be occupied, and the occupied sites will form a large cluster that extends from one end of the lattice to the other (see Figure 12.3c). Such a cluster is said to be a *spanning cluster*. Because there is no spanning cluster for small p, and there is a spanning cluster for p near unity, there must be an intermediate value of p at which a spanning cluster first exists (see Figure 12.3b). We shall see that in the limit of an infinite lattice, there exists a well-defined threshold probability p_c such that:

For $p < p_c$, no spanning cluster exists and all clusters are finite.

For $p > p_c$, a spanning cluster exists.

For $p = p_c$, a spanning cluster exists with a probability greater than zero and less than unity.

We emphasize that the defining characteristic of percolation is *connectedness*. Because the connectedness exhibits a qualitative change at a well-defined value of a continuous parameter, we shall see that the transition from a state with no spanning cluster to a state with a spanning cluster is an example of a *phase transition*.

An example of percolation that can easily be observed in the laboratory has been done with a wire mesh. Watson and Leath measured the electrical conductivity of a uniform metallic screen as the nodes connecting wire links were removed. The coordinates of the nodes to be removed were determined by a random number generator. The measured electrical conductivity is a rapidly decreasing function of the fraction of nodes p that are still present and vanishes below a critical threshold. A related conductivity measurement on a sheet of conducting paper with random holes has been performed (see Mehr et al.).

The applications of percolation phenomena go beyond metal-insulator transitions and the conductivity of wire mesh and include the spread of disease in a population, the behavior of magnets diluted by nonmagnetic impurities, the flow of oil through porous rock, the microstructure of fiber-reinforced concrete, and the characterization of gels. Percolation ideas have also been used to understand clusters in such diverse systems as granular matter and social networks. We concentrate on understanding several simple models of percolation that have an intuitive appeal of their own. Some of the applications of percolation phenomena are discussed in the references.

12.2 ■ THE PERCOLATION THRESHOLD

`PercolationApp` in Listing 12.1 generates site percolation configurations and initially shows the occupied sites as red squares of unit area. The state of each site is stored in `lattice`, which is an object of type `LatticeFrame`. `LatticeFrame.resizeLattice` is used to initialize the lattice to the desired size; `LatticeFrame.setAtIndex(site,value)` sets `site` to `value`. Although the lattice is two-dimensional, it is easier to represent the lattice as a one-dimensional array. (The site index at `(x,y)` is `x + L*y`.) An unoccupied site has value -2, and an occupied site that has not yet been assigned to a cluster has value -1. Values 0–6 are used to color the clusters. The `InteractiveMouseHandler` interface is used to allow the user to click on an occupied site. Then the `colorCluster` method iteratively colors all the sites in the associated cluster. That is, after a site is added to the cluster, we add it to the array `sitesToTest` so that we can test the site's neighbors for cluster membership. When `sitesToTest` is empty, all the possible sites have been added to the cluster.

Listing 12.1 The `PercolationApp` class.

```
package org.opensourcephysics.sip.ch12;
import java.awt.Color;
import java.awt.event.MouseEvent;
import java.util.Random;
import org.opensourcephysics.display.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.LatticeFrame;

public class PercolationApp extends AbstractCalculation implements
    InteractiveMouseHandler {
  LatticeFrame lattice = new LatticeFrame("Percolation");
  Random random = new Random();
  int L;
  int clusterNumber; // used to color clusters

  public PercolationApp() {
    lattice.setInteractiveMouseHandler(this);
    // unoccupied sites are black and have value -2
    lattice.setIndexedColor(-2, Color.BLACK);
    // occupied sites that are not part of an identified cluster are
    // red and have value -1
    lattice.setIndexedColor(-1, Color.RED);
    // following colors used to identify clusters when user clicks on
    // an occupied site
    lattice.setIndexedColor(0, Color.GREEN);
    lattice.setIndexedColor(1, Color.YELLOW);
    lattice.setIndexedColor(2, Color.BLUE);
    lattice.setIndexedColor(3, Color.CYAN);
    lattice.setIndexedColor(4, Color.MAGENTA);
    lattice.setIndexedColor(5, Color.PINK);
    lattice.setIndexedColor(6, Color.LIGHT_GRAY);
    // recycles cluster colors starting from green
  }

  // uses mouse click events to identify an occupied site and
  // identify its cluster
  public void handleMouseAction(InteractivePanel panel,
      MouseEvent evt) {
    panel.handleMouseAction(panel, evt);
    if(panel.getMouseAction()==InteractivePanel.MOUSE_PRESSED) {
      int site = lattice.indexFromPoint(panel.getMouseX(),
          panel.getMouseY());
      // test if a valid site was clicked (index nonnegative),
      // if occupied, but not yet cluster colored (index=-1).
      if(site>=0&&lattice.getAtIndex(site)==-1) {
        // color cluster to which site belongs
        colorCluster(site);
        // cycle through 7 cluster colors
        clusterNumber = (clusterNumber+1)%7;
        // display lattice with colored cluster lattice.repaint();
      }
    }
  }

  // occupies all sites with probability p
```