

On the evolution of music in a society of self-taught digital creatures

Eduardo Reck Miranda

University of Plymouth, UK

E.Miranda@plymouth.ac.uk

Abstract

We are investigating the potential of artificial life (Alife) models for musical creativity. This paper begins with a brief introduction to our research scenario and then revisits two cellular automata-based generative music systems of our own design: one for synthesising sounds (Chaosynth) and another for generating musical passages (CAMUS). Then, it presents the preliminary results of ongoing research aimed at the design of models to study the origins and evolution of music. Here we introduce a model where a society of distributed and autonomous but co-operative agents evolves repertoires of short melodies from scratch, by interacting with one another. Instead of mapping the behaviour of specific Alife algorithms onto sound or music, as in Chaosynth and CAMUS, this new research addresses the possibility of implementing collective generative music systems, using virtual musicians who learn how to compose and play by themselves and with people interacting with them. We demonstrate by means of a concrete example the important role of mimetic interactions for creativity in a society of software agents furnished with a vocal synthesiser, a hearing apparatus and a simple brain capable of making association between motor and perceptual parameters.

Keywords: artificial life, cellular automata, computer music, computer-aided musical creativity, evolutionary music.

1 Introduction

Perhaps one of the greatest achievements of artificial intelligence (AI) to date lies in the construction of machines that can compose music of incredibly high quality; e.g. (Cope 1991). These AI systems (Miranda 2000) are only as good, however, as the data they are trained on. They are either hard-wired to compose in a certain style or able to learn how to imitate a style by looking at patterns in training examples. Conversely, issues as to whether computers can create new kinds of music are much harder to study, because in such cases the computer should neither be embedded with particular models at the outset, nor learn from carefully selected examples. One plausible approach to address this problem is to program the computer with abstract models that embody our understanding of the dynamics of some compositional processes. Indeed, many composers have tried out mathematical models, which were thought to embody musical composition processes, such as combinatorial systems, stochastic models and fractals (Dodge 1985, Worrall 1996, Xenakis 1971). Some of these trials produced interesting music and much has been learned about using mathematical formalisms and computer models in composition. We believe artificial life (or a-life) modelling techniques to be a natural progression to pushing this understanding even further. By *a-life models*, we mean those computational models that display some form of emergent behaviour resembling natural phenomena; for example, cellular automata, genetic algorithms and adaptive games, to cite but a few (Kelemen and Sosik 2001).

There is a growing number of people investigating the potential of a-life algorithms for musical composition, notably L-systems (McCormack 1996) and genetic algorithms (Malt 2001), to cite but two examples.

We have been investigating the potential of a class of a-life models called cellular automata (CA) for music composition for almost a decade, but we have recently shifted our attention towards models inspired by the notion of adaptive distributed-agent systems. This paper begins with a brief review of two representative systems resulting from our research into cellular automata: a software synthesiser called *Chaosynth* and a generator of musical passages called *CAMUS*. Then, we discuss the potential and limitations of these two systems based upon the experience that we have gained from using them in professional composition. In the light of this short discussion, we introduce one of the new models that we have implemented to study the evolution of melodies in a society of distributed software agents. These models are aimed at a theoretical framework for studying the origins and evolution of music based upon the notion that music is an adaptive complex dynamic system. In simple terms, adaptive complex dynamic systems emerge from the overall behaviour of interacting autonomous elements in a non-hierarchical manner; that is, there is no central control driving these interactions. In our case, these individual elements are modelled as software agents geared to interact co-operatively with one another in a virtual society, under specific environmental conditions. We are particularly interested in establishing the fundamental properties and mechanisms that these agents, the interactions and the environment must possess in order to create music. The rationale here is that if we furnish these agents with proper cognitive and physical abilities, combined with appropriate interaction dynamics and adequate environmental conditions, then the agents should be able to evolve realistic musical cultures. The pressing questions that this research is addressing therefore are:

What are the appropriate agent interactions and what motivates them? What are the proper cognitive and physical abilities of these agents? What environmental constraints are necessary to foster music evolution?

As with the fields of psychoacoustics (Howard and Angus 1996, Cook 1999) and Artificial Intelligence (Balaban et al. 1992, Miranda 2000) which have greatly contributed to our understanding of music, we believe that the evolutionary simulations that we are experimenting with have the potential to reveal new fundamental aspects of our musical creativity that are waiting to be unveiled. From a pragmatic perspective, we are interested in developing new technology for implementing autonomous generative music systems. Instead of modelling specific styles of music or compositional processes, we are looking for ways to implement collective music-making systems, using virtual musicians who learn how to compose and play by themselves and with people interacting with them.

2 Cellular automata music investigation

Cellular automata (CA) are discrete dynamical systems often described as a counterpart to partial differential equations, which have the capability to describe continuous dynamical systems (Wolfram 1984). The meaning of discrete is that space, time and properties of the automata can have only a finite, countable number of states. The basic idea is not to try to describe a complex system using difficult equations, but rather to simulate this system by the interaction of its components following simple rules. CA are implemented as an array of identically programmed automata, or cells, that interact with one another. The arrays usually form either a 1-dimensional string of cells, a 2-D grid, or a 3-D solid. Most often the cells are arranged as a simple regular grid or matrix, but other arrangements, such as a honeycomb, are sometimes used. The essential features of CA are: the state of the cells and their neighbour-

hood. The state of a cell can be either a number or a property. For instance if each cell represents part of a landscape, then the state might represent the number of animals at each location or the type of forest cover growing there. The neighbourhood is the set of cells that a single cell interacts with. In a grid these are normally the cells physically closest to the cell in question.

2.1 From sound synthesis...

Chaosynth, our cellular automata audio synthesiser, is essentially a granular synthesiser (Miranda 2002). Granular synthesis works by generating a rapid succession of very short sound bursts called grains (e.g. 35 milliseconds long) that together form larger sound events. Whereas most granular synthesis systems so far have used stochastic methods to control the production of the grains (for example, to control the waveform and the duration of the individual grains), *Chaosynth* uses CA.

The automaton used in *Chaosynth* is an adapted version of a cellular automaton intro-

duced by Dewdney (1988): it tends to evolve from an initial random distribution of cell values on the grid, towards an oscillatory cycle of patterns. This automaton can be thought of as a grid of cells representing identical electronic circuits. At a given moment, cells can be in any one of the following conditions: quiescent, depolarised or burned. A cell interacts with its neighbours (4 or 8 nearest cells) through the flow of electric current between them. There are minimum (V_{min}) and maximum (V_{max}) threshold values that characterise the condition of a cell. If its internal voltage (V_i) is under V_{min} , then the cell is quiescent (or polarised). If it is between V_{min} (inclusive) and V_{max} values, then the cell is being depolarised. Each cell has a potential divider, which is aimed at maintaining V_i below V_{min} . But when it fails (that is, if V_i reaches V_{min}) the cell becomes depolarised. There is also an electric capacitor, which regulates the rate of depolarisation. The tendency, however, is to become increasingly depolarised with time. When V_i reaches V_{max} , the cell fires and becomes burned. A burned cell at time t is automatically replaced by a new quiescent cell at time $t + 1$.

Each configuration of cells c of the CA evolution in time t controls the frequency and duration values of each grain (Figure 1); the amplitude values are given via another procedure. The values (i.e. the colours) of the cells are associated to frequencies and oscillators are associated to a number of cells. The frequencies of the components of a grain at time t are established by the arithmetic mean of the frequencies associated with the values of the cells associated with the respective oscillators. Suppose, for example, that each oscillator is associated with 9 cells and that at a certain time t , 3 cells correspond to 110 Hz, 2 to 220 Hz and the other 4 correspond to 880 Hz. In this case, the mean frequency value for this oscillator at time t will be 476.66 Hz. The user can also specify the dimension of the grid, the amount of oscillators, the allocation of cells to oscillators, the allocation of frequencies to CA values, and various other CA-related parameters. An

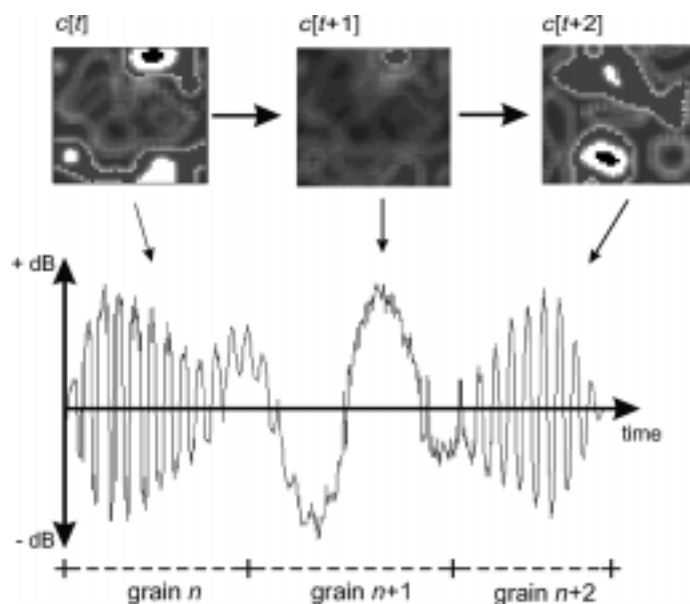


Figure 1. Each screen of the CA evolution controls the production of an individual granule.

example of a grid of 400 cells allocated to 16 oscillators of 25 cells each is shown in Figure 2.

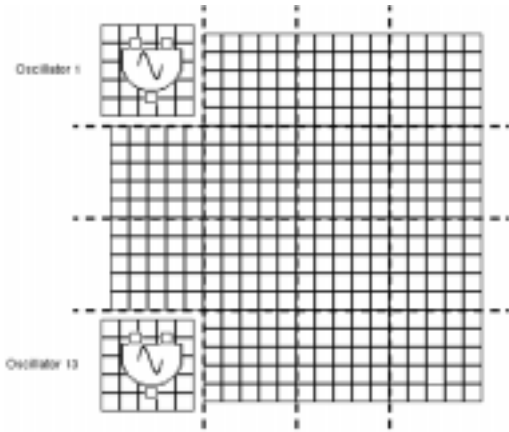


Figure 2. An example of a grid of 400 cells allocated to 16 digital oscillators.

The duration of a whole sound event is determined by the number of CA generations and the duration of the individual grains; for example, 100 iterations of 35 millisecond particles result in a sound event of 3.5 seconds of duration. More details about *Chaosynth* and references to related work can be found in a paper that appeared in *Leonardo* (Miranda 1995).

2.2 ... to musical forms

CAMUS uses two simultaneous CA to generate musical passages, or 'forms', in MIDI format: the *Game of life* and *Demon cyclic space*. Due to limitations of space, we will briefly introduce only the role of the *Game of life* (Gardner 1971) in the generative process. More information on *CAMUS* and references to related work can be obtained in a paper that appeared in the *Computer Music Journal* (McAlpine et al. 1999) or in this author's book *Composing music with computers* (Miranda 2001a). The *Game of life* is an attempt to model a colony of simple virtual organisms. The CA are defined as a matrix of

cells, each of which can be in one of two possible states: alive represented by the number one (coloured in black), or dead represented by the number zero (coloured in white) (Figure 3). The state of the cells as time progresses is determined by the state of the eight nearest neighbouring cells. There are essentially four rules that determine the fate of the cells of the *Game of life* CA:

Birth. A cell that is dead at time t becomes alive at time $t + 1$ if exactly three of its neighbours are alive at time t ;

Death by overcrowding. A cell that is alive at time t will die at time $t + 1$ if four or more of its neighbours are alive at time t ;

Death by exposure. A cell that is alive at time t will die at time $t + 1$ if it has one or none live neighbours at time t ;

Survival. A cell that is alive at time t will remain alive at time $t + 1$ only if it has either two or three live neighbours at time t .

A number of alternative rules can set, but not all of them produce interesting emergent behaviour.

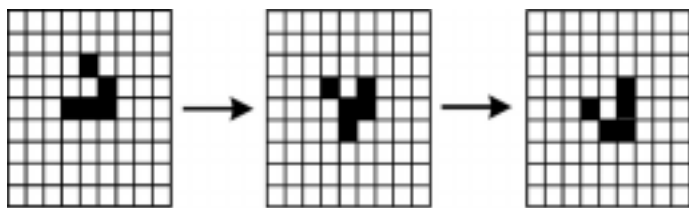


Figure 3. Game of Life in action.

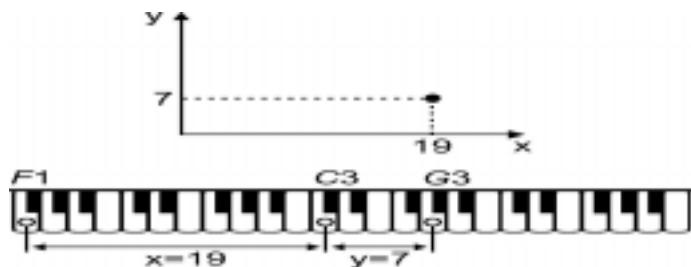
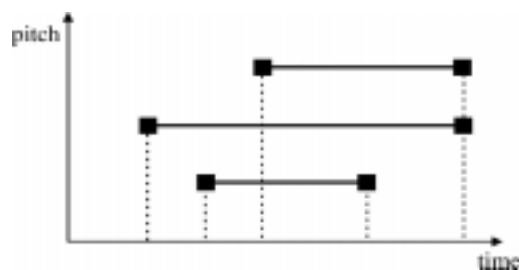


Figure 4. *CAMUS* uses a Cartesian model in order to represent a triple of notes.

We devised a Cartesian model to represent a triple of notes; that is, an ordered set of three notes that may or may not sound simultaneously. These three notes are defined in terms of the intervals between them. Given a starting note, the horizontal co-ordinate of the model represents the first interval of the triple and the vertical co-ordinate represents its second interval (Figure 4).

To begin the generative music process, the CA is set up with an initial random configuration of cell values and let to run. When the algorithm arrives at a live cell, its co-ordinates are taken to estimate the triple from a given lowest reference note. For example, if a cell at the position (19, 7) is alive, the co-ordinates (19, 7) describe the intervals of a triple of notes: a fundamental pitch is given, then the next note will be at 19 semitones above the fundamental and the last note 26 semitones above the fundamental (Figure 4). Although the cell updates occur at each time step in parallel, *CAMUS* plays the live cells column by column, from top to bottom. Each of these musical cells has its own timing, but the notes within a cell can be of different lengths and can be triggered at different times. Once the triple of notes for each cell has been determined, the states of the neighbouring cells are used to calculate a timing template, according to a set of temporal codes. More information about this temporal representation can be found in a paper that appeared in *Interface* (now called *Journal of New Music Research*) (Miranda 1993). As a brief example, if

Figure 5. An example of a template for the organisation of a cell's triplet. The horizontal axis represents time and the vertical axis pitch.



we assume that Figure 5 portrays the temporal template for a certain live cell (5, 5), then a musical passage that could be generated by this cell is given in Figure 6.



Figure 6. A musical passage generated by a single cell using the template portrayed in Figure 4.

2.3 Brief discussion of the results

A number of professional pieces were composed using *CAMUS*-generated material, such as *Entre o Absurdo e o Mistério* (Miranda 2001b), for chamber orchestra, the second movement of the string quartet *Wee Batucada Scotica* (Miranda 1998) and more recently *Grain Streams*, for (acoustic) piano, recorded electroacoustics, and live effects. *Chaosynth* has also proved to be a successful system in the sense that it is able to synthesise a large amount of unusual sounds that are not normally found in the real acoustic world, but nonetheless sound pleasing to the ear. As an example of an electroacoustic piece composed using the sounds of *Chaosynth* we cite *Olivine Trees*, which was awarded a bronze medal at the International Luigi Russolo Electroacoustic Music Competition in 1998, in Italy. The results of the CA experiments are very encouraging, as they are good evidence that both musical sounds and abstract musical forms might indeed share similar organisational principles with cellular automata, most notably the principles of pattern propagation and variation of musical forms. In general, we found that *Chaosynth* produced more interesting results than *CAMUS*. We think that this might be due to the very nature of the phenomena in question. The inner structures of sounds seem

more susceptible to CA modelling than large musical structures. As music is primarily a cultural phenomenon, we have come to the conclusion that it is necessary to take into account the dynamics of social formation and cultural evolution when designing generative music systems. We are currently devising a number of experiments aimed at models that take these aspects into account. The remainder of this paper introduces an experiment whereby a small community of digital creatures evolve a common repertoire of melodic patterns from scratch by interacting with one another.

CAMUS is available on the CD-ROM of the book *Composing music with computers* cited earlier and *Chaosynth* is commercially available from Nyr Sound in the UK. A demonstration version is available on the CD-ROM of the book *Computer sound design: synthesis techniques and programming* (Miranda 2002b).

3 Breeding digital cultures

We propose a mimetic model as an attempt to demonstrate that a small community of interactive distributed agents furnished with appropriate motor, auditory and cognitive skills can evolve a shared repertoire of melodies, or tunes, from scratch after a period of spontaneous creation, adjustment and memory reinforcement. Complementary information about this model can be found in a previous paper (Miranda 2002a). By way of related work, this model shares various features with models proposed by researchers in the field of evolutionary linguistics (Cangelosi and Parisi 2001).

The motivation of the agents is to form a repertoire of tunes in their memories and foster social bonding. In order to be sociable, an agent must form a repertoire that is similar to the repertoire of its peers. Sociability is therefore assessed in terms of the similarity of the agents' repertoires. In addition to the ability to produce and hear sounds, the agents are born with a basic instinct: to imitate what they hear.

3.1 The agents

The agents are equipped with a voice synthesiser, a hearing apparatus, a memory device and an enacting script. The voice synthesiser is essentially implemented as a physical model of the human vocal mechanism, but we scaled down its complexity in order to render our initial experiments simple. In short, the agents need to compute two parameters in order to produce a sound: $_n$ and $t(n)$, where $_n$ controls the pitch of the sound and $t(n)$ gives its triggering time; n corresponds to the ordering of these parameters in a sequence. As for the hearing apparatus, it employs short-term autocorrelation-based analysis to extract the pitch contour of a spoken signal. The algorithm features a parameter $_$ that regulates the degree of attention of the hearing apparatus, by controlling the resolution of the short-term autocorrelation analysis (Miranda 2001c). This resolution defines the sensitivity of the auditory perception of the agents.

The agent's memory stores its sound repertoire and other data such as probabilities, thresholds and reinforcement parameters. In the present simulation, the memory holds values for the following parameters (their meaning will be clarified as we go along):

P_a = creative willingness;
 P_b = forgetfulness disposition;
 H_T = bad history threshold;
 H^s = success history threshold;
 E_T = erase threshold;
 R^r = reinforcement threshold;
 J_T = jump threshold;
 $_$ = motor deviation coefficient;

$_$ = degree of attention (for the autocorrelation algorithm);

L_{\min} = minimum length (in terms of number of pitches in the tune) and

L_{\max} = maximum length.

An agent processes and stores tunes in terms of synthesis and analysis parameters. They have a dual representation of tunes in their memories: a

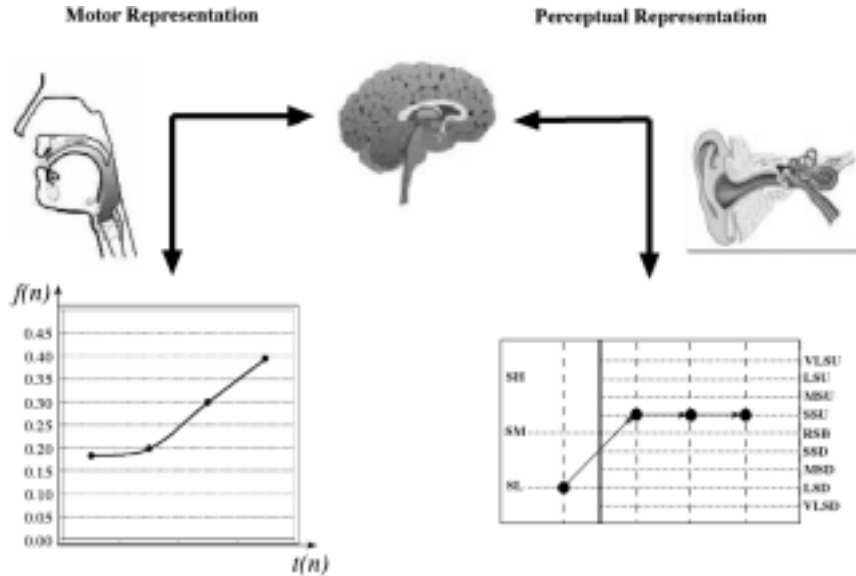


Figure 7. A motor representation example (left) and its corresponding perceptual representation (right). Imitation is defined as the task of hearing a tune and activating the motor system to reproduce it.

motor map (synthesis) and a perceptual representation (analysis). The motor representation is in terms of a function of motor (i.e. synthesis) parameters $f(n)$ and $t(n)$ (Figure 7) and the perceptual representation is in terms of the CARMEC representation scheme (refer to Appendix I).

Imitation is defined as the task of hearing a tune and activating the motor system to reproduce it. When we say that the agents should evolve a shared repertoire of tunes, we mean that the perceptual representation in the memory of the agents of the community should be identical, but the motor representation may be different.

3.2 The enacting script

The enacting script provides the agent with knowledge of how to behave during the interactions: the agent must know what to do when another agent produces a tune, how to assess an imitation, when to remain quiet, and so forth. The enacting script does not evolve in the present model; all agents are alike in this respect. Also, all agents have identical synthesis and listening apparatuses. Please refer to Appendix II for the main algorithm of the enacting script.

At each round, each of the agents in a pair from the community plays one of two different roles: the *agent-player* and the *agent-imitator*. The agent-player starts the interaction by producing a tune p_r , randomly chosen from its repertoire. If its repertoire is empty, then it produces a random tune. The agent-imitator

then analyses the tune p_r , searches for a similar tune in its repertoire, i_n , and produces it. The agent-player in turn analyses the tune i_n and compares it with all other tunes in its own repertoire. If its repertoire holds no other tune p_n that is more perceptibly similar to i_n than p_r is, then the agent-player replays p_r as a reassuring feedback for the

agent-imitator; in this case the imitation would be acceptable. Conversely, if the agent-player finds another tune p_n that is more perceptibly similar to i_n than p_r is, then the imitation is unsatisfactory and in this case the agent-player would halt the interaction without emitting the reassuring feedback; the agent-imitator realises that no feedback means imitation failure. In all these cases, the agents always add a small deviation to the motor realisation of the tunes they know; the amount of deviation is determined by the coefficient D . If the agent-imitator hears the reassuring feedback, then it will reinforce the existence of i_n in its repertoire and will change its perceptual parameters slightly in an attempt to make the tune even more similar to p_r (i.e. only if they are not already identical). In practice, the reinforcement is implemented as a counter that registers how many times a tune has successfully been used.

Conversely, if the agent-imitator does not receive the feedback then it will infer that something went wrong with its imitation. In this case, the agent has to choose between two potential courses of action. If it finds out that i_n is a weak tune (i.e. bad past success rate) in its memory, because it has not received enough reinforcement in the past, then it will try to modify its motor representation of i_n slightly, as an attempt to further approximate it to p_r . It is hoped that this approximation will give the tune a better chance of success if it is used again in another round. But if i_n is a strong tune (i.e.

good past success rate), then the agent will leave it untouched (because it has been successfully used in previous imitations and a few other agents in the community also probably know it), create a new tune that is similar to p_r , and include it in its repertoire. The new tune is created as follows: the agent produces a number of random tunes and then it picks the one that is perceptually most similar p_r to include in the repertoire. In case of failure, no reinforcement is applied to i_n . Before terminating the round, both agents perform final updates. Firstly they scan their repertoire and merge those tunes that are considered to be perceptibly identical to each other. Also, at the end of each round, both agents have a certain probability P_b of undertaking a spring-cleaning to get rid of weak tunes; those tunes that have not been sufficiently reinforced are forgotten. Finally, at the end of each round, the agent-imitator has a certain probability P_a of adding a new randomly created tune to its repertoire (Figure 8).

Bear in mind that an important presupposition in this model is that the action of singing tunes involves the activation of certain vocal motor mechanisms in specific ways. The recognition of tunes here thus requires knowledge of the activation of the right motor sequence (i.e. synthesis parameter values) in order to reproduce the tune in question. It does not matter, however, if one agent uses different values to its peers in order to produce identical tunes.

3.3 Example

In order to evaluate and discuss the results of our simulations, let us consider a simple case study whereby a society of 5 agents performed 5000 interactions, with the following settings:

P_a = probability of 0.5%;

P_b = probability of 20%;

H_r = 0.1% of total amount of interactions of the round;

H^r = 90% of interactions so far;

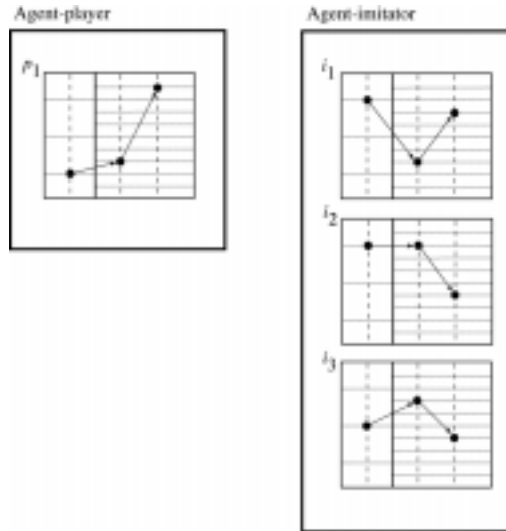


Figure 8. A case where the agent-player has only one melody in its repertoire whereas the agent-imitator has three. Since there is only one melody in the repertoire of the agent-player, any tune played by the agent-imitator will be considered to be an acceptable imitation, even though it might sound very different to an external observer. As far as this agent is concerned, both tunes are similar because it does not have yet the ability to distinguish between tunes.

E_r = 1% of total amount of interactions of the round;

R^r = 95% of interactions so far;

J_r = 0.25 in a scale from 0.0 to 1.0;

D = +/* 1%;

L_{\min} = 3 and

L_{\max} = 3.

That is, there is a 0.5% probability that an agent will insert a new random intonation in its repertoire after an interaction and a 20% probability that this agent will scan its repertoire in order to delete intonations that did not score well so far: if an intonation has been used for more than E_r times and has scored less 90% of the interactions so far (H^r), then it is deleted. In

case of an unsuccessful imitation, the agent-imitator checks whether the failed intonation has been already used at least H_t times and scored well for at least 95% of the interactions so far (R^*); if not, then the agent will attempt to correct the pattern. In a scale from 0.0 to 1.0, a pitch interval that is lower than 0.25 is considered as a small step, otherwise it is considered either medium, large and so forth (see Appendix D), according to a given perceptual equation. Finally, the motor deviation is set to $\pm 1\%$ of its current value and the length of the evolved melodies is fixed to 3 pitches.

The graph in Figure 9 shows the evolution of the average repertoire of the community, with snapshots taken after every 100 interactions. The agents quickly increase their repertoire to an average of between six and eight tunes per agent. At about 4000 interactions, more tunes appear, but at a lower rate. Identical behaviour has been observed in many such simulations with varied settings. The general tendency is to quickly settle into a repertoire of a certain size, which occasionally increases at lower rates. The pressure to increase the repertoire is mostly due to the creativity willingness parameter combined with the rate of new inclusions due to imitation failures. In this case the repertoire settled to 8 tunes between 1600 and 4000 interactions.

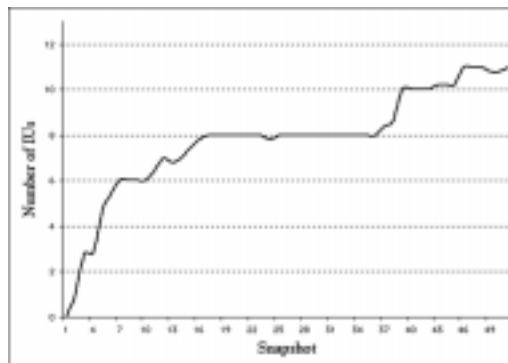


Figure 9. *The evolution of the average size of the repertoire of the whole community.*

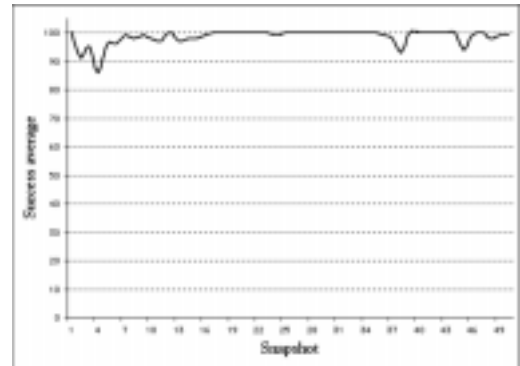


Figure 10. *The imitation success rate over time.*

The graph in Figure 10 plots the imitation success rate of the community, measured at every 100 interactions. Notice that the success rate drops within the first 1000 interactions, which coincides with the steep rising curve of the graph in Figure 9. This is the period in which the agents are negotiating how their repertoires should look in order to foster communication; this period is characterised by inclusions of tunes due to imitation failure and by motor adjustments due to imitation successes. At approximately 1800 interactions, the imitation rate goes back up to 100%. Then, occasional periods of lower success rate occur due to the appearance of new random tunes or eventual motor-perceptual inconsistencies that might be caused by pattern approximations. Although the repertoire tends to increase with time, the success rate stays consistently high. This is good evidence that the community does manage to foster social bonding. How about the other goal of the agents? Did they evolve a shared repertoire of tunes?

The answer is yes. Figures 11a and 11b portray the perceptual memory of two agents randomly selected from the community, after 5000 interactions. The repertoire of all five agents plotted on top of each other is shown in Figure 11c. Figure 11c demonstrates that at least

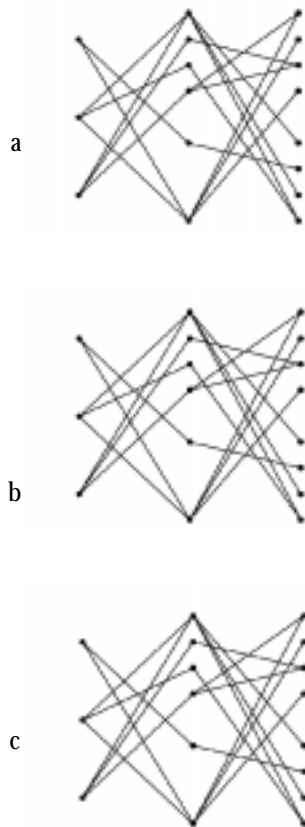


Figure 11. *The perceptual memory of the agents. For the sake of clarity, the background metrics and labels of the graph are not shown; refer to Appendix I.*

two agents (the ones whose memories are plotted in Figures 11a and 11b) share identical tunes with the whole community. Figures 12a and 12b plot the motor maps of two agents, also randomly selected from the community. Although both maps give an overall impression of similarity, the actual curves are slightly different. A better assessment can be made if both maps are plotted on the top of each other, as shown in Figure 12c. This figure is a concrete example of a case where different motor maps yield the same perceptual representation; that is, the two agents in question, randomly selected, have identical perceptual representations.

3.4 Discussion

What can we learn from this model? In the context of our research, it demonstrates that imitation is important for the evolution of music, more specifically melodic patterns. But

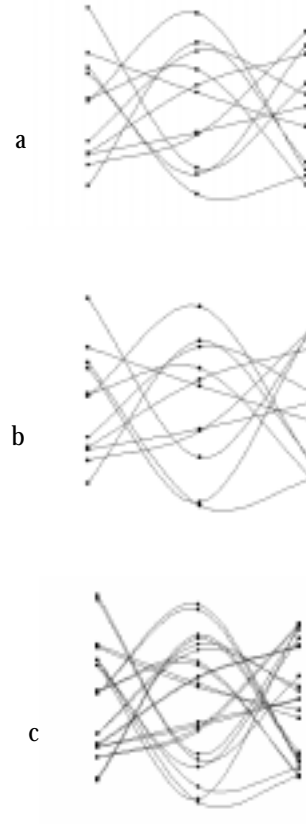


Figure 12. *The motor memories of the agents. For the sake of clarity the figure omits background metrics and labelling; only curves of the motor functions are displayed.*

what makes this model interesting is the fact that there is no global memory holding the repertoire of the community. The agents are autonomous individuals, each of them has its own memory and, most importantly, no other agent can 'see' what is stored in there. The only way to access the sound repertoire of a community is by considering the memories of all agents put together; e.g. Figure 10c. Moreover, the repertoire of melodies emerges from the interactions of the agents, but there is no global procedure supervising or regulating them. The actions of each agent are solely based upon its own internal rules. We do not assume the existence of a one-to-one mapping between perception and production. Although in this model we forged one-to-one mapping, it is clear that the agents learn by themselves how to correlate perception parameters (analysis) with production (synthesis) ones. The agents do not necessarily need to build the same motor

representations for what is considered to be perceptibly identical.

But why are such features important for generative music? Perhaps the breakthrough of this work is that we introduced the notion of active collective machine learning to generative music systems. Instead of modelling specific musical styles or compositional processes, this model is a first step towards the implementation of collective music-making systems whereby virtual musicians learn how to compose and play by themselves, and eventually with people interacting with them.

From a composer's point of view, although our agents are capable of evolving repertoires of simple melodies, we can hardly say that they can compose music at this stage. We will only be able to maintain that such a system can actually create music when more complex models for evolving multi-levelled generative procedures have successfully been implemented. There are still, however, a number of questions that need to be addressed before we set out to design such complex models. For example, we do not know what would happen if we gave the agents the choice of a sound-making device other than the voice, and/or created agents with diverse cognitive skills. Also, we need to consider that the agents should also be able to evolve (e.g. grow up, get smarter, get older) and therefore the enacting script should evolve as well.

References

- Balaban, M., Ebcioğlu, K. and Laske, O. (eds.) (1992) *Understanding music with AI: perspectives on music cognition*. The MIT Press, Cambridge, MA.
- Cangelosi, A. and Parisi, D. (eds.) (2001) *Simulating the evolution of language*. Springer Verlag, London.
- Cook, P. (ed.) (1999) *Music, cognition, and computerized sound*. The MIT Press, Cambridge, MA.
- Cope, D. (1991) *Computers and musical style*. Oxford University Press, Oxford.
- Dewdney, A. K. (1988) Computer recreations: the hodgepodge machine makes waves. *Scientific American* 251(8) 86–89.
- Dodge, T. and Jerse, T. A. (1985) *Computer music: synthesis, composition and performance*. Schirmer Books, New York, NY.
- Gardner, M. (1971) On cellular automata, self-reproduction, the garden of Eden and the game 'life'. *Scientific American* 224(2) 112–117.
- Howard, D. M. and Angus, J. (1996) *Acoustics and psychoacoustics*. Focal Press, Oxford.
- Kelemen, J. and Sosik, P. (eds.) (2001) *Advances in artificial life*. Lecture notes in artificial intelligence 2159. Springer, Berlin.
- McAlpine, K. B., Miranda, E. R., Hoggar, S. (1999) Making music with algorithms: a case study system. *Computer Music Journal* 23(2) 19–30.
- McCormack, J. (1996) Grammar-based music composition. In Stocker, R. et al. (eds) *Complex systems 96: from local interactions to global phenomena*. ISO Press, Amsterdam, pp. 321–336.
- Malt, M. (2001) In vitro – growing an artificial musical society. *Proceedings of the Workshop on Artificial Life Models for Musical Applications*. 6th European Conference on Artificial Life 2001. Editoriale Bios, Cosenza, Italy, pp. 29–36.
- Miranda, E. R. (1993) Cellular automata music: an interdisciplinary project. *Interface* 22(1) 3–21.
- Miranda, E. R. (1995) Granular synthesis of sounds by means of cellular automata. *Leonardo* 28(4) 297–300.
- Miranda, E. R. (1998) *Wee Batucada Scotica* (musical score). Goldberg Edições Musicais, Porto Alegre.
- Miranda, E. R. (ed) (2000) *Readings in music and artificial intelligence*. Harwood Academic Publishers, Amsterdam.
- Miranda, E. R. (2001a) *Composing music with computers*. Focal Press, Oxford.
- Miranda, E. R. (2001b) *Entre o Absurdo e o Mistério* (musical score). Goldberg Edições Musicais, Porto Alegre.
- Miranda, E. R. (2001c) Synthesising prosody with variable resolution. *Proceedings of 110th AES Convention* (CD-ROM), Audio Engineering Society.
- Miranda, E. R. (2002a) Emergent sound repertoires in virtual societies. *Computer Music Journal* 26(2) 77–90.
- Miranda, E. R. (2002b) *Computer sound design: synthesis techniques and programming*. 2nd edition. Focal Press, Oxford.

- Wolfram, S. (1984) Cellular automata as models of complexity. *Nature* 311(October) 419–424.
- Worrall, D. (1996) Studies in metamusical methods for sound image and composition. *Organised Sound* 1(3) 183–194.
- Xenakis, I. (1971) *Formalized music*. Indiana University Press, Bloomington, IN.

Eduardo Reck Miranda graduated in computing at UNISINOS and studied music at UFRGS, both in Brazil. He has an MSc in Music Technology from the University of York and a PhD in Music from the University of Edinburgh. Following academic and research positions held in Glasgow, Barcelona and Paris, he has recently been appointed Reader in Artificial Intelligence and Music at the School of Computing of the University of Plymouth, in the UK. Dr Miranda is a member of the editorial board of *Leonardo Music Journal*, *Organised Sound* and *Contemporary Music Review*.

Appendix I. Common Abstract Representation of Melodic Contour - Version 2 (CARMEC-2)

A melodic unit (MU) is represented in CARMEC-2 as a graph whose vertices stand for initial (or relative) pitch points and pitch movements, and the edges represent a directional path. Whilst the first vertex must have one outbound edge, the last one must have only one incoming edge. All vertices in between must have one incoming and one outbound edge each. Vertices can be of two types, initial pitch points (referred to as p-ini) and pitch movements (referred to as p-mov) as follows:

p-ini = {SM, SL, SH, SR}
 p-mov = {VLSU, LSU, MSU, SSU, RSB, SSD, MSD, LSD, VLSD}

where:

- SM = start MU in the middle register
- SL = start MU in the lower register
- SH = start MU in the higher register
- SR = start MU in a relative register

and

- VLSU = very large step up
- LSU = large step up
- MSU = medium step up
- SSU = small step up
- RSB = remain at the same band
- SSD = small step down
- MSD = medium step down
- LSD = large step down
- VLSD = very large step down

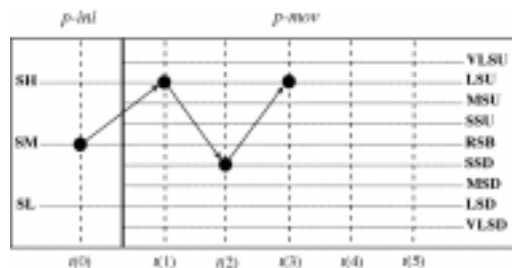


Figure 13. The representation of a melodic unit.

An MU will invariably start with a p-ini, followed by one or more p-movs. It is assumed that an MU can start at three different voice registers: low (SL), middle (SM) and high (SH). Then, from this initial point the next pitch might jump or step up or down, and so forth.

It is important to note that labels or absolute pitch values are not relevant here because CARMEC is intended to represent abstract melodic contours rather than a sequence of notes drawn from a specific tuning system. Figure 13 is a representation of the melody represented in Figure 14 using the standard Western classic music notation system.



Figure 14. An instantiation of the melody in Figure 13 using classic musical notation.

Since CARMEC is more general than the classic music notation system, there are surely many other instantiations of this melody in classic music notation.

Appendix II: The main algorithm of the enacting script

agent-player - AP	agent-imitator - AI
<pre>{ IF repertoire(AP) not empty pick motor control for pd; produce pd; ELSE generate random motor control for pd; add pd to repertoire(AP); produce pd; }</pre>	<pre>{ analyse pd } { build perceptual representation; } { IF rep(AI) not empty in = most perceptually similar to pd; ELSE generate random motor control for in; add in to repertoire(AI); produce in; }</pre>
<pre>{ analyse in; } { build perceptual representation; } { pn = most perceptually similar to in; } { IF pn = pd send positive feedback to AI; reinforce pd in repertoire(AP); ELSE send negative feedback to AI; }</pre>	<pre>{ IF feedback = positive approximate in to pd perceptually; generate appropriate motor control; reinforce in in repertoire(AI); } { IF feedback = negative IF in scores good HT; execute add_new_similar(snd); ELSE Modify motor representation of in towards pd; }</pre>
<pre>{ execute final_updates(AP); }</pre>	<pre>{ execute final_updates(AI); }</pre>

The *add_new_similar()* function works as follows: the agent produces a number of random intonations and then it picks the one that is perceptually most similar *pd* to include in the repertoire.