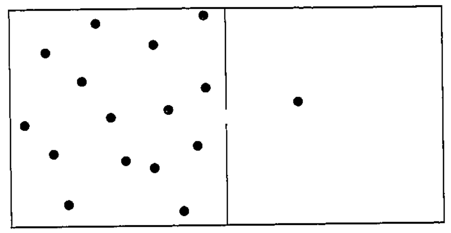

Figure 7.1 A box is divided into two equal halves by a partition. After a small hole is opened in the partition, one particle can pass through the hole per unit time.

algorithm. These algorithms are sometimes called *pseudorandom number generators* to distinguish their output from intrinsically random physical processes, such as the time between clicks in a Geiger counter near a radioactive sample.

For the present we will be content to use the random number generator supplied with Java, although the random number generators included with various programming languages vary in quality. The method `Math.random()` produces a random number r that is uniformly distributed in the interval $0 \le r < 1$. To generate a random integer i between 0 and $N-1$, we write

```
int i = (int)(N*Math.random());
```

The effect of the `(int)` cast is to eliminate the decimal digits from a floating point number. For example, `(int)(5.7) = 5`.

The algorithm for simulating the evolution of the model can be summarized by the following steps:

1. Use a random number generator to choose a particle at random.
2. Move this particle to the other side of the box.
3. Give the particle a random position on the new side of the box. This step is for visualization purposes only.
4. Increase the "time" by unity.

Note that this definition of time is arbitrary. Class `Box` implements this algorithm and class `BoxApp` plots the evolution of the number of particles on the left half of the box.

Listing 7.1 Class `Box` for the simulation of the approach to equilibrium.

```
package org.opensourcephysics.sip.ch07;
import java.awt.*;
import org.opensourcephysics.display.*;

public class Box implements Drawable {
   public double x[], y[];
   public int N, nleft, time;

   public void initialize() {
      // location of particles (for visualization purposes only)
      x = new double[N];
      y = new double[N];
      nleft = N; // start with all particles on the left
      time = 0;
```

```
      for(int i = 0;i<N;i++) {
         x[i] = 0.5*Math.random(); // needed only for visualization
         y[i] = Math.random();
      }
   }

   public void step() {
      int i = (int) (Math.random()*N);
      if(x[i]<0.5) {
         nleft--; // move to right
         x[i] = 0.5*(1+Math.random());
         y[i] = Math.random();
      } else {
         nleft++; // move to left
         x[i] = 0.5*Math.random();
         y[i] = Math.random();
      }
      time++;
   }

   public void draw(DrawingPanel panel, Graphics g) {
      if(x==null) {
         return;
      }
      int size = 2;
      // position of partition in middle of box
      int xMiddle = panel.xToPix(0.5);
      g.setColor(Color.black);
      g.drawLine(xMiddle, panel.yToPix(0), xMiddle,
           panel.yToPix(0.45));
      g.drawLine(xMiddle, panel.yToPix(0.55), xMiddle,
           panel.yToPix(1.0));
      g.setColor(Color.red);
      for(int i = 0;i<N;i++) {
         int xpix = panel.xToPix(x[i]);
         int ypix = panel.yToPix(y[i]);
         g.fillOval(xpix, ypix, size, size);
      }
   }
}
```

Listing 7.2 Target class for plotting the approach to equilibrium.

```
package org.opensourcephysics.sip.ch07;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

public class BoxApp extends AbstractSimulation {
   Box box = new Box();
   PlotFrame plotFrame = new PlotFrame("time", "number on left",
                                        "Box data");
   DisplayFrame displayFrame = new DisplayFrame("Partitioned box");

   public void initialize() {
      displayFrame.clearDrawables();
      displayFrame.addDrawable(box);
      box.N = control.getInt("Number of particles");
```