Chapter 1: Tinker-Toy Project

1.1 The Boltzmann factor

I could stop at this point. However, I wasn't quite satisfied with how idealistic my simulation was. Looking at the movement of each cell can be mesmerizing and seeing Brownian motion in action was quite a bit of fun. Yet, I knew it was incomplete.

To relax some of the assumptions, I decided to look back at rule 1 which states:

1. A cell can only be a 1 or 0.

To model what's going on down there, there should be multiple levels of energy that are discrete by nature as Max Planck first proposed in order to solve the Ultraviolet catastrophe. Hence, I decided to tinker rule 1 to be as follows:

- A cell can have integer values between 0 to 8.

In this situation, a 0 cell represents the minimum amount of energy that a cell/particle can take or its ground state while an 8-cell represents the maximum amount of energy that a cell can take.

This tweak adds a bit of realism to the simulation that changes the dynamics of how things work. After testing the prototype of version 2, I realized that a fundamental feature was missing, especially seeing that the clusters of heated particles had difficulty in dispersing their energy even at high move probabilities. To show what was wrong, we can look at the picture below to analyze the values.

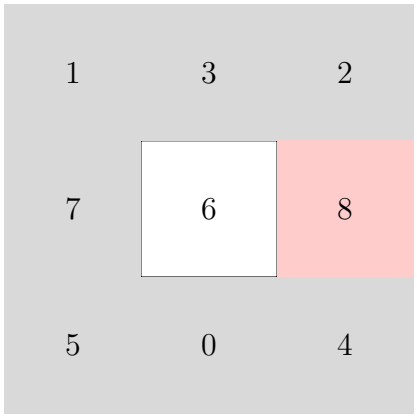

Figure 1.1: The cells that are colored are called the Moore neighborhood of the 6 cell

In Figure 1.1, the shaded regions are the Moore neighborhood of the 6 cell located in the middle. As with version 1, the rules of movement was not altered for the

prototype and 6 cell can transfer a unit of energy into any one of the grey cells with $\frac{1}{7}$ probability each.

At this point, I knew the assumption of $\frac{1}{7}$ probability was wrong as in Chemistry, an electron will try to occupy the lowest energy state first rather than going into the higher orbitals. For instance, the way the electron goes toward orbitals is in the following order: $1s^2, 2s^2, 2p^6, 3s^2, 3p^6, 4s^2, 3d^{10}, 4p^6$. Hence, I needed some method or function that will modify the probabilities of each direction that is dependent on each cell's energy value.

After some searching and reading articles on this topic, I came across the Boltzmann factor which conveniently addressed the problem quite well. The Boltzmann factor is given by:

$$\frac{p_i}{p_j} = e^{\frac{\epsilon_j - \epsilon_i}{kT}} \tag{1.1}$$

The p terms represent the probability at state i and state j while the ϵ terms represent the energy at state j and i respectively. At this point, I wasn't exactly sure how to implement the Boltzmann factor to Figure 1.1. Since I wanted the simulation to make use of this concept, I decided to modify the Boltzmann factor into:

$$p_j = e^{\frac{8 - \epsilon_j}{T}} \tag{1.2}$$

In this modified, version ϵ_j represents each cell's energy state within the Moore neighborhood and p_j represents the probability that a unit of energy is transferred to that particular cell.

Using this particular equation we can modify the probability that the 6 cell within Figure 1.1 transfers its energy to each of the cells shaded grey. To do that we need to compute all of the probabilities and normalize them with a partition function. The partition function is defined as:

$$\sum_{j=1}^{\kappa} e^{\frac{8-\epsilon_j}{T}} \tag{1.3}$$

Where κ is the number of cells within the neighborhood excluding 8s. So for the picture in Figure 1.2, $\kappa = 7$ as we only have 7 "available cells" that the energy from the 6 cell can go to.

Assuming that T=1, we can then divide each of Boltzmann's factors with 1.3. This yields our probabilities of $P = [0.2330, 0.0315, 0.0857, 0, 0.0116, 0.6333, 0.0043, 0.0006]$ for the cells labeled: $1, 3, 2, 8, 4, 0, 5, 7$ respectively. This means that there is a 61.6% chance that the 6 cell within Figure 1.2 will send a unit of energy to a 0 cell and a 0.06% chance that the 6 cell sends a unit of energy towards the 7 cell.

To better depict the probabilities, I have made the picture below where each of the cell's values has a small text beside it which is the probability that a unit of energy from the 6 cell goes into the other cells.

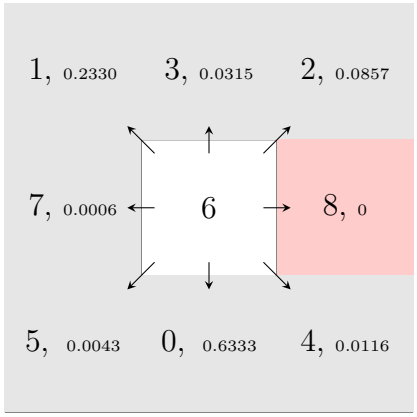

Figure 1.2: Probabilities of each of the cells within the Moore neighborhood of the 6 cell

We can then model all 8 outcomes depicted by the arrows. In the picture below, I decided on only drawing 2 possible outcomes depicted by arrows with the probabilities associated with them. The rest will be left as an exercise for the reader.

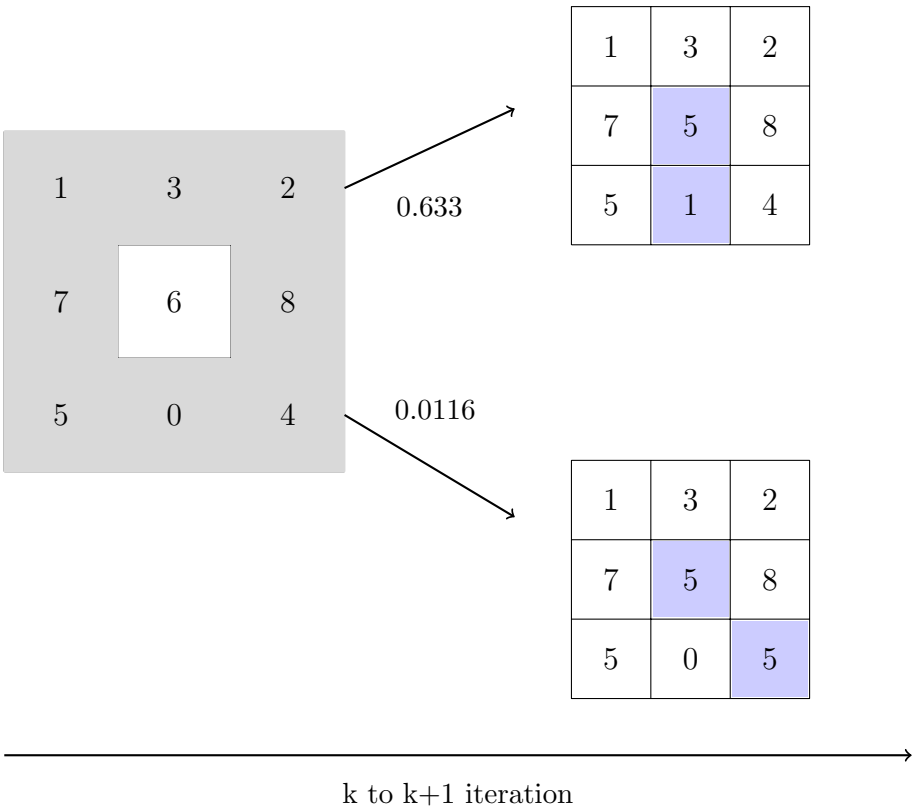

Figure 1.3: Possible configurations for the k+1 iteration. Blue cells are the cells whose value changed

As seen from Figure 1.3, there is a 63.3% chance that the grid would end up looking like the top and a 1.16% chance that the grid would look like the bottom in the next iteration. Using this strategy, I can then drift my simulation to be closer to reality.

1.2 Calculating entropy

Now that a strategy that governs energy movement has been devised, our next task is to find a way to compute entropy. In version 1, a crude method of using bitwise XOR is used. Details on how this is done can be seen within this document. However, such a method is not viable when the cells are no longer binary digits. Hence, a new strategy needs to be devised.

The first step in solving this problem is to look back at the definition of entropy that Ludwig Boltzmann provided. The entropy S is equal to:

$$S = k_b Ln(\Omega) \tag{1.4}$$

Now the Ω term is the total number of micro-states or possible configurations that the system can occupy. The challenge is then to find a way to compute Ω. Remembering the video that ParthG made in explaining what entropy is, it became clear that the strategy is to count the number of viable integers combinations that add up a number.

The technique I decided on is to first, break down the entire square grid into 2x2 sub-matrices. If we let l be equal to the length of the matrix and that $l \in 2\mathbb{Z}$, we would have $\frac{l^2}{4}$ different sub-matrices. In Figure 1.4, some of the splittings are depicted using arrows and the middle matrix is the grid. Note that the middle matrix is a square matrix, for convenience, it is depicted as a rectangle.

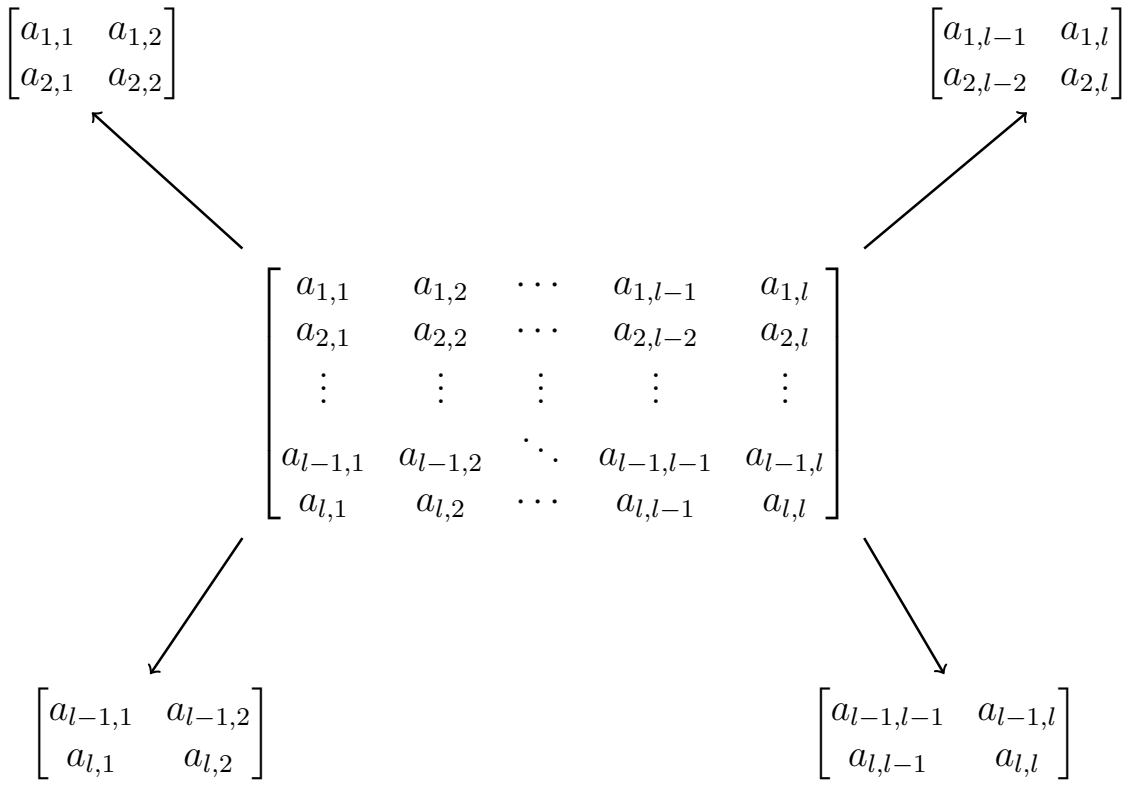

Figure 1.4: Submatrix splitting

Now let the grid be called M and its sub-matrices be called X_j. In this case, j goes from 1 all the way to $\frac{l^2}{4}$. At this point, the next step is $\forall X_j$, find the sum of its 4 elements. Now, let the sum of its four elements of each sub-matrix be depicted as Φ_j. Remembering the rules of the simulation the sum of its elements or its energy is bounded within:

$$0 \leq \Phi_j \leq 32, \quad \Phi_j \in \mathbb{Z}$$

Once this is completed, the next step is to find the number of integer combinations of the elements from X_j, such that its four elements:

$$a_j + b_j + c_j + d_j = \Phi_j \tag{1.5}$$

Where:

$$a_j, b_j, c_j, d_j \in [0, 8] \ \wedge \ a_j, b_j, c_j, d_j \in \mathbb{Z}$$

$$\sum_{j=1}^{\frac{l^2}{4}} \Phi_j = E \tag{1.6}$$

Now equation 1.6 is the conservation of energy and the simulation is designed such that this is always true.

Conveniently, there are hundreds of sample codes out there in stack overflow that solve something similar to equation 1.5, therefore solving this problem is thankfully not that difficult. The number of possible combinations such that $a_j+b_j+c_j+d_j = \Phi_j$ is the number of micro-states for each sub-matrix. Lets call this ω_j. Some of the calculated values of ω_j corresponding to its Φ_j is shown in the table down below:

Sub-matrix energy (Φ_j)	Micro-states (ω_j)
0	1
1	4
2	10
4	35
6	84
9	216
11	324
14	456
16	489

The above table can map Φ_j to its associated ω_j for each sub-matrix X_j. However, in order to find the total configurations that the grid M can take we have to multiply each of the sub-matrix micro-state where Ω from equation 1.4 is:

$$\Omega = \prod_{j=1}^{\frac{l^2}{4}} \omega_j \tag{1.7}$$

For large grids where $l \geq 150$, Ω becomes a huge number and in some cases, float32 operations in Python may become insufficient. Using the results from equation 1.7 and combining the product rule for logarithms, equation 1.4 can be rewritten as:

$$S = k_b \sum_{j=1}^{\frac{l^2}{4}} ln(\omega_j) \tag{1.8}$$

To sum up, while $k \leq I$, and I is the number of iterations. We have:

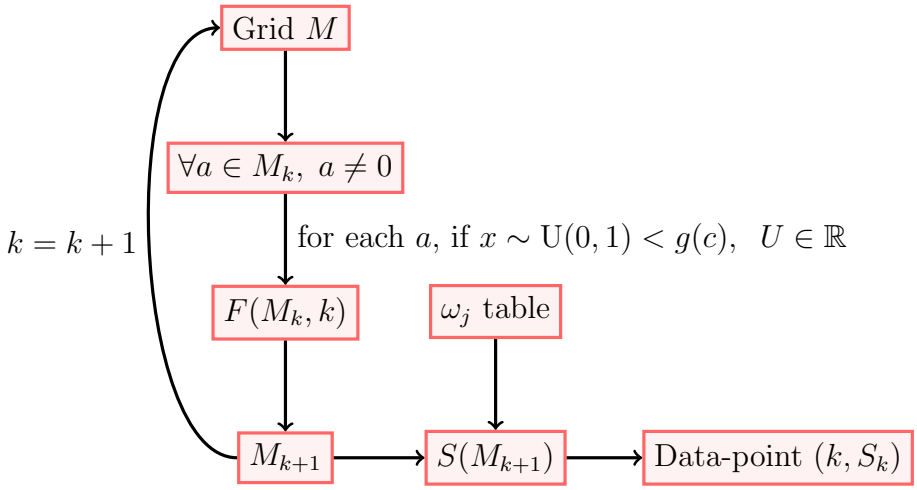

Figure 1.5: Process or Functional flow diagram

Figure 1.5 represents a high-level overview of the process steps needed to produce and run the simulation for 1 iteration. a is the elements of the grid/matrix M. $U(0,1)$ is a uniform distribution from 0 to 1. $F(M,k)$ is the function responsible for movement, ω_j is the micro-state table, $S(M_k,k)$ is the entropy function.