /*Iterate the Henon Map to pass the transient phase*/
 for (i=0; i<200; i++) Henon(HENON_A, HENON_B);

 /*Generate bass line*/
 mapHenon(bassLine,HENON_A, HENON_B);

 /*Generate Melody*/
 mapStandard(melody, STANDARD_K);

 printSong();
}
```

7.2 Questionnaire Results

	Question Number																	
Participent	1	2	3_1	3_2	4_1	4_2	5_1	5_2	6_1	6_2	7_1	7_2	8_1	8_2	9_1	9_2	10_1	10_2
A	3	2	5	3	2	2	5	3	4	3	6	4	5	7	3	2	Y	Y
B	2	2	2	2	2	2	3	3	5	5	5	6	1	1	1	4	Y	Y
C	2	1	1	5	1	3	1	4	2	5	1	5	2	4	2	5	N	Y
D	3	1	6	8	3	3	6	7	6	7	6	6	5	7	5	7	Y	Y
E	5	3	2	7	1	3	2	6	4	5	2	6	4	7	1	6	N	Y
F	4	3	2	7	1	2	2	6	5	6	5	6	4	7	1	7	N	N
G	4	2	1	4	1	2	2	4	1	4	1	5	3	3	1	4	N	N
H	1	1	3	4	3	3	5	6	5	6	6	7	3	4	3	4	N	N
I	5	2	2	2	1	2	1	2	3	3	1	1	2	1	1	1	N	N
J	1	1	6	7	2	3	5	6	6	7	4	5	8	7	5	7	Y	Y

Figure 7.1:

7.2.1 Comments

Person B

Timing. There is a note at every point required throughout the bar in a steady fashion without gaps. I think this is what makes it feel a little lifeless and timing is why a beginner sounds like a beginner!

The first sounds like it's hitting flat notes but is more inventive.

Person C

The first one is quite painful to listen to but the second one is much more realistic and enjoyable.

Person E

Output 1 does essentially seem a sequence of random notes - quite a few are "wrong" ones (as far wrong as one can be in an improvisatory sense), and it doesn't really seem to flow, or have a discernable tune to it. Output 2 on the other hand is better. There are some nice moments in 2 - the minor 3rd to root "dee-doo-dee-doo" on the 5th bar of the 1st chorus is cool, sounds almost like something a human might play at that point. There are a couple of other points like that in the second one as well, where what the computer comes up which sound almost like someone with a rudimentary/intermediate knowledge of improvisation might play. A quite a lot of the phrases do have a melodic shape to them, and it's generally much more listenable to.

I've given my answers and comments in the context that it's a computer you've programmed to improvise (and hats off to you!). Obviously on a scale that went from your first example at 1 all the way up to top pro jazz musicians at 10, the second would only score about 1.5 - but that sort of bias obviously not going to help your survey very much!!

Person F

if you eliminated the notes which sound "out of tune" youd have a very acceptable basic blues improvisation in output2 in my opinion, and to say you programmed a computer to improvise it... thats a top effort

Person I

Output 1: If the rhythm were swung it might sound merely like a very 'kooky' sort of solo, although there's no sort of groove to it at all, even from one note to the next. The notes are computer generated but also played by a computer; if they were played by a good human musician maybe it would sound much better than the awful, non-descript computer sound on this recording.

Output 2: The melody seems slightly more structured here, but the rhythm is worse than the first: it's far too regular and un-syncopated. There also seem to be rather odd random chords throughout which sound like someone accidentally hitting them on a keyboard, and which are distracting.

Bibliography

[1] R Bidlack. Chaotic systems as simple (but complex) compositional algorithms. *Computer Music Journal*, 16(3):33–47, 1992.

[2] R Boulanger. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing,and Programming*. The MIT Press, 2000.

[3] D J Burr and Y Miyata. Hierarchical recurrent networks for learning musical structure. In *Neural Networks for Signal Processing [1993] III. Proceedings of the 1993 IEEE-SP Workshop*, pages 216–225, 1993.

[4] A R Burton and T Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Jounal*, 23(4):59–73, 1999.

[5] D Cope. One approach to musical intelligence. *Intelligent Systems, IEEE [see also IEEE Expert]*, 14(3):21–25, 1999.

[6] Alfonso Ortega de la Puente, Rafael Sanchez Alfonso, and Manuel Alfonseca Moreno. Automatic composition of music by means of grammatical evolution. In *APL '02: Proceedings of the 2002 conference on APL*, pages 148–155. ACM Press, 2002.

[7] R L Devany. *A First Course in Chaotic Dynamical Systems*. Addison-Wesley Publishing Company, 2003.

[8] S Dubnov, G Assayag, O Lartillot, and G Bejerano. Using machine-learning methods for musical style modeling. *Computer*, 36(10):73–80, 2003.

[9] D Eck and J Schmidhuber. Finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 747–756, 2002.

[10] P M Gibson and J A Byrne. Neurogen, musical composition using genetic algorithms and cooperating neural networks. In *Artificial Neural Networks, 1991., Second International Conference on*, pages 309–313, 1991.

[11] J Leach and J Fitch. Nature, music and algorithmic composition. *Computer Music Jounal*, 19(2):22–33, 1995.

[12] M Marques, V Oliveira, S Vieira, and A C Rosa. Music composition using genetic evolutionary algorithms. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 714–719, 2000.

[13] James Anderson Moorer. Music and computer composition. *Commun. ACM*, 15(2):104–113, 1972.

[14] S W Smoliar. Algorithms for musical composition: a question of granularity. *Computer*, 24(7):54–56, 1991.

[15] Belinda Thom. Bob: an interactive improvisational music companion. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 309–316. ACM Press, 2000.

[16] I Xenakis. *Formalised Music*. Indiana University Press, 1971.