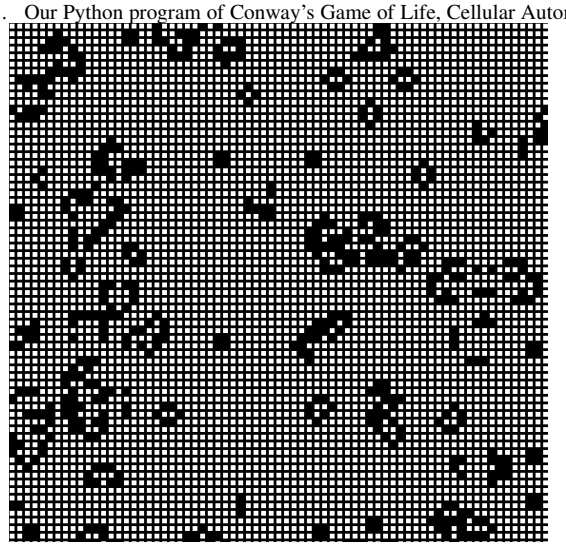
A Brief History and Symposium On Cellular Automaton

Joseph Cox [1], Kelvin S. P. Dong[1], Bishop Clark [1]and Mauzor Ilonzo [1]

***Abstract*— Cellular Automaton was conceived in the 1940s by John Von Neumann and Stanislaw Ulamwhile as colleagues at Los Alamos National Laboratory. They defined life as an organism or a creation that can be defined by a Turing Machine, which had recently been invented by Allan Turing in 1936. Although Cellular Automaton, also referred to as C.A., was studied throughout the 1950s and 1960s, it was not until 1970 when a British mathematician by the name of John Horton Conway devised "The Game of Life", which was published on October 1970 in Scientific America. The Game of Life is a two-dimensional Cellular Automata that caused a substantial amount of interest in the subject. In the 1980s, Stephen Wolfram started a methodical study of Cellular Automata which culminated in 2002 with the publication of "A New Kind of Science", which is an anthology of Cellular Automata consisting of over a thousand pages that attempts to demonstrate how Cellular Automata could be used across many different applications in many scientific fields. Our group project is to deliver a survey of Cellular Automata, its history and how it is used in modern science. In this paper we will provide a summary of Von Nuemann's original C.A. and Self Replicating Machine, describe Conway's Game of Life, explore Wolfram's Elementary Cellular Automata and his publication "A New Kind of Science", describe Boids Flocking Algorithm and its relevance to Cellular Automata, and provide a brief summary of the future of hardware utilizing Quantum Cellular Automata.**

Fig. 1. Our Python program of Conway's Game of Life, Cellular Automata



I. An Introduction To C.A.

A Cellular Automata is a discrete structure utilized across a manifold of scientific fields such as biology, physics, chemistry, and mathematics. Cellular Automata consists of a group of cell spaces which makes a grid. Each cell starts at a starting state S_0, and transitions to another at a time t, from a list of finite states. The grid can be an infinite number of dimensions and size. Each cell transitions to a new state based on the state of each cell surrounding it called the cell's "neighborhood". At time t, each cell is assigned to a state by its transition function $\delta(t)$. At time $t+1$, each cell changes to a different state based on the state of its relative neighbors and a fixed set of rules specified by the transition function δ. Depending on the rules and the neighborhood of each cell, the entire grid changes its formation. Each grid at a time t is called a generation. As time progresses, each generation formulates a system that simulates cellular life in various contexts.

II. Von Nuemann's C.A. and Conway's Game of Life

A. Introduction to Von Nuemann's C.A.

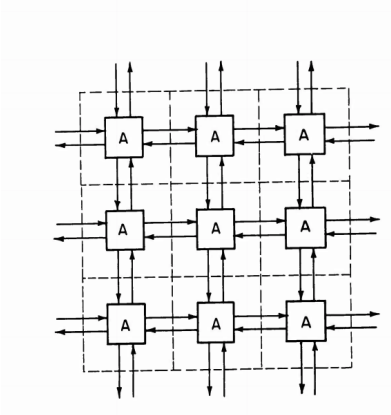
Conway's game of life was inspired by John Von Neumann and Stanislaw Ulamwhile. A detailed conceptual proposal for a physical non-biological self-replicating system was first put forward by John Von Neumann in lectures delivered in 1948 and 1949, when he proposed a kinematic self-reproducing automaton model as a thought experiment. Von Neumann described a model of a self-reproducing machine in a paper called **The General and Logical Theory of Automata**. Von Neumann struggled to create a self replicating system mathematically and could not fully solve the problem. Stanislaw Ulamwhile suggested using a grid of cells to solve the problem. Utilizing this suggestion, Von Neumann formulated a mathematical machine that was fully capable of self-reproduction. Von Neumann eventually published the concept of Cellular Automaton and his self-replication machine in his book called **Theory of Self-Reproducing Automata**, which was completed and published after his death by Arthur Walter Burks in 1966 [1].

B. John Von Nuemann's C.A. The Self-Replicating Machine

John Von Nuemann thought that the future of computing would be to advance science with models that could produce natural systems found in physics, chemistry, and biology by utilizing formal logic. This was stated by Arthur Walter Burks in 1966, "The late John Von Neumann once pointed out that, in the past, science has dealt mainly with problems of energy, power, force and motion. He predicted that in the future science would be much more concerned with problems of control, programming, information processing, communication, organization, and systems. General purpose digital computers provide an excellent opportunity for studies of this kind, and Von Neumann started a theory of automata based on them [1]." Von Nuemann prefaced his concept of Cellular

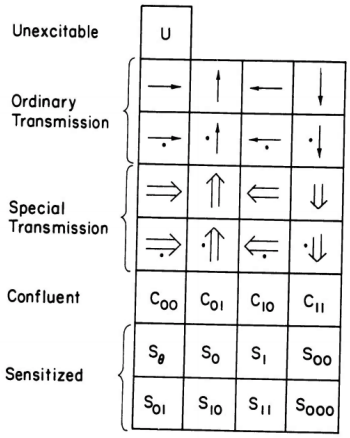
Automata by proposing a problem that he eventually solved which is: "What kind of logical organization is sufficient for an automaton to control itself in such a manner that it reproduces itself [1]?" Von Nuemann eventually explained his notions and described a cellular system in his essays. According to Von Nuemann's work, a cellular system constitutes a grid called a "space" where automaton events occur. One can create basic rules of governance for the system to operate and control it over discrete time intervals. As stated by Von Nuemann and Arthur Burks in **Von Neumann's Self-Reproducing Automata** "A cellular automaton system (or "cellular system," for short) is specified by giving a finite list of states for each cell, a distinguished state (called the "blank state"), and a rule which gives the state of a cell at time $t + 1$ as a function of its own state and the states of its immediate neighbors at time t. We will call the list of states for a cell together with the rule governing the state transition of the cell a transition function. The distinguished state corresponds to the blank state of a square of a computing machine tape, and it is required of a transition function that if a cell and its neighbors are in the blank state at time t, the cell is in the blank state at time $t + 1$. Thus a cellular automaton system consists of a cellular space and a transition function defined over that space. A cellular automaton state is specified by a finite list of cells together with the cell state assigned to each, it being understood that all other cells are in the blank state." This is illustrated by Figure 1 which is Von Nuemann's original drawing depicting his Cellular Automata [1].

Fig. 2. Von Nuemann's original drawing to describe the cell space of his C.A. [1].



Von Neumann's cellular system is a complex system and is fundamentally a finite state machine consisting of 29 unique states. Von Neumann and Arthur Burk's original work is 113 pages long and for the purposes of this paper we will only show concepts from sections one through three, and for the sake of brevity only summarize the work and give a simple demonstration. Essentially, Von Nuemann's replicating machine consists of a set of rules governing the system and a notation that utilizes orthogonal, directional arrows and **RGB** 3-tuple that describes colors. The 29 states are categorized into five different classes as seen in Figure 3 [1]. A brief explanation of the notation is provided as

Fig. 3. Von Nuemann's original drawing to describe the 29 states [1].



follows:

1) **U** - the ground state also referred to as the "empty" state.
2) **The Transition or Sensitised States** - consisting of 8 sub-states:
   - S- newly sensitised
   - S_0- sensitised, having received no input for one cycle
   - S_{00}- sensitised, having received no input for two cycles
   - S_{000}- sensitised, having received no input for three cycles
   - S_{01}- sensitised, having received no input for one cycle and then an input for one cycle
   - S_1 - sensitised, having received an input for one cycle
   - S_{10} - sensitised, having received an input for one cycle and then no input for one cycles
   - S_{11}- sensitised, having received an input for two cycles
3) **The Confluent States** - Has 4 states of excitation, and the following rules are applied:
   (a) Confluent states do not pass data between each other.
   (b) Confluent states take input from one or more ordinary transmission states, and deliver output to transmission states, ordinary and special, that are not directed toward the confluent state.
   (c) Data are not transmitted against the transmission state direction property.
   (d) Data held by a confluent state is lost if that state has no adjacent transmission state that is also not pointed at the confluent state.
   (e) Confluent-state cells are used as "bridges" from transmission lines of ordinary- to special-transmission state cells.
   (f) The confluent state applies the AND operator to inputs, only "saving" an excited input if all potential inputs are excited simultaneously.

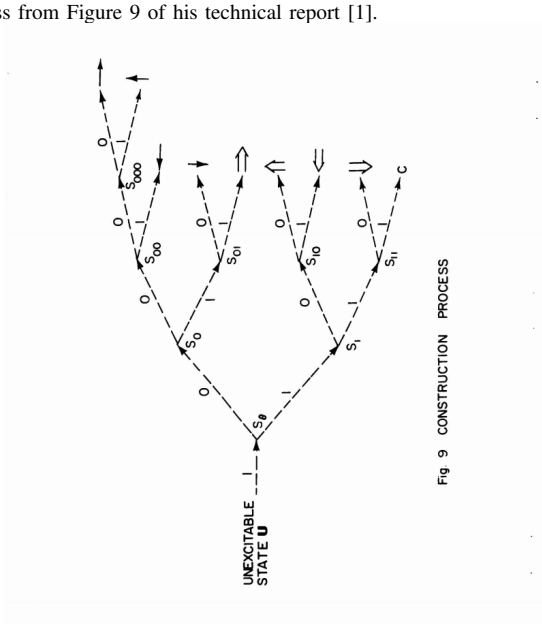
An initially quiescent automaton is defined as a finite

area of the cellular structure where every cell is in one of the ten quiescent states. Each transmission element has a quiescent (passive, no-pulse, zero) state and an excited (active,"pulse," one) state, as follows:

- C_{00} - quiescent and will also be quiescent next cycle
- C_{01} - next-excited now quiescent, but will be excited next cycle
- C_{10} - excited but will be quiescent next cycle
- C_{11} - excited next-excited (currently excited and will be excited next cycle)

4) **The Ordinary Transmission States**:
   - North-directed (excited and quiescent) denoted:↑
   - South-directed (excited and quiescent) denoted:↓
   - West-directed (excited and quiescent) denoted:←
   - East-directed (excited and quiescent) denoted:→
5) **The Special Transmission States** - states in 4 directions, excited or quiescent, making 8 states:
   - North-directed (excited and quiescent) denoted:⇑
   - South-directed (excited and quiescent) denoted:⇓
   - West-directed (excited and quiescent) denoted:⇐
   - East-directed (excited and quiescent) denoted:⇒
6) **Construction Rules:** The space of the cellular automaton, starts at the ground state "blank" called U. When input is taken, the excitation from a "neighborhood" in the ground state becomes "sensitized," transitioning through a series of states before "waiting/resting" at a quiescent transmission, also called the confluent state. The destination state of the cell is determined by the sequence of input signals. Therefore, the transition/sensitized states are visualized in Figure 4 [1] by a tree called the construction process beginning at the ground-state and ending at each of the quiescent transmissions and confluent states.

Fig. 4. Von Nuemann's original drawing to describe the construction process from Figure 9 of his technical report [1].



7) **Destruction Rules:**
   (a) An input into a confluent-state cell from a special-transmission state cell will result in the confluent state cell being reduced to the ground state.
   (b) An input into an ordinary transmission-state cell from a special-transmission state cell will result in the ordinary-transmission state cell being reduced to the ground state.
   (c) An input into a special transmission-state cell from an ordinary-transmission state cell will result in the special-transmission state cell being reduced to the ground state.

The processes of Construction and Destruction are represented in a cellular space as a C.A. shown in Figure 5 and Figure 6 [1]. Although these process are trivial compared to the full Von Nuemann's self-replication machine, we will provide and explain a simple example of a Von Nuemann's C.A. to illustrate how these rules produce a replication machine. Von Nuemann's original illustrations are as follows:

Fig. 5. Von Nuemann's original drawing to describe the construction process as a C.A. [1].

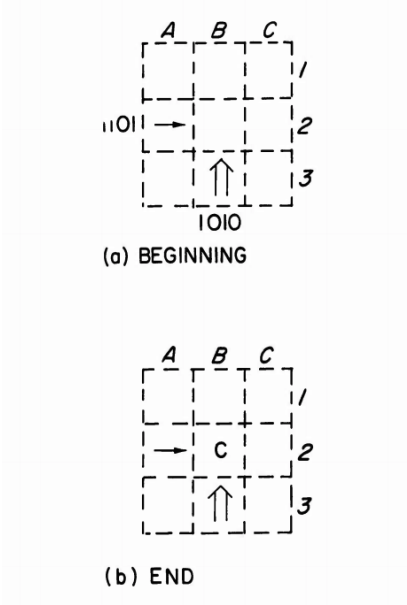
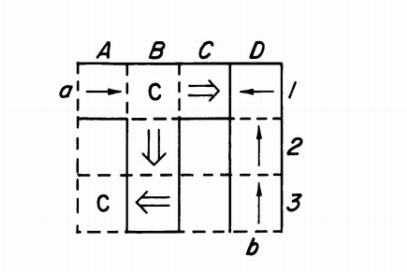


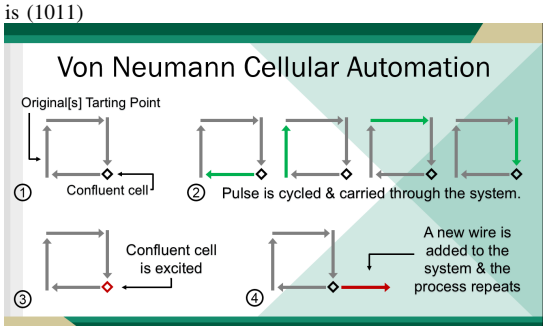
Fig. 6. Von Nuemann's original drawing to describe the destruction process [1].



To illustrate this process in action we provided a visual that shows a simple circuit that adds to itself over

a period of time t. The transition function $\delta(t)$ is illustrated by the coloring of the arrows. As the pulse is carried through the system it reaches the confluence state and builds upon itself. Figure 7 demonstrates this process:

Fig. 7. A simple configuration in von Neumann's Cellular Automaton. A pulse is passed repeatedly around the black wire loop, using excited and quiescent ordinary transmission states. A confluent cell replicates the signal onto a length of red wire consisting of special transmission states. The signal passes down this wire and constructs a new cell at the end. This particular signal is (1011)



C. Introduction to Conway's adaptation The Game of Life

As discussed earlier John Conway's inspiration for his Game of Life originated with John Von Nuemann's research. Many believe the idea originated with a thought experiment of trying to model complex reality with as simple logic as possible. In other words, what is the simplest set of logic that can produce a model for complex life? John Conway began doing experiments in 1968 with a mixture of different 2-Dimensional Cellular Automaton rules and configurations for the neighborhood laws that govern the system. Conway's initial vision was to define an interesting and unpredictable Cellular Automata, that could represent life as a whole, not just a reproductive process. Interestingly, much of this work was for fun and was a hobby, not for thorough scientific research. A scientific investigation did not become available until Wolfram investigated it for his own research in particle physics. For many years, an open problem was to prove that Conway's Game of Life satisfied both conditions that Von Nuemann proposed. According to Wolfram Science, "An immense amount of effort was spent finding special initial conditions that give particular forms of repetitive or other behavior, but virtually no systematic scientific work was done (perhaps in part because even Conway treated the system largely as a recreation), and almost without exception only the very specific rules of Life were ever investigated [2]."

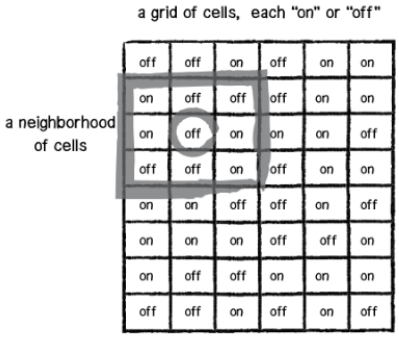
D. Conway's Transition function $\delta(t)$

As mentioned above, Conway's Game of Life is a infinite 2-Dimensional Cellular Automata. The space is defined as a orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each discrete moment of time called a "step" the transition function $\delta(t)$ updates each cell defined by the following criteria:

1) A live cell with fewer than two live neighbors dies. (Under population Rule)
2) A live cell with two or three live neighbors lives on to the next generation. (Still life Rule)
3) A live cell with more than three live neighbors dies by overpopulation. (Over population rule)
4) A dead cell with three live neighbors becomes a live cell by reproduction. (Birth Rule)

Pictorially this is shown below in Figure 8:

Fig. 8. 8-Cell Neighborhood [3]



The first generation can be conceived randomly, after each step in time the transition function $\delta(t+1)$ updates all the cells by applying the above rules simultaneously, which then creates a new generation. The below algorithm is a snippet of python code from our version of Conway's Game of Life defined on a finite set of cells that wrap around on the edges utilizing the modulus of the grids size. The grid is constructed as a square meaning all the edges are congruent in length, and the size N represents the length of the columns and rows of the array simultaneously. The transition function is defined in Figure 9.

Fig. 9. Our algorithm for the transition function in python

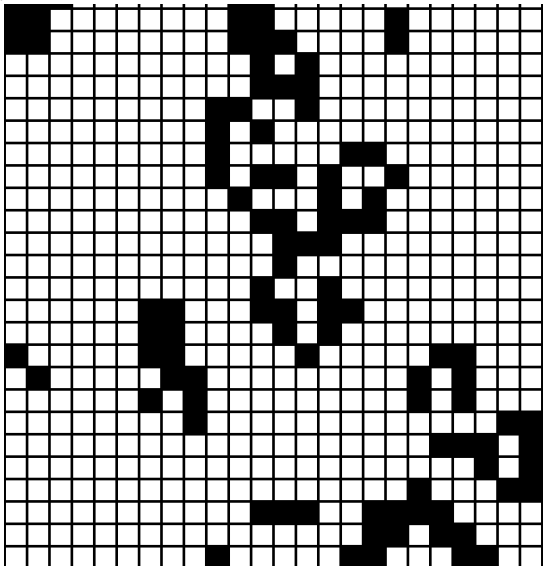
```
# Input N:size of the grid wraps around
# Input cellSpace: An array implementation of a cellular space
def transition (cellSpace,N):
    newcellSpace =cellSpace.copy()
    for i in range(N):
        for j in range(N):

            # compute 8-neghbor sum
            # Allowing for wrap around
            total =  (cellSpace[i][(j-1)%N]+ cellSpace[i][(j+1)%N] +
            cellSpace[(i-1)%N] [j] + cellSpace[(i+1)%N][j] +
            cellSpace[(i-1)%N] [(j-1)%N] + cellSpace[(i-1)%N] [(j+1)%N] +
            cellSpace[(i+1)%N] [(j-1)%N]+ cellSpace[(i+1)%N][(j+1)%N])

            # apply Conway's rules
            if cellSpace[i] [j]  == 1:
                if (total < 2) or (total > 3):
                    newcellSpace[i] [j] = 0
            else:
                if total == 3:
                    newcellSpace[i] [j] = 1

    return newcellSpace
```

Figure 10 provides a close up view of our Game of life program. The white space represents the dead cells, but can also be thought of as empty space. The black filled cells are the living cells that produce fascinating patterns and movements as the system moves forward in time.

E. Pattern Classifications in The Game of Life

Consequently, the above algorithm creates a complex model for cellular life. There are three classes that describe

Fig. 10. Close Up View of Conway's Game of life provided by our Python Program

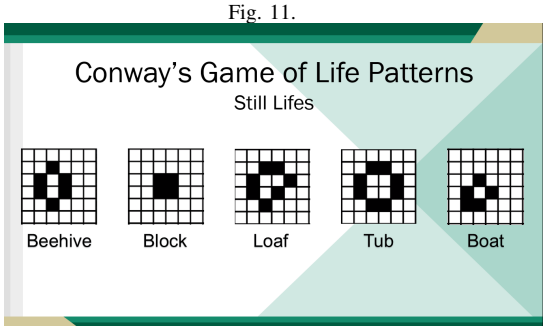


all the patterns in this model. The classes are as follows:

1) **Still Life** - Is a pattern that does not change from one generation to the next. The transition function δ at time t to $t+1$ does not change the pattern
2) **Oscillators** - Is a pattern that repeats itself over a period of time t. Some times these patterns are called blinkers and can be calculated as 1 divided by the time it takes to return to its initial configuration: $1/t_{blinkingperiod}$
3) **Spaceships** - Also called gliders are a pattern that replicates itself of a period of time t. After each transition it reappears in a different location in space. It is essentially a kinematic animation and its speed can be calculated as number of cells that the pattern traverses during its period divided by the period length. $(NumberofCellsTraversed)/t_{finalcell} - t_{initialcell}$

The following images Figure 11, Figure 12, and Figure 13 are from our python generated Game of Life that show the pattern classifications:

Fig. 11.



F. Game of Life Significance

In short the Game of Life was the first descriptor of autonomous agents. An autonomous agent can be thought of as a object that simulates unpredictable behavior which

Fig. 12.

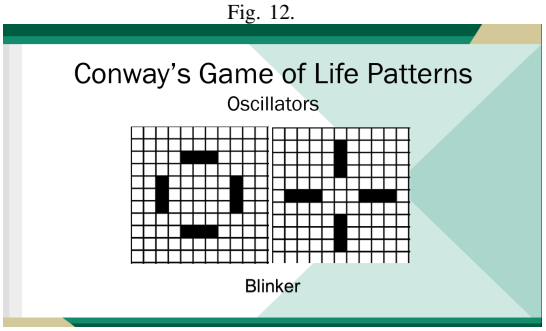
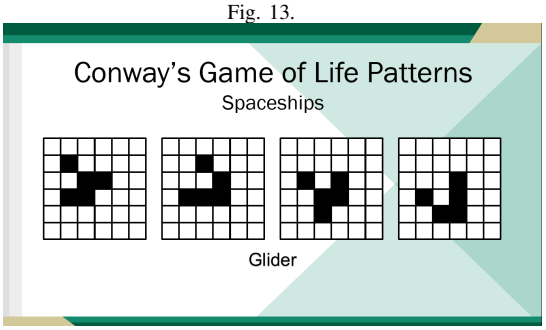


Fig. 13.



will be discussed in section IV of the paper. It models free will, emergence, and self-organization of an agency or group, which was later refined by Craig Reynolds and his Boids Algorithm. The Game of Life also provides symmetric patterns that were later heavily investigated by Stephan Wolfram.

III. Elementary Cellular Automata

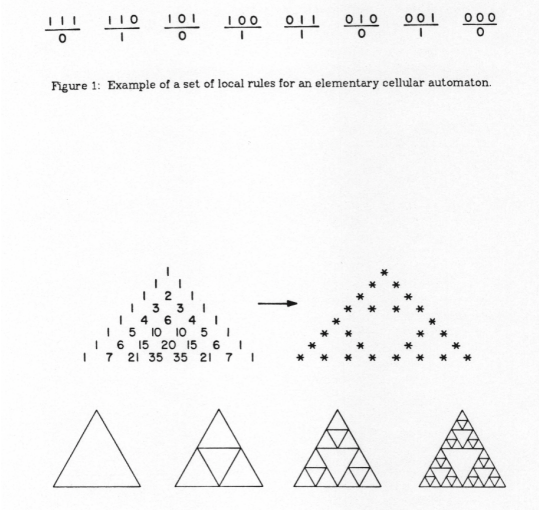
A. Introduction to Wolframs's Research

In the early 1980s Stephen Wolfram produced a series of papers that provided a scientific investigation of C.A. The research was partially for fun and partially for his own work in particle physics. Wolfram heavily invested his time into developing the most fundamental Cellular Automata known as the Elementary Cellular Automata, also referred to as E.C.A. Elementary Cellular Automata has been used in graphical renderings. These renderings have been studied in a neighboring field of mathematics referred to as Fractals. The most recent famous example of Fractals were used to create snow flakes in Disney's animation Frozen. Wolfram's Research produced the most in-depth publication ever created on Cellular Automata titled **A New Kind of Science**. The book covers hundreds of different E.C.As and their potential uses in science. This publication is over a thousand pages long. In this section we will provide a brief explanation for his work in E.C.A.

B. Wolfram's Elementary Automata

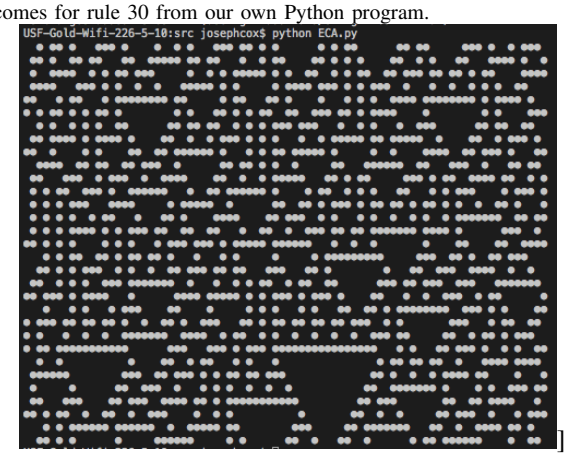
Elementary Cellular Automata is the simplest form of Cellular Automata. E.C.A. is modelled by a 1-dimensional grid with each cell having one of two states, 1 or 0. Wolfram's model of Generational Elementary Automata states that there

Fig. 14. F
ollowing figure describes neighborhood rules and generational evolution. [3]



are eight possible configurations of a neighborhood represented as binary numbers 0(000) to 7(111). This is shown in Figure 14. The possible states for a cell and its neighborhood produce 2^8(256) total Cellular Automata. Every Wolfram elementary Cellular Automata is considered a rule-set [4], [5]. A rule-set is the outcome of the continuing generations of a Cellular Automata. For example, the rule-set: 00011110 (rule 30) defines the outcome of each neighborhood. The first 0 derives from the neighborhood 000, the second 0 derives from the neighborhood 001, and so on. The cell in the next generation can be described by the following function: Cell state$_t$ = $f(Cell_{neighborhood(t-1)})$. Which reads as a function of the previous generation. An example output for the function created by our python code is shown in Figure 15.

Fig. 15. This figure shows the first 30 (amount of rows) generations of outcomes for rule 30 from our own Python program.



C. Behaviors of Elementary Cellular Automata

Each of the 256 Elementary Cellular Automata graphs can be described by one of four behaviors.

1) Uniformity - Every neighborhood produces the same results.
2) Repetition - Alternating neighborhood outputs.
3) Random - Outputs that have no perceivable pattern.
4) Complexity - Outputs that have patterns at certain places, but these places appear random.

Examples of uniform behavior include Rule 0 and Rule 255. Each neighborhood outputs all 0 or all 1. No other rule is as completely uniform as these rules. There exist rules such as Rule 170 and Rule 85 that model repetitive output. Neighborhood outputs produce 0 to 1 or 1 to 0 repeatedly. Random behavior can be seen in several E.C.A., for example, Rule 90 and Rule 30 which have no discernible pattern. Wolfram introduced Rule 30 in 1983. This Rule is unique because it can model the chaotic or random patterns on the shell of a Corus Textile Snail. Rule 30 has also been used as a Random Number Generator in computing programs. Complex behavior is seen in Rule 110, where patterns are located in random places. This rule is considered Turing Complete; any computer program is able to be expressed with this automaton.

D. Modern Research

Numerous uses of Elementary Cellular Automata are being used to model systems. For example, Rule 184 can be used to model traffic flow in a single lane highway; it can be used to predict waves of stop-and-go traffic. It also has the capability to solve the Majority Problem, and describe particle-systems. Due to the complexity of the Majority Problem and the fact that it is an open unsolved problem in computation theory, we will not be going into further detail on the matter. Rule 90 is used to model the logic operation, exclusive or. This means that the exclusive or of a cell's neighborhood determines the next generation of the cells state. E.C.A. also models fractals which are infinite geometric patterns. Fractals are researched extensively in E.C.A. For example, Disneys Frozen (2013) used research from UCLA on Cellular Automata to model their snows graphics [6]. E.C.A is also used to model physical reality. Consider a crystal. If 1 represents a solid and 0 represents any other state of matter, then the next generation of cells adjacent to the 1s will become solid, represented as a 1.

IV. CELLULAR AUTOMATA APPROACH IN FLOCKING

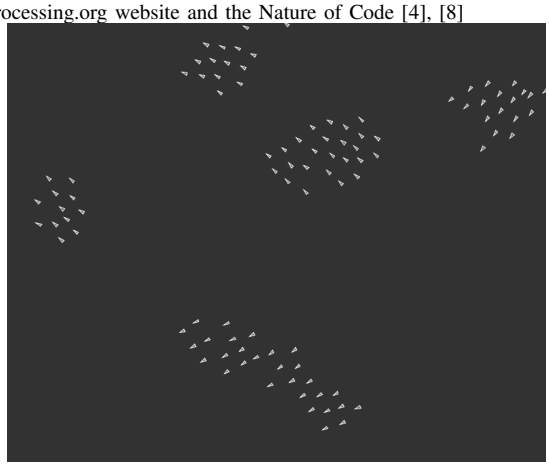
A. Introduction to Flocking

In this section, we will explore how Cellular Automata is useful for automating a large quantity of moving entities, also known as the flock behaviour. In real-life, flocking behaviour can be observed in animals such as birds flying in flocks, fish swimming in schools, and insects such as locust swarming together [7]. Travelling as a flock has many advantages. One such example would be attaining a higher level of security as being in a large crowd will make the animals less likely to be singled out by predators.

In a simulated environment, flocking is a process which allows autonomous agents to interact with their immediate

surroundings and neighbouring agents to produce complex, emergent behaviour can be very computationally expensive depending on the size of the flock [7].” Although flocking has been used for simulation for quite a while now, the computational costs that it incurs becomes relatively high as more and more entities are introduced into the simulation [7]. A figure of flocking provided by Daniel Shiffmen is provided in Figure 16 for illustrating a flocking system.

Fig. 16. Boid Flocking System developed by Daniel Shiffman, located on the processing.org website and the Nature of Code [4], [8]

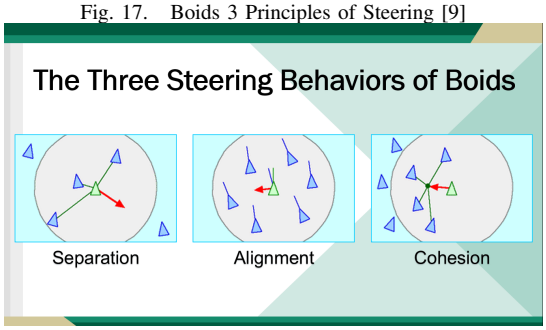


B. Boids Flocking Algorithm

One of the first flocking algorithms is called Boids Flocking Algorithm. This flocking algorithm was created by Craig Reynolds in 1986. It was designed to simulate the motions of bird flocks and fish schools [9]. The algorithm is used to illustrate how an entity would behave around the locations and speeds of its neighbours based on the following three steering behaviours [9] and are presented in Figure 17:

- Separation - Steer to avoid crowding local flock mates.
- Alignment - Steer towards the average heading of local flock mates.
- Cohesion - Steer to move toward the average position of local flock mates.

Fig. 17. Boids 3 Principles of Steering [9]



Although Boids is able to simulate the flocking behaviour of entities, it does have several performance issues. For instance, as the number of entities increases, the cost of computations grows exponentially [7]. This means that Boids is not effective in simulating large quantity of entities. Moreover, in order to take entities who are within their line of sight into consideration, the entities would have to look through every other entity [7]. This way of searching is not cost-effective.

C. Cellular Automata Based Approach

In this section, we will be discussing about how a researcher, James Shannon, managed to use the concept of Cellular Automata to implement a flocking simulation. In this approach, James Shannon stated that Alignment principle could be neglected due to the discrete, cell-based nature of movement in cellular automata [7].” The approach considers the distance between each entity and keep them separated while at the same time has directional forces which keep the flock together and propels them towards their destination [7].

Fig. 18. Cellular Automata Based Flocking by James Shannon [7]

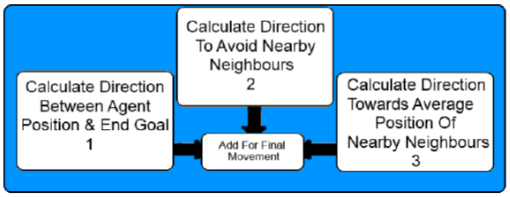
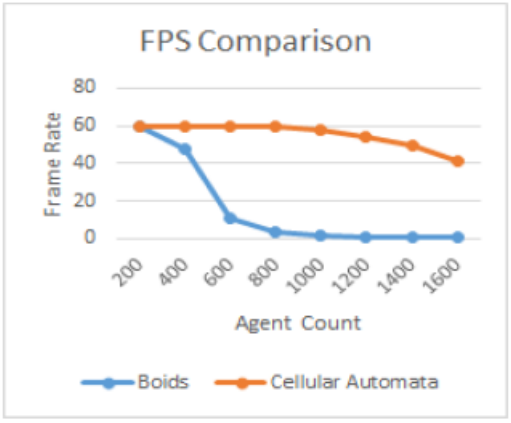


Figure 18 illustrates the steps that are applied to each entity in the simulation for every movement. In the steps, the entity may need to compute the following directional forces needed: to move towards its flocks destination, to move away from other neighbouring entities, and to stay together with its flock. Once the average of the sum of the forces found from the previous steps is computed, an angle can be computed to give the direction towards which each entity should move [7].

Fig. 19. Frames Per Second (FPS) comparison Between C.A. Approach And Boids by James Shannon [7]



As seen in Figure 19, the Cellular Automata approach was shown to perform better than the Boids Flocking Algorithm. The Cellular Automata approach was able to simulate over 1200 entities and still maintains a constant 60 frames per second, whereas the Boids approach started to slump at

around 200 entities [7]. The new approach also scaled well with additional entities as the time increase was linear while the Boids approach scaled exponentially with each additional entity.

V. QUANTUM CELLULAR AUTOMATION

A. Introduction to Quantum Cellular Automation

A growing subject of interest in modern uses of Cellular Automata is the concept of Quantum Cellular Automation (Q.C.A), also known as Quantum Dot Cellular Automata. Q.C.As are models of quantum computation that are based on Von Neumanns models of Cellular Automata. The origins of Q.C.As can be traced back to Richard Feynmans introduction of quantum computation in 1982, specifically with his archetypal quantum computer [10]. Afterwards, David Deutsch would define the Quantum Turning Machine in 1985 which formalized Feynman's concept [11]. The actual concept of Q.C.As was introduced by Gerhard Grossing and Anton Zeilinger, who coined the term and provided the first definition in 1988 [12]. After Grossing and Zeilinger introduced Q.C.As, many other people developed their own models and research on the matter. For example, John Watrous introduced his own one-dimensional Q.C.A model in 1995 [13]. As a result of the various different models that were introduced in the last two decades, there is not a single Q.C.A definition that has universal agreement but some of the Q.C.A models themselves have demonstrated that they are universal in terms of computation [14].

B. The Basics of Q.C.A Cells

The most basic Q.C.A device contains a single cell that has four quantum dots positioned in the cells corners and two mobile electrons [15]. A basic diagram of a Q.C.A cell can be found in Figure 20. These four quantum dots serve as sites that electrons may inhabit as they tunnel through the cell. A Q.C.A cell can achieve three different states depending on the how the barriers within the tunneling are raised and lowered through a local electric field [15]. The first state is the Null State which occurs when all barriers are lowered, allowing for active electrons to access any quantum dot. The other two states are achieved by raising a barrier which polarizes the cell and limits the electrons to two of the quantum dots on the cell. These two states are commonly referred to as $P = +1$, which is used to represent the binary logic 1, and $P = -1$ which is used to represent the binary logic 0. The orientation of these two polarized states are showcased in Figure 21. This basic Q.C.A device can be used to construct all of the possible components of a circuit, including the computational elements and the wires [15]. Due to the complexity of Q.C.A circuits, we will not be discussing the actual components in further detail.

C. Quantum Cellular Automation Replacing C.M.O.S

One of the reasons why Quantum Cellular Automata has attracted so much attention in modern research is its possibility of replacing Complementary Metal Oxide Semiconductors (C.M.O.S). With the continuous scaling of C.M.O.S, it will eventually reach the maximum limit of transistors that can be placed in a single semiconductor [16]. Q.C.As on the other hand do not transfer information through the use of an electrical current like C.M.O.S, but rather they transfer that information through the propagation of a polarization state [17], [18]. As a result, Q.C.As have been described as a transistorless computational paradigm which can achieve device density of 10^{12} devices/cm^2 and operating speed of THz [1].

Fig. 20. Diagram from Quantum-Dot Cellular Automata (Q.C.A) Circuit Partitioning: Problem Modeling and Solutions [15]

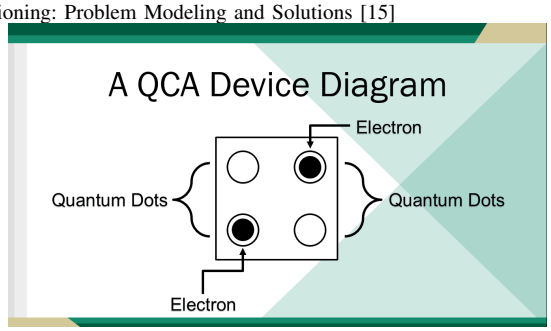
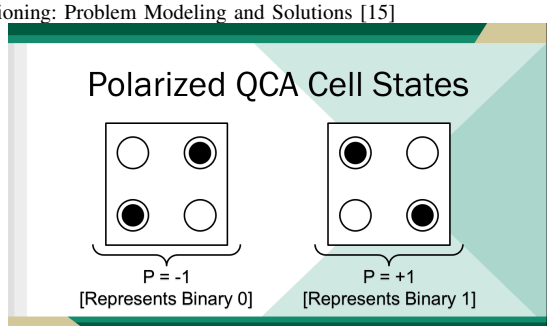


Fig. 21. Diagram from Quantum-Dot Cellular Automata (Q.C.A) Circuit Partitioning: Problem Modeling and Solutions [15]



VI. CONCLUSIONS

A. What We Learned

Besides exploring one of the most lengthy and beautiful publications on Computational Theory and Mathematics, we learned a lot about graphics generation and how Automata Theory applies to graphics, visual renderings, and the future of hardware. Some of the topics that we were originally going to investigate became too cumbersome to tackle, such as the Firing Synchronization Problem and the Majority Problem. These are well known problems within Cellular Automata research that we could not tackle without an understanding of Wolfram's work. Some of these problems did not have a solution and are still open problems in Mathematics. Instead we decided to talk more about Wolfram's work which does have solutions and algorithms to follow and learn about. This step became natural to us because it is well documented with numerous examples and demonstrations. This led us to replace the Majority Problem, Firing Synchronization Problem and the Majority problem with Elementary Cellular Automata and Boids Flocking System.

B. What Surprised Us

We did not think that this was going to be an easy research topic, but even with that in mind this research was extremely cumbersome and complex. The shear volume of research provided by Wolfram was shocking. In context, our paper is only a few pages long but the vast amount of information provided by Wolfram's thousand page plus masterpiece alone was shocking and something that none of us expected. So doing a proper survey on this topic was pretty much impossible and the topics we chose to talk about are necessary. It would be foolish to talk about C.A. without talking about Wolfram's research or Von Nuemann's sentinel papers. Without covering these topics we would have embarrassed ourselves. We originally wanted to talk about John Conway's Game of Life but that evolved into discussing about flocking systems as well. This change was inspired by Daniel Shiffman's publication **The Nature of Code** and his YouTube channel **The Coding Train** which provided a lot of insight on how to code Conway's Game of Life and E.C.A. which we did in Python [3], [20]. Our biggest disappointment was not being able to build our own Flocking System in Python. The amount of time and energy to build it ourselves proved to be extremely frustrating. We learned a lot in our attempts, but providing a third chuck of code when we already had two working code samples made the venture bitter sweet.

ACKNOWLEDGMENT

University of South Florida Computer Science and Engineering Department and Dr. Valentina Korzhova Automata Theory/Formal Languages Instructor.

REFERENCES

[1] A. W. Burks, Von Neumann's Self-Reproducing Automata, Office of Research Administration, Ann Arbor, MI, tech., 1969.
[2] S. Wolfram, A New Kind of Science. Wolfram Media, 2002.
[3] S. Wolfram, Cellular Automata as Simple Self-Organizing Systems.
[4] D. Shiffman, The Nature of Code. 2012.
[5] E. W. Weisstein, Elementary Cellular Automaton, Wolfram MathWorld.[Online].Available: http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.
[6] K. E. Stiefel, Fractals in Frozen, Science On, 29-Dec-2016. [Online]. Available: https://scienceonblog.wordpress.com/2017/01/19/fractals-in-frozen/.
[7] J. Shannon, Exploring the real world applications of cellular automata and its application to the simulation of flocking behaviour, Aug. 2013.
[8] D. Shiffman, Flocking. [Online]. Available: https://processing.org/examples/flocking.html.
[9] C. Reynolds, Boids: Background and Update, Steering Behaviors For Autonomous Characters. [Online]. Available: https://www.red3d.com/cwr/boids/.
[10] R. Feynman. Simulating physics with computers. International Journal of Theoretical Physics, 21:467488, June 1982.
[11] D. Deutsch, Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer, Proc. R. Soc. Lond., Vol. A400 (1985) 97-117
[12] G. Grossing and A. Zeilinger. Quantum cellular automata. Complex Syst., 2:197208, 1988.
[13] J. Watrous. On one-dimensional quantum cellular automata. In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, pages 528537, October 1995.
[14] K. Wiesner, Quantum Cellular Automata, Aug. 2008.
[15] D. A. Antonelli, D. Z. Chen, T. J. Dysart, X. S. Hu, A. B. Kahng, P. M. Kogge, R. C. Murphy, and M. T. Niemier, Quantum-Dot Cellular Automata (QCA) Circuit Partitioning: Problem Modeling and Solutions.
[16] International Technology Roadmap for Semiconductors (ITRS), 2015 Edition
[17] C.G. Smith. Computation without current. Science, 284:274, 1999.
[18] P.D. Tougaw and C.S. Lent. Logical devices implemented using quantum cellular automata. Journal of Applied Physics, 75:1818, 1994.
[19] U. Mehta and V. Dhare, Quantum-dot Cellular Automata (QCA): A Survey.
[20] D. Shiffman, "The Coding Train" [Online]. Available: https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1aw