Cellular Automata Simulations

Tan Kok Cheng, Anthony
School of Physics, Georgia Institute of Technology
School of Physical and Mathematical Sciences, Nanyang Technological University

Shah, Karan
School of Physics, Georgia Institute of Technology
College of Computing, Georgia Institute of Technology

Stanley, David
School of Physics, Georgia Institute of Technology

Abstract: Cellular automata are known for their remarkable ability to exhibit complex behaviors from a few simple rules. With the aid of Java applets and visualization hardware, we explore 1 to 2 dimensional cellular automation which includes elementary cellular automata, the Game of Life and spiral waves. And also when applicable, explore the possible application of them.

1. INTRODUCTION

Cellular automata (CA) are discrete, rule-based computational systems first proposed by von Neumann in early 1950s. They have been observed to exhibit remarkably complex behavior from simple rules which can be useful to model complex systems in the scientific field.[1]

The simple model consist of a lattice of regularly spaced cells, each with a finite number of state. Theoretically, this lattice can be of any dimensions but the model is largely studied in our physical dimensions ranging from 1 to 3 dimensions. The number of cells in each dimension can be infinite. To facilitate the study of CA, hardware and software simulations were created. However due to visualization purposes the study has the following limitations

- Number of cells is limited to the resolution of our simulating system, model
- Studies up to 2 dimensions
- Only 2 cell states – on, off

The report will briefly introduce the simulation program and hardware used for the study which covers interesting topics of CA in 1 and 2 dimensions namely elementary Cellular Automata (1D), Spiral Waves (2D) and The Game of Life (2D).

2. SIMULATION PROGRAM AND HARDWARE

2.1. Software

A java applet was developed to help facilitate the visualization of 1 and 2 dimensional cellular automata. The applet can be found in the following website: http://ca.karanprime.science/. The applet has 4 modes as follows

1. The Game of Life simulation
2. Spiral Wave simulation
3. Elementary Simulation
4. Comparison for GOL simulation

The user interface of the applet is shown in figure 2.1. The interface is programmed to take in common variables and items such as the number of cells, start and speed buttons, and also parameters unique to the mode (run the applet for details).

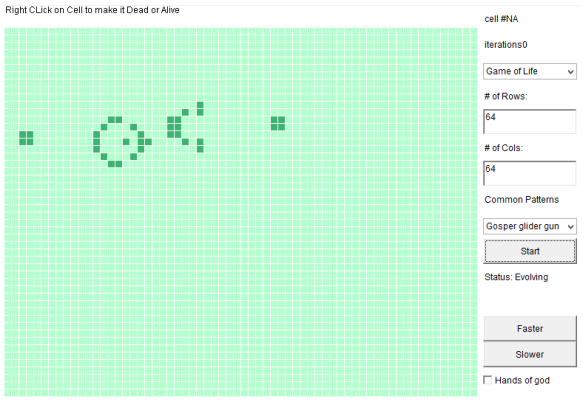


Figure 1: UI of the cellular automata applet used in the study

2.2. Hardware

Arduinos were used as the main drivers to control physical LED matrices which mimics the lattice of cells. And in some cases, a random number generator was implemented by using the Arduino to control a LCD display.

The Arduinos used are of various models, mainly Uno and Mega. The following things were implemented,

1. The Game of Life on a 8x8 LED matrix
2. Elementary cellular automata of various rules on a 8x8 LED matrix
3. Random number generator (based on rule 30 of elementary cellular automata) on a LCD display

3. THE GAME OF LIFE

The game of life is a 2 dimensional cellular automata. The game only requires an initial start state, which will evolve after discreetly in time according to the rules governing it (covered below). It is termed the game of life because the evolving patterns closely resembles the struggles of living organisms. Therefore naturally each cell has only 2 states, which are dead or alive. The game was invented by mathematician John Conway in 1970. Conway carefully choses the rules to meet these criteria:

- No explosive growth
- Existence of small initial patterns that evolves into chaotic outcomes
- Potential candidates for Neumann Universal constructor

3.1. Rules

The playing field of the game is a 2 dimensional grid of regularly spaced square cells. The number of cells can be infinite however the resources behind this simulation are finite the field studied limited to the resolution of the system running it.

Each cell interacts with 8 nearest neighbors shown in figure 2(a) where the light green cell is the cell in question and the 8 dark green cells are its nearest neighbors.

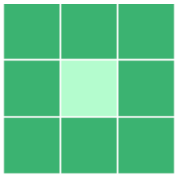


Figure 2(a): Eight nearest neighbors of a cell

The rules for each cell are as follows:

1. Live cell with less than two live neighbors dies as if caused by under-population.
2. Live cell with two or three live neighbors lives on to the next iteration.
3. Live cell with more than three live neighbors dies as if caused by overcrowding.

4. Dead cell with exactly three live neighbors becomes alive resembling reproduction.

3.2. Patterns

Some frequently occurring patterns in the game can be categorize as still lifes and oscillators and space ship.

3.3. Still Lifes

Still lifes are static patterns that stay in the state forever if undisturbed. The most common still life is the block (see figure 2(b)). Other common still lifes are shown in figure 3, 4 and 5.

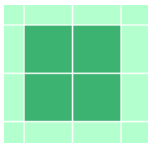


Figure 2(b): Still Life known as the block

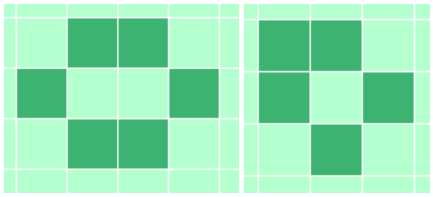


Figure 3: Still Life known as the Beehive(left) and Boat

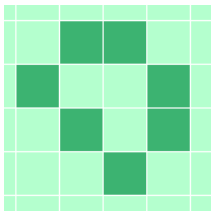


Figure 4: Still life known as the Loaf

Pseudo still life made up of 2 or more “island” can exist even though the individual “island” alone might not be a still life. Figure 5 and 6 shows some commonly found pseudo still lifes.

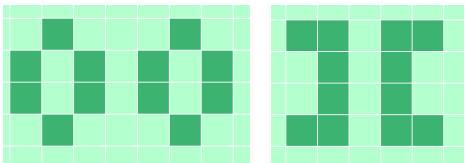


Figure 5: Pseudo life known as Bi-Hive (left) consist of 2 still life "island" while the right pesudo life consist of 2 non-still life "island"

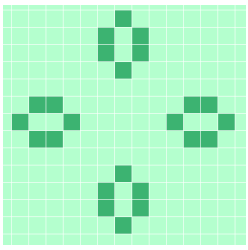


Figure 6: Pseudo Still Life known as the Honey Farm which consist of 4 islands

3.4. Oscillators

A still life can be seen as repeating itself with a period of 1 and naturally there should be life with periods more than 1. This patterns are termed as oscillators. Figure 7-9 shows oscillators of period 2, 5 and 8 along with their corresponding patterns in each generation. Oscillators are not limited to what is shown here, in fact there are many more known oscillators with various periods in the game.

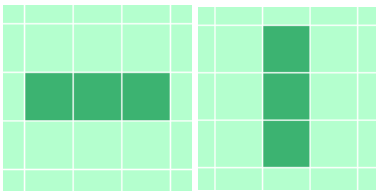


Figure 7: Period 2 oscillator, Blinker

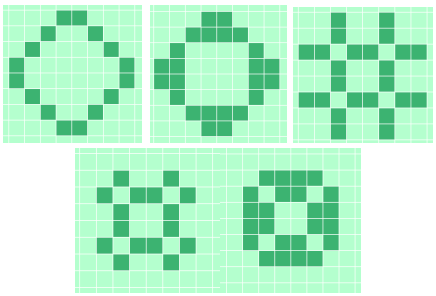


Figure 8: Period 5 Oscillator, Octagon

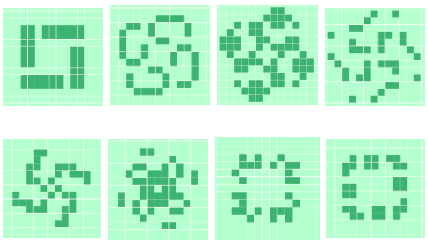


Figure 9: Period 8 Oscillator, Kok's Galaxy

Spaceships

There are also patterns that reappears a finite period however in a different position. The pattern can be seen as travelling in space with a velocity and hence such patterns are termed as spaceships. Like all patterns introduced, there are numerous kinds of spaceships, the 2 simplest one along with their trajectory are shown in figure 10 and 11.

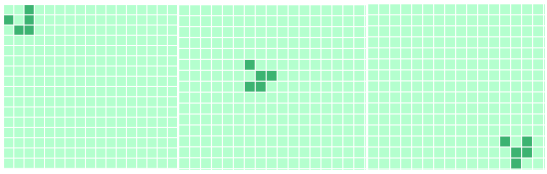


Figure a0: Spaceship known as the Glider travelling diagonally across space

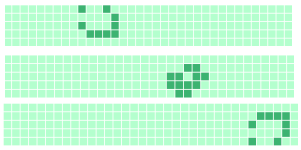


Figure 11: Spaceship known as the Lightweight spaceship (LWSS) travelling horizontally across space

Other Interesting Patterns

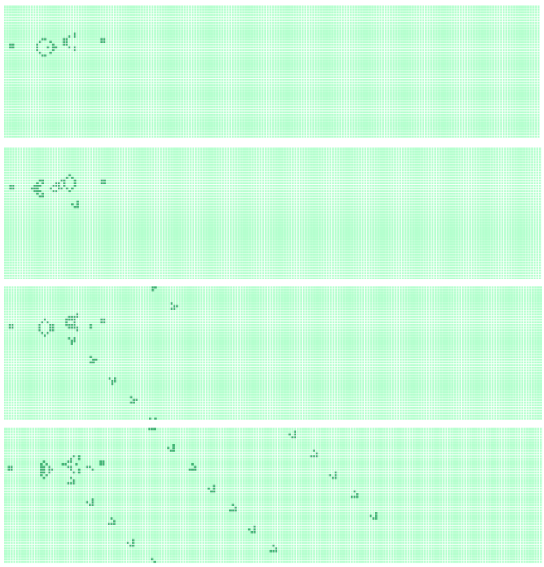
Guns

During the early stages of the game, Conway posed a challenge to anyone who can find a initial configuration that will grow infinitely. This led to several very interesting findings. Firstly the challenge was won by a team from MIT with their discovery of what is known as the "Gosper gun" (see figure 12). This pattern acts literally like a gun shooting out gliders (see figure 10) periodically (every 15 iteration, see figure 13). The gun can be seen as a life giving machine, giving birth to infinitely many living

Figure12: The Gosper Gun

spaceships that travels across space. Smaller patterns that exhibit infinite growth were found subsequently.

There is an interesting phenomenon with the Gosper gun that will be discussed. But before that we will talk about periodic space. Imagine that the ends of the playing field are stitched together; the top to the bottom and left to right leading to a toroidal array. In this way the pattern is allowed to evolve freely without losing them at the edges. The stitched field can also be seen as the surface of a sphere where they are no ends to it. Now putting a Gosper gun in such a playing field will eventually shoot itself to "death" when the gliders generated by it wraps around to destroy it. A series of snapshots of the event is shown in figure 13. Future more, upon destruction, there will be a surge of activities taking place resembling explosive after effects.



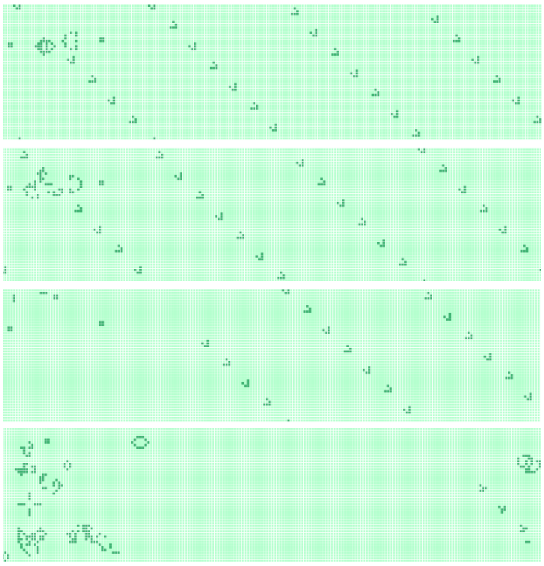


Figure13: Snapshots of the Gosper Gun generating gliders within a toroidal array which eventually destroy itself in spectacular fashion

Methuselahs

Also of interesting properties are patterns called "Methuselahs" which evolve over long periods before reaching equilibrium. One of the most intriguing is called the Acorn (shown in figure 14). At merely 7 initial living cells, the pattern takes 5206 generations to settle down with an astonishing 633 living cells including 13 gliders. Such a model resembles a living organism at work, and surprising the process of only 7 cells and 4 simple rules!

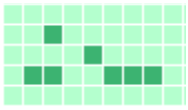


Figure 14: Methuselahs known as the Acorn

4. SPIRAL WAVES

There are numerous spiral wave like patterns observed in reality such as in the brain and heart, combs of bees, and in chemical reaction such as Belousov-Zhabotinsky reaction, oxidation of carbon monoxide on platinum surfaces. They have also appeared in numerical simulations like the reaction-diffusion systems[2].

Figure 14(a): Belousov-Zhabotinsky Reaction

Figure 14(b): Spiral combs of bees

We will first talk about the rules to simulate a spiral wave like the one shown in figure 17. This wave when generated will last forever. However we are not interested in such spiral waves, instead it will be the base where we attempt to simulate spiral waves observed in nature.

Rules

The playing field is a 2 dimensional grid of square cells which may be infinitely many. However the resources behind the simulation are finite therefore the grid studied in this report is limited to the resolution of the system running the simulation.

Figure 15: Four nearest neighbors of a cell

Each cell interacts with 4 nearest neighbours shown in figure 15 where the red cell is the cell in question and the 4 dark green cells are its neighbours.

The rules for each cell are as follows:

1. The start state of every cell is deactivated unless they are activated in the initial condition.
2. The cell will activate if any of its neighbors are activated.
3. Cell will deactivate after activating.

The duration to remain activated when activated and the duration to remain deactivated after activation will be a set as a variable of the simulation.

Initial conditions

Two of many initial conditions to generate a spiral wave are shown in figure 16. Both conditions generate the same kind of spiral wave shown in figure 5.

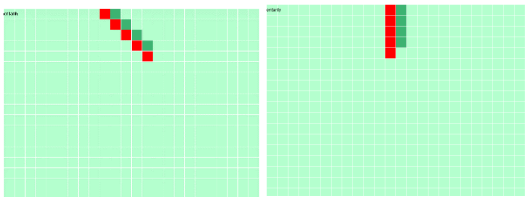


Figure 16: Two different initial conditions to generate a spiral wave

The red cells in figure 16 are ones that will have to stay deactivated for a fixed duration after the simulation starts while the dark green ones will activate for a fixed duration and the rest of the light green cells are deactivated. Note that the light green cells can activate once rule 2 is observed contrary to the red cells who have to be deactivated for a duration before being able to activate.

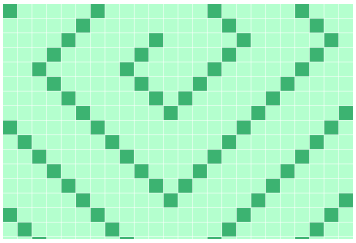


Figure 17: Resulting spiral wave from the initial conditions given in figure 16

Human Spiral Wave

To simulate each cell as a human being we took the following assumptions:

- Time is needed to react, and this time varies according to individual.
- There will be an error in determining the instructed duration to stay activated once activated
- There will also be an error in determining the instructed duration to stay deactivated once deactivated

For simplicity, we will define the duration to stay activated once activated as activation duration and similarly for deactivation as deactivation duration. Note that due to time constrains, only the latter 2 assumptions were implemented which means each cell takes exactly 0.1s to react (time step used is 0.1s). The simulation ran with a 0.5 ± 0.1s activation duration with a 1.0 ± 0.1s deactivation duration and the results are shown in figure 18.

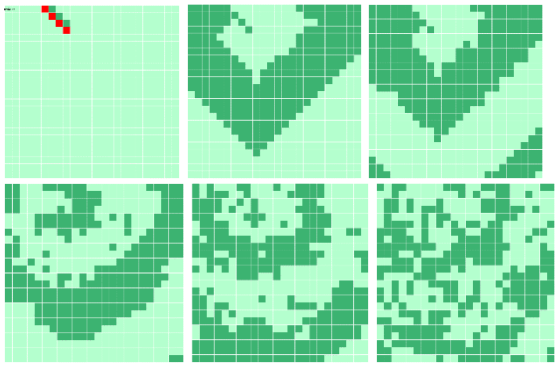


Figure 18: Snapshot of the various iterations of the human wave simulation. 0^{th} (Top left), 19^{th}, 108^{th} (top right), 557^{th} (bottom left), 1362^{nd} and 2233^{rd} iteration.

The simulation result has successfully simulated a circular spiral wave after a number of iterations. Unfortunately it is still far from reality[2] shown in figure 19, which the wave eventually dies off after approximately 26s.

One of the assumptions made to model over the human wave was not implemented which we

believe is a significant modelling factor. It is observed that the simulated wave is spiralling faster than the real observation which we believe implementing a varying reaction time for each cell will significantly slow it down to more accurately model the human wave.

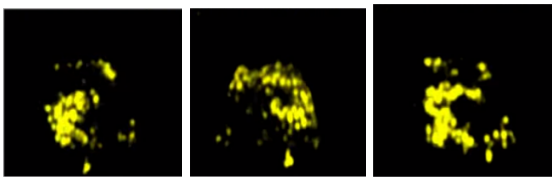


Figure 19: Snapshot of a video clip showing a real human spiral wave[3]

Belousov-Zhabotinsky Reaction

Remarkably, running the simulation on a 128×128 grid with an exact activation duration of 0.2s, a deactivation duration of 1.0 ± 0.1s with an exact reaction time of 0.2s (time step used is 0.2s) generated a pattern that starkly resembles the spiral waves generated in the Belousov-Zhabotinsky reaction (see figure 20 and 21).

Figure 20: Spiral waves observed in Belousov-Zhabotinsky Reaction[4]

The simulation (see figure 9) started off with a single spiral wave followed with 2 and multiple spiral waves were generated. Although the simulated spiral waves look less rounded than reality, it might be due to the low resolution used (grid size). Varying the parameters accordingly might yield a pattern closer to what is observed in figure 8.

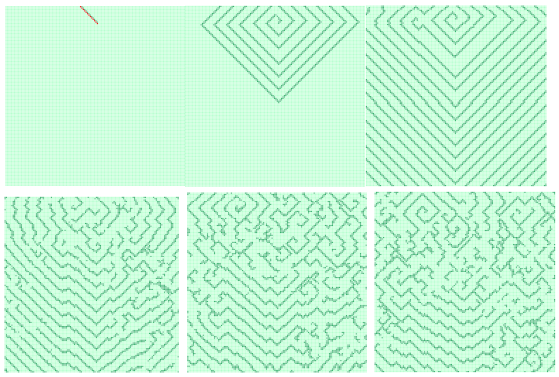


Figure 21: Snapshots of the simulation on the 0th, 59th, 960th, 1962nd, 2901st, and the 3259th iteration.

5. ELEMENTARY CELLULAR AUTOMATA

Elementary cellular automata are 1 dimensional cellular automata. They evolve generation by generation following a rule set such that each generation is a function of the previous generation. Each generation is an array of cells, with an on/true/alive state and an off/false/dead state.

While they have been studied in their primitive form since 1940s, the bulk of research on elementary cellular automata was done by Stephen Wolfram. He started studying them independently in the 1980s and published his magnum-opus, "A New Kind of Science", a 1280 page treatise on cellular automata in 2002. Elementary cellular automata are very powerful and useful for modelling systems. Some cellular automata are also capable of producing Turing complete machines (universal computers) with the right initial conditions. In this paper, we will explore one such application, generating random numbers using Rule 30. Even Wolfram's Mathematica uses Rule 30 to generate random numbers.[5]

5.1 Rules

Since elementary cellular automata are one dimensional, they can be represented as arrays of bits. As stated previously, the number of cells can be infinite however the resources behind this simulation are finite the field studied limited to the resolution of the system running it. We define the neighborhood of each cell as the cell itself and its two adjacent neighbors: one to the left and one to the right. The neighborhood of the first and the last cells of each generation is dependent on the boundary condition (discussed in the next section).
Now the state of each cell is determined the state of its neighborhood in the previous generation. We can set the initial generation (called generation 0) arbitrarily or however we want. The state of each cell in subsequent generations is determined by the formula:

$$\boldsymbol{Cell\ state\ at\ gen\ n} = \boldsymbol{f(Cell\ neighborhood\ at\ gen\ (n-1))}$$

Equation 1 Determining cell state at gen n

Since each neighborhood consists of three bits, $2^3 = 8$ unique neighborhoods are possible. A rule set is an 8 bit array that maps the new cell state to each unique neighborhood. Rules are named by converting the 8 bit array into decimal. So Rule 30 corresponds to 0b00011110.

In our simulation, we used this algorithm to generate cellular automata:

1. Define ruleset as an array of 8 bits, eg {0, 1, 0, 1, 1, 0, 1, 0} for Rule 90.
2. Set the generation 0 array. Default is 1 living cell at the center, eg {0, 0, 1, 0, 0}
3. Form a 3 bit binary number from the neighborhood of each cell in the current generation (subject to boundary conditions), eg {0, 1, 0} for the center cell.
4. Set the decimal equivalent of that 3 bit number to a variable "x" ("x" will be a number from 0 to 7), eg "x" = 2
5. Using "x" as the index, set the state of that cell as the bit at the $(x+1)^{th}$ position in the ruleset array. So new gen center cell = ruleset[x] = ruleset[2] = 0.

5.2 Rules and Classification

(Note: All automatons on this section(5.2) are of size 33, i.e the size of the array is 33.)

Since there are $2^8 = 256$ unique combinations of 8 bit arrays, there are 256 rules, numbered from Rule 0 to Rule 255. These rules are functions which map the new cell state to each unique neighborhood. In the program, this parameter can be set by typing a rule number in the ruleset text field. If the rule number is out of the 0-255 range, only the first 8 bits in the binary representation of the rules will be used.

Wolfram[6] classified Rules of Cellular Automata into 4 categories:

1. Class 1: Evolution leads to a stable state. All cells reach an equilibrium. Examples: Rule 222, Rule 0 etc.

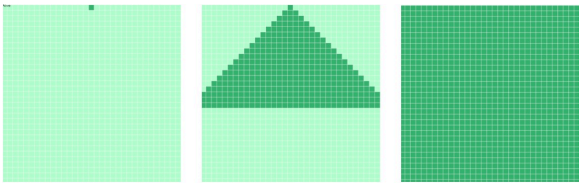


Figure 21 A class 1 automaton (Rule 222) at generations 0, 16 and 100 respectively

2. Class 2: Evolution leads to a set of separated simple stable or periodic structures.
   All cells oscillate among 0 and 1. Example: Rule 190

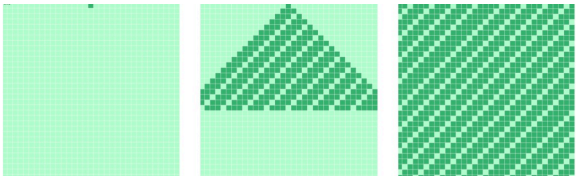


Figure 22 A class 2 automaton (Rule 190) at generations 0, 16 and 100 respectively

3. Class 3: Evolution leads to a chaotic pattern. These are random and chaotic. The state of the cells cannot be determined without numerical simulation. Example: Rule 30

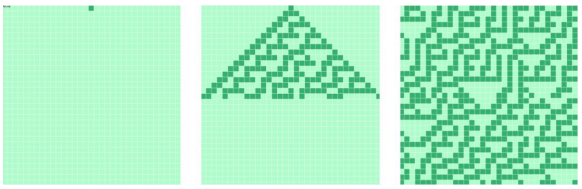


Figure 23 A class 3 automaton (Rule 30) at generations 0, 16 and 100 respectively

4. Class 4: Evolution leads to complex localized structures, sometimes long-lived. These can be thought of as a mix between class 2 and class 3. One can find repetitive, oscillating patterns inside the CA, but where and when these patterns appear is unpredictable and seemingly random. Example: Rule 110

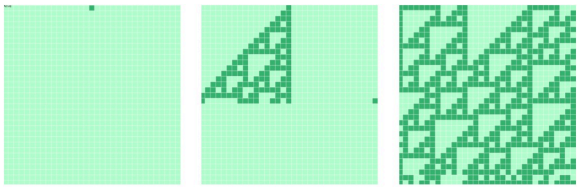


Figure 24 A class 4 automaton (Rule 110) at generations 0, 16 and 100 respectively

5.3 Parameters

We designed the simulator such that the following parameters can be varied.

5.3.1 Boundary Conditions

Boundary conditions affect the first and the last cells of the array. Boundary conditions determine what neighborhood will be used for those cells. We can use two types of boundary conditions in the program:

1. Circular Boundary: In this case, the array is treated as a ring. So, for the first cell, the neighborhood would consist of the last cell in that generation, the cell itself, and the second cell. For the last cell, the neighborhood consists of the cell to the right of the last cell, the last cell itself and the first cell.
2. Fixed Boundary: In this case, the first and the last cell are set to a constant off /0/false state. The ruleset is only applied to cells between the first and the last cell.

Changing the boundary conditions often lead to a drastic changes in the automaton as it evolved. However, they only affect the outcome when state changes occur at the boundaries.

Example : When fixed boundary conditions were applied to Rule 30, it reached an equilibrium state in 200(dependent on the size of array) and started behaving like a class 1 automaton. When a circular boundary condition was applied, it started behaving like a chaotic class 3 automaton again.

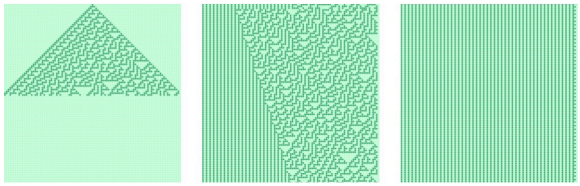


Figure 25 A Size 100 Rule 30 with fixed boundaries at generations 50, 100 and 200 respectively

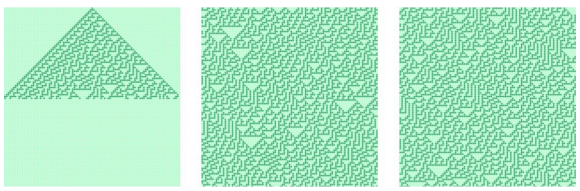


Figure 26 A Size 100 Rule 30 with circular boundaries at generations 50, 100 and 200 respectively

We think this happens because Rule 30 is chaotic in the middle and has a fixed pattern at its ends. This becomes obvious when we view multiple generations with respect to time.

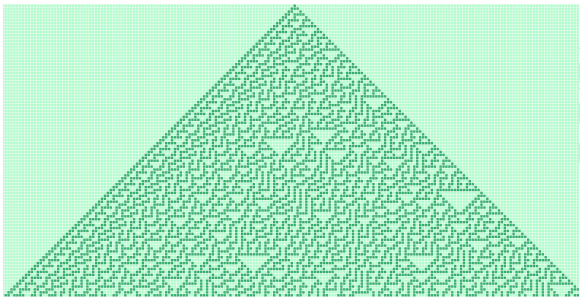


Figure 27 A Size 200 Rule 30 at generation 100

From figure 27, we can say that the left part of the automaton is shows a stable pattern which translates leftwards with every generation. However, once it reaches the left boundary, it starts replicating itself when the boundaries are fixed and gradually takes over the entire automaton (see figure 25). When the boundaries are fixed, the program only fills the automaton with the leftmost cells. When the boundaries are circular, the program fills the automaton with central cells irrespective of the size of that generation.

5.3.2 Initial State

Changes in the initial state, or generation 0 also lead to big changes in the automaton. Our program lets the user change the initial state by clicking on cells to toggle their states. The default generation 0 is that all cells will be dead except the middle cell.

Example: Rule 90 is a beautiful automaton that produces the Sierpinsky Triangle. However, when a random initial condition is applied, it makes different pattern, which looks like intereference between multiple Sierpinsky triangles.

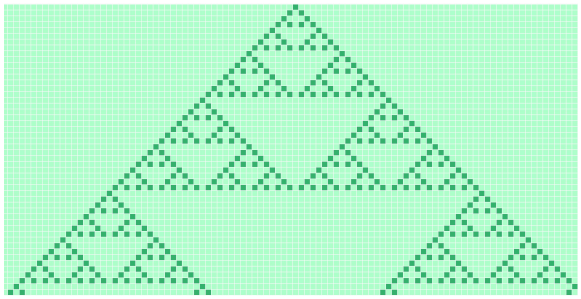


Figure 28 A Size 100 Rule 90 at generation 50 with default initial conditions

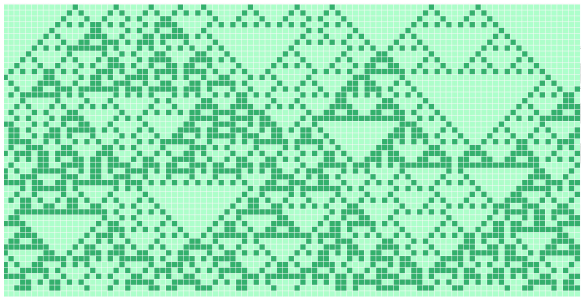


Figure 29 A Size 100 Rule 90 at generation 50 with arbitrary initial conditions

5.3.3 Size of the array

The size can be changed by entering the required size in the “col:” text field. While running the program with different conditions, we noticed that in some cases, the output was not as expected. We think these are special cases that occur because of resolution limits.

Example: Disappearing Sierpinsky Triangle at size 64. When we ran rule 90 at size 64, all the cells died after 32 generations. When the dead automaton continued evolving, the pattern on our 64x64 board disappeared after 96 iterations.

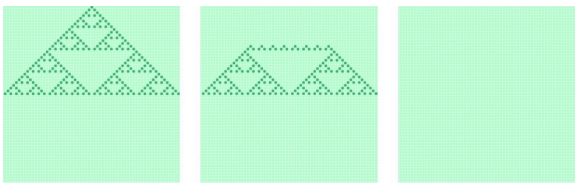


Figure 30 A Size 64 Rule 30 on a 64x64 board with circular boundaries at generations 32, 77 and 96 respectively

5.4 Applications

Elementary cellular automata have widespread applications. Applications include fractal generation, chaotic system simulation using class 3 automata: traffic systems modelling[7], password generation, computing machines and other complex machines using class 4 automata.

5.4.1 Custom Random Number Generator using Rule 30

After we found out that Mathematica uses Rule 30 to generate random numbers, we wrote our own Random Number generator using Rule 30 that uses the central column of a size 10 rule 30 automata.

Algorithm for generator:

1. Take a seed value from the user.
2. Run the algorithm in Section 5.1 with an array of size 10 for (seed + 10) iterations.
3. From iterations seed to seed + 10, save the value of the central element in a 10 bit array.
4. Convert that array to floating point decimal.
5. That is your random number between 0 and 1024.

We implemented this in Java (RandGen.java) and Arduino (RandGenArduino.ino).

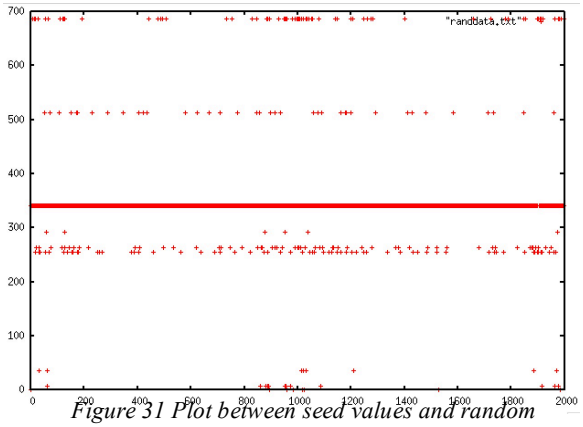


Figure 31 Plot between seed values and random value outputs for seed values 0 to 2000.

However, this did not produce truly random numbers, it produced some discrete values with disproportionate probabilities. Given the limited time frame, we could only experiment with this and a few other algorithms.
We will look into this matter with more resources.

6. LIMITATIONS

The biggest limitation we had for this study was the time constraint. Cellular Automata is a vast topic with a lot of breadth as well as depth. We could only scratch the surface of cellular automata in the time allotted.

From a programming point of view, the biggest problem was managing the screen real estate in our very visual app. Because of limited screen resolution, we could only allow low resolution (size 200 by 200) cellular automatons.

Our random number generator also didn't work as expected. We will look into that when time permits.

We felt somewhat limited when reading other papers on cellular automata because we of our lack of understanding of formal logic theory.

7. CONCLUSION

The study has covered the basic principles behind elementary CA, The Game of Life and spiral waves. And also implemented elementary CA as a random number generator using rule 30 and employed the spiral wave CA to model patterns observed in the Belousov-Zhabotinsky reaction and the human spiral wave; by mainly using the Java applet. However these are only a few of the many applications of cellular automata which include but not limited to applications in computer processors, cryptography, error corrections, solving of PDEs, traffic patterns.

It is remarkable that so much was born from a model with a few simple rules – from phenomenon observed in nature, to applications like random number generator to problem solving in mathematics.

8. REFERENCES

[1] Stephen, W. (1983). Cellular Automata. *Los Alamos Science, 9*, 2-21.

[2]Sandstede, B., & Scheel, A. (n.d.). Absolute versus convective instability of spiral waves. Physical Review E, 7708-7714.

[3]Largest human spiral wave ever made [Motion picture]. (2014). United States: Georgia Institute of Technology, School of Physics.

[4]Belousov–Zhabotinsky reaction - Turing - Cargo example design. (n.d.). Retrieved April 20, 2015, from http://cargocollective.com/turing/Belousov-Zhabotinsky-reaction

[5] Weisstein, Eric W. "Rule 30." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Rule30.html

[6] Wolfram, S. (1984, January 1). UNIVERSALITY AND COMPLEXITY IN CELLULAR AUTOMATA. Retrieved April 25, 2015, from http://new.math.uiuc.edu/im2008/dakkak/papers/files/wolfram.universityofca.pdf

[7] David, R., & Gerhenson, C. (n.d.). A Model of City Traffic Based on Elementary Cellular Automata. Retrieved April 25, 2015, from http://www.complex-systems.com/pdf/19-4-1.pdf