Assignment 5: Due November 30 @ 11:59:59PM

- This assignment is to be done individually. You may discuss the assignment with each other and/or in the course newsgroup, but you have to write the code yourself.
- Use the handin command to hand in your assignment electronically. In your home directory make a subdirectory cs405. In ~/cs405 put a subdirectory a5. Put all the files to handin in ~/cs405/a5/. Then issue the command “handin cs405 a5”. You can resubmit until the due date. If you have not submitted by the due date you can still submit, but only once.
- This assignment is worth a total of 100 points. It is an optional assignment, meaning it will be ignored if your final mark in the course would be higher without it.
- For late and other penalties see the course website.

Crowding Behaviour

Consider an audience at a play, concert, speech, or some such event. Most of the time everyone sits quietly but when excitement arises a few people will clap, shout, or boo. Occasionally the claps will spread over the whole hall and we get a massive applause.

We will model this behaviour by assuming a single persons behaviour is determined 1) the level of stimulation and 2) the excitement of the people sitting close to this person. We will model “excitement” of an individual by a number

\[ s = -1, -(q - 1)/q, -(q - 2)/q, \ldots, -1/q, 0, 1/q, \ldots, (q - 1)/q, 1 \]

where \( q \) is an overall integer constant characterizing the model. \(-1\) is interpreted as sitting quietly, and \( +1 \) is the maximum level of excitement. We will model the crows with a statistical physics inspired model, and will interpret the temperature of this model as the level of external stimulation.

We assume the audience to be seated in a regular \( N \times N \) grid. At each gridpoint \((i, j)\) we have a state variable \( s_{ij} \) which has \( 2q + 1 \) possible values. The “energy” \( E \) of the system is the sum of two contributions. The first is “crowding behaviour” term \( E_1 \) which is the negative of the sum of the products of \( s \) on adjacent grid points.

\[ E_1 = \sum_{neighbors (ij)(kl)} -s_{ij}s_{kl}. \]

For simplicity will use periodic boundary conditions, meaning that we consider \( s_{N+1,j} \) to be the same as \( s_{1,j} \) and similar for the \( j \) direction, meaning \( s_{N,j} \) is considered a neighbour of \( s_{1,j} \).
The equilibrium behaviour which minimized the energy is thus one where all the $s$ are $-1$ (all sitting quietly) or $+1$ (all shouting/clapping).

The second contribution $E_2$ to the total energy $E = E_1 + E_2$ is given by the sum over all points

$$E_2 = b \sum_{ij} s_{ij},$$

with $b$ a constant called the “bias”. So if $b > 0$ energy is lowered by calming down.

We now model the system at non-zero temperature $T$ by assuming a pdf) for the state which is

$$Z^{-1}e^{-E/T}$$

with $Z$ a normalization constant (which we actually don’t need to know). We define the “total magnetization” analogous to the Ising model as sum of all $s$, divided by the number of grid points:

$$M = \sum_{ij} s_{ij}/N^2.$$  

At $T = 0$ and $b = 0$ we will have $M = -1$ or $M = +1$ depending on which lowest energy state we are in. At non-zero temperature the measured value of $M$ is the average over all states distributed according to their pdf.

The model described here is known as the “Potts Model”.

We are going to use it to create sound effects for applause. The driver application PottsDemo uses the Potts model to create a Markov chain of states according to the Metropolis algorithm and uses the resulting sequence of states to drive sound. A single clap sound is emitted by each of $N^2$ clappers with volume determined by the state $s$. The spatial location of the sound is determined by the location on the grid. By changing the parameters $T$ and $b$ it is thus possible to control the crowds behaviour and control the crowds sound in real time.

This is all done for you, and your task is to write the java class PottsModel that is called by the audio application.

Once you have written your class you can compile and run it with the invocations described in the readme.txt file.

Depending on the musclepower of your CPU you will be limited to lowish values of $N$. I can run up to $N = 30$. If $N$ is too high, the audio starts to break up. If you have problem with audio you can try to increase the variable bufferSizeJavaSound in PottsDemo.java. This buffer size controls the latency between audio and the visual display.

1 Write the constructor and the init() method. It should initialize all state variables to $-1$, reset the running average of $M$ (see below), calculate and remember the energy.

2 We are going to generate states according to the Metropolis algorithm. Write a method update(int i, int j) that changes the state by moving $s_{ij}$ one step up or down (equal probability) unless they are in their minimum (maximum) state when you move $s_{ij}$ one step up (down). Accept or reject this candidate update according to the Metropolis algorithm.
After each `update()` you should update the total energy and the running average of the magnetization. The running average is the average over all nodes in the Markov chain you have encountered thus far, from the last call to `resetRunningAverage()`. Since each update involves only a single grid point you should update the energy and average of $M$ with an $O(1)$ method, i.e., don’t iterate over the whole grid. $M$ is displayed in the title bar of your visual display.

3 Implement `sweep()` which performs a Metropolis update step `update(i,j)` consecutively for each grid point.

4 Run your code with the driver program for $q = 1$. It will print the running average of $M$ in the title bar, and render the audio. Try controlling the audience by the parameters $T$ and $b$. Try some other values of $q$. (You don’t have to document this item, I hope you can have some fun with it....)

Hand in `PottsModel.java` and the coversheeta5.txt.