

run, can be saved by including line-by-line comments while the intent is fresh in your mind.

THE GUIDELINE:

Aim to comment every single line of code

Let's reiterate the guidelines, in case you were asleep:

- #10. Avoid looping whenever possible.
- #9. Don't waste your time trying to find obscure (and unreadable) non-looping solutions.
- #8. Use nested arrays to keep related things together.
- #7. Don't go ape over each (``).
- #6. Don't be afraid of files.
- #5. Document in writing every component of every file.
- #4. Spend ten percent of your APL time learning more APL.
- #3. Aim to have no global variables in your workspace, and minimize the passing of globals between functions.
- #2. Include comments at the beginning of every function that document intent, syntax, and assumptions.
- #1. Aim to comment every single line of code.

Follow these guidelines and you will improve APL's reputation—as well as your own. ■

Gary A. Bergquist, A.S.A. is president of Zark Incorporated, an APL consulting firm that works mainly on insurance applications. Among the products supported by Zark are: the Zark APL Tutor, the Zark Library of Utility Functions, the quarterly publication Zark APL Tutor News, and the Variable Products Administration (VPA) system. Gary can be reached at GABergquist@SNET.net or at 860-872-7806.

Programming Cellular Automata in APL2

—by *Manuel Alfonso*
Madrid, Spain

Universidad Autonoma de Madrid
Dept. Ingenieria Informatica
Manuel.Alfonseca@ii.uam.es

Acknowledgment: *This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project numbers TIC-96-0723-G02-01 and TEL97-0306.*

THIS PAPER REVIEWS THE CONCEPT OF CELLULAR AUTOMATA and describes a simple way to implement them in APL2. Two cellular automata are programmed: the well-known Conway's game of life, and one that simulates an ecosystem and displays a behaviour similar to that represented by the Volterra differential equations. Cellular automata are shown to be a powerful simulation tool for this kind of systems.

We shall start with a few definitions:

Finite automata

A *deterministic finite automaton* [1] consists essentially of three elements:

- A finite set of input symbols.
- A finite set of states.
- A transition function defining the next state of the automaton given its input and its state. This function is deterministic (there is a single next state for every combination of input and current state) and is usually described as a *transition table*.

A fourth element usually considered is the *initial state* of the automaton, a distinguished member of the set of states.

A *probabilistic finite automaton* consists of the same elements as a deterministic finite automaton, but the transition function is probabilistic; i.e., for every combination of input and current state there are several possible next states, each with a given probability.

Cellular automata

A *cellular automaton* [2–5] is a regular grid of points, to each of which is associated a finite automaton, which may differ from other automata in the grid only in its initial state.

- The input to the automaton associated to a given point is the set of states of the automata associated to the neighboring points in the grid.

Cellular automata may differ in the following:

- The shape and size of the grid, usually square, rectangular or triangular, which may be infinite.
- The definition of the set of neighbors to a given grid point.
- The actual finite automaton associated to each point in the grid. If this automaton is deterministic/probabilistic, the cellular automaton is deterministic/probabilistic.
- The set of initial states of all the automata.

In cellular automata, specially those that use an infinite grid, the set of states of the finite automaton associated to the grid points usually includes a special symbol (the *empty* state). The number of automata not initially in the empty state is assumed to be finite.

The game of Life

Introduced by John Conway [6], it is a very simple cellular automaton that may give rise to extremely complicated behaviors and has been proved to be computationally complete; i.e., it is able to perform any computation which may be performed by a digital computer, a Turing machine, a genetic algorithm or a neural network.

The cellular automaton associated to the game of Life is defined thus:

- The grid is rectangular and potentially infinite.
- Each finite automaton has two states: empty (also called dead, represented by a zero or a space character) and full (also called alive, represented by a one or a star symbol, *). The set of states is thus represented by the two boolean numbers {0,1} or by the two characters ' *'.
- The transition function is defined by the following simple rules:
 - ▶ If the automaton associated to a cell is in the empty state, it goes into the full state if and only if the number of its neighbors in the full state is exactly three.
 - ▶ If the automaton associated to a cell is in the full state, it goes into the empty state if and only if the number of its neighbors in the full state is less than two or more than three.
 - ▶ In any other case, the automaton remains in the same state.

Each time step is called a “generation.” The set of all the cells alive at a given time step is called the “population.”

The fact that the grid is potentially infinite makes the game of life difficult to implement. However, restricted versions, associated to a grid of finite dimensions, are very simple, at the cost of

losing computational completeness. The implementation is almost trivial in APL, since the states of all the automata in the grid may be represented by a boolean matrix. Listing 1 shows a program that implements the game of Life.

```
[0] Y LIFE N;X1;⊞IO
[1] ⍝ The game of life
[2] ⊞IO←1
[3] →((0=⊞NC 'Y')∧0≠⊞NC 'A')/L
[4] G←1
[5] A←2=⍶20 20⍶2
[6] L:'Generation ',(⍶G),' Population ',⍶+/,A
[7] '-','-',[1](' *'[A+1],[1]'-'),'-'
[8] →(∧/,A=0)/0
[9] →(N≤G)/0
[10] G←G+1
[11] X1←0,0,[1](A,[1]0),0
[12] X1←(1⊖X1)+(¬1⊖X1)+(1⊖X1)+(¬1⊖X1)+(1⊖X1)+
(¬1⊖X1)+(1⊖X1)+¬1⊖X1
[13] X1←1 1←¬1 ¬1←X1
[14] A←(X1=3)∨A∧X1∈2 3
[15] →L,⊞
```

Listing 1: An APL program implementing the game of Life

The cellular automata for all the nodes in the grid are implemented simultaneously in just four sentences (11–14), with no loops. The first three compute for every cell the number of neighbors in the full state, while sentence 14 performs the change of state by applying the three rules indicated above.

This program performs subsequent generations of the game of Life up to generation number N. Two global variables are used: A (the boolean matrix defining the initial states of all the automata in the grid) and G (the initial number of generations). The program is written in such a way that it is possible to stop the execution of a run and continue it later from the point at which it was left. If the global variables do not exist, the program

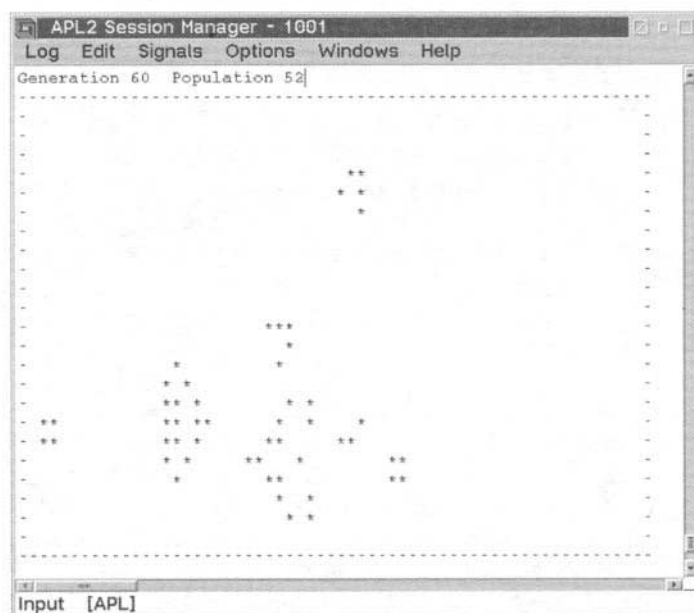


Figure 1

generates a random boolean matrix and starts a new generation count. This may also be done simply by specifying any value for the optional left argument of the function.

Figure 1 shows a classical situation in the game of life: a glider throwing cannon.

Ecosystem simulation by cellular automata

Cellular automata similar to the game of life, but more complicated, have been defined in the literature [7]. Some of these automata are able to model complicated systems, such as gas diffusion, usually represented by differential equations and simulated by means of continuous simulation languages [8].

In a previous paper [9] we have used an extension of the Volterra differential equations [10] to model different complex ecosystems. The result of this work can be seen at the following WWW address:

<http://www.ii.uam.es/~epulido/ecology/simul.htm>

The work presented here models the same system by means of a specially designed cellular automaton with the following characteristics:

- Each cell may contain up to four individuals of the prey (x), each pointing in a different direction.
- Each cell may also contain up to eight predators, which may exist in two different states (a, b) and point in one of the four main directions.
- The state of the automaton associated to each cell is defined by the individuals (up to twelve) currently occupying the cell and may be represented by a 12 bit boolean vector.
- The total cellular automaton state at a given time step may be represented by a $12 \times N \times M$ boolean APL object, where N and M is the size of the rectangular grid.
- The next state of each automaton is the result of applying two successive transition operations: a set of collisions and a set of movements.
- The transition function for the collisions is defined by the following rules:
 - ▶ The prey reproduces if there are at least two and at most three individuals in the same cell. The new individual generated takes one of the available (empty) orientations with the same probability.
 - ▶ A predator in the a state dies if there is no prey in the same cell.

- ▶ A predator in the a state goes into the b state if there are at least two prey individuals in the same cell. In this case, one of the prey individuals dies (is eaten).
 - ▶ A predator in the b state goes into the a state if there is no prey in the same cell.
 - ▶ A predator in the b state becomes two predators in the a state (reproduces) if there are at least two prey individuals in the same cell. In this case, one of the prey individuals is consumed. The new predator is generated only if there is an empty orientation in the a state in the cell. The actual orientation is chosen at random.
- After all the possible collisions have taken place, all the individuals in each cell move to the neighboring cell in the direction they are oriented. The grid is assumed to be a plane torus (i.e., the upper and lower rows are contiguous, as well as the right and left columns).

Listing 2 on the next page shows the APL2 program that implements the cellular automaton. It may be observed that, as in the game of life, there is a single loop for the generations (this loop is unavoidable), while all the collisions and movements at a given step are performed at the same time for all the cells, by using matricial operations.

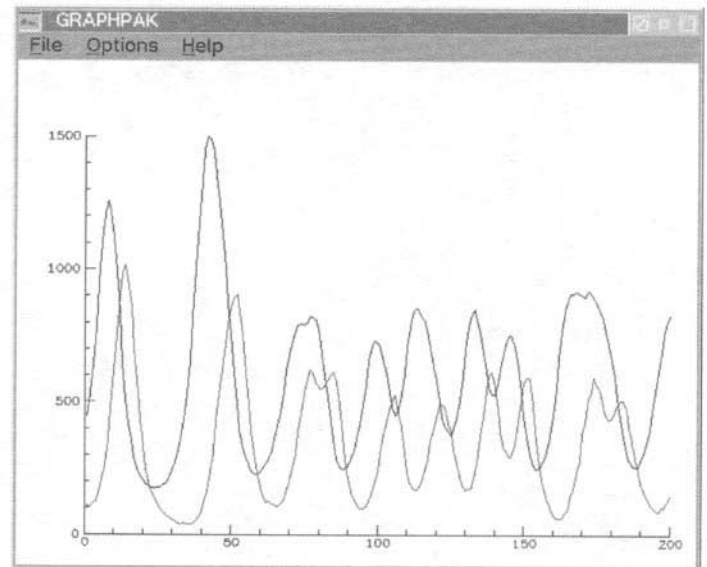


Figure 2

It is curious to observe how the evolution of this cellular automaton mimics the Volterra differential equations that regulate the predator/prey interaction, without making use of any of the tools usually associated to continuous simulation. Figure 2 shows the evolution of the predator/prey populations along time for one execution of the program. The similarity of the curves to the solution of the Volterra equations is obvious: the predator population follows the ups and downs in the prey population with a certain phase delay and a lower absolute amplitude fluctuation.

```

[0] Y ECOLOGY N;A1;X1;ΠIO
[1] * Simulation of an ecological system with a cellular automaton
[2] ΠIO+1
[3] -(0=ΠNC 'Y')^0#ΠNC 'A')/EO
[4] GEN+1
[5] C+R+10
[6] A+12 10 20p0
[7] A[4;]+(4,1+pA)p(0 0 0 1)[?(*/4,1+pA)p4]
[8] A[5 6 7 8;]+(4,1+pA)p(0 0 0 0 0 0 0 0 1)[?(*/4,1+pA)p10]
[9] →EO
[10] L:'Generation ',(vGEN),' Population ',v+/,A
[11] H+H,+/,A[4;];
[12] C+C,+/,A[4+18;];
[13] 'Herbivores ',(v~1pR),' Carnivores ',v~1pC
[14] X1<(v/A[4;]+(2*v/A[5 6 7 8;]+)4*v/A[9 10 11 12;];]
[15] '-','-',[1](' xaabbbb'[X1+1],[1]'-','-','-')
[16] EO:~(v/A,A=0)/0
[17] ~(N#GEN)/0
[18] GEN+GEN+1
[19] * Collisions: a+0x+
[20] * a+2x~b+x
[21] * b+0x~a
[22] * b+2x~2a+x
[23] * 2x ~3x
[24] A1+A
[25] A[5;]+A1[5;];]A^v/A1[4;];] * a+0x+
[26] A[6;]+A1[6;];]A^v/A1[4;];]
[27] A[7;]+A1[7;];]A^v/A1[4;];]
[28] A[8;]+A1[8;];]A^v/A1[4;];]
[29] X1+A1[5;];]A1<+A1[4;];] * a+2x~b+x
[30] A[5;]+A[5;];]A~X1
[31] A[9 10 11 12;];]X1 SET A[9 10 11 12;];]
[32] A[4;];]X1 RESET A[4;];]
[33] X1+A1[6;];]A1<+A1[4;];]
[34] A[6;];]+A[6;];]A~X1
[35] A[9 10 11 12;];]X1 SET A[9 10 11 12;];]
[36] A[4;];]X1 RESET A[4;];]
[37] X1+A1[7;];]A1<+A1[4;];]
[38] A[7;];]+A[7;];]A~X1
[39] A[9 10 11 12;];]X1 SET A[9 10 11 12;];]
[40] A[4;];]X1 RESET A[4;];]
[41] X1+A1[8;];]A1<+A1[4;];]
[42] A[8;];]+A[8;];]A~X1
[43] A[9 10 11 12;];]X1 SET A[9 10 11 12;];]
[44] A[4;];]X1 RESET A[4;];]
[45] A[9;];]+A[9;];]A~X1+A1[9;];]A~v/A1[4;];] * b+0x~a
[46] A[5;];]+A[5;];]vX1
[47] A[10;];]+A[10;];]A~X1+A1[10;];]A~v/A1[4;];]
[48] A[6;];]+A[6;];]vX1
[49] A[11;];]+A[11;];]A~X1+A1[11;];]A~v/A1[4;];]
[50] A[7;];]+A[7;];]vX1
[51] A[12;];]+A[12;];]A~X1+A1[12;];]A~v/A1[4;];]
[52] A[8;];]+A[8;];]vX1 * b+2x~aa+x
[53] X1+A1[9;];]A1<+A1[4;];]
[54] A[5;];]+A[9;];]A~X1
[55] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[56] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[57] A[4;];]X1 RESET A[4;];]
[58] X1+A1[10;];]A1<+A1[4;];]
[59] A[10;];]+A[10;];]A~X1
[60] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[61] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[62] A[4;];]X1 RESET A[4;];]
[63] X1+A1[11;];]A1<+A1[4;];]
[64] A[11;];]+A[11;];]A~X1
[65] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[66] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[67] A[4;];]X1 RESET A[4;];]
[68] X1+A1[12;];]A1<+A1[4;];]
[69] A[12;];]+A[12;];]A~X1
[70] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[71] A[5 6 7 8;];]X1 SET A[5 6 7 8;];]
[72] A[4;];]X1 RESET A[4;];]
[73] X1+1<+A1[4;];] * 2x+3x
[74] A[4;];]X1 SET A[4;];]
[75] * Movements
[76] A[1;];]+10A[1;];] * North
[77] A[5;];]+10A[5;];]
[78] A[9;];]+10A[9;];]
[79] A[2;];]+~10A[2;];] * East
[80] A[6;];]+~10A[6;];]
[81] A[10;];]+~10A[10;];]
[82] A[3;];]+~10A[3;];] * South
[83] A[7;];]+~10A[7;];]
[84] A[11;];]+~10A[11;];]
[85] A[4;];]+~10A[4;];] * West
[86] A[8;];]+~10A[8;];]
[87] A[12;];]+~10A[12;];]
[88] →L,Π

```

Listing 2: An APL program implementing an ecological system

Two successive executions of the program are never identical, even with the same initial conditions, as there are random effects during the collisions, which depend on the initial value of the random seed. This is different from the Volterra equations, but give this ecological simulation an even more realistic appearance.

Conclusion

The generality of cellular automata and their use to simulate systems usually modelled by means of continuous mathematical tools has been described. Other more complicated cellular automata have also been developed, that generate even more realistic simulations of complex ecosystems, including up to three different trophic levels. ■

References

1. Linz, P.: "An introduction to Formal Languages and Automata," D.C. Heath and Co., Lexington, 1990.
2. Von Neumann, J.: "Theory of Self-Reproducing Automata," Univ. of Illinois Press, Urbana, 1966.
3. Burks, A.W., ed.: "Essays on Cellular Automata," Univ. of Illinois Press, Urbana, 1970.
4. Wolfram, S.: "Theory and Application of Cellular Automata," World Sci. Publ., Singapore, 1986.
5. Kari, J.: "Cellular Automata. An Introduction," in "Artificial Life: Grammatical Models," ed. by G. Paun, Black Sea Univ. Press, Bucharest, 1995.
6. Berlekamp, E.R.; Conway, J.H.; Guy, R.K.: "Winning Ways for your Mathematical Plays," Academic Press, 1982.
7. Succi, S.: "Cellular Automata Modeling on IBM 3090/VF," IBM European Scientific Center, ICE-0014, 1987.
8. Y.Monsef, "Modelling and Simulation of Complex Systems," SCS Int., Erlangen, 1997.
9. M.Alfonseca, J.de Lara, E. Pulido: "Educational simulation of complex ecosystems in the World-Wide Web," Proc. ESS'98, SCS Int., p.248-252, 1998.
10. Volterra, V.: "Leçons sur la Théorie Mathématique de la Lutte pour la Vie," Gauthier-Villars, Paris, 1931.

Manuel Alfonseca is the Subdirector of Research at Escuela Técnica Superior de Informática, Universidad Autónoma de Madrid, Spain. He can be reached at "Manuel.Alfonseca@ii.uam.es". You can see more information about Manuel and his work on his website, "www.ii.uam.es/~alfonsec".