```
   public void massDistribution() {
      mass = new double[L];
      double xcm = 0;
      double ycm = 0;
      for(int n = 0;n<occupiedNumber;n++) {
         xcm += xs[n];
         ycm += ys[n];
      }
      xcm /= occupiedNumber;
      ycm /= occupiedNumber;
      for(int n = 0;n<occupiedNumber;n++) {
         double dx = xs[n]-xcm;
         double dy = ys[n]-ycm;
         int r = (int) Math.sqrt(dx*dx+dy*dy);
         if((r>1)&&(r<L)) {
            mass[r]++;
         }
      }
   }
}
```

The target class is shown in Listing 13.2. Note the use of the Open Source Physics `LatticeFrame` class. When the user stops the cluster growth, a log-log plot of the mass distribution is shown.

Listing 13.2 Class `SingleClusterApp` displays the site percolation cluster and the mass distribution.

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

public class SingleClusterApp extends AbstractSimulation {
   SingleCluster cluster = new SingleCluster();
   PlotFrame plotFrame = new PlotFrame("ln r", "ln M",
        "Mass distribution");
   LatticeFrame latticeFrame = new LatticeFrame("Percolation cluster");
   int steps;

   public void initialize() {
      // not occupied or tested
      latticeFrame.setIndexedColor(0, Color.BLACK);
      latticeFrame.setIndexedColor(1, Color.BLUE);     // occupied
      // perimeter or growth site
      latticeFrame.setIndexedColor(2, Color.GREEN);
      // permanently not occupied
      latticeFrame.setIndexedColor(-1, Color.YELLOW);
      cluster.L = control.getInt("L");
      cluster.p = control.getDouble("p");
      cluster.initialize();
      latticeFrame.setAll(cluster.site);
   }

   public void doStep() {
      cluster.step();
      latticeFrame.setAll(cluster.site);
```

```
      latticeFrame.setMessage("n = "+cluster.occupiedNumber);
      if(cluster.perimeterNumber==0) {
         control.calculationDone("Computation done");
      }
   }

   public void stopRunning() {
      plotFrame.clearData();
      cluster.massDistribution();
      double massEnclosed = 0;
      int rPrint = 2;
      for(int r = 2;r<cluster.L/2;r++) {
         massEnclosed += cluster.mass[r];
         if(r==rPrint) { // use logarithmic scale
            plotFrame.append(0, Math.log(r), Math.log(massEnclosed));
            rPrint *= 2;
         }
      }
      plotFrame.setVisible(true);
   }

   public void reset() {
      control.setValue("L", 61);
      control.setValue("p", 0.5927);
      setStepsPerDisplay(10);
      enableStepsPerDisplay(true);
   }

   public static void main(String[] args) {
      SimulationControl.createApp(new SingleClusterApp());
   }
}
```

We will use the Leath or single cluster growth algorithm in Problem 13.3 to generate a spanning cluster at the percolation threshold. The fractal dimension is determined by counting the number of sites M in the cluster within a distance r of the center of mass of the cluster. The center of mass is defined by

$$\mathbf{r}_{\rm cm} = \frac{1}{N}\sum_i \mathbf{r}_i, \tag{13.4}$$

where N is the total number of particles in the cluster. A typical plot of $\ln M(r)$ versus $\ln r$ is shown in Figure 13.4. Because the cluster cannot grow past the edge of the lattice, we do not include data for $r \approx L$.

Problem 13.3 Single cluster growth and the fractal dimension

(a) Explain how the Leath algorithm generates single clusters in a way that is equivalent to the multiple clusters that are generated by visiting all sites. More precisely, the Leath algorithm generates percolation clusters with a distribution of cluster sizes equal to sn_s. For example, if you grow 10 clusters of size $s = 2$, then $n_s = 10/2 = 5$. The additional factor of s is due to the fact that each site of the cluster has an equal chance of being the seed of the cluster, and hence the same cluster can be generated in s ways.