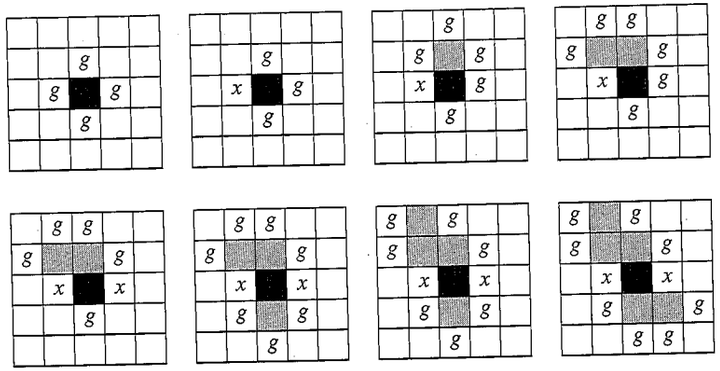

Figure 13.3 An example of the growth of a percolation cluster. Sites are occupied with probability p. Occupied sites are represented by a shaded square, growth or perimeter sites are labeled by g, and tested unoccupied sites are labeled by x. Because the seed site is occupied but not tested, we have represented it differently than the other occupied sites. The growth sites are chosen at random.

2. For each perimeter site, generate a uniform random number r in the unit interval. If $r \leq p$, the site is occupied and added to the cluster; otherwise, the site is not occupied. In order that sites be unoccupied with probability $1 - p$, these sites are not tested again.
3. For each site that is occupied, determine if there are any new perimeter sites, that is, untested neighbors. Add the new perimeter sites to the perimeter list.
4. Continue steps 2 and 3 until there are no untested perimeter sites to test for occupancy.

Class `SingleCluster` implements this algorithm and computes the number of occupied sites within a radius r of the seed particle. The seed site is placed at the center of a square lattice. Two one-dimensional arrays, `pxs` and `pys`, store the x and y positions of the perimeter sites. The status of a site is stored in the byte array `s` with `s[x][y] = (byte) 1` for an occupied site, `s[x][y] = (byte) 2` for a perimeter site, `s[x][y] = (byte)-1` for a site that has already been tested and not occupied, and `s[x][y] = (byte) 0` for an untested and unvisited site. To avoid checking for the boundaries of the lattice, we add extra rows and columns at the boundaries and set these sites equal to `(byte)-1`. We use a byte array because the array `s` will be sent to the `LatticeFrame` class which uses byte arrays.

Listing 13.1 Class `SingleCluster` generates and analyzes a single percolation cluster.

```
package org.opensourcephysics.sip.ch13.cluster;
public class SingleCluster {
   public byte site[][];
   public int[] xs, ys, pxs, pys;
   public int L;
   public double p;            // site occupation probability
   int occupiedNumber;
   int perimeterNumber;
   // displacement x to nearest neighbors
   int nx[] = {1, -1, 0, 0};
   // displacement y to nearest neighbors
   int ny[] = {0, 0, 1, -1};
```



```
   // mass of ring, index is distance from center of mass
   double mass[];

   public void initialize() {
      site = new byte[L+2][L+2]; // gives status of each site
      xs = new int[L*L];         // location of occupied sites
      ys = new int[L*L];
      pxs = new int[L*L];        // location of perimeter sites
      pys = new int[L*L];
      for(int i = 0;i<L+2;i++) {
         site[0][i] = (byte) -1; // don't occupy edge sites
         site[L+1][i] = (byte) -1;
         site[i][0] = (byte) -1;
         site[i][L+1] = (byte) -1;
      }
      xs[0] = 1+(L/2);
      ys[0] = xs[0];
      site[xs[0]][ys[0]] = (byte) 1; // occupy center site
      occupiedNumber = 1;
      for(int n = 0;n<4;n++) { // perimeter sites
         pxs[n] = xs[0]+nx[n];
         pys[n] = ys[0]+ny[n];
         site[pxs[n]][pys[n]] = (byte) 2;
      }
      perimeterNumber = 4;
   }

   public void step() {
      if(perimeterNumber>0) {
         int perimeter = (int) (Math.random()*perimeterNumber);
         int x = pxs[perimeter];
         int y = pys[perimeter];
         perimeterNumber--;
         pxs[perimeter] = pxs[perimeterNumber];
         pys[perimeter] = pys[perimeterNumber];
         if(Math.random()<p) {        // occupy site
            site[x][y] = (byte) 1;
            xs[occupiedNumber] = x;
            ys[occupiedNumber] = y;
            occupiedNumber++;
            for(int n = 0;n<4;n++) { // find new perimeter sites
               int px = x+nx[n];
               int py = y+ny[n];
               if(site[px][py]==(byte) 0) {
                  pxs[perimeterNumber] = px;
                  pys[perimeterNumber] = py;
                  site[px][py] = (byte) 2;
                  perimeterNumber++;
               }
            }
         } else {
            site[x][y] = (byte) -1;
         }
      }
   }
```