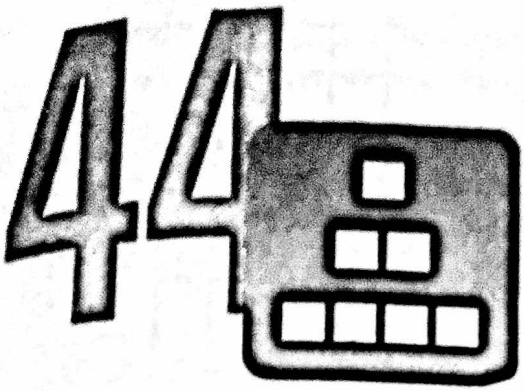

# CELLULAR AUTOMATA

## *The Game of Life*

In a darkened room, eager faces peer at a steadily evolving pattern of little white squares on a display screen. Within this pattern, some populations of squares may be growing while others appear headed for extinction; thus the name *life* given by its creator, John Conway, a Cambridge mathematician, to a game that has intrigued millions. When it was first described by Martin Gardner in the October 1970 issue of *Scientific American*, the game quickly established itself as a major spare-time preoccupation of senior and graduate students having access to a graphics computer. Even some faculty found themselves drawn into the magic circle surrounding those scintillating screens, perhaps watching something like the succession of patterns in Figure 44.1.

The game of Life is an example of a cellular automaton. Formally, we must think of an infinite square grid in which each cell exists in one of two states, "living" or "dead." Each cell is a simple automaton that at every tick of a great clock must decide which state it will be in until the next tick. It makes this decision on the basis of not only its present state but also those of its eight neighbors, four adjacent along sides and four adjacent at corners. Here are the rules on which that decision is based:

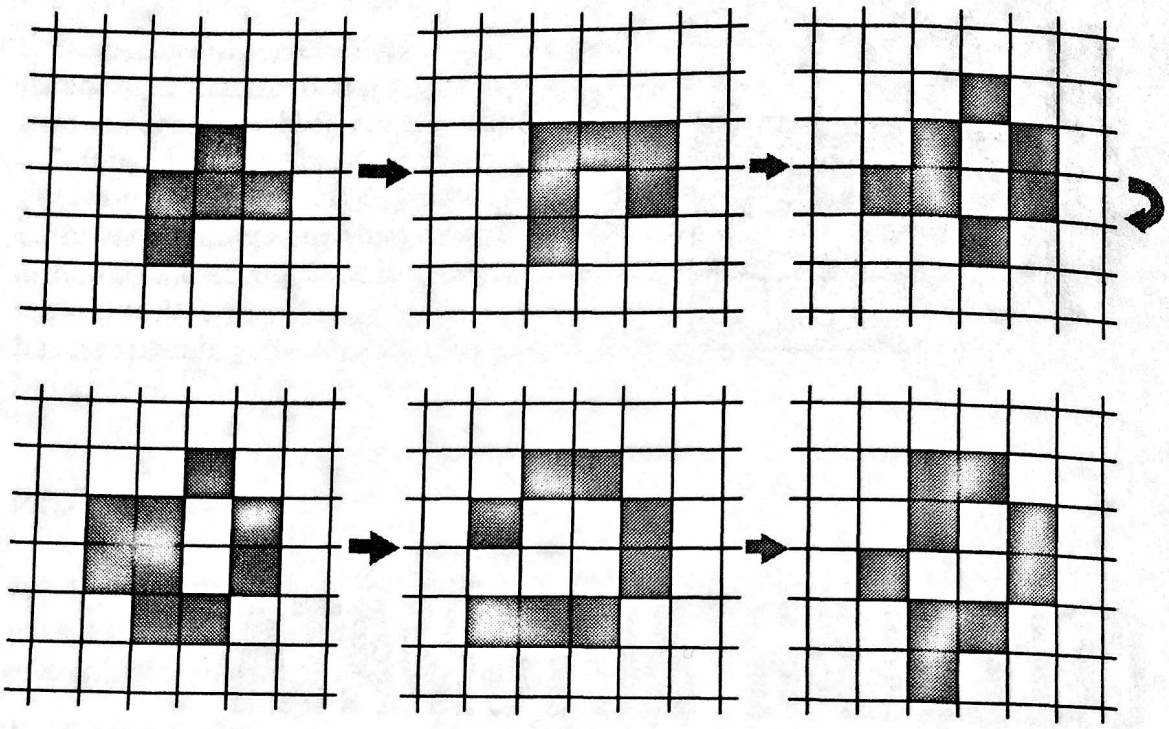

**Figure 44.1** Six generations of Life

**1.** If the cell is alive at time $t$, it will remain alive at time $t+1$ if it is not "overcrowded" or "undernourished." In other words, it must have at least two living neighbors and no more than three.
**2.** If the cell is dead at time $t$, it will remain dead unless it has three "parents." That is, the cell must have three living neighbors in order to be born again.

Although they sound almost arbitrary to the uninitiated, Conway went to some trouble to discover rules which made the behavior of populations of live cells as difficult as possible to predict. On the way, he experimented with dozens of different rules, discarding each set in turn. Among the notes he sent to Gardner for the 1970 article was the prediction that no population could grow without limit—sooner or later, every population either would become extinct (all cells eternally dead) or would fall into an endlessly repeating cycle of patterns.

The game of Life can be played by hand on a ruled grid using counters. It is difficult, however, to play the game with just one color of counter because one must remember, in going from one generation to the next, just which counters were present in the former. For this reason, it is much better to use two colors, one for already living cells (say, white) and the other for newly born cells (say, gray). Given the $t$ generation of white counters, first go around putting in gray

counters wherever a new cell is born. Then remove white counters representing cells which are to die. Finally, replace the gray counters by white ones.

Of course, if one has access to a computer, much more can be done. Programming the game of Life is reasonably straightforward, and even if the computer has no graphics terminal attached, successive generations can be printed as patterns of X's on a printer. In some ways, having hard copy is an advantage, especially for examining past populations for the purpose of detailed analysis. It was by using a computer that Conway and many others caught up in the excitement were able to find novel and interesting patterns. An initial pattern of live cells would be typed into the computer and appear on the screen, and with a press of a key, successive generations of the pattern would appear.

Figure 44.2 shows *(a)* the evolution (with the 13 intermediate generations not shown) of a row of seven live cells into a "honey farm" consisting of four "beehives" and *(b)* a five-cell configuration called a *glider.* The glider repeats itself every 4 cycles but ends up in a new position!

With so many people playing Life, it was not long before someone discovered a counterexample to Conway's conjecture that no populations could grow without limit. A group of six students at the Massachusetts Institute of Technology discovered a "glider gun," a configuration that emits a glider every 30 generations. A pattern which grows into the glider gun after 39 moves is shown in Figure 44.3. With gliders flying out of the gun every 30 moves, the total number of live cells obviously grows without limit. The same group of students managed to arrange 13 gliders crashing together to form a glider gun!

The algorithm displayed below computes successive generations for Life in matrix $L$. A 1 in the $i, j$th entry represents a live cell, and a 0 represents a dead cell.

LIFE

```
1. for i ← 1 to 100
   1. for j ← 1 to 100
      1. s ← 0
      2. for p ← i − 1 to i + 1          /compute effect
            for q ← j − 1 to j + 1       /of neighbors
               s ← s + L(p, q)
      3. s ← s − L(i, j)
      4. if (s = 3) or (s + L(i, j) = 3)
            then X(i, j) = 1             /store life or death
            else X(i, j) = 0             /in auxiliary array X
2. for i ← 1 to 100
   1. for j ← 1 to 100
      1. L(i, j) ← X(i, j)               /refresh L
      2. display L(i, j)                 /display L
```

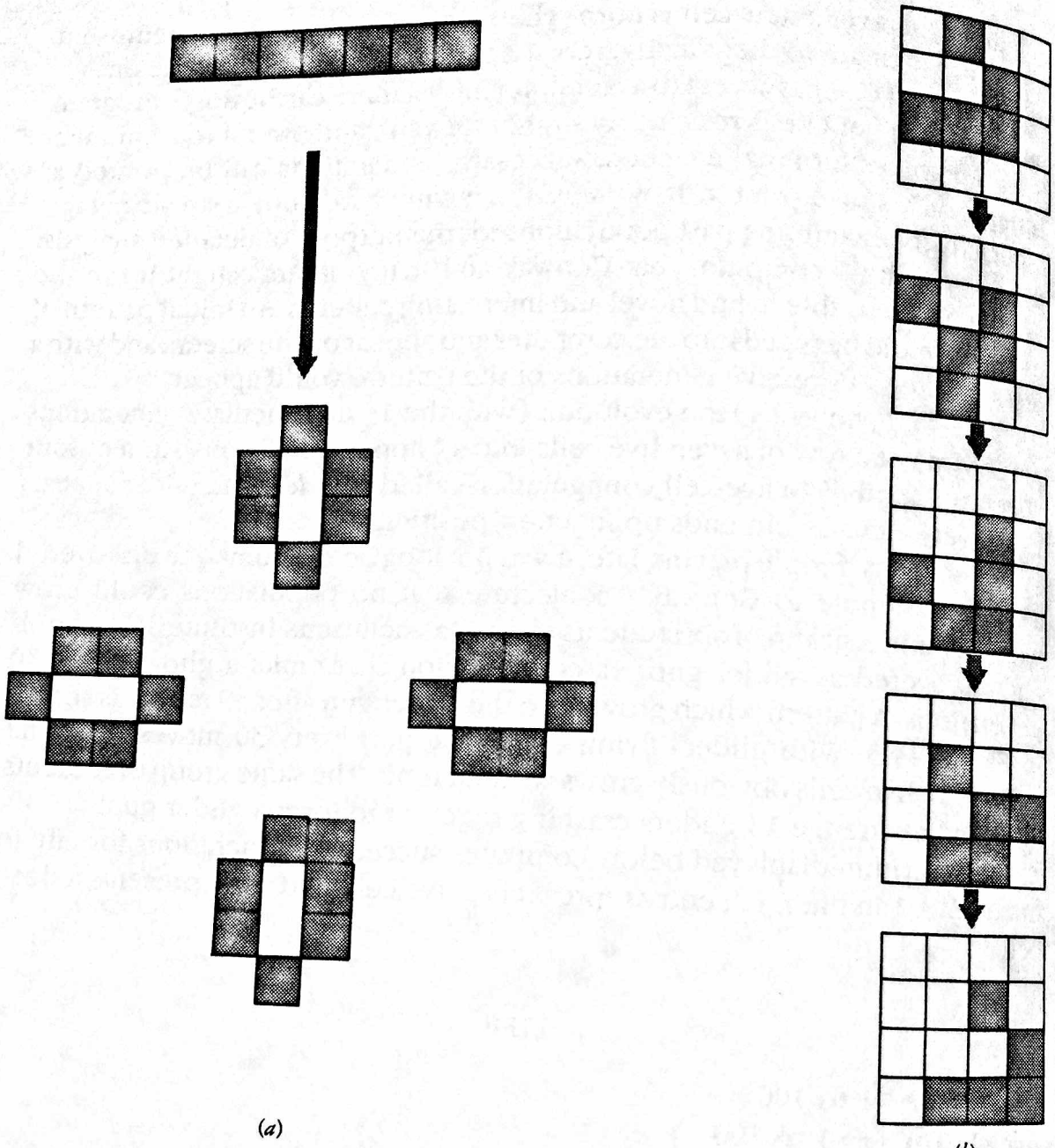


**Figure 44.2** Evolution of a honey farm (left) and motion of a glider (right)

In writing a program based on this design, the syntax peculiar to the language used must, of course, be substituted for the replacement arrows (representing assignment statements) and indentations (indicating the scope of **for** and **if** statements).

This algorithm is relatively straightforward. The first double-**for** loop computes new values for the $(i, j)$th entry of $L$ by scanning the $3 \times 3$ neighborhood

of $(i, j)$. The variable $s$ contains the sum of these values (with the value of $L(i, j)$ subtracted away). The statement 1.1.4 embodies the life-and-death criteria mentioned earlier. The auxiliary array $X$ holds the new value of $L(i, j)$. This value cannot yet be placed in $L$ since the old value of $L(i, j)$ must participate in four more computations of $L$ values. Only at step 2.1.1 do the new $L$ values (carried in $X$) finally replace the old ones. At the same time, so to speak , the new values are displayed on one's screen at line 2.1.2.

This algorithm computes just one generation of Life. It must be embedded in yet another loop that the programmer may structure according to taste: a fixed number of generations may be computed or the program may be terminated by a keystroke.

Living configurations that reach the edge of the $100 \times 100$ matrix boundary will automatically die. One may prevent this by "wrapping the matrices around" using modular arithmetic. For example $L(100,1)$ will be adjacent to $L(1,1)$. In this case, however, the computation of $s$ in the loop at 1.1.2 must be modified to reflect the new rules of cell adjacency.

Any program based on this algorithm must include the appropriate initialization statements.

The academic area called cellular automata has fascinated many computer scientists and mathematicians since John von Neumann invented the subject in 1950. His aim was to construct a self-reproducing "machine." On being persuaded that cellular spaces were an ideal setting for gedanken experiments in this area, von Neumann arrived, finally, at a cellular automaton each cell of which had 29 states. He proved the existence of (but did not explicitly state!) a configuration of about 200,000 cells which would self-reproduce. This meant that when the cellular automaton was put into this configuration, it would, after a definite period, result in two such configurations, side by side. Since von Neumann's time, much simpler self-reproducing cellular automata have been found.

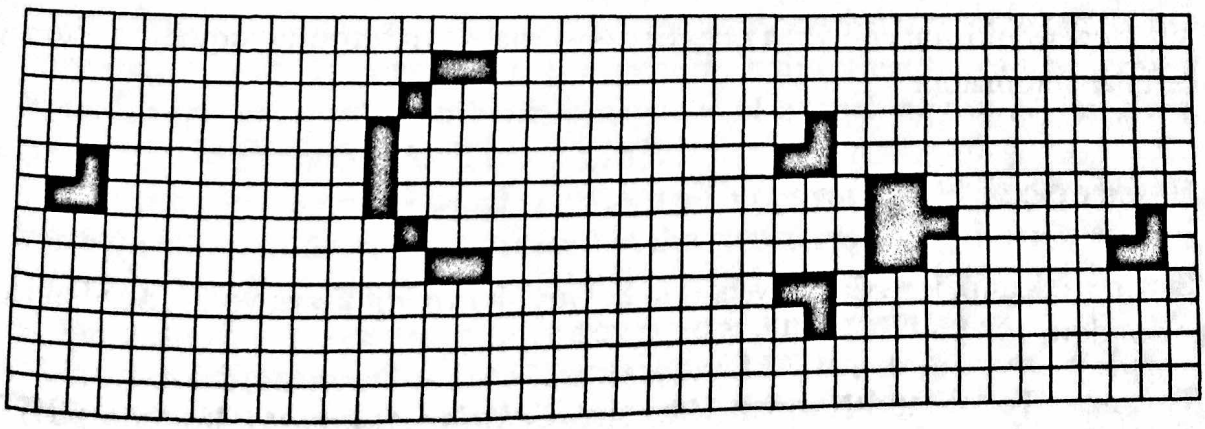

**Figure 44.3** A glider gun

In formal terms, a *cellular automaton* consists of three things:

1. An automaton, a copy of which is associated with each cell in an infinite $n$-dimensional grid.
2. A neighborhood function that specifies which of the cells adjacent to a given one may affect it.
3. A transition function that specifies, for each combination of states in "neighboring" cells, what the next state of the given cell will be.

Besides constructing self-reproducing automata, researchers have developed cellular automata which can carry out computations. This is usually done by embedding the equivalent of a Turing machine in a cellular space: Moving patterns of states represent the read/write head, scanning what other state patterns represent as a tape. There are, at least in "mental" existence, universal computational cellular automata, even ones that self-reproduce (see Chapter 46). Indeed, it was recently shown that the game of Life itself has this property!

## Problems

1. Write the Life algorithm as an actual program in a language of choice.

2. A one-dimensional cellular automaton coinsists of an infinite strip of cells. A great advantage of such automata is that one can watch the history of a given binary configuration unfold on a display screen if successive generations are displayed as successive rows. Write a program to do exactly this. Use neighborhoods consisting of two cells on either side of the central one. Employ the following rule: If two or four of the cells in a given cell's neighborhood are alive (state 1) at time $t$, the given cell will be alive at time $t + 1$. Otherwise, it will be dead. Note that each cell is a member of its own neighborhood.

3. How would you convert a one- or two-variable differential equation into a cellular automaton?

## References

E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways,* vol. 2, Academic, London, 1982.

Tommaso Toffoli and Norman Margolis. *Cellular Automata Machines.* MIT Press, Cambridge, MA, 1987.