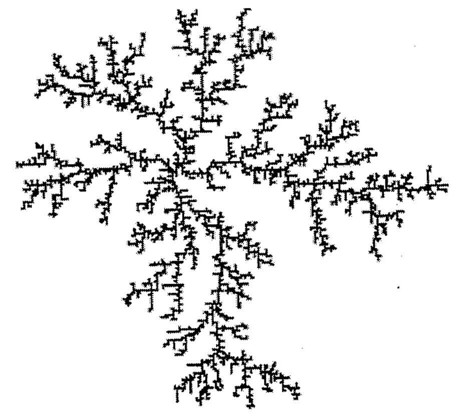

Figure 13.11 A DLA cluster of 4284 particles on a square lattice with $L = 300$.

million) until a large cluster is formed. A typical DLA cluster is shown in Figure 13.11. Some of the properties of DLA clusters are explored in Problem 13.9.

The following class provides a reasonably efficient simulation of DLA. Walkers begin just outside a circle of radius `startRadius` enclosing the existing cluster and centered at the seed site. If the walker moves away from the cluster, the step size for the random walker increases. If the walker wanders too far away (further than `maxRadius`), the walk is restarted.

Listing 13.5 Class for simulating diffusion limited aggregation.

```java
package org.opensourcephysics.sip.ch13;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.LatticeFrame;
import java.awt.Color;

public class DLAApp extends AbstractSimulation {
   LatticeFrame latticeFrame = new LatticeFrame("DLA");
   byte s[][];                       // lattice on which cluster lives
   int xOccupied[], yOccupied[]; // location of occupied sites
   int L;                            // linear dimension of lattice
   int halfL;                        // L/2
   int ringSize;                     // ring size in which walkers can move
   int numberOfParticles;            // number of particles in cluster
   // radius of cluster at which walkers are started
   int startRadius;
   // maximum distance from origin before a new walk is started
   int maxRadius;

   public void initialize() {
      latticeFrame.setMessage(null);
      numberOfParticles = 1;
      L = control.getInt("lattice size");
      startRadius = 3;
      halfL = L/2;
      ringSize = L/10;
      maxRadius = startRadius+ringSize;
      s = new byte[L][L];
```

```java
      s[halfL][halfL] = Byte.MAX_VALUE;
      latticeFrame.setAll(s);
   }

   public void reset() {
      latticeFrame.setIndexedColor(0, Color.BLACK);
      control.setValue("lattice size", 300);
      setStepsPerDisplay(100);
      enableStepsPerDisplay(true);
      initialize();
   }

   public void stopRunning() {
      control.println("Number of particles = "+numberOfParticles);
      // add code to compute the mass distribution here
   }

   public void doStep() {
      int x = 0, y = 0;
      if(startRadius<halfL) {
         // find random initial position of new walker
         do {
            double theta = 2*Math.PI*Math.random();
            x = halfL+(int) (startRadius*Math.cos(theta));
            y = halfL+(int) (startRadius*Math.sin(theta));
         } while(walk(x, y));
      }
      if(startRadius>=halfL) { // stop the simulation
         control.calculationDone("Done");
         latticeFrame.setMessage("Done");
      }
      latticeFrame.setMessage("n = "+numberOfParticles);
   }

   public boolean walk(int x, int y) {
      do {
         double rSquared = (x-halfL)*(x-halfL)+(y-halfL)*(y-halfL);
         int r = 1+(int) Math.sqrt(rSquared);
         if(r>maxRadius) {
            return true;       // start new walker
         }
         if((r<halfL)&&(s[x+1][y]+s[x-1][y]+s[x][y+1]+s[x][y-1]>0)) {
            numberOfParticles++;
            s[x][y] = 1;
            latticeFrame.setValue(x, y, Byte.MAX_VALUE);
            if(r>=startRadius) {
               startRadius = r+2;
            }
            maxRadius = startRadius+ringSize;
            return false;      // walk is finished
         } else {              // take a step
            // select direction randomly
            switch((int) (4*Math.random())) {
            case 0:
               x++;
               break;
```