A visual demonstration of Entropy

A simulation of random processes using cellular automata.

By: Bowen Shiro Husin

25th February 2023

Acknowledgements

This project is intended to keep my brain busy while I am *job-searching*. Likewise, it's meant to be a coding challenge that I've come up with. This document is meant to declare the mathematics and my line of thinking in designing the rules and entropy computation. Should you read this part, feel free to use the code and tinker with it. If you are a physics teacher, feel free to use it to teach your students about the fascinating concept of entropy.

To view how my code is implemented check my GitHub repository.

Abstract

Entropy is a fascinating concept with far-reaching consequences for our daily lives. In this document, a cellular-automata-inspired approach is proposed. Namely, a 2-dimensional array consisting of cells with integer values ranging from 0-8 is constructed for visualizing entropy increase over time.

For version 2, the Boltzmann factor alongside temperature is implemented to count for energy differences between cells and alter their probabilities. Likewise, a new method is used to compute the entropy of the simulation. Namely, the use of the bitwise XOR function is abandoned for version 2 and is replaced with a more elementary operation that is closer to how entropy is computed by the famous Boltzmann equation. Finally, a custom function is used to map the conductivity or c with move probability or α.

Results from the simulation show that the underlying structure of the 2-dimensional array will become more uniformly distributed no matter the starting configuration. Plots show that both conductivity and temperature have tremendous effects on the final entropy after a set amount of iterations. Conductivity increases it asymptotically while temperature also flattens it asymptotically.

Contents

Chapter 1: Introduction

Cellular Automata (CA) is a fascinating topic within the field of computer science. Its principles have been used to model complex phenomena in the area of computational biology, physics, and mathematics.

Last month, I was exploring the applications of cellular automata, in particular, the "game of life" which was created by John Horton Conway in the 1970s. While I was reading the Wikipedia page I connected 2 dots. I realized that you could use the principles of cellular automata to demonstrate the 2nd law of thermodynamics and why it will always tend to increase in isolated systems.

In the previous document, I have discussed the basic idea behind the implementation and the rules. In this document, I would like to revisit the rules and fine-tune them to relax certain assumptions of the simulation in version 1.

To recap, the simulation in version 1 contains 3 rules and 1 feature. These are:

- A cell can only be a one or zero.
- A one cell can transfer its "energy" towards a 0 cell that is surrounding it or its Moore's neighborhood.
- The number of one cells within the simulation cannot change.
  - Feature: There is a parameter called alpha or "move probability" which determines the likelihood that the code responsible for movement will execute for each 1s.

These 3 rules and features are enforced to a large square matrix that can be easily initialized using the NumPy library in Python. Lastly, version 1 employs a very crude method to compute the entropy of the entire system. The strategy used to calculate it involves flattening the entire NumPy grid into a 1-dimensional array of 1s and 0s. Next, the flattened array is divided into chunks of 2. Afterward, a bitwise XOR operation is performed on each of the chunks moving 1 square at a time. Finally, the truth values are all summed up and plotted in a graph of entropy versus iterations. This process produces 1 data point and is performed every time for a single iteration of the grid.

The main idea behind this method is that if the grid is more entropic. there should be more occurrences of alternating 1s and 0s rather than repeating 1s or repeating 0s. This means sequences such as $(0, 1)$ or $(1, 0)$ should be more likely if the grid is more entropic compared to sequences such as $(0, 0)$ or $(1, 1)$. This is where the use of bitwise XOR shines through as alternating 1s and 0s return a value of 1 while

the others yield a 0. While flawed, the assumption works if the number of occupied cells (which cannot change) is a small percentage of the total number of cells of the entire grid.

As a result, chosen strategy to compute the entropy (H) yields values that can be bounded in the following inequality:

$$0 \leq H \leq 2E$$

Where E is the total energy of the grid or is the number of one cells within the grid. The upper bound is 2E as the bitwise XOR function can double count the number of alternating 1s and 0s as it moves one square at a time.

Chapter 2: Random processes

2.1 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.

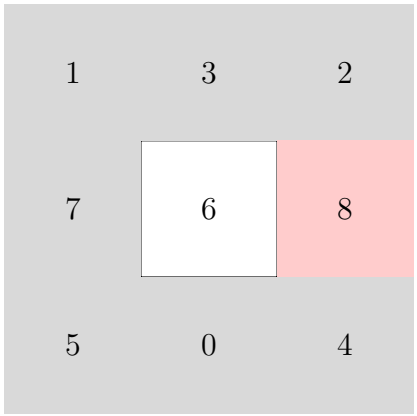


Figure 2.1: The cells that are colored are called the Moore neighborhood of the 6 cell

In Figure 2.1, 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:

$$\frac{p_i}{p_j} = e^{\frac{\epsilon_j - \epsilon_i}{kT}} \tag{2.1}$$

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 2.1. Since I wanted the simulation to make use of this concept, I decided to modify the Boltzmann factor into:

$$p_j = e^{\frac{8 - \epsilon_j}{T}} \tag{2.2}$$

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 2.1 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:

$$\sum_{j=1}^{\kappa} e^{\frac{8-\epsilon_j}{T}} \tag{2.3}$$

Where κ is the number of cells within the neighborhood excluding 8s. So for the picture in Figure 2.2, $\kappa = 7$ 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 2.3. 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 2.2 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.

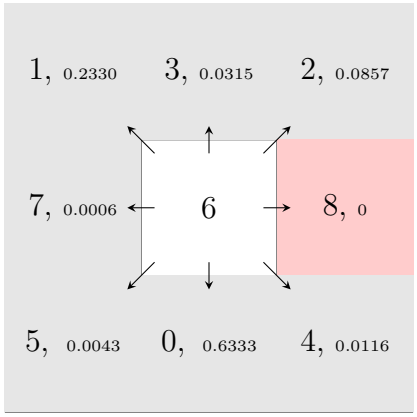


Figure 2.2: Probabilities of each of the cells within the Moore neighborhood of the 6 cell

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.

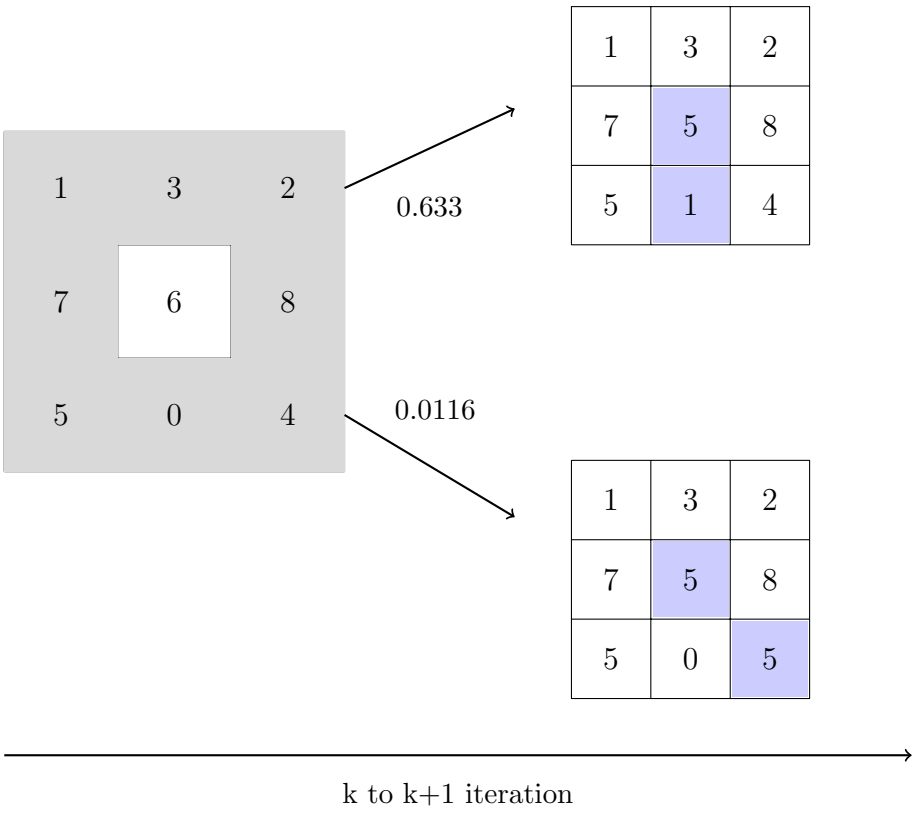


Figure 2.3: Possible configurations for the k+1 iteration. Blue cells are the cells whose value changed

As seen from Figure 2.3, 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.

2.2 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:

$$S = k_b Ln(\Omega) \tag{2.4}$$

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 2.4, 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.

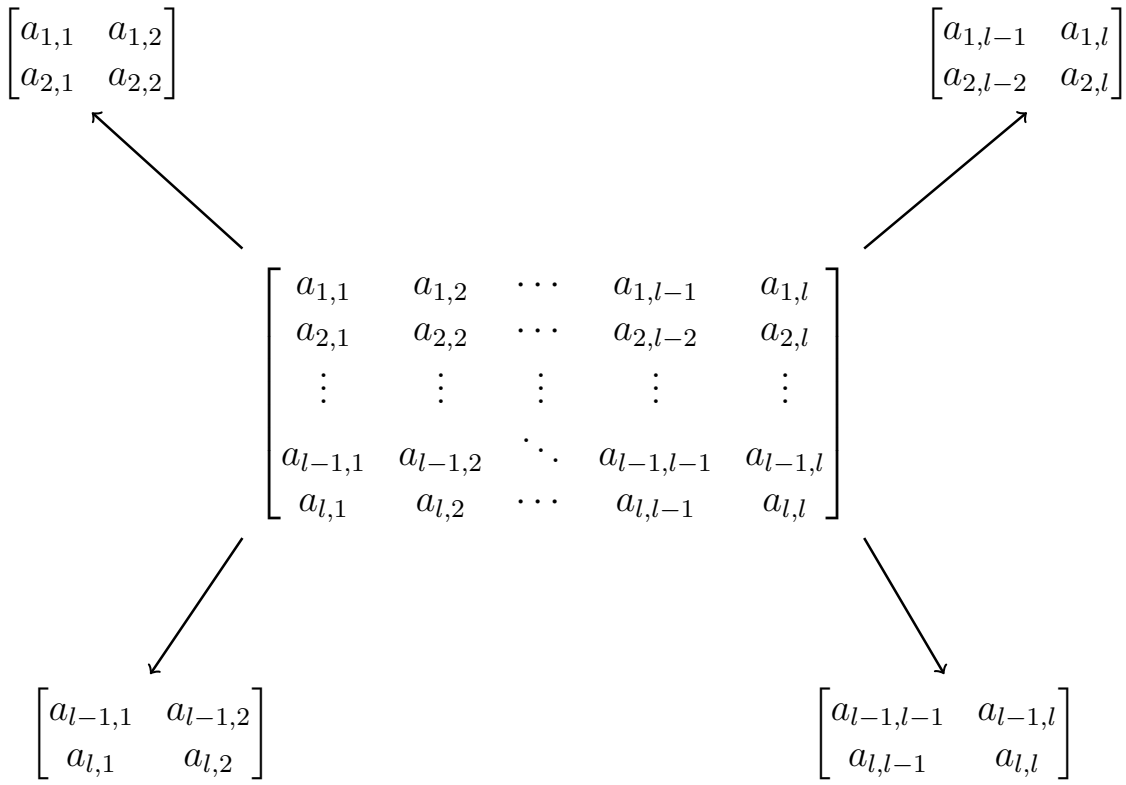


Figure 2.4: Submatrix splitting

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:

$$a_j + b_j + c_j + d_j = \Phi_j \tag{2.5}$$

Where:

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

$$\sum_{j=1}^{\frac{l^2}{4}} \Phi_j = E \tag{2.6}$$

Now equation 2.6 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 2.5, 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 2.4 is:

$$\Omega = \prod_{j=1}^{\frac{l^2}{4}} \omega_j \tag{2.7}$$

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 2.7 and combining the product rule for logarithms, equation 2.4 can be rewritten as:

$$S = k_b \sum_{j=1}^{\frac{l^2}{4}} ln(\omega_j) \tag{2.8}$$

2.3 Conductivity

Now that a strategy for calculating entropy has been devised. The next step is to introduce conductivity into the mix and relate it to "move probability" or α from Chapter 1. In version 1, move probability determines the likelihood that the code responsible for movement is executed. If this value is 1. This means that the code executes at every iteration. If this number is 0. The function responsible for updating the grid does nothing.

From my instincts, it seems like this is a good proxy for conductivity (c). For small values, α or move probability is very small but for large values, $\alpha = 1$. We could use the logistic function to map conductivity with α, however, defining the logistic function to be undefined or 0 when $c \leq 0$ is quite difficult. Likewise, I wanted a function to be a small non-zero value when the user puts $c = 0$, climbs rather quickly to 0.4 when $0 \leq c \leq 400$, and steadily reaches 1 when $c \geq 3000\pi$. After some experimenting with GeoGebra, I decided to use a custom sin function where the mapping function $g(c)$ is defined as:

$$g(c) = \begin{cases} 1 & \text{if } c > 3000\pi \\ sin(\frac{1}{6000}c)^{\frac{0.2}{0.1c}} & \text{if } 0 < c \leq 3000\pi \\ 0.014 & \text{if } c = 0 \\ \text{undefined} & \text{if } c < 0 \end{cases} \tag{2.9}$$

In this way, I obtain a function with the behavior I want. In version 2, the user can dynamically change the conductivity and temperature T within equation 2.1

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

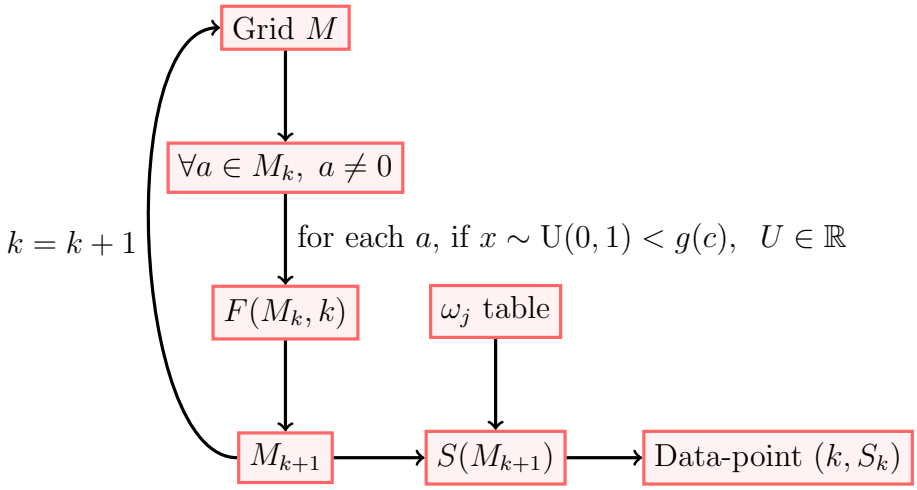


Figure 2.5: Process or Functional flow diagram

Figure 2.5 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.

Chapter 3: Results

3.1 Implementation

In order to start the simulation, the user needs to input certain conditions. Such as grid length, conductivity, and temperature. Version 2 replaces version 1's use of the Python console with a GUI (Graphical user interface). The graphical UI is shown below:

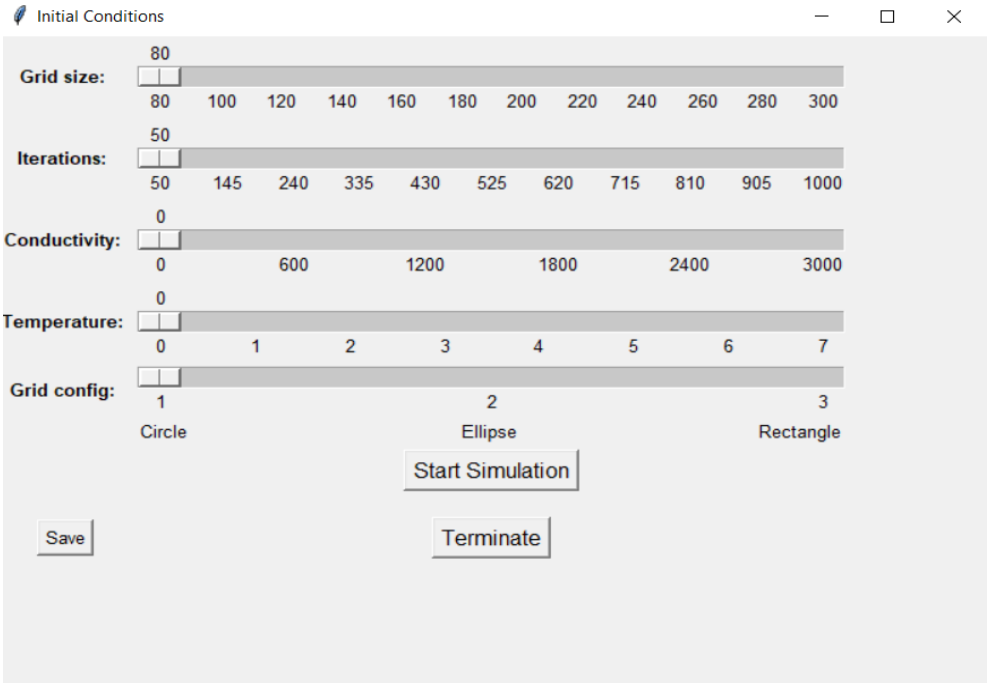


Figure 3.1: Input values for the simulation

Grid size is l, iterations or generations is I, Conductivity is c, Temperature is T within equation 2.2, and the grid config is what the initial grid looks like at the very first iteration or iteration 0. The user can input one of 3 choices. The "Start" button starts the simulation which will disable all other functionality other than the "Terminate" button which will delete the matplotlib image. The "save" button is meant to save a copy of the animation as a gif file. It is recommended that each action is taken one at a time to avoid unexpected errors. Likewise, the "Terminate" button acts as a reset button which will re-enable all functionality.

To dynamically change the values of the graph you would need to alter the slider widgets within matplotlib. There are 2 slider widgets within that one of them is

a conductivity slide and the other is the temperature slide. You can dynamically change that yourself to the values you desire during the simulation.

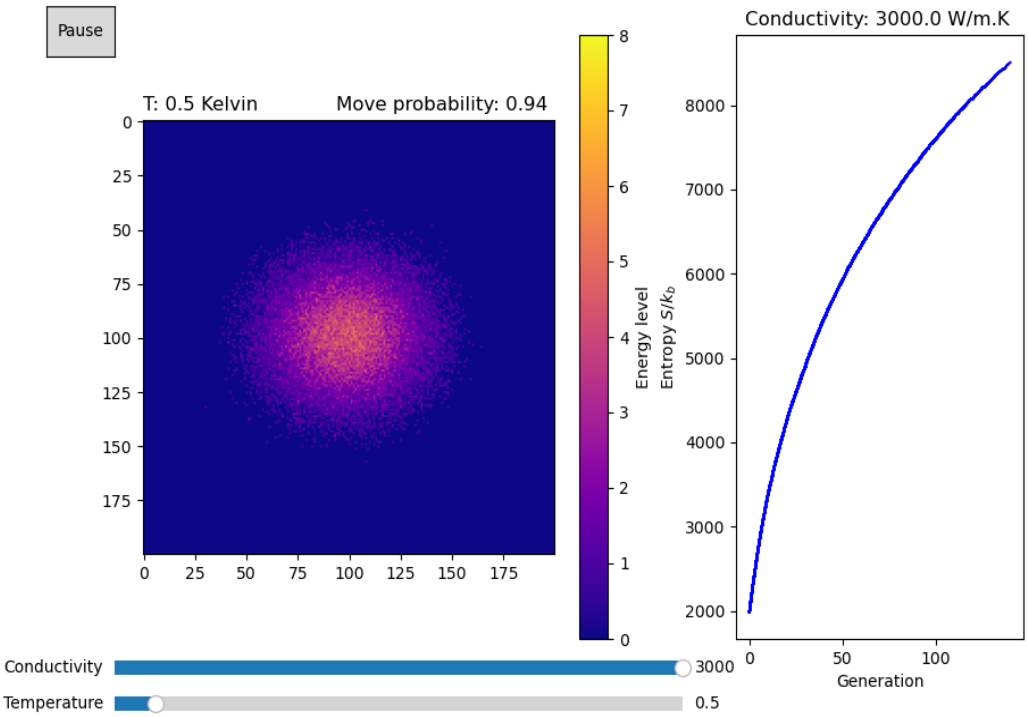
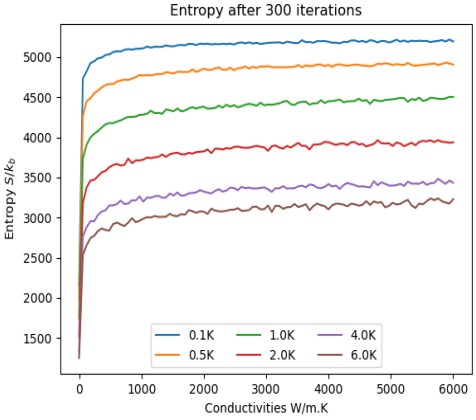


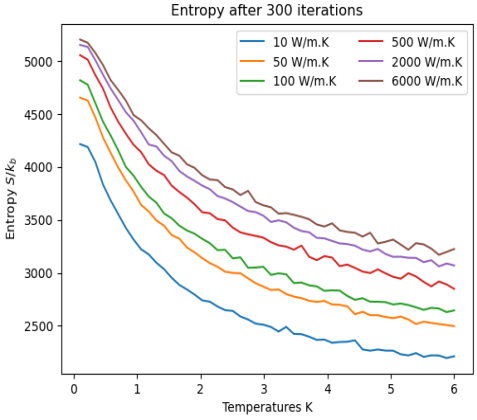
Figure 3.2: Slider widgets within the simulation window

3.2 Results

Now that the GUI system is complete. We could run some analysis by using some standardized initial conditions. For the plots presented below, the length of the grid is $l = 100$, the number of iterations is $I = 300$, and the grid config is set as "circle". The conductivity and the temperature are made to vary. The results of the simulation are:



(a) Entropy against conductivity



(b) Entropy against temperature

Figure 3.3: An explanation of the 2nd law

As seen from Figure 3.3a, the final entropy is highly sensitive to changes when conductivity $0 < c < 500$. This is because of the high growth rate of equation 2.9 for the first 500 numbers of c. Temperature also plays quite a large role as entropy is decreased when T is greater. Looking at Figure 3.3b, it is clear that a downward trend exists for increasing temperature levels. Likewise, higher temperature makes the data look more "noisy" with ever-increasing variations for the final entropy measurement.

While this appears counter-intuitive, there is a very good mathematical explanation of why that is. Looking again at equation 2.2 differences between energy levels are diminished when T is a large big number. Likewise, the omission of the Boltzmann constant k_b in calculations exaggerates the effects. As seen from this image in Wikipedia's definition of Boltzmann distribution. The ratios of the probability rapidly approach 1 when T becomes very large.

$$lim_{T\to\infty}\frac{p_i}{p_j} = 1$$

Reflecting back to equation 2.2, the exponent term becomes:

$$lim_{T\to\infty}\frac{8-\epsilon_j}{T} = 0$$

Therefore, for large values of T, Figure 2.2 would look more like:

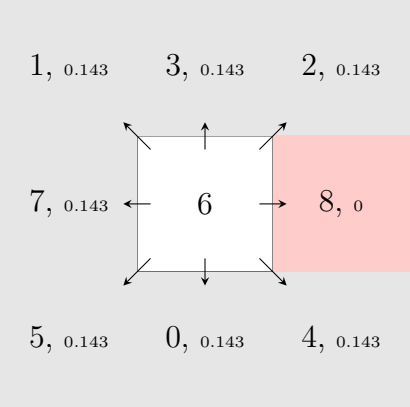


Figure 3.4: Probabilities of $\frac{1}{7}$ each for the 6 free cells at high T.

Another interpretation of this is that high temperatures cause a lot of thermodynamic noise, which diminishes the effects of local differences. Now one flaw that my simulation currently has is that it doesn't have a feature that visually tells that the temperature is high. What I might do next is to probably alter the color mapping of matplotlib such that the 0 cells emit more light to display that they are radiating energy. In that case, it's only the differences between cell energy that matter.

Chapter 4: Closing remarks

Overall, this was a passion project that is of my own doing. Well, not exactly. To be fair, 30% of the code was done by Chat-GPT3. However, the problem-solving strategy and creative output are almost entirely my own work. In early January of this year, I had a realization. The fact is that I can use the principles of cellular automata and some probability in order to demonstrate why the 2nd law of thermodynamics works and that there is no mysterious force behind it. It is only the case that entropy always increases because it's much more likely to do so than not.

Overall, this was a self-confidence boost that I desperately needed. After a very unproductive and depressing 2nd half of 2022, a part of me wanted to prove to myself that I could actually engage in high-level problem-solving and that I am capable of doing something intellectually demanding. Likewise, working on this was also an exorcism of my own demons of self-doubt and self-hatred that would otherwise bring me down to absolute mediocrity and waste the scientific skills and abilities that I acquired over the 22 years of my life.

This was a work born out of desperation and I could safely say that I feel more accomplished than getting my testamur 3 months ago in December 2022! Anyways, life goes on and the show must go on. Life was a long hard journey but I decided should no longer betray what I wanted to do as a child, which is to be a scientist.