CHAPTER

14 Complex Systems

We introduce cellular automata, neural networks, genetic algorithms, and growing networks to explore the concepts of self-organization and complexity. Applications to sandpiles, fluids, earthquakes, and other areas are discussed.

14.1 ■ CELLULAR AUTOMATA

Part of the fascination of physics is that it allows us to reduce natural phenomena to a few simple laws. It is also fascinating to think about how a few simple laws can produce the enormously rich behavior that we see in nature. In this chapter we will discuss several models that illustrate some of the new ideas that are emerging from the study of *complex systems*.

The first class of models that we will discuss are known as *cellular automata*. Cellular automata were introduced by von Neumann and Ulam in 1948 and are mathematical idealizations of dynamical systems in which space and time are discrete and the quantities of interest have a finite set of discrete values that are updated according to a local rule. A cellular automaton can be thought of as a lattice of sites or a checkerboard with colored squares (the cells). Each cell changes its state at the tick of an external clock according to a rule based on the present configuration of the cells in its neighborhood. Cellular automata are examples of discrete dynamical systems that can be simulated exactly on a digital computer.

Because the original motivation for studying cellular automata was their biological aspects, the discrete locations in space are frequently referred to as cells. More recently, cellular automata have been applied to a wide variety of physical systems ranging from fluids to galaxies. We will usually refer to sites rather then cells, except when we are explicitly discussing biological systems. The important characteristics of cellular automata include the following:

1. Space is discrete and consists of a regular array of sites. Each site has a finite set of values.
2. The rule for the new value of a site depends only on the values of a *local* neighborhood of sites near it.
3. Time is discrete. The variables at each site are updated *simultaneously* based on the values of the variables at the previous time step. Hence, the state of the entire lattice advances in discrete time steps.

We first consider one-dimensional cellular automata and assume that the neighborhood of a given site is the site itself and the sites immediately to the left and right of it. Each site is

t:	111	110	101	100	011	010	001	000
$t+1$:	0	1	0	1	1	0	1	0

Figure 14.1 Example of a local rule for the evolution of a one-dimensional cellular automaton. The variable at each site can have values 0 or 1. The top row shows the $2^3 = 8$ possible combinations of three sites. The bottom row gives the value of the central site at the next iteration. For example, if the value of a site is 0 and its left neighbor is 1 and its right neighbor is 0, the central site will have the value 1 in the next time step. This rule is termed 01011010 in binary notation (see the second row), the modulo-two rule or rule 90. Note that 90 is the base ten (decimal) equivalent of the binary number 01011010, that is, $90 = 2^1 + 2^3 + 2^4 + 2^6$.

assumed to have two states (a Boolean automaton). An example of such a rule is illustrated in Figure 14.1, where we see that a rule can be labeled by the binary representation of the update rule for each of the eight possible neighborhoods and by the base ten equivalent of the binary representation. Because any eight digit binary number specifies a one-dimensional cellular automaton, there are $2^8 = 256$ possible rules.

Class `OneDimensionalAutomatonApp` takes the decimal representation of the rule as input and produces the rule array `update`, which is used to update each lattice site using periodic boundary conditions. The `OneDimensionalAutomatonApp` class manipulates numbers using their binary representation. Note the use of the bit manipulation operators `>>>` and `&` (AND) in method `setRule`. To understand how the right shift operator `>>>` works, consider the expression `13 >>> 1`. In this case the result of the shift operator is to shift the bits of the binary representation of the integer 13 to the right by one. Because the binary representation of 13 is 1101, the result of the shift operator is 0110. (The left-hand bits are filled with 0s as needed.) To understand the nature of the `&` operator, consider the expression `0110 & 1`, which we can write as `0110 & 0001`. In this case the result is `0000` because the `&` operator sets each of the the resulting bits to 1 if the corresponding bit in both operands is 1; otherwise, the bit is zero.

We use the `LatticeFrame` class to represent the sites and their evolution. At a given time, the sites are drawn in the horizontal direction; time increases in the vertical direction. In method `iterate`, the `%` operator is used to determine the left and right neighbors of a site using periodic boundary conditions. Also, note the use of the left shift operator `<<` in method `iterate`. A more complete discussion of bit manipulation is given in Section 14.6.

Listing 14.1 One-dimensional cellular automaton class.

```
package org.opensourcephysics.sip.ch14.ca;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

public class OneDimensionalAutomatonApp extends AbstractCalculation {
   LatticeFrame automaton = new LatticeFrame("");
   // update[] maps neighborhood configurations to 0 or 1
   int[] update = new int[8];

   public void calculate() {
      control.clearMessages();
      int L = control.getInt("Linear dimension");
      int tmax = control.getInt("Maximum time");
      // default is lattice sites all zero
      automaton.resizeLattice(L, tmax);
      // seed lattice by putting 1 in middle of first row
```