```
   public void doStep() {
      freeway.step();
   }

   public void reset() {
      control.setValue("Number of cars", 10);
      control.setValue("Road length", 50);
      control.setValue("Slow down probability", 0.5);
      control.setValue("Maximum velocity", 2);
      control.setValue("Steps between plots", 1);
      enableStepsPerDisplay(true);
   }

   public void resetAverages() {
      freeway.flow = 0;
      freeway.steps = 0;
   }

   public static void main(String[] args) {
      SimulationControl control = SimulationControl.createApp(new FreewayApp());
      control.addButton("resetAverages", "resetAverages");
   }
}
```

Problem 14.5. Cellular automata traffic models

a. Run `FreewayApp` for 10 cars on a road of length 50, with $v_{\max} = 2$ and $p = 0.5$. Allow the system to evolve before recording the flow rate. Repeat the simulation with a different initial configuration at least several more times to estimate the uncertainty in the data. Repeat for 1, 2, 5, 20, 30, and 40 cars. Plot the flow rate versus the density. This plot is called the *fundamental diagram*. Explain its qualitative shape. At what density do traffic jams begin to occur?

b. Repeat part (a) with a road of length 500 and the same car densities. Use other road lengths to determine the minimum road length needed to obtain results that are independent of the length of the road.

c. Add methods to your classes to compute the velocity and gap distributions, where the gap is defined as the distance between two cars.

d. For a fixed road length compare your results for $v_{\max} = 1$ with your results for $v_{\max} = 2$. Also consider $v_{\max} = 5$. Are there any qualitative differences in the behavior of the cars?

e. Explore the effect the speed reduction probability by considering $p = 0.2$ and $p = 0.8$.

f. Add on- and off-ramps separated by a fixed distance. One way to do so is to choose a car at random and have it slow down as it approaches the off-ramp and exits. To maintain a constant density, allow a car to enter the on-ramp whenever a car leaves the highway. What is the effect of adding the on- and off-ramps?

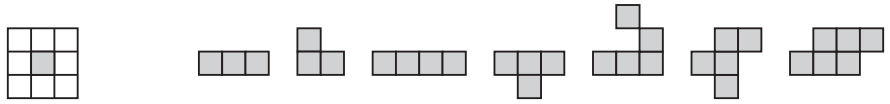

Figure 14.2: (a) The local neighborhood of a site in the Game of Life is given by the sum of its eight neighbors. (b) Examples of initial configurations for the Game of Life, some of which lead to interesting patterns. Live cells are shaded.

g. Modify your program to simulate a two lane highway. You will need to choose rules for moving from one lane to the other. Some possibilities to explore include the following. One reason for a car to move to the left lane is that the car is moving at less than the maximum speed and cannot increase its speed due to the car in front of it. Such a car could move to the left lane if there were a free space to the left. One reason for a car to move to the right lane is that there is a car immediately behind it. How does the behavior of the two lane highway differ from that of the one lane highway?

h. Modify your two lane simulation so that there are two kinds of vehicles (for example, cars and trucks) with different values of $v_{\max}$. How do the gap and velocity distributions change? Compute separate values for the truck and car flows as well as the total flow. Compute the average speed of the trucks and compare it with that of cars.

Because one-dimensional cellular automata models are limited, we consider several two-dimensional models. The philosophy is the same except that the neighborhood contains more sites. For the eight neighbor sites shown in Figure 14.2a, there are $2^9 = 512$ possible configurations for the eight neighbors and the center site, and 2^{512} possible rules. Clearly, we cannot go through all these rules in any systematic fashion as we did for one-dimensional cellular automata. For this reason, we will choose our rules based on other considerations.

The Game of Life. The rules used in `LifeApp` implement a popular two-dimensional cellular automaton known as the *Game of Life*. This model, invented in 1970 by the mathematician John Conway, produces many fascinating patterns. The rules of the game are simple. For each cell determine the sum of the values of its four nearest and four next-nearest neighbors (see Figure 14.2a). A "live" cell (value 1) remains alive only if this sum equals 2 or 3. If the sum is greater than 3, the cell will "die" (become 0) at the next iteration due to overcrowding. If the sum is less than 2, the cell will die due to isolation. A dead cell will come to life only if the sum equals 3.

Listing 14.5: Implementation of the Game of Life.

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

public class LifeApp extends AbstractSimulation {
   LatticeFrame latticeFrame = new LatticeFrame("Game of Life");
   byte[][] newCells;
```