Researchers have long used cellular automata (CA), and in general parallel generative devices such as Lindenmayer systems, to produce images, fractals, growth patterns in two dimensions, and sound. They usually implement these models on special-purpose architectures that often depend on the type of presentation being generated. (See the “Related Work” sidebar for more details about previous research.)

We have developed our own prototype for generating content, called Extended Cellular Automata with Pluggable Multimedia Elements (ExcapeMe). This prototype acts as an open environment that lets users execute CA in three dimensions and manage simultaneous rendering of CA using different forms of presentation. In principle, we can attach any type of multimedia presentation as a plug-in. The environment already supports 1D, 2D, 2.5D, and 3D presentations, and we can apply sound rendering to CA of any dimension. The environment also supports interactively defining, editing, and executing CA as well as defining the mapping between the CA configurations and their actual rendering.

Prospective users include researchers and students interested in studying CA dynamics, either to explore a given law’s characteristics or use CA as simulation tools to observe the possible evolutions of a modeled system. In particular, researchers can use the different types of representation we provide to gain insight into such evolutions. Another class of users is computer artists interested in generating visual or sound patterns for performances. Because we endowed ExcapeMe with network communication mechanisms, it also acts as a source of parameter configurations for other applications.

Besides letting us use several dimensions, both in defining the universe and in its pictorial representation, ExcapeMe exploits some extensions to the mathematical model. (See the “Cellular Automata and ExcapeMe Extensions” sidebar for more specifics.)

Interactively managing CA ExcapeMe users can exploit the automaton definition environment to define 1D, 2D, or 3D universes.

Users can write any next-state function by using the Evolution Function Language (EFL), which lets them specify evolution rules through expressions of the form \((q, \text{cond}, q')\), where \(q\) is the cell’s current state, \(\text{cond}\) is a logical condition to be evaluated on the cell neighborhood, and \(q'\)
is the cell’s new state if the condition is verified. For example, the law \( (1, (\text{Moore}(\text{atMost} 2, = 1) \text{ or } \text{Moore}(\text{atLeast} 5, = 1)), 0) \) indicates that a cell moves from state 1 to state 0 if, in its neighborhood, there exist fewer than two (isolation) or more than five (overcrowding) cells in state 1. This is a part of the function defining Conway’s famous Game of Life.\(^1\) By writing several of these expressions, users can define the set of laws governing the CA.

Defining the next-state function requires that users define the neighborhood. Users can select the predefined Moore (a neighbor is a cell touched in a vertex, hence there are eight neighbors in the 2D case) or von Neumann (a neighbor is a cell with an adjacent face, hence there are four neighbors in the 2D case) neighborhoods, or generalize them as surrounding and axis, respectively. We can use some predefined operators as well. For example, in the 2D case, the condition \( \text{vonNeumann}(\text{atLeast} 2, > 5) \) is satisfied if at least two cells in the von Neumann neighborhood have a state from the subset \( \{q_0, \ldots, q_m\} \); the condition \( \text{surrounding}(\text{radius} 2, \text{maximum} 3, \text{time} = 1) \) is satisfied if no more than three cells in a surrounding of radius 2—that is, in a neighborhood of size 25—have been in their current states for exactly one instant of time. The condition \( \text{Time}(-1, 0) = \text{Time}(1, 0) \) is verified if the neighbor on the left has been in its current state for a time length different from that of the neighbor on the right.

Figure 1 shows the user interface for the next-state function’s definition while the user is entering an expression.

Users select the \( q \) and \( q' \) states for a transition via the combo boxes at the top of the interaction mask. They either directly write the cond component in the text field or create it by using buttons labeled by predefined operators. ExcapeMe signals syntactical errors and performs a semantical check to issue a warning in case a law allows creation from nil. Users can ignore this warning.

As conditions don’t need to be mutually exclusive, the order in which the user enters the rules indicates their priority, so that at each evolution step cells with the same state and neighborhood configuration are guaranteed to evolve in the same way. Moreover, rules that have the same \( q \) and \( q' \) components can be aggregated by ANDing or ORing them.

**Interactively managing multimedia elements**

ExcapeMe’s architecture follows the coopera-

---

**Cellular Automata and ExcapeMe Extensions**

An informal definition of cellular automaton sees it as a potentially infinite grid (usually 2D but possibly with any number \( n \) of dimensions) of cells, each occupied by an identical replica of a Moore automaton with a finite set of states \( Q \). Each automaton can communicate with its neighbors—where the definition of a neighborhood can vary. In general, we speak of a cell to indicate the automaton that occupies it. The evolution of the grid is synchronous: at each step, each cell performs a state transition based on its current state and the current state of its neighbors.

The set \( Q \) presents a designated state \( q_0 \), called the quiescent state. The law of evolution is usually such that if a cell is in the state \( q_0 \) and it only has quiescent neighbors, it remains in \( q_0 \). Hence, if we start from a grid configuration with only a finite set of nonquiescent cells, at each step, the set of nonquiescent cells will remain finite. We call the subset of nonquiescent cells the core. Often, researchers are interested in the minimum (\( n \)-dimensional) rectangle enclosing the core and in all the elements that belong in the neighborhood of at least one element of the core.

ExcapeMe adopts two significant extensions to the CA model. First, we let the neighborhood be any size—that is, we can specify a dependency on an element both at a relative position with respect to the current cell and at a fixed place. Hence, the model resembles more of an automata network than purely CA. In any case, the neighborhood and the evolution law are uniform over the whole universe.

Second, we introduce time in the laws, so that a law can take into account the number of time instants that an automaton has remained in the same state. If only constants are involved in time comparisons, this extension can be simulated by enlarging the set of states. However, if we want to allow comparisons of arbitrary times, this can’t in principle be simulated with a finite number of states. Although these extensions don’t affect the basic architecture, they let us realize some interesting artistic effects. In this article, we define ExcapeMe as if it were developed for supporting the basic CA model.

---

**Figure 1. Interactive definition of a next-state function.**
The model layer maintains the additional information needed to represent the cellular automaton—for example, the association between a state symbol and its color and shape.

In turn, we uniformly structure each layer, according to a pattern that lets us generalize the aspects relative to the different dimensions (1, 2, and 3) of the cellular automaton.

The diagram in Figure 2 summarizes the application’s structure. Although the logic and model layers are fixed, we can extend the GUI layer at will by introducing new presentation forms. This architecture replicates the observer–observable pattern at the model level so that the model component is notified of modifications in the automaton and notifies in turn the presentation layers to update themselves.

Different presentation components obtain data about the states and process them to produce suitable representations of the current configuration. We also exploit this communication protocol to produce responses to interactive user actions. Each user action on the editor is translated into a request to the model to steer changes to the cellular automaton, triggering the notification and update process.

Such an organization lets ExcapeMe distribute interaction events according to whether the presentation, model, or automaton itself must be affected. For example, changing the perspective in a 3D view will only affect the presentation, choosing a different color for a state will affect the model (and consequently the presentation), and suspending the automaton to change the value of some of its states will affect the logic level (and hence the logic levels above it). ExcapeMe can associate multiple views with the same logic objects.

Presentation components don’t depend on one another, so we can make them work concurrently, achieving multimedia effects. We achieve a weak form of synchronization between components when we impose that they evolve only according to the cellular automaton evolution.

This organization lets us expand the application by adding multimedia off-the-shelf plug-ins as they become available. For each plug-in, we must devise a suitable wrapper to conform the new component to the interface for presentation components. Such a wrapper will also be responsible for processing the data received by the model layer to generate a proper rendering. Data are pushed to the presentation component at each evolution step in the form of a set of quadruples <coordinates, state, shape, color>. In this way, wrapped plug-ins only need to interface with model components, without having to refer to the underlying logic.

Moreover, because communication is based on notification, we can encapsulate communication with the model component in the wrapper, which thus hides any plug-in-dependent protocol.

For each presentation component, users can define the mapping between states and presentation objects to be rendered in the component. For example, they can select the shape, color, or text to represent a state. For audio presentation components, they also can select the instruments and notes for each state’s representation. Finally,
the dimensionality of the rendering is completely independent of that of the universe. Hence, we can make a 3D automaton play or have a 3D view of a 1D universe.

2D elements
A configuration’s visual representation exploits text and color codes to represent several pieces of information (see Figure 3). Each cell displays its coordinates on the discrete Cartesian plane, and the state’s numeric value in textual format. Shapes such as pyramid, cube, sphere, as well as colors, can be associated with states. We can also use such a coding to represent properties of individual cells in relation to the universe’s current configuration. Thus, a cell can be an internal cell if it’s alive (that is, not in the \( q_0 \) state) and has no dead neighbor, a border cell if it’s alive and has at least one dead neighbor, or a shore cell if it’s dead and has some live neighbor.

A third, alternative representation highlights all the states in the core so that users can get an overall qualitative impression of the configuration (see Figure 4). Users can select the form of representation to be used and move from one to another during a session, without interrupting the execution. The execution is suspended if users select a state and modify it interactively or, of course, if they decide to change the execution’s current law.

We can also use the 2D representation to visualize monodimensional or bidimensional CA. With a monodimensional CA, we can represent the CA activity’s story by presenting the configuration at time \( t_{i+1} \) above the configuration at time \( t_i \). In the 3D case, we can observe the universe’s planar projections according to the Cartesian axis (for example, XY or YZ). Finally, any bitmap can be the starting state of a 2D CA, so we can make a picture evolve.

3D elements
We have devised some effective representations of 3D universes. They can also be used for other numbers of dimension, so we can achieve 3D representations of, say, 2D universes. Figure 5 (next page) shows the relief view, in which the cell state, besides defining its color, is also taken as the elevation on the \( z \) axis. The resulting landscape is represented by a texture of straight triangles connecting von Neumann neighbors, and we can explore it with the usual mechanisms for rotation, panning, and zooming of 3D representations.

We obtain a full-fledged 3D representation by associating 3D solid shapes with states, as Figure 6 shows. Each solid is rendered by a certain number of triangles. A less detailed, but more computationally efficient, effect is obtained by defining 2D textures for each shape so that two triangles are sufficient for each cell.

ExcapeMe can also exploit an efficient, though less effective, representation in which a cube is associated with each cell so as to construct two rendering triangles from each cube face and select the triangle to use for rendering for each face (the same for all cells).
Audio elements

We have implemented two plug-ins for audio rendering. In the Synthesizer plug-in (see Figure 7), we can use four channels and associate a different instrument with each channel. We generate the sound by considering a cell’s x coordinate as defining a note to be played and using the state’s value (mod 4) as a channel selector. Similarly, in the Sequencer plug-in, 47 instruments are used and the state value (mod 47) determines the instrument to be played, while the x coordinate (mod 16) determines the beat at which to play. Users can vary the number of beats per minute, affecting the speed at which the CA evolves, which ExcapeMe synchronizes with the presentation.

In both cases, we used the MIDI peripheral and adjusted different parameters according to the type of rendering. A SAVE utility lets us record and subsequently play the generated music.

Researchers can use ExcapeMe’s interactive and multimedia features for both scientific and artistic purposes, as the sidebar “Sample Uses for ExcapeMe” illustrates.

Conclusions

ExcapeMe is a testbed for exploring properties of models formalized through CA, with an open architecture that allows several extensions. Developers can add new forms to ExcapeMe by providing suitable wrappers for them and interfacing them with the system exploiting a basic communication protocol. Because of decoupling, it’s also possible to use only ExcapeMe’s computational abilities to produce universe configurations, which users can communicate to other applications without exploiting the user interface. For example, users can feed 1D universe configurations as sequences of parameters to a system for generating multimedia events, where interaction is based on recognizing a dancer’s movements.

Conversely, ExcapeMe’s components can represent bidimensional configurations produced by other systems if they’re properly formatted. For example, we can exploit them for visualizing the evolution of models based on multidimensional parallel systems, such as in studies on diffusion of pollutants or nutrients in the environment.

References


Readers may contact Paolo Bottoni at bottoni@di.uniroma1.it.
Sample Uses for ExcapeMe

In a typical use of ExcapeMe, a user can explore a 3D universe by simultaneously examining bidimensional projections (the projection on the XY plane is currently shown) and rotating the 3D representation of the same universe (see Figure A). At the same time, the user can map the XY projection onto the Sequencer audio renderer. A black spot in an entry identifies that the cell at the position \((X \mod 16, Y \mod 47)\) is in a non-quiescent state.

We have also used ExcapeMe to produce artistic performances. The law of Figure B shifts active cells to the right in state 1.

Figure C shows a configuration of a 1D universe, which in turn effects the computer graphics application of Figure D. A position’s absolute value (mod 8) is associated with a graphical fountain. Every time a cell enters state 1, the corresponding graphical fountain emits a “water” jet, to which the fall of drops follows, according to gravity. In the performance’s execution, we use the same sequence of states to provide parameters driving the production of a musical piece. Because ExcapeMe can communicate on TCP/IP connections, the graphical engine and the musical execution occurred on different computers. An artist can modify the laws to obtain different effects.