we expect that one Monte Carlo step per spin is proportional to the average time between single spin flips observed in the laboratory. Hence, we can regard single spin flips as a time dependent process and observe the relaxation to equilibrium. In the following, we will frequently refer to the application of the Metropolis algorithm to the Ising model as *single spin flip* dynamics.

In Problem 15.11 we use the Metropolis algorithm to simulate the one-dimensional Ising model. Note that the parameters J and kT do not appear separately, but appear together in the dimensionless ratio J/kT. Unless otherwise stated, we measure temperature in units of J/k and set $B = 0$.

Problem 15.11 One-dimensional Ising model

(a) Write a program to simulate the one-dimensional Ising model in equilibrium with a heat bath. Modify method `doOneMCStep` in `IsingDemon` (see class `IsingDemon` on page 600 or class `Ising` on page 611). Use periodic boundary conditions. Assume that the external magnetic field is zero. Draw the microscopic state (configuration) of the system after each Monte Carlo step per spin.

(b) Choose $N = 20$ and $T = 1$ and start with all spins up. What is the initial effective temperature of the system? Run for at least 1000 mcs, where mcs is the number of Monte Carlo steps per spin. Visually inspect the configuration of the system after each Monte Carlo step per spin and estimate the time it takes for the system to reach equilibrium. Does the sign of the magnetization change during the simulation? Increase N and estimate the time for the system to reach equilibrium and for the magnetization to change sign.

(c) Change the initial condition so that the orientation of each spin is chosen at random. What is the initial effective temperature of the system in this case? Estimate the time it takes for the system to reach equilibrium.

(d) Choose $N = 50$ and determine $\langle E\rangle$, $\langle E^2\rangle$, and $\langle M^2\rangle$ as a function of T in the range $0.1 \leq T \leq 5$. Plot $\langle E\rangle$ as a function of T and discuss its qualitative features. Compare your computed results for $\langle E\rangle$ to the exact result (for $B = 0$):

$$E(T) = -N \tanh \beta J. \tag{15.22}$$

Use the relation (15.19) to determine the T-dependence of C.

(e) As you probably noticed in part (b), the system can overturn completely during a long run and thus the value of $\langle M\rangle$ can vary widely from run to run. Because $\langle M\rangle = 0$ for $T > 0$ for the one-dimensional Ising model, it is better to assume $\langle M\rangle = 0$ and compute χ from the relation $\chi = \langle M^2\rangle/kT$. Use this relation (15.21) to estimate the T-dependence of χ.

(f) One of the best laboratory realizations of a one-dimensional Ising ferromagnet is a chain of bichloride-bridged Fe^{2+} ions known as FeTAC (see Greeney et al.). Measurements of χ yield a value of the exchange interaction J given by $J/k = 17.4\,\mathrm{K}$. Note that experimental values of J are typically given in temperature units. Use this value of J to plot your Monte Carlo results for χ versus T with T given in Kelvin. At what temperature is χ a maximum for FeTAC?

(g) Is the acceptance probability an increasing or decreasing function of T? Does the Metropolis algorithm become more or less efficient as the temperature is lowered?

(h) Compute the probability $P(E)$ for a system of $N = 50$ spins at $T = 1$. Run for at least 1000 mcs. Plot $\ln P(E)$ versus $(E - \langle E\rangle)^2$ and discuss its qualitative features. ■

We next apply the Metropolis algorithm to the Ising model on the square lattice. The `Ising` class is listed in the following.

Listing 15.5 The `Ising` class.

```
package org.opensourcephysics.sip.ch15;
import java.awt.*;
import org.opensourcephysics.frames.*;

public class Ising {
   public static final double criticalTemperature =
        2.0/Math.log(1.0+Math.sqrt(2.0));
   public int L = 32;
   public int N = L*L;                    // number of spins
   public double temperature = criticalTemperature;
   public int mcs = 0;                    // number of MC moves per spin
   public int energy;
   public double energyAccumulator = 0;
   public double energySquaredAccumulator = 0;
   public int magnetization = 0;
   public double magnetizationAccumulator = 0;
   public double magnetizationSquaredAccumulator = 0;
   public int acceptedMoves = 0;
   // array to hold Boltzmann factors
   private double[] w = new double[9];
   public LatticeFrame lattice;

   public void initialize(int L, LatticeFrame displayFrame) {
      lattice = displayFrame;
      this.L = L;
      N = L*L;
      lattice.resizeLattice(L, L); // set lattice size
      lattice.setIndexedColor(1, Color.red);
      lattice.setIndexedColor(-1, Color.green);
      for(int i = 0;i<L;++i) {
         for(int j = 0;j<L;++j) {
            lattice.setValue(i, j, 1); // all spins up
         }
      }
      magnetization = N;
      energy = -2*N; // minimum energy
      resetData();
      // other array elements never occur for H = 0
      w[8] = Math.exp(-8.0/temperature);
      w[4] = Math.exp(-4.0/temperature);
   }

   public double specificHeat() {
      double energySquaredAverage = energySquaredAccumulator/mcs;
      double energyAverage = energyAccumulator/mcs;
      double heatCapacity =
           energySquaredAverage-energyAverage*energyAverage;
```