characteristic scale, and the system is said to be *scale invariant*. This terminology reflects the fact that power laws look the same on all scales. For example, the replacement $s \to bs$ in the function $N(s) = As^{-\alpha}$ yields a function $\widetilde{N}(s)$ that is indistinguishable from $N(s)$, except for a change in the amplitude A by the factor $b^{-\alpha}$.

Contrast the nature of the power law dependence of $N(s)$ in (14.1) to the result of combining a large number of independently acting random events. In this case we know that the distribution of the sum is a Gaussian (see Problem 7.15), and $N(s)$ has the form

$$N(s) \sim e^{-(s/s_0)^2} \quad \text{(characteristic scale).} \tag{14.2}$$

Scale invariance does not hold for functions that decay as in (14.2), because the replacement $s \to bs$ in the function $e^{-(s/s_0)^2}$ changes s_0 (the characteristic scale or size of s) by the factor b. Note that for a power law distribution, there are events of all sizes, but for a Gaussian distribution, there are, practically speaking, no events much larger than the characteristic scale s_0. For example, if we take $s_0 = 100$, there would be one large event of size 1000 for every 2.7×10^{43} events of size one!

A simple example of self-organized critical phenomena is an idealized sandpile. Suppose that we construct a sandpile by randomly adding one grain at a time onto a flat surface with open edges. Initially, the grains will remain where they land, but after we add more grains, there will be small avalanches during which the grains move so that the local slope of the pile is not too big. Eventually, the pile will reach a statistically stationary (time-independent) state, and the amount of sand added will balance the sand that falls off the edge (on the average). When a single grain of sand is added to such a configuration, a rearrangement might occur that triggers an avalanche of any size (up to the size of the system), so that the mean slope again equals the critical value. We say that the statistically stationary state is critical because there are avalanches of all sizes. The stationary state is *self-organized* because no external parameter (such as the temperature) needs to be tuned to force the system to this state. In contrast, the concentration of fissionable material in a nuclear chain reaction has to be carefully controlled for the nuclear chain reaction to become critical.

We consider a two-dimensional model of a sandpile and represent the height at site i by the array element `height[i]`. One grain of sand is added to a random site j, `height[j]++`, at each iteration. If `height[j]` = 4, then we remove the four grains from site j and distribute them equally to its nearest neighbors. A site whose height is equal to four is said to *topple*. If any of the neighbors now have four grains of sand, they topple as well. This process continues until all sites have less than four grains of sand. Grains that fall outside the lattice are lost forever.

Class `Sandpile` implements this idealized model. The lattice is stored in a `LatticeFrame` and the arrays `toppleSiteX` and `toppleSiteY` store the coordinates of the sites with four grains of sand. The array `distribution` accumulates the data for the number of sites that topple at each addition of a grain of sand to the pile. It is possible, though rare, that a site will topple more than once in one step. Hence, the number of toppled sites may be greater than the number of sites in the lattice.

Physically, it is not the actual height that determines toppling but the mean local slope between a site and its nearest neighbors. Thus, what we call the "height" really should be called the "slope." However, in the literature many authors use the term "height."

Listing 14.6 Implementation of the two-dimensional sandpile model.

```
package org.opensourcephysics.sip.ch14.sandpile;
import java.awt.Graphics;
import org.opensourcephysics.frames.*;

public class Sandpile {
   int[] distribution; // distribution of number of sites toppling
   int[] toppleSiteX, toppleSiteY;
   LatticeFrame height;
   int L, numberToppledMax;
   int numberToppled, numberOfSitesToTopple, numberOfGrains;

   public void initialize(LatticeFrame height) {
      this.height = height;
      height.resizeLattice(L, L); // create new lattice
      // size of distribution array
      numberToppledMax = 2*L*L+1;
      // could use HistogramFrame instead
      distribution = new int[numberToppledMax];
      toppleSiteX = new int[L*L];
      toppleSiteY = new int[L*L];
      numberOfGrains = 0;
      resetAverages();
   }

   public void step() {
      numberOfGrains++;
      numberToppled = 0;
      int x = (int) (Math.random()*L);
      int y = (int) (Math.random()*L);
      int h = height.getValue(x, y) + 1;
      height.setValue(x, y, h); // add grain to random site
      height.render();
      if(h==4) { // topple grain
         numberOfSitesToTopple = 1;
         boolean unstable = true;
         int[] siteToTopple = {x, y};
         while(unstable) {
            unstable = toppleSite(siteToTopple);
         }
      }
      distribution[numberToppled]++;
   }

   public boolean toppleSite(int siteToTopple[]) { // topple site
      numberToppled++;
      int x = siteToTopple[0];
      int y = siteToTopple[1];
      numberOfSitesToTopple--;
      // remove grains from site
      height.setValue(x, y, height.getValue(x, y)-4);
      height.render();
      // add grains to neighbors
      // if (x,y) is on the border of the lattice, then some
      // grains will be lost
      if(x+1<L) {
```