```
      rule[LU|RD|RI] = RU|LD|RI;
      rule[LU|LE|RD] = RU|LE|LD;
      rule[RU|LE|LD] = LU|LE|RD;
      // two-particle cyclic rules
      rule[LE|RI] = RU|LD;
      rule[RU|LD] = LU|RD;
      rule[LU|RD] = LE|RI;
      // four-particle cyclic rules
      rule[RU|LU|LD|RD] = RU|LE|LD|RI;
      rule[RU|LE|LD|RI] = LU|LE|RD|RI;
      rule[LU|LE|RD|RI] = RU|LU|LD|RD;
      // stationary particle creation rules
      rule[LU|RI] = RU|S;
      rule[RU|LE] = LU|S;
      rule[LU|LD] = LE|S;
      rule[LE|RD] = LD|S;
      rule[LD|RI] = RD|S;
      rule[RD|RU] = RI|S;
      rule[LU|LE|LD|RD|RI] = RU|LE|LD|RD|S;
      rule[RU|LE|LD|RD|RI] = LU|LD|RD|RI|S;
      rule[RU|LU|LD|RD|RI] = RU|LE|RD|RI|S;
      rule[RU|LU|LE|RD|RI] = RU|LU|LD|RI|S;
      rule[RU|LU|LE|LD|RI] = RU|LU|LE|RD|S;
      rule[RU|LU|LE|LD|RD] = LU|LE|LD|RI|S;
      // add all rules indexed with a stationary particle (dual rules)
      for(int i = 0;i<S;i++) {
         // ^ is the exclusive or operator
         rule[i^(RU|LU|LE|LD|RD|RI|S)] = rule[i]^(RU|LU|LE|LD|RD|RI|S);
      }
      // add rules to bounce back at barriers
      for(int i = BARRIER;i<NUM_RULES;i++) {
         int highBits = i&(LE|LU|RU); // & is bitwise and operator
         int lowBits = i&(RI|RD|LD);
         rule[i] = BARRIER|(highBits>>3)|(lowBits<<3);
      }
   }
   static { // set average site velocities
      // for every particle site configuration i, calculate total net
      // velocity and place in vx[i], vy[i]
      vx = new double[NUM_RULES];
      vy = new double[NUM_RULES];
      for(int i = 0;i<NUM_RULES;i++) {
         for(int dir = 0;dir<NUM_CHANNELS;dir++) {
            if((i&(1<<dir))!=0) {
               vx[i] += ux[dir];
               vy[i] += uy[dir];
            }
         }
      }
   }
   public void initialize(int Lx, int Ly, double density) {
      this.Lx = Lx;
      this.Ly = Ly-Ly%2;                      // Ly must be even
      // approximate total number of particles
      numParticles = Lx*Ly*NUM_CHANNELS*density;
      // density = particles divided by maximum number possible
```

```
      lattice = new int[Lx][Ly];
      newLattice = new int[Lx][Ly];
      int sevenParticleSite = ((1<<NUM_CHANNELS)-1); // equals 127
      for(int i = 0;i<Lx;i++) {
         // wall at top and bottom
         lattice[i][1] = lattice[i][Ly-2] = BARRIER;
         for(int j = 2;j<Ly-2;j++) {
            // occupy site by 0 or 7 particles, average occupation will
            // be approximately the density
            int siteValue =
                 Math.random()<density ? sevenParticleSite : 0;
            lattice[i][j] = siteValue; // random particle configuration
         }
      }
      for(int j = 3*Ly/10;j<7*Ly/10;j++) {
         lattice[2*Lx/10][j] = BARRIER; // obstruction toward the left
      }
   }

   public void step() {
      // move all particles forward
      for(int i = 0;i<Lx;i++) {
         // define the columns of a 2D array
         int[] left = newLattice[(i-1+Lx)%Lx];
         // use abbreviations to align expressions
         int[] cent = newLattice[i];
         int[] rght = newLattice[(i+1)%Lx];
         for(int j = 1;j<Ly-2;j += 2) {
            // loop j in increments of 2 to decrease reads and writes
            // of neighbors
            int site1 = lattice[i][j];
            int site2 = lattice[i][j+1];
            // move all particles in site1 and site2 to their neighbors
            rght[j-1] |= site1&RIGHT_DOWN;
            cent[j-1] |= site1&LEFT_DOWN;
            rght[j] |= site1&RIGHT;
            cent[j] |= site1&(STATIONARY|BARRIER)|site2&RIGHT_DOWN;
            left[j] |= site1&LEFT|site2&LEFT_DOWN;
            rght[j+1] |= site1&RIGHT_UP|site2&RIGHT;
            cent[j+1] |= site1&LEFT_UP|site2&(STATIONARY|BARRIER);
            left[j+1] |= site2&LEFT;
            cent[j+2] |= site2&RIGHT_UP;
            left[j+2] |= site2&LEFT_UP;
         }
      } // handle collisions, find average x velocity
      double vxTotal = 0;
      for(int i = 0;i<Lx;i++) {
         for(int j = 0;j<Ly;j++) {
            int site = rule[newLattice[i][j]]; // use collision rule
            lattice[i][j] = site;
            // reset newLattice values to 0
            newLattice[i][j] = 0;
            vxTotal += vx[site];
         }
      }
      int scale = 4;
```