Conway's Game of Life:

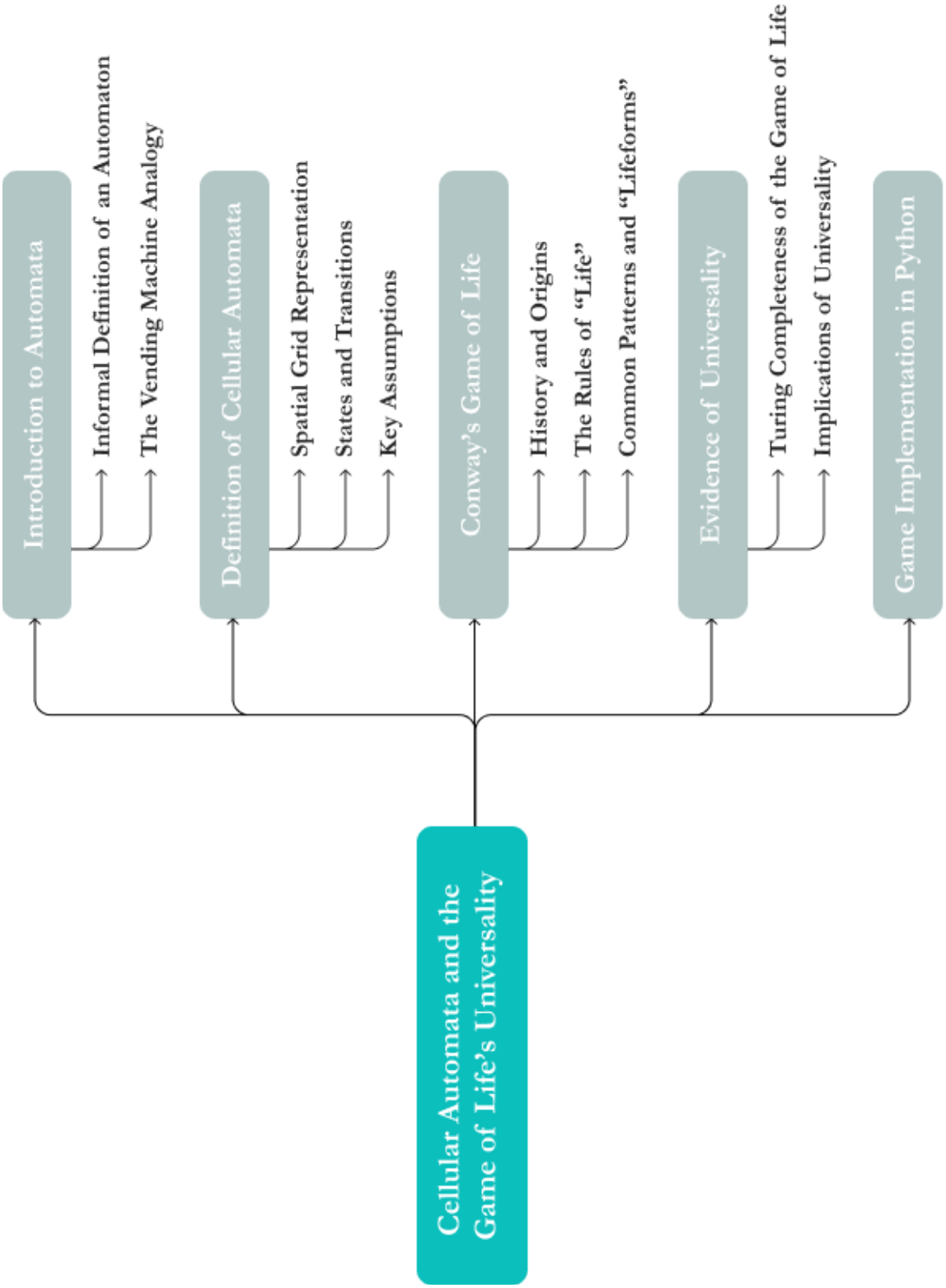
A Brief Overview of Cellular Automata and the Game of Life's Universality

Submitted by:
Daniel Philippe F. Lachica
11987642

Summary

In the 1970s, mathematician John Conway devised a cellular automaton which simulates the propagation and extinction of a population in evolutionary stages. Known as the Game of Life or simply "Life," this cellular automaton is governed by a set of rules and possesses the computational power to solve complex problems at the level of a Universal Turing Machine. This paper aims to discuss the mechanics behind Conway's Game of Life and provide evidence of its universality.

Schematic Diagram



Discussion

In the most basic sense of the word, an *automaton* (plural: *automata*) is a simple, yet powerful, computing device with a finite, distinct set of *states* and formally defined *transitions* which dictate how the machine is to facilitate changes in state (i.e. move from one state to another). An automaton may receive external input which could possibly trigger a change in state or even result in the production of output. Ladner (n.d.) likens automata to soft-drink vending machines, with good reason. Much like an automaton, a vending machine can receive input (in the form of coins, bills, and "choice" buttons), update its state according to the aforementioned input (i.e. only dispense the drink once it arrives at a final state wherein enough cash has been given), and lastly, produce output, which in this case, is the beverage of choice.

Cellular Automata

The universe as we know it is governed by an extensive set of rules. One happening may be traced back as the result of another, not necessarily related happening—the cycle of causality can persist indefinitely. In their book on cellular automata, Toffoli & Margolus (1987) provide an abstract, conceptual definition of such machines following this "universe" analogy: "Cellular automata are stylized, synthetic universes defined by simple rules much like those of a board game."

As discussed in the brief introduction above, automata are characterized by states and transitions. Cellular automata are no different; however, their "states" are represented spatially, by a uniform (usually two-dimensional) grid composed of *cells*, (hence the name) with each cell carrying some form of data. This state representation is illustrated in Figure 1. Transitions, on the other hand, occur in discrete measures of time. For example, the configuration of the cells in a cellular automaton's grid (or simply, its *state*) at time *t1* may or may not change upon transitioning to time *t2* depending on some input (Figure 2), which in this case, are arbitrary rules adhered to by the machine, or "laws of the universe," as expressed by Toffoli & Margolus (1987).

Fig 1. Visual Representation of a Cellular Automaton's State Space (*Ladner, n.d.*)

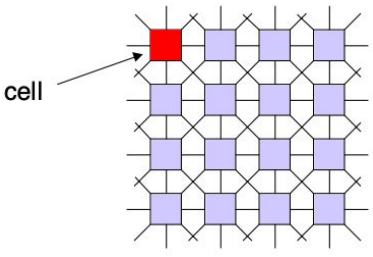
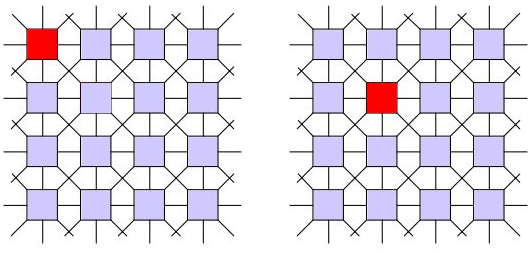


Fig 2. Cellular Automaton Transition



Kozliner (2019) formally defines cellular automata as computational models which adhere to the state-transition ideology described above, but are further characterized by a grid-like representation with cells which hold a value and signify an exact location on the grid. Like any other system, cellular automata are bound by certain assumptions. Firstly, all cells on the grid change state simultaneously (Ladner, n.d.). Secondly, the rules that govern cellular automata are *local*—cells are only concerned with their immediate surroundings or neighbors before transitioning to another state; and *uniform*—the rules apply to all cells on the grid at any given time (Toffoli & Margolus, 1987). Thus, with these assumptions at hand, the beauty and computational complexity of a cellular automaton can be seen at a glance: "Once set in motion, it runs by itself" (Toffoli & Margolus, 1987).

Conway's Game of Life

"Complex behaviors are often the result of simple computational rules" (Ladner, n.d.). This statement is perfectly exemplified by mathematician John Horton Conway's work on a cellular automaton known simply as, "the Game of Life." Invented in 1970, the Game of Life is perhaps one of the most widely known and researched cellular automata and has since piqued the interest of professionals in the field of computing (Toffoli & Margolus, 1987). At a glance, one may view the Game of Life as a rule-based simulation which illustrates the effect of "propagation and extinction tendencies" in live organisms (*cells*) of some population (*grid*) over the passage of time.

Moukarzel & Haber (2019) define the Game of Life as a zero-player mathematical "game." As discussed, the Game of Life takes place on a two-dimensional finite or infinite grid. Each cell on the grid can take on one of two states at any given time: "alive" or "dead." The game starts with an arbitrary grid configuration with a number of cells designated as alive, while the rest are dead. This initial configuration then determines the next state of the grid, and all cells update their state (i.e. being "alive" or "dead") simultaneously at the end of the current iteration. This repetitive loop of state generation is viewed as the "evolutionary" aspect of the Game of Life and can either go on indefinitely or for a predetermined amount of time. With these assumptions at hand, one must not forget the most important aspect of this cellular automaton: its rules.

The rules of the Game of Life are relatively simple and can be expressed as follows, as stated by Heaton in 2018:

1. Any live cell with zero or only one live neighbors dies due to "underpopulation";
2. Any live cell with two or three neighbors remains alive because its neighborhood is "just right";
3. Any live cell with more than three neighbors dies due to "overpopulation"; and
4. Any dead cell with *exactly* three neighbors becomes alive by means of "reproduction."

A more systematic way to view the rules of the game is through the use of a look-up table. Since a cellular automaton rule is a function on a finite set (i.e. one or more input can result in only one distinct output), look-up

tables are a common, structured implementation of such rules, as opposed to linguistic sentences. Moreover, Toffoli & Margolus (1987) state that this representation of rules drastically speeds up state computation. The look-up table below shows a few rows of the 512 possible configurations of a cell and its eight neighbors, represented by arrows in each of the eight cardinal directions, to illustrate Heaton's four rules of the game. Note that a 1 represents a cell in the "alive" state while a 0 represents a "dead" cell.

Rule	Current Cell State	↑	→	↓	←	↗	↘	↖	↙	Next Cell State
1	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0	0	1
2	1	1	0	1	0	1	0	0	0	1
3	1	0	1	1	1	1	1	1	1	0
4	0	0	1	1	1	1	1	1	1	0
4	0	1	1	1	0	0	0	0	0	1

Table 1. State Diagram Illustrating the Four Principal Rules of the Game of Life

One interesting aspect about the Game of Life worth mentioning is its flexibility with initial configurations. Certain placements of live cells next to dead cells at the initial state can give rise to complex and visually-stunning patterns with each iteration of the game. Izhikevich et al. (2015) mention a few such as the stable block known as the "Still Life," the repetitive "Oscillator," the perplexing "R-Pentomino," and the persistently moving "Glider." Some patterns such as the "Still Life" and the "Boat" are termed as "stable" because the cells which form them do not change state unless another moving pattern gets close enough to disrupt this stability. Others like the "Glider" and "Lightweight SpaceShip," remain in constant motion across iterations of the game until other patterns disrupt their configurations and cause them to die out (Lipa, n.d.).

Fig 3. Common Patterns in the Game of Life

Evidence of Universality

The Game of Life has been proven to be a universal Turing machine (i.e. *Turing-complete*) in that it can compute anything that is possibly computable (Kozliner, 2019; Moukarzel & Haber, 2019). Furthermore, the latter source adds to this by stating that the Game of Life is deterministic as well, meaning that the state which results from a transition entirely depends upon input from the previous state, which in this case, is the configuration of "alive" cells on the grid and their immediate neighbors. Conway was able to illustrate this universality in a stream of Gliders produced by the Glider Gun "lifeform." Rendel (2014) goes on further to say that mere collisions between these Gliders results in transference of information and the necessary logic to construct a computer with a finite amount of storage.

References

Heaton, R. (2018). “Programming Projects for Advanced Beginners #2: Game of Life.” Retrieved from https://robertheaton.com/2018/07/20/project-2-game-of-life/.

Izhikevich, E. M., Conway, J. H., & Seth, A. (2015). “Game of Life.” Scholarpedia, 10(6):1816.

Kozliner, E. (2019). “Algorithmic Beauty: An Introduction to Cellular Automata—An overview of simple algorithms that generate complex, life-like results.” Retrieved from https://towardsdatascience.com/algorithmic-beauty-an-introduction-to-cellular-automata-f53179b3cf8f.

Ladner, R. (n.d.). “Cellular Automata: The Game of Life or a New Kind of Science?” Retrieved from https://homes.cs.washington.edu/~ladner/ca/info/Mathday_presentation.pdf.

Lipa, C. (n.d.). “Chaos and Fractals: Conway’s Game of Life.” Retrieved from http://pi.math.cornell.edu/~lipa/mec/lesson6.html.

Moukarzel, J. & Haber, M. (2019). “From Scratch: The Game of Life.” Retrieved from https://towardsdatascience.com/from- scratch-the-game-of-life-161430453ee3.

Rendel, P. (2014). “Turing Machine Universality of the Game of Life.” University of the West of England.

Toffoli, T. & Margolus, N. (1987). “Cellular Automata Machines: A new environment for modeling.” Massachusetts Institute of Technology.

Author’s Note

In order to better understand the mechanics behind Conway’s Game of Life, I decided to code a simple implementation of the game in Python. You can see the Game of Life in motion once you run the file `GameOfLife.py`. If the `colorama` library hasn’t been installed on your machine yet, run `pip install colorama` first. If, for any reason, you see color codes (like `esc[31mMy`) appearing on the command line or terminal window instead of colored asterisks (representing cells), you may edit the source code and uncomment line #39 to solve the problem. Lastly, in the main function of the program, there are three variables you may experiment with: boundaries `GRID_WIDTH` and `GRID_HEIGHT`, and a `THRESHOLD` value which affects how many “alive” cells are initialized at random when the game starts.