Kenneth McAlpine,[*] Eduardo Miranda,[†] and Stuart Hoggar[*]
[*]Department of Mathematics
University of Glasgow
Glasgow G12 8QW, Scotland
{km,sgh}@maths.gla.ac.uk
[†]Sony Computer Science Laboratory
6, rue Amyot
Paris 75005, France
miranda@csl.sony.fr

Making Music with Algorithms: A Case-Study System

Musicians, perhaps more so than any other group of artists, have always been quick to embrace technology in all its forms. From early attempts at synthesis with the Telharmonium (Roads 1996) to the latest digital-audio workstations (see, for example, Lehrman 1997; Manning 1993), musicians have looked to science to provide them with new and challenging ways of working.

One composition method that has benefited greatly from advances in technology is *algorithmic composition*. Algorithmic composition is by no means new—formal techniques for melody composition date back to at least 1026, when Guido d'Arezzo proposed a scheme that assigned pitches to each vowel sound in a religious text (Loy 1989)—but the advent of affordable and powerful computing resources has meant that more and more people, armed with no more than a little programming knowledge and some musical ideas, have been able to realize their composition algorithms from the comfort of their own desktops.

The use of computers as composition generators was pioneered in the mid-1950s by such people as Lejaren Hiller (Hiller and Isaacson 1959), whose 1956 work, *The Illiac Suite for String Quartet*, is recognized as being the first computer-composed composition. Iannis Xenakis (Xenakis 1960, 1971), although not the first to publish a computer-composed work, had been composing stochastically generated pieces by hand for some time before the premier of *The Illiac Suite*. Later, Gottfried Koenig developed the Project 1 and 2 composition programs (Koenig 1970a, b).

Computer-based algorithmic composition continues the post-war trend toward formalized and systematic composition methods, such as Arnold Schoenberg's tone-row and Anton Webern's serialism methods (Roads 1996). Considered thus, it is apparent that the computer is merely a tool for the realization of abstract design constructs, and is best employed as a labor-saving device to free the composer from performing menial calculations by hand. Using a computer, it is possible to preview and test the musical capabilities of many different algorithmic types and judge whether they have the potential to be developed into full-fledged composition systems. This is also the view subscribed to by the authors. Computers in music, and indeed, the algorithmic composition systems themselves, are best thought of as composition tools rather than one-stop music solutions. It was with this in mind that we developed CAMUS 3D, a composition system of our own design that likens the process of music composition to that of pattern propagation. In this article, we introduce the system and show how two classes of algorithms are used to generate musical data.

Two Classes of Musical Algorithms

We now present a brief summary of two classes of algorithms that have been used for music composition: *stochastic algorithms* and algorithms based on

Computer Music Journal, 23:2, pp. 19–30, Summer 1999

cellular automata. We choose to present only these two algorithmic techniques here, because of their relevance to the discussion of our system. The technical details behind these processes are also relatively straightforward for a nonmathematician to comprehend. This comprehensibility was one of the main reasons for the selection of these algorithms, and we believe that the system is particularly well suited to the musician who wishes to explore formal composition systems, but who is either unwilling or unable to learn about the complexities of the underlying mechanisms.

Stochastic Algorithms

Of all the compositional algorithms, stochastic algorithms are undoubtedly the easiest, both to comprehend and to implement. A stochastic algorithm is one that is intrinsically dependent on the laws of probability, making it impossible to predict the precise outcome of the process at any point in the future (Roads 1996). Stochastic processes may be natural, such as with radioactive decay, or artificial (that is, made by humans), as is the case for compositional algorithms. It is partly due to the accessibility of stochastic algorithms that they have become so extensively employed for compositional purposes. Most, if not all, composers engaged in algorithmic composition have used stochastic processes in one form or another, whether as the basis for an entire composition algorithm or as an incidental decision-making routine.

Probability Lookup Tables

The outcome of a stochastic process depends on certain underlying *probabilistic distributions*, a term used to describe the way that the probabilities are divided among each of the possible outcomes. The simplest and most widely utilized method of modeling and implementing these distributions is that of *probability lookup tables*. As the name suggests, these are simply stored tables of values that correspond to the likelihood of occurrence of one or more events. The table entries range in value from 0, which indicates that the corresponding event will never occur, to 1, which indicates that the event will occur with absolute certainty. Thus, a value of 0.25 would equate to a one-in-four chance of occurrence.

In fact, this definition is not strictly correct. It is possible for a single outcome to have probability 0 and still occur. However, for the (discrete) stochastic selection routines given below, it is certainly the case that a probability of 0 ensures that an event will never take place. It is for this reason that we have presented the definition as above. For a more complete discussion on the nature of probability, the interested reader is referred to any introductory text, such as that of Murphy (1991).

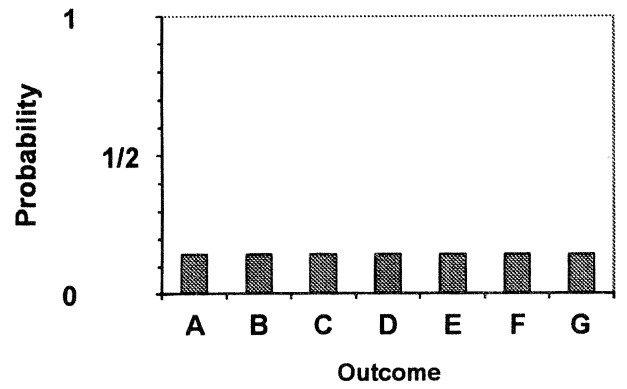
If the sum of all the entries in the probability table is equal to 1, then we say that the table is *normalized*. This is a very desirable state of affairs, because it implies that when a selection is made according to the probability table, we can say with absolute certainty that there will be some output, since the events that index the table partition the probability space.

The simplest type of probability distribution is the *uniform distribution* (see Figure 1a). Here, each possible outcome is assigned an equal probability of success. Since a normalized table requires the sum of the probabilities of the possible outcomes to be equal to 1, this means that if we have N equally probable outcomes, the probability of each will be $1/N$.

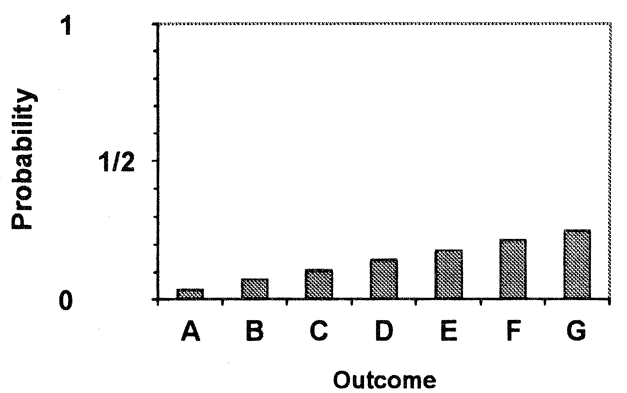
It is likely that a composer who is using probability tables for note selection would wish to use distributions that favor certain outcomes over others. There are many alternative distributions that can be used:

- *Linear distributions* (see Figure 1b) give a line of fixed, increasing or decreasing probability between two limits. Clearly, the uniform distribution is a special case of the linear distribution, in which the line of probability has zero gradient and the limit points are equal.
- *Exponential distributions* (see Figure 1c) give an exponential curve between two endpoints.
- *Bell-shaped distributions* (see Figure 1d) are one of the most common naturally occurring distributions. Traditionally, the term bell-shaped is

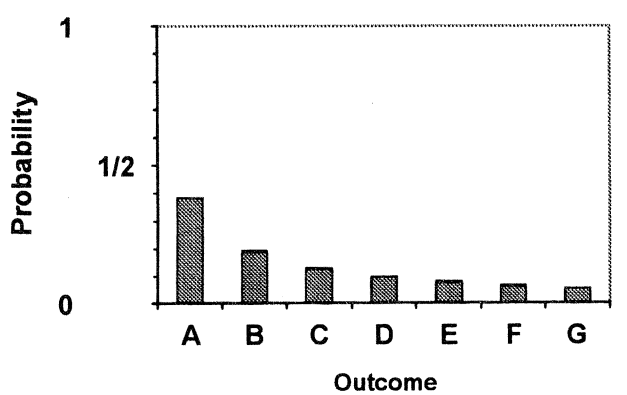
Figure 1. Common probability distributions: uniform (a), linear (b), exponential (c), bell-shaped (d), and U-shaped (e).



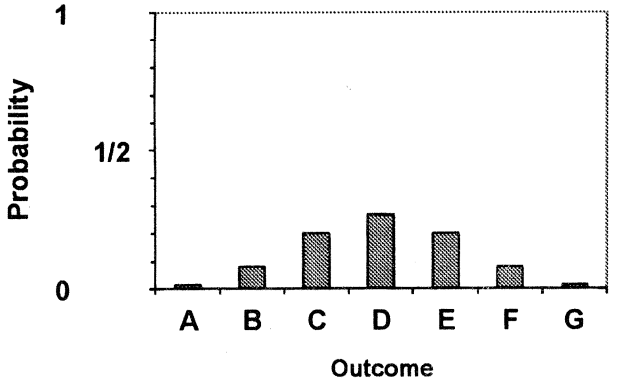
(a)



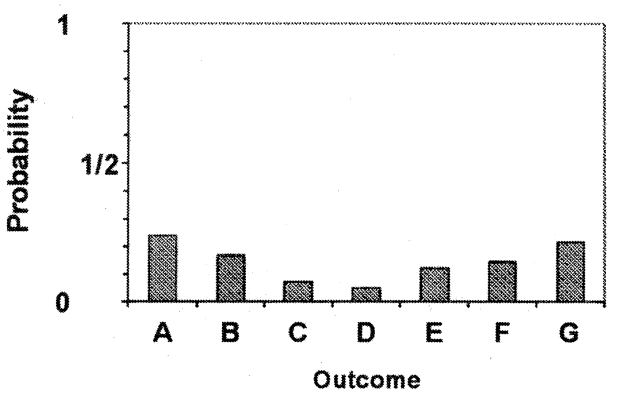
(b)



(c)

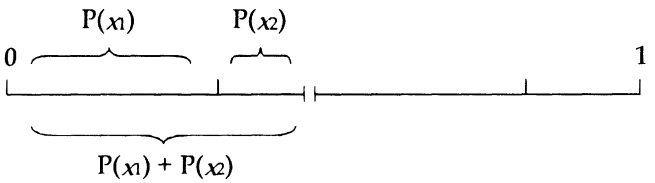


(d)



(e)

Figure 2. A normalized probability distribution partitions the real line segment [0,1] into regions whose interval lengths are the probabilities of the corresponding outcomes.



used to describe the normal distribution. Here, however, we use the term to denote any distribution that exhibits a distinctive *n*-shaped curve. Bell-shaped distributions may or may not be symmetrical and have their minima at the endpoints.

U-shaped distributions (see Figure 1e) are essentially upturned bell-shaped distributions. They peak at the endpoints, and need not be symmetrical.

Probability tables may be utilized as, say, part of a decision-making routine as follows: The composer specifies a number, *n*, of possible outcomes. With each of these outcomes is associated a procedure that will be executed if that particular outcome is selected, and a real number between 0 and 1 that gives the probability that the outcome will be selected. Upon normalization, this probability distribution corresponds to a partition of the real line segment [0, 1] into *n* distinct regions, whose interval widths are the probabilities of the respective outcomes. A random-number generator is used to generate a real number between 0 and 1. The decision can then be made by observing in which of the *n* segments the number lies (see Figure 2). For further information on the implementation of probability tables, see the work of Lorrain (1980).

Markov Chains

Markov chains are discrete probability systems in which the probability of future events depends on one or more past events (Jones 1981). In other words, Markov chains are stochastic processes that retain memory of past events to influence the outcome of future events. The number of past events that are taken into consideration at each stage is known as the *order* of the chain. Thus, a Markov chain that takes only an event's predecessor into account is of first order, a chain that considers both an event's predecessor and the predecessor's predecessor is of second order, and so on.

In general, an *N*th-order Markov chain can be represented by a state-transition matrix—an $N + 1$-dimensional probability table. The state-transition matrix gives us information on the likelihood of an event's occurring, given the previous *N* states. Figure 3 shows a possible state-transition matrix for a first-order Markov chain with four possible outcomes. Here, the previous states are listed vertically, and the transition states are listed horizontally. Thus, if we wish to find, for example, the probability of state B occurring immediately after state A, we simply find state A in the first column, and then move along horizontally to the B column. The entry here is 0.5, so there is a 1 in 2 chance of this transition occurring.

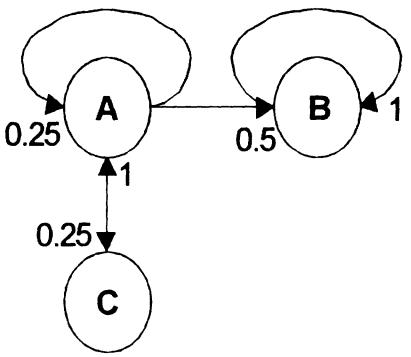
For a Markov chain, a state X is said to be *reachable* from a state Y if it is possible to reach state Y from state X after a finite number of steps. If state Y is reachable from state X and state X is reachable from state Y, then the two states are said to *communicate*. For example, in Figure 3, states A and C communicate, since clearly C is reachable from A and A is reachable from C. However, neither A and B nor B and C communicate: although B is reachable from A (and thus also from C), neither state is reachable from B, which is always followed by itself.

It can be shown fairly easily that the communication relation on a Markov chain is an *equivalence*. That is, the communication relation is *reflexive* (since a state always communicates with itself), *symmetric* (since if a state X communicates with a state Y, then clearly Y communicates with

Figure 3. A first-order Markov chain as a state-transition matrix (a), and a labeled directed graph (b).

	A	B	C
A	0.25	0.5	0.25
B	0	1	0
C	1	0	0

(a)



(b)

X), and *transitive* (since if a state X communicates with a state Y, and state Y communicates with a state Z, then state X also communicates with state Z). Grouping communicating states together, we can partition the states of the chain into equivalence classes of communicating states.

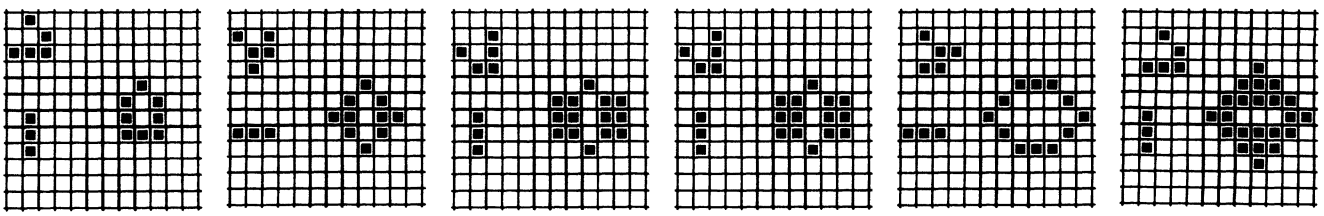
Those states that are certain to occur again once they have been reached by the chain are called *recurrent*, and the equivalence class they belong to is known as a *recurrent class*. States that may never occur again (i.e., those that are not recurrent) are called *transient*, and the class they belong to is known as a *transient class*. It can be shown that every Markov chain consists of at least one recurrent class and some number (possibly none) of transient classes.

Markov processes are exceptionally well suited for rhythm selection. Rhythmic lines often exhibit semi-cyclic behavior in that short phrases often repeat exactly or slightly altered as the line progresses—the human ear tends to like this sort of regularity. Similar behavior can be engineered by careful manipulation of the recurrent and transient classes within the Markov chain. A Markov chain for rhythm selection may be constructed as follows: The composer first decides on the order of the Markov chain that is to be used in the composition, and then initializes an array whose dimensionality is 1 greater than the order of the chain, and whose size is determined by the number of different rhythmic components that are to form the states. This array will correspond to the state-transition matrix of the Markov chain. Each rhythmic figure that the composer wishes to use is then assigned a unique integral value that is used to index the state-transition matrix. The index of the previous state is stored as a temporary variable, and used to select the correct row of the state-transition matrix. The probabilities stored in this row are then used to drive a stochastic selection routine like that described in the previous section.

Cellular Automata

Cellular automata are very important to scientists as modeling and simulation tools. They have found applications in many different disciplines, from physics, chemistry, and biology to philosophy and sociology (Peitgen, Jürgens, and Saupe 1992). Cellular automata are discrete *dynamical systems*; that is, they change some feature with time. We often view a cellular automaton as an array of elements, referred to as *cells*, to which we apply some evolution rule that determines how the automaton develops in time. Each cell can exist in one of p possible states, represented by the integers 0, 1, 2,..., $p - 1$. We often refer to such an automaton as a *p-state cellular automaton*. To specify fully and run a cellular automaton, we need one further piece of information: an initial cell configuration. When this is specified, the automaton can be set to run and the cellular evolution can be observed.

Figure 4. Six successive time steps from the Game of Life automaton.



It is important to notice that any long-term global trends that arise in the automaton's development are examples of *emergent behavior*. The evolution rules in a cellular automaton are concerned only with local neighborhoods around the cell under consideration. Global trends are not explicitly coded beforehand.

Game of Life

The Game of Life (Wolfram 1984) is a two-dimensional cellular automaton that attempts to model a colony of simple organisms. The automaton consists of an array of $(m \times n)$ cells, each of which can exist in two states—alive, represented by 0, or dead, represented by 1. The rule that determines the development of the automaton is: *a cell will be alive at time-step $t + 1$ if and only if it has precisely three live neighbors (other than itself) at time-step t.* Figure 4 shows six successive steps of the Game of Life.

Demon Cyclic Space

The Demon Cyclic Space (Wolfram 1984) is a two-dimensional p-state automaton of $(m \times n)$ cells. The evolution of the automaton is determined by the following rule: *a cell that is in state j at time-step t will dominate any neighboring cells that are in state $j - 1$, so that they increase their state to j at time-step $t + 1$.* It is important to note, however, that the two-dimensional automaton space is *cyclic*. Thus, cells in state 0 dominate cells in state $n - 1$, so that the influence that each cell exerts on its neighbors has far-reaching consequences, often extending beyond the eight cells that immediately border the cell in question. The cells in the Demon Cyclic Space are, in general, initially randomized. The behavior of the system is such that after a number of time steps, the cells self-organize to a patchwork pattern like that of Figure 5.

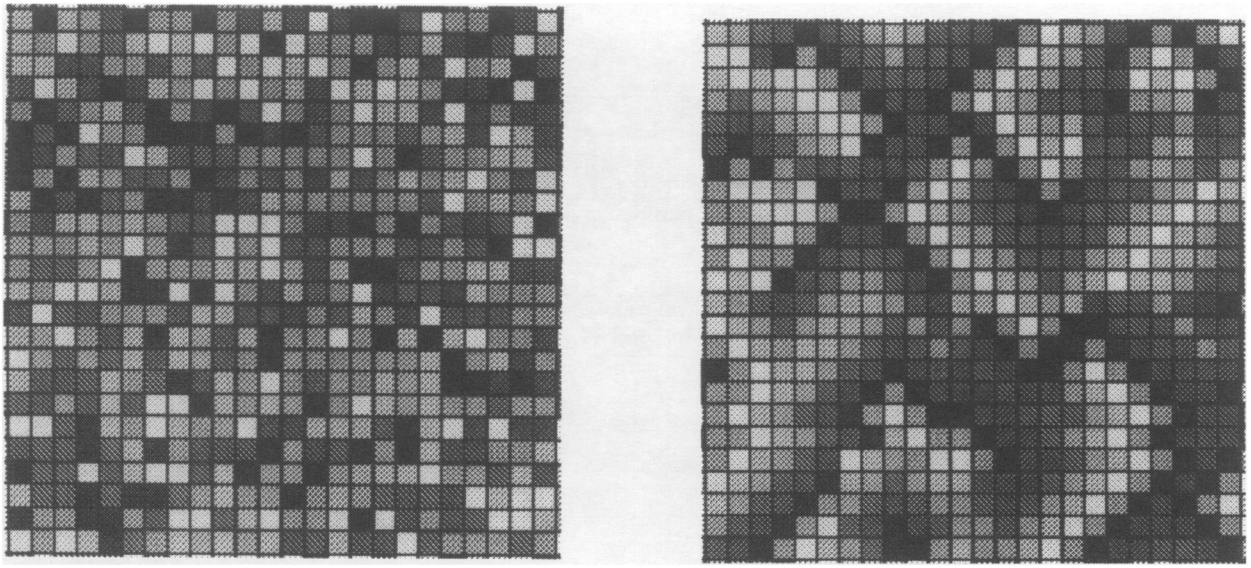
We illustrate the musical usage of cellular automata in the following section. The interested reader is also referred to the works of Beyls (1997) and McAlpine, Miranda, and Hoggar (1997a, b).

CAMUS 3D

CAMUS 3D (Cellular Automata MUSic in 3 Dimensions) is part of an ongoing research project. The system is a development of an earlier two-dimensional system (Miranda 1993, 1994; McAlpine, Miranda, and Hoggar 1997a, b), and uses three-dimensional extensions of the Game of Life and Demon Cyclic Space automata to generate compositions. The composition process is modeled on pattern propagation; we may view each theme in a composition as a separate pattern. As the composition progresses, the patterns are subjected to certain transformations (such as straight repetition, transposition, inversion, augmentation, and so on), according to the formal structure that the composer has chosen for the work. This structure can be rigidly adhered to or used as a general guiding principle, but so long as certain design constructs are in place to guide the temporal development of the composition, we can say that we have a system of pattern propagation according to some predetermined constraints.

Traditionally, composers have employed pattern propagation intuitively, but algorithmic composition techniques, such as those described below, allow the pattern propagation to be formalized, albeit at a much higher level. Here, the composer does not in general apply specific transformations

Figure 5. Initial randomized state of the Demon Cyclic Space automaton (left), and the same region of the automaton after a number of time steps (right).



to a particular pattern. Instead, all of the musical patterns evolve according to the rules and constraints that have been specified at the design stage. Thus, any common stylistic musical features that emerge from the use of cellular automata as music generators can be said to be a sonification of the emergent behavior of the automaton; the temporal development arises as a result of the local evolution rules, and is not specified in full in advance.

Cellular automata are used to drive the CAMUS 3D composition process, since these are a well-known and well-understood form of pattern propagation. Stochastic selection routines are also quite widely utilized within the system, as they offer a quick and efficient method of specifying long-term structure while avoiding the often laborious task of specifying details at each step.

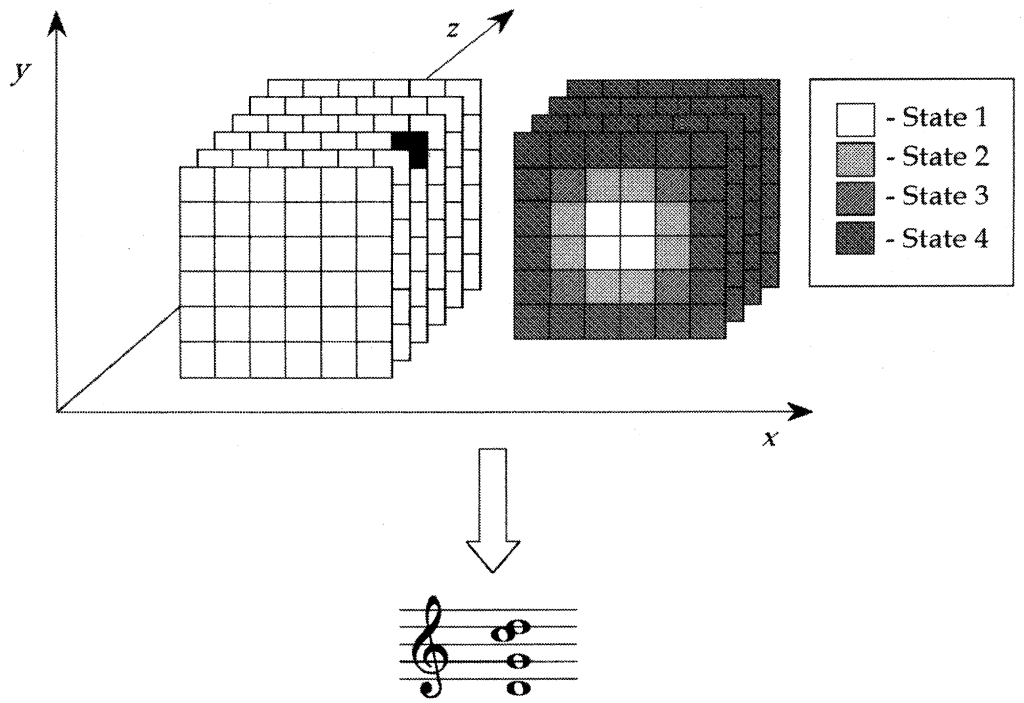
To begin the composition process, the Game of Life automaton is initialized with a starting cell configuration, the Demon Cyclic Space automaton is initialized with random states, and both are set to run. At each time step, the coordinates of each live cell are analyzed and used to determine a four-note chord: a set of four (not necessarily distinct) notes that may or may not sound simultaneously and that will be played at the corresponding moment in the composition. The state of the corresponding cell of the Demon Cyclic Space automaton is used to determine the instrumentation of the piece. This configuration is demonstrated in Figure 6. In this case, the cell in the Game of Life at (5, 5, 2) is alive, and thus constitutes a musical event.

The coordinates (5, 5, 2) describe the intervals in a four-note chord: The x cell position (starting at 0 in the bottom-left corner) defines a semitone interval from a fundamental pitch to the second-lowest pitch of the chord. The y cell position defines a semitone interval from the second-lowest to the second-highest pitch in the chord. The z cell position defines a semitone interval from the second-highest pitch to the top note of the chord. Note that if the cell position is 0 (corresponding to the first cell in each direction), the "higher" pitch defined by the associated interval will be identical to the "lower" pitch.

The corresponding cell in the Demon Cyclic Space is in state 4, which means that the sonic event would be played by instrument number four (for example, by using MIDI channel 4). Note that for the sake of clarity the first two layers of the Demon Cyclic Space have been omitted in Figure 6.

The use of a discrete three-dimensional Euclidean space to represent musical intervals is an ex-

Figure 6. Configuration for a typical time step of the CAMUS 3D algorithm, showing the mapping that CAMUS 3D employs to convert cell data from the automata into music.



tension of the two-dimensional von Neumann Music Space used in an earlier version of the system (Miranda 1993, 1994).

Having established the intervallic content of the chord associated with a live cell, we must then establish the fundamental note to specify fully each of the pitches in the chord. This can be done either manually, by reading note data sequentially from a user-specified list, or automatically, by using stochastic selection routines that allow the composer to specify the relative weightings of the pitch values for the fundamental pitches.

To avoid a piece that is composed entirely of block chords, we must implement a routine that staggers the starting (and possibly ending) times of each of the notes of the chord. It is a simple matter to calculate that there are 24 different ways of arranging the starting order of these 4 (nonsimultaneous) notes. For CAMUS 3D, a stochastic selection routine is used that consults a user-specified table for the associated probabilities of each of the 24 possible starting arrangements (see Figure 7). When a starting arrangement has been chosen, CAMUS 3D calculates the precise note durations. This is achieved by means of a first-order Markov chain.

The probabilities in the transition matrix of the Markov chain are, again, user specified. Note lengths are quantized to sixteenth-note resolution, and simultaneous note events are catered for by allowing starting times of duration 0.

Two methods for specifying the probabilities in the Markov state-transition matrix are offered by CAMUS 3D. The standard option provides the composer with a graphical means of viewing and altering the probability values. The probabilities are represented on-screen by a graded color scheme that ranges from pure red (probability value 0) to pure green (probability value 1). This scheme is natural to use, as it ties in with the natural color schemes of the physical world: red often signifies a warning or danger (impossible), while green indicates safety (certain).

The standard window is probably best used to view the probability data to see quickly which note lengths are likely to arise. To actually set the probability values, the advanced window allows

Figure 7. The note-orderings dialog box allows the user to set the probability of exposition of each of the 24 possible note orderings.

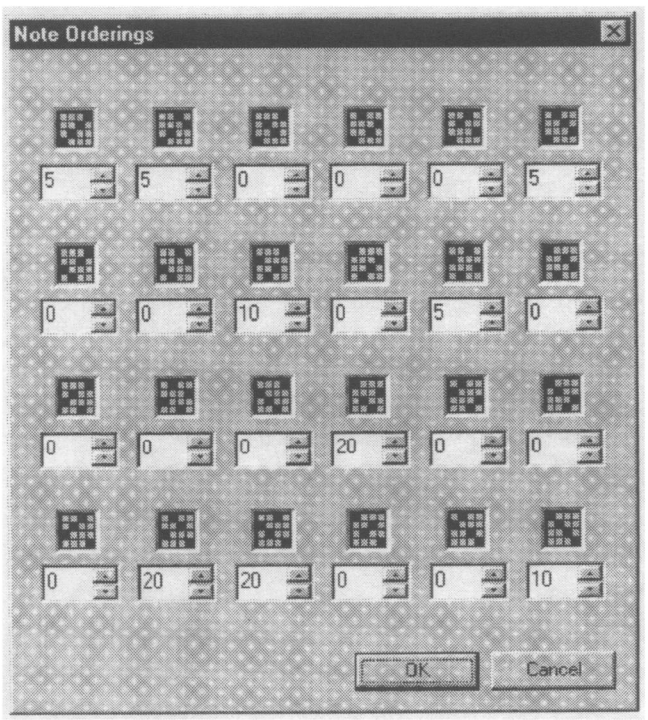
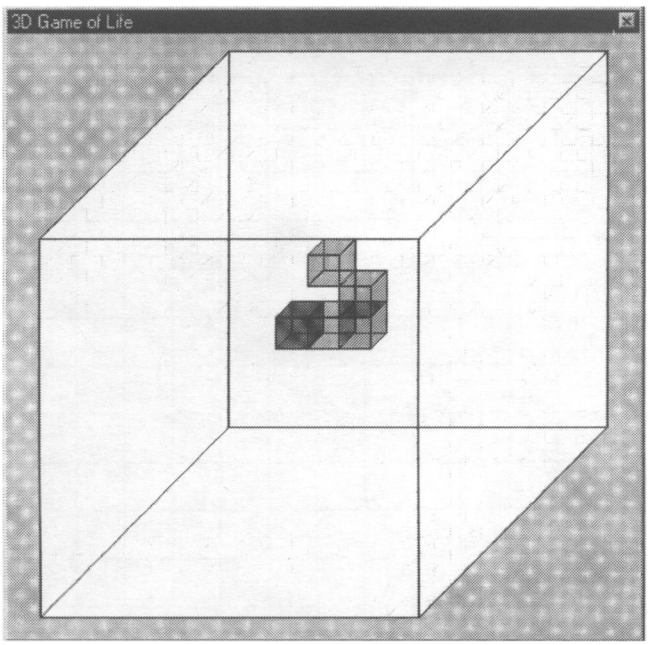


Figure 8. The 3-D Game of Life dialog box.



direct access to the numerical values. The advanced window presents exactly the same information as the standard option, but does so by using numerical values. As with all of the probability tables described herein, CAMUS 3D automatically renormalizes the probability data whenever changes are made.

When the composition process is started, the music is performed in real time, and can be saved as a type-0 standard MIDI file. In addition, CAMUS 3D allows the user to save the composition as a set of parameters that correspond to the states of the automata and the probability tables for the selection routines. While this allows the composer to recreate the "same" composition, the resulting music may sound quite different. Whereas the automata are wholly deterministic—and so produce identical chord sequences and instrumentations each time they run with identical initial configurations—the stochastic selection routines may lead to very different fundamental pitches, note orderings, and note durations.

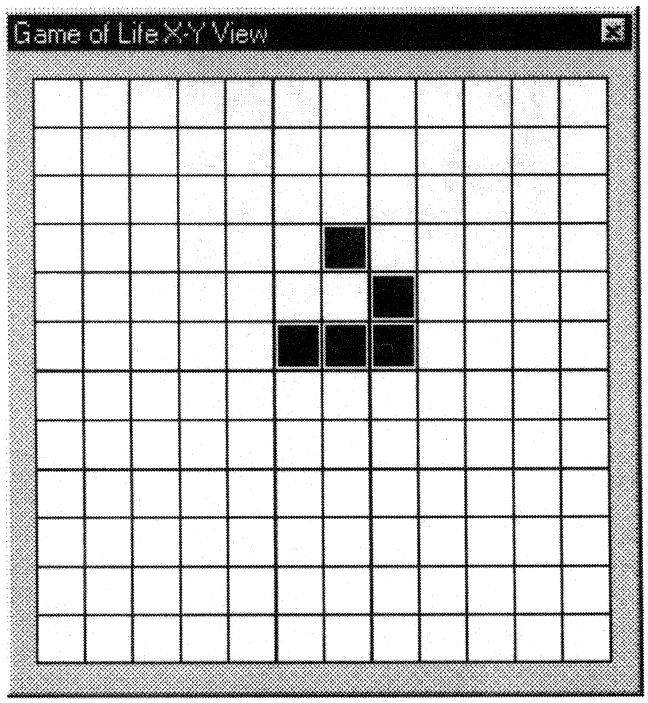
Composing with CAMUS 3D

We now illustrate the workings of the CAMUS 3D music algorithm by presenting a brief example of the system in use.

When the system is first loaded, the user is presented with a two-dimensional isometric display of the three-dimensional Game of Life. This serves as the musical score, in the sense that the live cells in this graphical display correspond to the chord events in much the same way that the dots on manuscript paper correspond to note events. This window, as shown in Figure 8, combines the information from the Game of Life and Demon Cyclic Space automata, by coloring the live cells from the Game of Life according to the state of the corresponding Demon Cyclic Space cell. The composer may choose to view the automaton data as a plan, elevation, or end elevation by selecting one or more of the view options from the menu (see Figure 9).

The initial configuration of the Game of Life automaton may be set using the CAMUS 3D toolbar.

Figure 9. The x, y-plane view of the 3-D Game of Life.



From here, the user can access the following tools for altering the states of cells in the Game of Life:

1. The *clear* option clears the Game of Life of all live cells.
2. The *randomize* option assigns each cell in the Game of Life a random state.
3. The *invert cell* option displays the invert-cell dialog box, which allows the user to invert the state of the cell at position (x, y, z).
4. The *line* option displays the line dialog box, which allows the user to invert the states of those cells that lie on the unique line between the cells (a, b, c) and (x, y, z). It should be noted, however, that since the Game of Life space is discrete and quite small ($12 \times 12 \times 12$), the quantization of the line may cause it to look blocky, and in extreme cases, almost resemble a curve.
5. The *cube* option displays the cube dialog box, which allows the user to invert the states of those cells that lie within a cube with sides of length *l* cells, whose top-left, nearest corner is given by the cell (x, y, z). Hollow cubes can thus be specified by splitting the cube into a boundary and a hollow interior. The cells that lie in the interior of the cube are brought to life first. Then, the larger boundary is placed on top, bringing those cells that lie on the boundary to life, and killing those that lie within the interior.

Using these tools, the user can specify complex patterns of cells quickly and easily while getting visual feedback from the four available displays of the automaton. This configuration of views and tools for placing primitive shapes within the working area is very similar to that used by 3-D art packages and modelers.

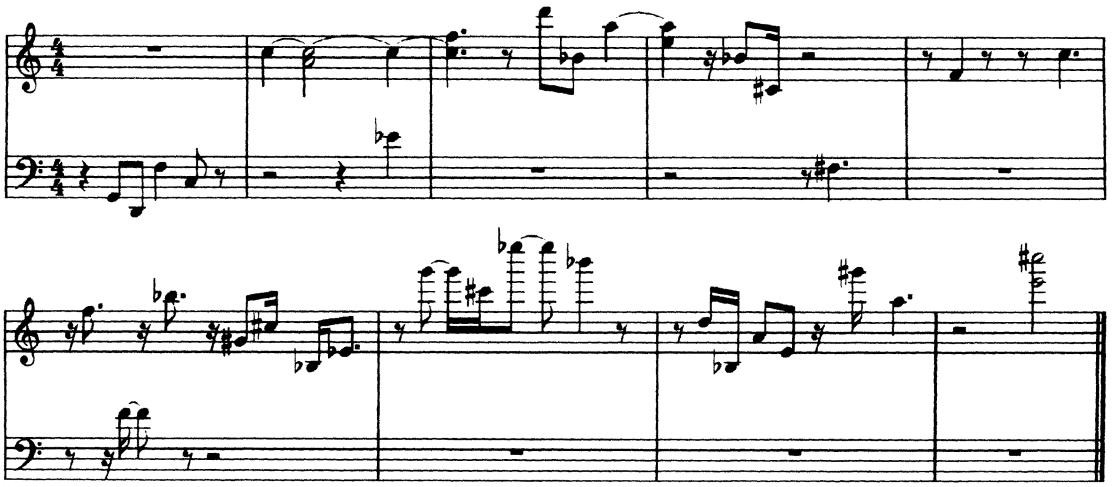
The composer may also wish to alter the evolution rules for the two automata. Toward this end, CAMUS 3D allows the user to alter the number of possible states in which each Demon Cyclic Space cell can exist, the number of neighboring cells that cause Game of Life cells to live and die, and whether the automata are treated as true three-dimensional spaces or as a number of stacked two-dimensional spaces. The next stage in the composition process is to specify the probabilities involved with the pitches, note orderings, and the Markov selection routines. Finally, the user specifies the instrumentation of the composition by associating each possible state of the Demon Cyclic Space with a General MIDI instrument.

With the initialization in place, all that remains is for the composer to set the process running. This can be done either one step at a time using the Step Through function, or continuously by selecting Go. Figure 10 shows a musical passage generated by the system.

Further Research and Conclusions

Having proven itself to be a viable mechanism for driving musical composition, CAMUS 3D has been successfully used to compose a number of works. The music that is generated in general exhibits a specific formal style centered around a four-note musical event. This arises as a direct result of the al-

Figure 10. Music generated as a result of the CAMUS 3D system.



gorithm design, and represents part of the system's emergent behavior. For the composer who is prepared to put a little effort into the system, the results can be very pleasing, and often sound much more natural than compositions obtained using comparable algorithmic techniques. We believe that this is owing to the organic nature of the automata used to generate the raw compositional data and to the careful choice of mapping from these abstract mathematical systems to the musical output.

Further developments for the system include implementing a number of different forms of pattern propagation, such as fractal zooms and other types of automata, to drive the composition. More complex evolution rules and tools for manipulating the Game of Life cells are also planned, along with a dynamics processor that will color the music in a natural way.

One significant limitation of the system at present is that it will only generate music one cell at a time, using a fixed checking order. This means that the music is generated monotimbrally, and suffers from a fixed order of chord progression. The solution to this problem seems to lie with the introduction of parallel cell processing, which will not only enable the system to generate several musical parts at once, but will also overcome the problem of the fixed chord sequence.

Parallel processing will, however, raise several further difficulties (Miranda 1998). For example, note ordering will become much more complex as we move from considering just four notes at any one time to a maximum of $12 \times 12 \times 12 \times 4 = 6{,}912$ notes per time step. This is clearly not a trivial matter, and will require a system considerably more intricate than the current stochastic decision routine.

There is also the issue of harmonic content; when playing back several simultaneous note events, the likelihood is that a sizeable portion of the music will sound extremely unpleasant. Thus, when implementing parallel cell checking, we also intend to implement a system of chord filtering that will examine the note events that are generated on each time step and decide which cell combinations produce music that most closely matches the composer's aesthetic preferences. This will be accomplished by using a neural network (Haykin 1994) that is designed to classify chords and is trained to respond to harmony in a similar manner to the human composer operating the system.

We are aware that research of this sort is a never-ending process, and we are by no means searching for the ultimate composition system. Rather, we seek to develop creative tools that serve a particular purpose or generate specific kinds of musical passages.

An earlier version of the CAMUS 3D system was used by Eduardo Miranda to compose a piece entitled *Entre l'Absurde et le Mystère* for chamber orchestra. The composition was premiered in

Edinburgh in March 1995 by The Chamber Group of Scotland, conducted by Martyn Brabbins. The score is available by request from the authors, as is a full-featured demonstration version of the CAMUS 3D software.

Acknowledgments

Many thanks to the Carnegie Trust for the Universities in Scotland, who generously provided the funding that has enabled this research. Thanks also to the University of Glasgow for providing both a research position and the facilities to undertake research in this field.

References

Beyls, P. 1997. "Aesthetic Navigation: Musical Complexity Engineering using Genetic Algorithms." *Proceedings of Journées d'Informatique Musicale*. Lyon: Grame, pp. 97–105.

Haykin, S. 1994. *Neural Netrworks: A Comprehensive Foundation*. Indianapolis: Macmillan.

Hiller, L., and L. Isaacson. 1959. *Experimental Music*. New York: McGraw-Hill.

Jones, K. 1981. "Compositional Applications of Stochastic Processes." *Computer Music Journal* 5(2):45–61.

Koenig, G. M. 1970a. "Project 1: A Programme for Musical Composition." *Electronic Music Reports* 2:32–44.

Koenig, G. M. 1970b. "Project 2: A Programme for Musical Composition." *Electronic Music Reports* 3:1–16.

Lehrman, P. D. 1997. "4 Score! Digidesign ProTools 4.0 Software." *Sound on Sound* 12(9):162–173.

Lorrain, D. 1980. "A Panoply of Stochastic Cannons." *Computer Music Journal* 4(1):53–81.

Loy, G. 1989. "Composing with Computers—A Survey of Some Compositional Formalisms and Music Programming Languages." In M. Mathews and J. R. Pierce, eds. *Current Directions in Computer Music Research*. Cambridge, Massachusetts: MIT Press, pp. 292–396.

Manning, P. 1993. *Electronic and Computer Music*, 2nd ed. Oxford: Claredon Press.

McAlpine, K. B., E. R. Miranda, and S. G. Hoggar. 1997a. "Dynamical Systems and Applications to Music Composition: A Research Report." *Proceedings of Journées d'Informatique Musicale*. Lyon: Grame, pp. 106–113.

McAlpine, K. B., E. R. Miranda, and S. G. Hoggar. 1997b. "A Cellular Automata-Based Music Algorithm: A Research Report." *Proceedings of IV Brazilian Symposium on Computer Music*. Brasília: Brazilian Computer Music Association (NUCOM), pp. 7–17.

Miranda, E. R. 1993. "Cellular Automata Music: An Interdisciplinary Project." *Interface* 22:3–21.

Miranda, E. R. 1994. "Music Composition using Cellular Automata." *Languages of Design* 2:105–117.

Miranda, E. R. 1998. "Parallel Computing for Musicians." *Proceedings of the V Brazilian Symposium on Computer Music*. Belo Horizonte: Brazilian Computer Music Association (NUCOM), pp.9-20.

Murphy, I. S. 1991. *Probability: A First Course*. Stirling: Arklay.

Peitgen, H. O., H. Jürgens, and D. Saupe. 1992. *Chaos and Fractals: New Frontiers of Science*. New York: Springer-Verlag.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press.

Wolfram, S. 1984. "Universality and Complexity in Cellular Automata." *Physics* 10:1–35.

Xenakis, I. 1960. "Elements of Stochastic Music." *Gravesaner Blätter* 18:84–105.

Xenakis, I. 1971. *Formalized Music*. Bloomington, Indiana: Indiana University Press.