```
      int injections = (int) ((flowSpeed*numParticles-vxTotal)/scale);
      for(int k = 0;k<Math.abs(injections);k++) {
         int i = (int) (Math.random()*Lx); // choose site at random
         int j = (int) (Math.random()*Ly);
         // flip direction of horizontally moving particle if possible
         if((lattice[i][j]&(RIGHT|LEFT))==(
              (injections>0) ? LEFT : RIGHT)) {
            lattice[i][j] ^= RIGHT|LEFT;
         }
      }
   }

   public void draw(DrawingPanel panel, Graphics g) {
      if(lattice==null) {
         return;
      }
      // if s = 1 draw lattice and particle details explicitly
      // otherwise average velocity over an s by s square
      int s = spatialAveragingLength;
      Graphics2D g2 = (Graphics2D) g;
      AffineTransform toPixels = panel.getPixelTransform();
      Line2D.Double line = new Line2D.Double();
      for(int i = 0;i<Lx;i++) {
         for(int j = 2;j<Ly-2;j++) {
            double x = i+(j%2)*0.5;
            double y = j*SQRT3_OVER2;
            if(s==1) {
               g2.setPaint(Color.BLACK);
               for(int dir = 0;dir<NUM_CHANNELS;dir++) {
                  if((lattice[i][j]&(1<<dir))!=0) {
                     line.setLine(x, y, x+ux[dir]*0.4, y+uy[dir]*0.4);
                     g2.draw(toPixels.createTransformedShape(line));
                  }
               }
            }
            // draw points at lattice sites
            if((lattice[i][j]&BARRIER)==BARRIER||s==1) {
               Circle c = new Circle(x, y);
               c.pixRadius =
                    ((lattice[i][j]&BARRIER)==BARRIER) ? 2 : 1;
               c.draw(panel, g);
            }
         }
      }
      if(s==1) {
         return;
      }
      for(int i = 0;i<Lx;i += s) {
         for(int j = 0;j<Ly;j += s) {
            double x = i+s/2.0;
            double y = (j+s/2.0)*SQRT3_OVER2;
            double wx = 0, wy = 0;
            // compute coarse grained average velocity
            for(int m = i;m!=(i+s)%Lx;m = (m+1)%Lx) {
               for(int n = j;n!=(j+s)%Ly;n = (n+1)%Ly) {
                  wx += vx[lattice[m][n]];
```



```
                  wy += vy[lattice[m][n]];
               }
            }
            Arrow a = new Arrow(x, y, arrowSize*wx/s, arrowSize*wy/s);
            a.setHeadSize(2);
            a.draw(panel, g);
         }
      }
   }
}
```

Listing 14.14 Listing of the `LatticeGasApp` class.

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

public class LatticeGasApp extends AbstractSimulation {
   LatticeGas model = new LatticeGas();
   DisplayFrame display = new DisplayFrame("Lattice gas");

   public LatticeGasApp() {
      display.addDrawable(model);
      display.setSize(800, (int) (400*Math.sqrt(3)/2));
   }

   public void initialize() {
      int lx = control.getInt("lx");
      int ly = control.getInt("ly");
      double density = control.getDouble("Particle density");
      model.initialize(lx, ly, density);
      model.flowSpeed = control.getDouble("Flow speed");
      model.spatialAveragingLength =
           control.getInt("Spatial averaging length");
      model.arrowSize = control.getInt("Arrow size");
      display.setPreferredMinMax(-1, lx, -Math.sqrt(3)/2,
           ly*Math.sqrt(3)/2);
   }

   public void doStep() {
      model.flowSpeed = control.getDouble("Flow speed");
      model.spatialAveragingLength =
           control.getInt("Spatial averaging length");
      model.arrowSize = control.getDouble("Arrow size");
      model.step();
   }

   public void reset() {
      control.setValue("lx", 1000);
      control.setValue("ly", 500);
      control.setValue("Particle density", 0.2);
      control.setAdjustableValue("Flow speed", 0.2);
      control.setAdjustableValue("Spatial averaging length", 20);
      control.setAdjustableValue("Arrow size", 2);
      enableStepsPerDisplay(true);
```