A visual demonstration of Entropy

A simulation of stochastic processes using cellular automata.

By: Bowen Shiro Husin

30th January 2023

Acknowledgements

This project is intended to keep my brain busy while I am unemployed. 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 1s and 0s is constructed for visualizing entropy increase over time.

To declare the initial function for running the simulation. 3 Laws were proposed alongside an additional feature called α which is responsible for controlling the probability of movement. To calculate the entropy, Shannon's entropy alongside simplified techniques from mathematical papers was adopted for the specific problem.

Results from the simulation show that the underlying structure of the 2-dimensional array will become more uniformly distributed. The Move probability declared as α is shown to have a tremendous effect on the final results. The proxy measurement of entropy shows a large increase within the first hundred iterations followed by a decrease in its gradient. Slight variations of the proxy entropy measurement are also visible due to random walks that each of the 1s can take according to the 3 rules.

Contents

Chapter 1: Introduction

1.1 What is cellular automata (CA)

In short, CA is a subset of computer science that studies discrete systems. These systems are mathematical idealizations of actual physical systems that aid physicists, biologists, and mathematicians in understanding real-life phenomena.

The playing field of cellular automata is typically an infinite 2-dimensional vector space that has a discrete value in each site or cell [1]. This means that the numbers within each cell should be integers or natural numbers. This system evolves according to a set of rules which are applied simultaneously throughout the grid. In terms of functions, the process flow of cellular automata is shown in Figure 1.1.

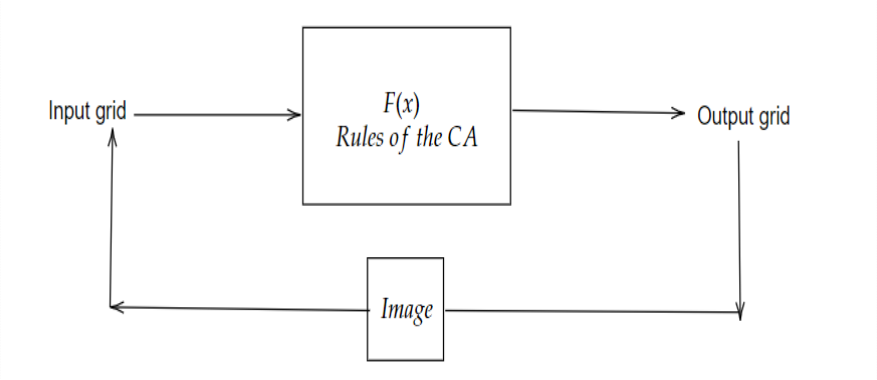


Figure 1.1: Process flow of Cellular Automata

The CA evolves in discrete integer time steps where the output grid depends on the state of the input grid. The input grid is then replaced with the output of the previous time step and the image box is there to visualize how the system evolves over time. The above loop can run forever or it can halt when the number of desired iterations is reached.

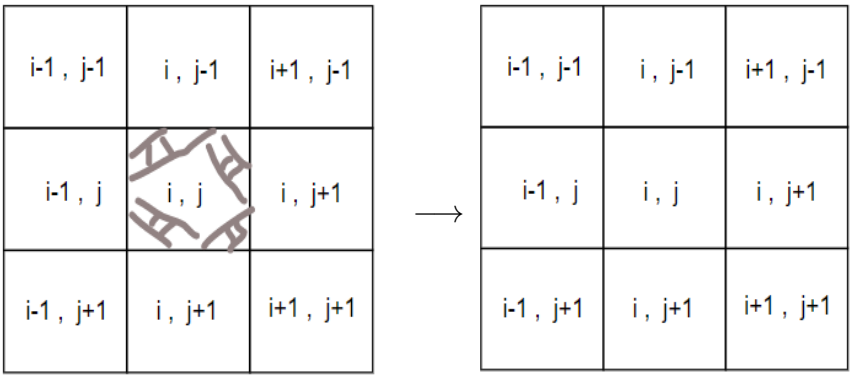
The modeling of cellular automata has been used extensively within the field of computational biology. For instance, naturally occurring CA's can be found within seashells [2] in accordance to rule 30 from Stephen Wolfram's work in 1983 [1].

Cellular automata was formally introduced in the 1940s by Polish Physicist Stanisław Ulam and mathematician John Von Neumann. However, the big hit piece that put cellular automata within the public sphere was John Conway's work in 1970 known as **The game of life** [3].

Going back to Figure 1.1, constructing F(x) requires us to lay the foundations of the rules. In Conway's game of life, the rules are:

- A cell can only be alive or dead.
- Any live cell with fewer than two live neighbors dies, as if by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

To get a better representation in coding terms, imagine the following array where the cell at i,j position is shaded (1) and the rest of cells 1 square surrounding it is white (0):



(a) Initial grid (b) Output grid

Figure 1.2: Initial and final grid state

Figure 1.2 displays the consequence of applying the rules toward the initial grid. The example above is a simple case, however, things can get far more complex. By following the same rules under the right initial grid configuration, one can actually make a "Gosper gun". The associated gif is available in Wikipedia. There are many more fascinating patterns that emerge within the game of life. Some of these patterns form a stable structure with a certain periodicity while others form chaotic structures where there is no way of predicting the final outcome. Lastly, an important property of Conway's game of life is that it is Turing complete. This means that it can perform any sort of computation and it would indeed be possible to make the game of life within the game of life within the game of life!

1.2 An introduction to Entropy

Before I introduce my simulation, a definition of entropy needs to be established. The normal zeitgeist definition of entropy is that it is a measure of disorder within a system. This is partially true. However, I think this definition is coming from statistical mechanics.

Another way to think about entropy is that it measures the amount of "useless" energy within a system. If the system is of higher entropy, there is not much you can do to extract energy from it. If the system is of a lower entropy state, then there is a lot one can do to extract energy from it. Consider the following 2 examples:

- Building a hydroelectric dam on 2 reservoirs of water of a different height versus water at even footing.
- Using fuel to drive pistons within an engine that drives movement.

In both examples, there is a lot more that one can do in the initial state (2 reservoirs of unequal height or fuel) rather than the final state (water of even footing or CO2 gases).

1.2.1 Statistical mechanics definition of entropy

To understand the statistical mechanics' definition, I would need to introduce the notion of a **macro-state** and **micro-state** .To get an intuitive feeling of what a micro-state is consider the following sequence of randomly generated 8 bits of information:

$$S_1 = [0, 1, 0, 1, 1, 1, 0, 1]$$

$$S_2 = [1, 0, 0, 1, 0, 1, 1, 1]$$

$$S_3 = [1, 1, 1, 0, 0, 1, 0, 0]$$

There are exactly 2^8 or 256 possible sequences of 1s and 0s, each of them unique. Each of these unique configurations of S_k where $k \in (1, 256)$ is a micro-state. For example, S_1 is a micro-state so is S_2 and so is S_3, and all the way to S_{256} are all possible configurations or micro-states.

To understand what is meant by a macro-state, consider the following parameter a that counts the total number of 1s within this 8-bit sequence. There is only 1 possible sequence that yields $[0, 0, 0, 0, 0, 0, 0, 0]$ *no ones* or $[1, 1, 1, 1, 1, 1, 1, 1]$ *all ones*. On the other hand, there are 8 unique combinations for the sequence of $a = 1$, $[1, 0, 0, 0, 0, 0, 0, 0]$ where each of the 1 is in different positions. Likewise, there are 28 possible unique combinations for $a = 2$ one of them being $[1, 0, 0, 1, 0, 0, 0, 0]$ where the two 1s are placed at different positions.

We can compile the results for all possibilities of a and tally up the total number of sequences associated with a in the following table:

Number of 1s (a)	No of sequences	Probability
0	1	0.0039
1	8	0.031
2	28	0.11
3	56	0.22
4	70	0.27
5	56	0.22
6	28	0.11
7	8	0.031
8	1	0.0039

Adding each of the elements within the number of sequences adds up to 256 and the associated probabilities to 1.

Going back to what is meant by a macro-state, the macro-state in this context is the measure of the Number of 1s. So for the macro-state of having four 1s $a = 4$, there are 70 possible micro-states or combinations where there are four 1s in the randomly generated sequence S_k. Likewise, there are 56 possible micro-states associated with the macro-state of having three 1s or five 1s.

Now that notions of macro-state and micro-state are defined. I can finally put **entropy** into the mix :) So what is this entropy? Entropy is simply a direct measurement of the number of micro-states a system is associated with its macro-state. In statistical mechanics, Ludwig Boltzmann defined entropy as:

$$S = k_b ln(\omega)$$

Where entropy is S and the number of micro-states corresponding to its macro-state is ω. For my example, I will omit the Boltzmann constant k_b and modify the equation as:

$$S_a = log_{10}(\omega_a)$$

In this equation, the entropy associated with the macro-state a is given by S_a. Applying this equation to the above table yields:

Macro-state (a)	Micro-states	Entropy S_a
0	1	0
1	8	0.903
2	28	1.447
3	56	1.748
4	70	1.845
5	56	1.748
6	28	1.447
7	8	0.903
8	1	0

As seen from the table, the entropy for 3,4, and 5 is a lot larger than the entropy values of 1 or 7. By this section, the definition of entropy is complete. However, there is another piece of the puzzle that needs to be addressed.

1.2.2 The first law of thermodynamics

The picture of entropy isn't quite complete without 1st law of thermodynamics. The first law formally states:

$$\Delta U = Q \pm W \tag{1.1}$$

In plain English, this states that the change in the internal energy of a system is equal to the energy supplied to the system, (denoted by Q) added by the thermodynamic work to the system, (denoted as W). The $\pm$ sign is there depending on 2 situations. If thermodynamic work is added to the system, it's a $+$, if work is extracted from the system it's a - sign.

The above situation describes any **closed system** in which energy can flow but matter cannot. Examples of closed systems include:

- Gas within a fully enclosed box where no gas can enter but can still get heated.
- The room of a teenage emo where the parents can't come in after a temper tantrum but can still call his/her phone lying within their room.
- The particle accelerator at CERN where no gas or matter can come in to disrupt the experiment of mad geniuses but the pipe housing the proton collision can still dissipate heat and get colder.

There is also the idea of an **isolated system**. In this case, neither matter nor energy can flow into the system. Examples of isolated systems include:

- Gas within a perfectly enclosed and insulated box where no gas can enter and no energy can flow in or out.
- The Universe

For an isolated system Equation 1.1 reduces to:

$$\Delta U = 0 \tag{1.2}$$

This means that for an isolated system, energy cannot be created or destroyed which is essentially the first law of thermodynamics.

1.2.3 Putting the puzzle pieces together

To understand the interplay between the 1st law of thermodynamics with entropy. Consider the following game:

Imagine that there are 2 systems within an isolated system. Each system is denoted as:

$$S_1 = [1, 0, 0, 1, 0, 1, 0, 0.....]$$

$$S_2 = [0, 1, 0, 0, 1, 1, 0, 0.....]$$

Where both systems are:

- Each of set length l containing l elements and the length of both is $l + l = L$
- Each element is discrete and only takes binary values.
- The number of 1s between these 2 systems is constant E.

  Means: The number of 1s between S_1 and S_2 must be equal and constant

  Mathematically: $n_1 + n_2 = E$, where $n_1, n_2 \in N$

  For the math nazis: For simplicity sake, consider $0 \in N$

To initialize the game, let $E \leq l$ where $l \in N$ and $E \in N$.

If I choose the $l = 12$ and $E = 12$. There are 13 possible pairs of $n_1 + n_2$ such that $n_1 + n_2 = 12$. Some of these pairs of $n_1 + n_2$ are:

- 0+12, $n_1 = 0$, $n_2 = 12$
- 7+5, $n_1 = 7$, $n_2 = 5$
- 8+4, $n_1 = 8$, $n_2 = 4$
- 3+9, $n_1 = 3$, $n_2 = 9$

The total number of permutations or micro-states that this game can take is 2^{2l} or 2^{24} in the case above. However, the total number of combinations the game can take for a set amount of E is:

$$^{L}C_E = \frac{(L)!}{E!(L-E)!} \tag{1.3}$$

Likewise, each of the 13 pairs of $n_1 + n_2$ has an associated number of configurations that satisfy the pair. In this case, the number of configurations must add up to $\frac{(24)!}{12!(24-12)!} = 2704156$. We can tabulate all possible sequences corresponding to the possible pairs of n_1 and n_2 as follows:

n_1	n_2	Micro-states	Probability	Entropy
0	12	1	0.00	0
1	11	144	0.00	2.16
2	10	4356	0.00	3.64
3	9	48400	0.02	4.68
4	8	245025	0.09	5.39
5	7	627264	0.23	5.8
6	6	853776	0.32	5.93
7	5	627264	0.23	5.8
8	4	245025	0.09	5.39
9	3	48400	0.02	4.68
10	2	4356	0.00	3.64
11	1	144	0.00	2.16
12	0	1	0.00	0

Now consider S_1 and S_2 to be closed systems where each sequence can transfer a 1 to each other in any way such that $n_1 + n_2 = E = 12$ is conserved.

If I start with 2 sequences such that $n_1 = 10$ and $n_2 = 2$. What is the probability that n_1 increases spontaneously? It turns out that this number is astronomically low as $\frac{145}{2704156}$.

So if you imagine the sequence S_1 being a hot object with $\frac{10}{12}$ elements being heated and sequence S_2 being a cold object with $\frac{2}{12}$ elements being heated. The chances that S_1 gets hotter is $\frac{145}{2704156}$. Basically, it almost never happens spontaneously.

The chances that a hot object gets hotter and a cold object gets colder becomes even more unlikely when the length of the sequences l and energy units E are scaled up. Now consider $l = 100$ and $E = 100$. If I start with S_1 having an $n_1 = 30$ and S_2 having a $n_2 = 70$, the chances that S_1 gets colder spontaneously is 0.00000001155163215. Now that is only for 100 particles in real life a typical object like a mug of 300 ml of water contains 16.7 moles of water. If you'd imagine 2 different cups and each particle of water having the same property of being heated or not heated. The chances that the hotter cup gets hotter is in the order of magnitude of 10^{-100}. It's probably much smaller than that!

Real-life phenomena constantly alternate between each micro-state at random with each of them being of equal probability. In my example of 1.3, there are 2704156 possible micro-states that satisfy $n_1 + n_2 = 12$. Each of the micro-states is equally likely with one another. If we started with S_1 having six 1s and S_2 having six 1s. The chance s are that we are going to see a huge drop in entropy from 5.93 to anything equal or lower than 3.64 is $\frac{9002}{2704156}$. Which is a very small number. Scaling this up to real-life scenarios would just mean that a noticeable drop in entropy due to spontaneous arrangement is neigh impossible.

This brings us to another fact: **Entropy can decrease but it will always *tend* to increase**. This is the 2nd law of thermodynamics.

Concerning, orderliness and disorderliness. I would like to rephrase the question as energy being concentrated versus energy being spread out. In 1.3, a configuration of orderly structures are pairs of $(12, 0), (0, 12), (1, 11), (11, 1), (10, 2), (2, 10)$. As seen from the table, there are many more configurations where the energy units are more evenly distributed between the 2 sequences S_1 and S_2, the associated micro-states that are available for all other pairs is 2695154 which means that if I were to look at the game at random. There is a 99.7% chance that the energy between the 2 sequences is more spread out.

Additionally, if I started with 1 sequence having 10 energy units $n_1 = 10$ and the other with 2 energy units $n_1 = 2$, it is almost a guarantee that it will no longer be in the same configuration pair of $(10, 2)$ after some time and will most likely be in n_1, n_2 pairs of $(4, 8), (6, 6), (5, 7), (7, 5)$. There is an associated increase of entropy accompanying this phenomenon but there is absolutely zero force driving the direction toward randomness. It is simply because being random and disorderly is more likely than it is to be orderly.

Chapter 2: Stochastic processes

2.1 Extending the idea

In section 1.2.3, I introduced the idea that you could actually have 2 sequences S_1 and S_2 with a set number of 1s in each. Now I would like to extend the idea further and have a sequence $S_k = [1, 0, 0, 1, 0, 0, 1, 0, 0....]$ of length l, l times. This means the final system will be:

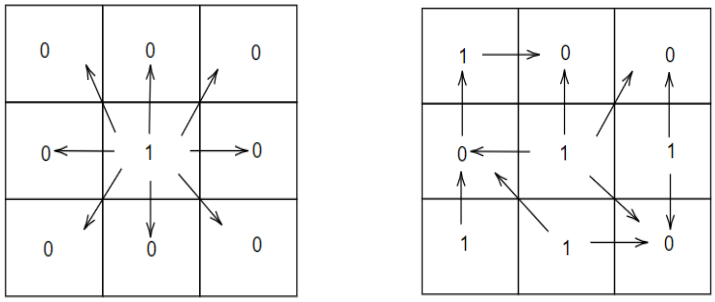
$$M = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ ... \\ S_l \end{bmatrix}$$

What we end up with is a square matrix M with l^2 elements. Each of these elements will still be discrete with it only taking binary values of 1 or 0.

To model real-life phenomena of heat transfer, conservation of energy, and conduction. I would introduce 3 things these are:

- **Law 1:** A cell can either be a 1 or a 0, on or off.
- **Law 2:** The number of 1s (E) within the Matrix M cannot change.
- **Law 3:** Each of the 1s can move to an empty square (0) that is 1 square surrounding it.
- **Features:** There is a probability multiplier labelled α which modifies the chances of movement. If $\alpha = 0$ nothing will happen. If $\alpha = 10\%$, the arrows at 2.1a have a 1.25% chance of happening each. If $\alpha = 100\%$, the arrows are 12.5% each. Generally, this means that if it can move it will move if it can't it won't.

We can then enclose these 3 bullet points within a function $F(M, \alpha)$. For k iterations, the output is $M_{k+1} = F(M_k, \alpha)$. This is related to Figure 1.1 where the output at a specific iteration is fed back as the input for the next iteration until a certain number of iterations is reached. The second rule is an analog of the 1st law of thermodynamics. The third law governs heat transfer and the feature mirrors that of heat conduction. To provide a clearer way of how the third law works consider the following Figure 2.1.



(a) Valid directions for a 1 surrounded by 0

(b) Valid directions for multiple 1s.

Figure 2.1: An explanation of the 2nd law

The arrows depict the possible directions that an occupied cell (1) can go. Under appropriate code implementation, the 1s should not be able to jump 2 spaces apart. In the special situation where an occupied cell is completely surrounded by occupied cells, the only possible outcome is for it to remain in the same position.

2.1.1 Computing entropy

The next task is to find a way to compute the entropy of the matrix M_k against its associated iteration. Allowing for k to be 600 means that the final entropy plot will contain 600 pairs of $(k, H(M_k))$ where $H(M)$ is the entropy function. We can compute the entropy in 2 ways:

- Adopt a modified version of 1.3 and find integer combinations of $n_1 + n_2 + n_3 + ... + n_l = E$ and find their unique micro-state tally.
- Adopt an imperfect approximation using Shannon entropy.

Method 1 works for small numbers of l but if $l \geq 50$. The computation of method 1 gets extremely tedious and computationally demanding. Likewise, I am too lazy to even think about the possible micro-states possible associated with a unique integer combination of $n_1 + n_2 + n_3 + ... + n_l = E$. Therefore, I will adopt method 2 within my simulation.

The Shannon entropy of a system is given as follows:

$$H = -\sum_{i=1}^{N}(P_i)log_2(P_i) \tag{2.1}$$

However, I prefer the alternative version of Shannon entropy explained by Josh Starmer which is:

$$H = \sum_{i=1}^{N}(P_i)log_2(\frac{1}{P_i}) \tag{2.2}$$

In plain English, Equation 2.2 says that for the associated probability, multiply it by its *surprise* and sum it up for all the probabilities.

To compute the entropy for each of the iterations of M_k. I would rearrange matrix M and flatten it to a 1-dimensional sequence. This sequence G can be written as:

$$G = [S_1, S_2...S_l]$$

The sequence G will almost look like a tape of 1s and 0s with length l^2. The next step is to partition the tape into 2-bit binary digits. This means that there will be $\frac{l^2}{2}$ partitions of 2-bit binary digits if l^2 were even. If l^2 is an odd number, the number of partitions of 2-bit binary digits is $\lfloor \frac{l^2}{2} \rfloor - 1$. Now, that the partitioning is done, we can then calculate the Shannon entropy for each of the 2-bit binary partitions. The computation of this simple algorithm is demonstrated in the table down below:

While we can stop here, a specific situation could arise where if l^2 were an odd number and the digit at the last index of G is a 1, the algorithm I described would miss it entirely. Hence, I modified it as shown in 2.2.

Parameter	1s	0s
Probability	$\frac{E}{2}$	$\frac{2-E}{2}$
Surprise	$\log_2 \frac{2}{E}$	$\log_2 \frac{2}{2-E}$

Otherwise, note that E is a function that just calculates how many 1s are within a defined binary string of length l. The possible sequences of a 2-bit binary string are $(0,0),(1,0),(0,1),(1,1)$. For the situation of $(0,0)$ the E within the above table is 0. This means that the Shannon entropy for the $(0,0)$ string is:

$$H((0,0)) = \frac{0}{2}\log_2 \frac{2}{0} + \frac{2}{2}\log_2 \frac{2}{2}$$

The first term is undefined while the 2nd term yields a 0. Getting an undefined result is OK within the context of Shannon entropy as the surprise of an event that never happens can be considered to be 0. Hence for the string $(0,0)$ the Entropy associated with it is 0. This is also the case for the string $(1,1)$ where the entropy also adds up to 0.

For the situation of the strings $(1,0)$ and $(0,1)$ the entropy calculation yields:

$$H((1,0)) = \frac{1}{2}\log_2 \frac{2}{1} + \frac{1}{2}\log_2 \frac{2}{1} \longrightarrow 1$$

The results for computing 4 different permutations of 2-bit binary strings are shown in the table below:

Bit	1	0
1	**False**	**True**
0	**True**	**False**

False (0) values are the result of having $(1,1),(0,0)$ pairs, and True (1) is returned when the pairs have different elements such as $(1,0),(0,1)$. This is the **XOR** gate.

The XOR gate has been used in determining the entropy for randomly generated binary strings. For instance, the work by Grenville J. Croll [4] analyzed the entropy of a finite binary string. His algorithm is to compute the entropy of the original string, store that value and then take the XOR value of the original binary string of length l 2-bits at a time. The resulting 1st *derivative* of the string is a string of length $l-1$. We can then compute the entropy of the 1st derivative and then store this value. Next, we apply the XOR function again to the first derivative resulting in the 2nd *derivative* and take its entropy. This algorithm stops until its last derivative is all 0s or 1s. Based on this information, we can determine whether the original string contains periodic patterns. Finally, after we computed the entropy for each derivative we multiplied it with a weight w and sum them up to find the entropy of the original binary string.

While the above algorithm will give an accurate measurement of the entropy of a binary string of a flattened matrix M or G, it would be computationally demanding to implement it alongside the original simulation. This is especially true for large square matrices such as 200x200 ($l = 200$) with 40000 elements in matrix G. Hence for the simulation, I decided to opt for just applying the XOR function once which will result in a $l^2 - 1$ bit binary string.

Once the $l^2 - 1$ bit binary string is computed, the next step is to just sum up all the 1s of this string which reveals the number of unique occurrences of pairs of $(1,0),(0,1)$ within the original string. Depicting how this works is shown within Figure 2.2 down below:

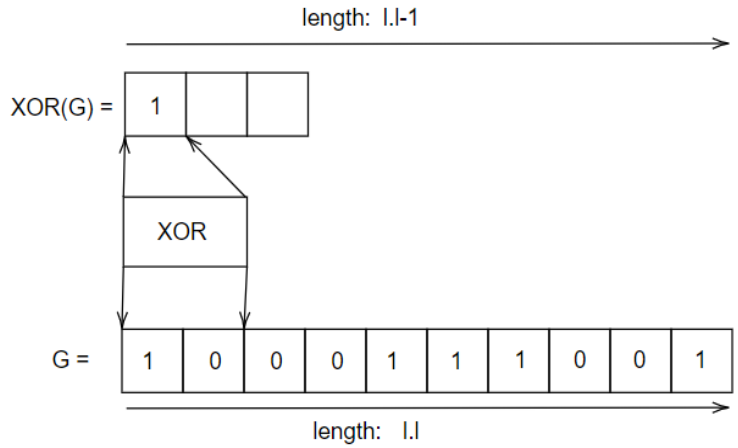


Figure 2.2: The XOR machine moves 1 square right per iteration.

Using the algorithm depicted above, it is possible to determine the upper bound sum of the number of 1s of XOR(G). Noting that I defined E as the number of 1s in matrix M and that E is a constant. The upper bound and lower bounds of this proxy entropy (H) are:

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

The only way H can actually be equal to $2E$ is if only if the sequence G is in the form of $G = [0,1,0,1,0,1....1,0]$. A fully alternating binary string with the first and last index positions being a 0. In this scenario, the function XOR(G) will count unique occurrences of $(1,0),(0,1)$ twice and will double count the number of 1s in G.

The method that I described above can be criticized as being rather *caveman* like in its way of thinking and some major flaws are:

- The algorithm fails when $E \geq 0.5l^2$.
- $G = [0, 1, 0, 1, 0, 1....1, 0]$ is not exactly disordered or entropic.

However, for a large square Matrix M where $l = 200$. I would want to limit E to be $E \leq 0.05l^2$ which in this case, applies the game rules to matrix M such that $M_{k+1} = F(M_k, \alpha)$ the underlying grid will be more disordered if there are more unique occurrences of $(1, 0), (0, 1)$.

The code for implementing this monster is posted on my GitHub page with its specific repository at Here. The source code is named Automata.py

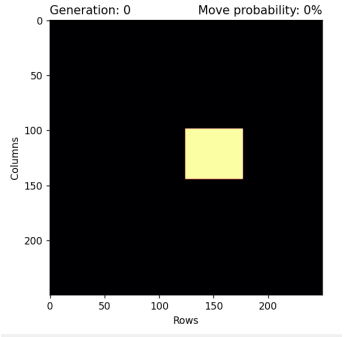
Chapter 3: Interesting results

3.1 Implementation

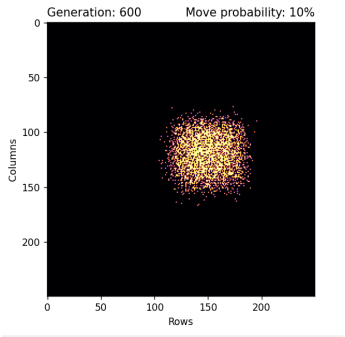
Referring back to Figure 1.1. We can then implement the iteration of matrix M by employing the function $F(M, \alpha)$ that encodes the rules of the simulation. The way it iterates is While $k \leq 600$:

$$M_{k+1} = F(M_k, \alpha)$$
$$G_k = f(M_k)$$
$$H_k = \sum XOR(G_k)$$

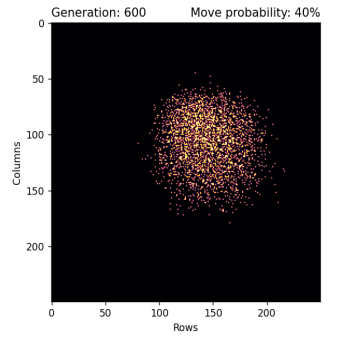
Where f just flattens the matrix M. Now, the task is to display M_{k+1} and plot 600 points of (k, E_k) side by side. A potential way to look at how the heat conduction analog, α affects the final entropy after 600 iterations are to check H_{600} for different α values. Some of the static images of the effect of α are shown in Figure 3.1.



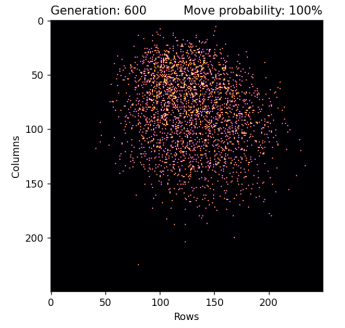
(a) The initial grid



(b) $\alpha = 10\%$



(c) $\alpha = 40\%$



(d) $\alpha = 100\%$

Figure 3.1: The Grid after 600 iterations

The grid is a 250x250 matrix where $E = 2438$. The initial grid M_1 is Figure 3.1a which is a large rectangular box. Activated cells (1) are colored yellow while the other cells (0) are colored black. As seen from the figures, it can be seen that α has a large effect on how spread out the activated cells are.

The general effects of α on the proxy entropy measurement H can be seen on Figure 3.2 shown below:

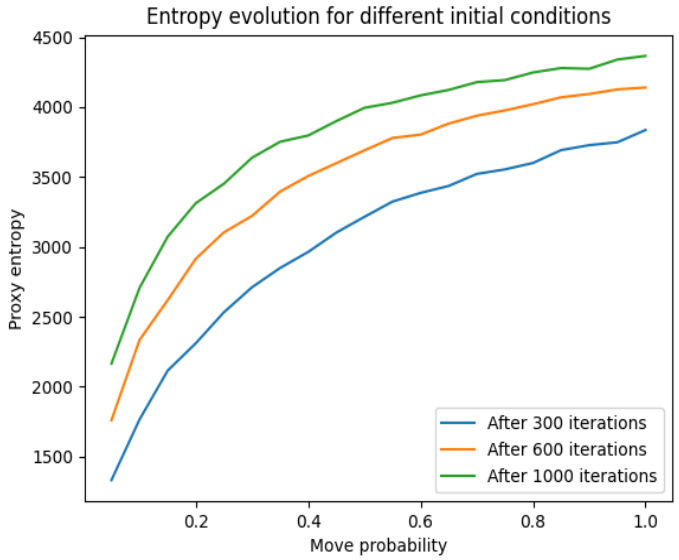
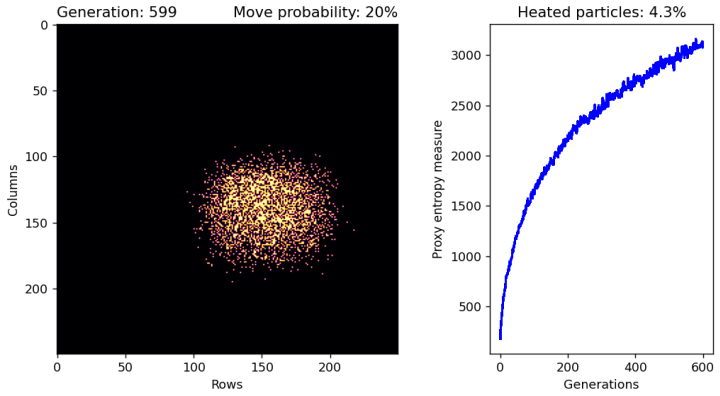


Figure 3.2: The effects of α on the final entropy after set amount of simulations

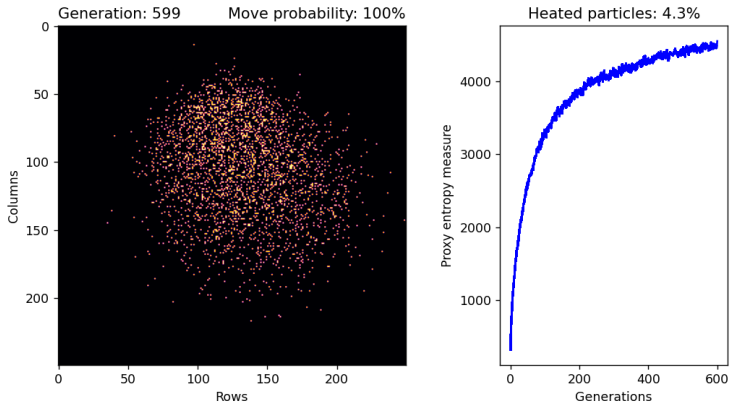
The above figure runs have 20 different points for α (Move probability) ranging from 0 to 1 in increments of 0.05. For each of these points, 5 simulations were run and the arithmetic means of these 5 simulations of 300,600, and 1000 iterations were collected. The final points that are associated with the α are then plotted for 300, 600, and 1000 iterations. Here the Matrix M is a standardized square matrix of 250x250 with an E value of 2438.

As seen from Figure 3.2 the graph shows that α has a tremendous effect on the final proxy entropy H, with diminishing returns. A similar trend of increasing H followed by a decrease in the gradient of the graph is seen in Figure 3.3

For Figure 3.3, the simulation's proxy entropy H is plotted against its iteration. For $\alpha = 1$, H shows a sharp increase in the first few iterations and the gradient of the graph decreases over the iterations. The trend indicates that maximum entropy will be reached when the iteration number becomes very large. While the graph trends upwards, there are slight variations for H. This is because the random nature of the simulation might decrease the number of unique occurrences of $(1,0),(0,1)$. However, the number of unique alternative $(1,0),(0,1)$ will increase as the underlying matrix gets more homogeneous simply because many more configurations match a homogeneous distribution rather than an ordered structure.



(a) $\alpha = 0.2$



(b) $\alpha = 1$

Figure 3.3: Results after 600 iterations for different alpha values

3.1.1 Known Bugs

From Figure 3.1d, it is visible that the colored cells tend to go up. Investigating further reveals that there might be a bug within the implementation of $F(M, \alpha)$. We can investigate the problem further by identifying which directions the code chooses. There are 8 pairs of directions it can choose, these are:

$$([0,-1],[-1,0],[1,0],[0,1],[-1,-1],[1,-1],[1,1],[-1,1])$$

We can mark these directions into an appropriate string, such as:

$$[\text{"}T\text{"},\text{"}L\text{"},\text{"}R\text{"},\text{"}B\text{"},\text{"}TL\text{"},\text{"}TR\text{"},\text{"}BR\text{"},\text{"}BL\text{"}]$$

Each corresponds to the pairs. In this case, T is top direction L is left, R is right, TL is top-left and BL is bottom right. Finally, we can run a while loop for 6000 iterations and plot the results on a histogram to see which letter it chooses the most.

In Figure 3.4 it can be seen that the string "T", associated with the direction vector $[0, -1]$ is chosen slightly more than the rest, with a rate of 13.5%. The differences are small but they are noticeable as in Figure 3.1d, the colored points tend to go slightly upwards. As for the current situation, I have yet to know what is causing this slight drift within the directions.

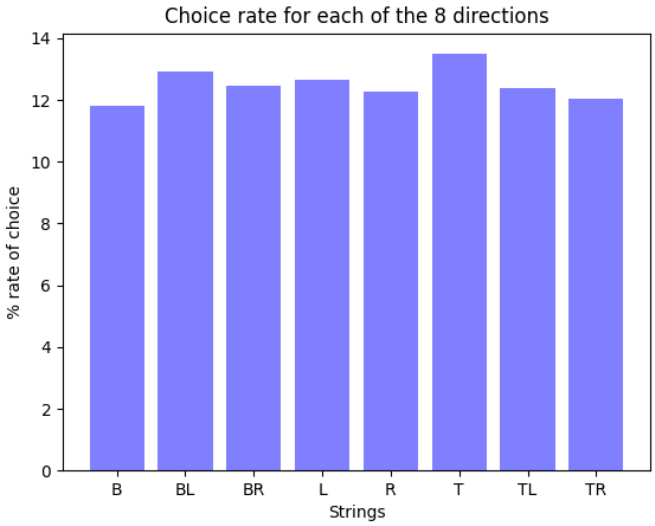


Figure 3.4: Frequency density of the 8 possible directions after 6000 iterations

3.1.2 Further plans for Version 2

In Quantum mechanics, certain discrete energy levels are allowed. Such a system is usually depicted in Figure 3.5 below:

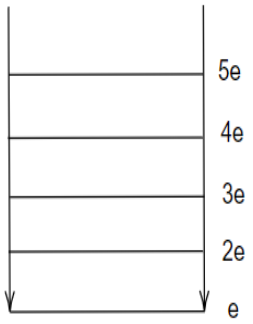


Figure 3.5: Discrete energy levels

The level corresponding to "e" is the ground state or the lowest energy state that a particle can carry. This hypothetical particle can take up to 5e units of energy.

Things get interesting when one questions how many different energy levels are possible when there are 2 or more particles involved with a set energy level. Defining E is the total amount of energy as before, we can count the total amount of energy for 2 particles to be $e_1 + e_2$. Our job is to then find the total number of integer combinations where $e_1, e_2 \geq 1$, and both of them add up to E. If E=4 then there are 3 pairs of e_1 and e_2. These are $(1,3), (3,1), (2,2)$ which corresponds to 3 different micro-states. To see how more particles affect the different number of combinations you can view a video made by Parth G in which he does a really good job in explaining what is happening.

Going back to the rules, which are:

- **Law 1:** A cell can either be a 1 or a 0, on or off.
- **Law 2:** The number of 1s (E) within the Matrix M cannot change.
- **Law 3:** Each of the 1s can move to an empty square (0) that is 1 square surrounding it.
- **Features:** There is a probability multiplier labeled α which modifies the chances of "heat transfer". Generally, this can be read that if it can move it will move, if it can't it won't.

I wondered what if I modified Law 1 while keeping everything else constant. What if Law 1 is restated as:

- **Law 1:** A cell can take up to 9 energy levels with the ground state corresponding to 0 and the maximum energy corresponding to 8.

This means that each of the cells at Matrix M can only take integer values of $0, 1, 2, 3, 4, 5, 6, 7, 8$ while the rules of heat transfer are still governed. Meaning that any cell that is less than 8 is a vacant cell in which energy can transfer.

Such a change within the rules would be interesting to simulate. However, entropy will be quite a problem to recompute. In version 1 of Automata, Entropy was simply calculated by applying the XOR function to a flattened matrix M called matrix G. I could make an array of truth values such that any value bigger than 0 will be captured as 1 and that counting unique occurrences of $(1, 0), (0, 1)$ would reveal how spread out the energy is. Following that, a spread-out configuration corresponds to more micro-states which means higher entropy.

While this could work, such a technique is ridden with problems and will require a lot more thought to reformulate. Alas, more research would need to be conducted and a careful approach to picking a solution matched to my problem would need to be adopted.

Chapter 4: Conclusion

Simulation runs to demonstrate the evolution of 1s according to the rules show that the underlying matrix M will get more homogeneous. This is indicated by the increase within the proxy entropy measurement H which calculates the unique occurrences of $(1,0),(0,1)$ patterns. The feature α referred to as the Move probability is also influential with the final entropy measurement and simulation runs showing a more modest influence.

There was one bug that was discovered during the observation of simulation runs. The code shows a slight bias of choosing the vector direction associated with the "Top" direction or the vector $[0,-1]$. This is seen in 3.1d where after 600 runs the scattered colored pixels tend to the upwards direction. Currently, consultation from a better coder is needed to identify the cause.

Finally, more plans are underway for the next version of the simulation. One way of tinkering with the rules is to allow for each cell within matrix M to take 9 possible values of $0,1,2,3,4,5,6,7,8$. This is to match with real-life quantum-mechanical systems in which each particle can have multiple energy levels. In this case, 0 corresponds to the ground state and 8 represents the maximum amount of energy or quanta that each particle can take.

Bibliography

[1] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, 1983.

[2] S. Coombes, "The geometry and pigmentation of seashells," *"University of Nottingham"*, 2009.

[3] M. Gardner, "The fantastic combinations of john conway's new solitaire game 'life," *Scientific American*, 1970.

[4] G. J. Croll, "Bi-entropy: The approximate entropy of a finite binary string," *arXiv preprint arXiv:1305.0954*, 2013.