# Cellular Automata and Their Implementation

Carter Sifferman

For Dr. Branton, CSCI 474, Drury University

A cellular automaton is a discrete model consisting of a grid of cells which evolves as time advances via a set of rules. These rules govern when the state of cells change depending on the state of surrounding cells [1, p. 601]. The number of states any one cell can be varies between cellular automata, while many use two states: an "on" and "off" state. While much of cellular automata theory was developed without the help of computers, modern technologies allow us to implement them rather easily. These implementations, including the one presented alongside this paper, are a useful tool for teaching artificial intelligence, program structure, and computer graphics.

Cellular automata were first conceived by John Von Neumann and Stanislaw Ulam in the 1950s. They were originally used to model motion of fluids, and Von Neumann used the idea to create more theoretical self-replicating systems [2, p. 15]. Cellular automata became of more interest to computer scientists when, in 1969, Alvy Ray Smith showed that they are computationally universal [3]. In the 1970s John Conway's "Game of Life" Cellular Automaton became widely known and has since been proven able to emulate a Turing machine [4]. More modern efforts have actually done so, and very recently, hobbyists have amazingly created Tetris in Conway's Game of Life [5].

There are an infinite number of conceivable rules for cellular automata. Consider the fact that an automaton can have infinitely many states, and can, in theory, take place in infinitely many dimensions. One-dimensional cellular automata have been classified succinctly by Steven Wolfram, giving a unique number and therefore ordering to the 256 possible rules, called the

rule's "Wolfram Code" [6]. Such a system is only conceivable in one dimension, as a two-dimensional variant would require $2^{512}$ numbered rules in order to uniquely identify each. Wolfram also classified all cellular automata into four complexity classes, with the first exhibiting no complex behavior, and increasing complexity up to the fourth class exhibiting very complex behavior (S. Wolfram). He conjectured that all class four rules are capable of universal computation. This is the class containing Conway's Game of Life [7]. Other classifications have existed since before Wolfram. A cellular automaton is reversible if there is exactly one previous state which leads to any one current state [7, p. 436]. This can be thought of like a one-to-one function. Additionally, automaton can have varying "neighborhoods", often called the Von Neumann neighborhood or Moore neighborhood. This refers to which surrounding cells are checked on each update [8].

Theory and practice differ greatly in the world of cellular automata. This is evident by the fact than much of early research on the subject was done without the help of computers. With modern computers, however, real-time simulations of cellular automata are possible, and are easy enough for an undergraduate computer science student to implement. These simulations can be useful for research, or for simply teaching students in an engaging way. There are, however, a few considerations to be had. Firstly, traditional cellular automata take place on an infinite plane. For the sake of practicality and simplicity, many implementations will handle this in one of three ways, listed from simplest to most complex: by "trimming" the edges of the simulation each iteration, by wrapping the edges of the simulation back on each other, as if on a toroidal shape, or by creating a (nearly) infinitely expanding grid. Considerations must also be made for the efficiency of the simulation. For example, in a 1000 by 1000 cell simulation, there are over a million cells, each with as many as eight neighbors. To optimize for this, the simulation can be

programmed to only check for cells which are eligible to be changed. For example, in the "seeds" rule, cells are only eligible to turn to the "on" state if they were in the "off" state in the previous frame. This means it is unnecessary to check the neighbors of cells that are in the "on" state each iteration. With more effort, further optimizations can be made, such as ignoring static structures, and keeping a list of cells eligible to change. Both of these techniques work by reducing comparisons.

Cellular automata can be implemented graphically in any language with a graphics library available and may be a good way to introduce one's self or students to a specific graphics library, as they only require drawing differently colored squares. The most effective language for cellular automata may be C++ for its speed and graphics capabilities. The best language, however, depends on what the user will be able to understand. According to ACM, the "most popular introductory teaching language" in the U.S. is Python [9]. It is for this reason that the companion program to this paper is written in Python.

Many factors were taken into account when writing the companion program. For simplicity, it is restricted to two-state, two-dimensional automata. Additionally, cells near the edge of the simulation are trimmed each iteration. These decisions help to strike a balance between features and simplicity. Command line parameters are supported that allow the user to change the simulation size, render scale, iteration time, and rule. A help parameter is also supported. Controls have been implemented that work during execution, allowing "drawing" on the grid, "erasing", pausing, and stepping one iteration at a time. The program is split into two files: cellular_automaton.py, and rules.py. The former houses most of the program logic, command line parameter handling, graphics, and the "main" method. The latter simply gives the former the rules by which to advance the grid. Each function in rules.py receives a grid and

returns the next grid. A new function, or "rule" can be added to rules.py, and can be passed in via the "rule" command line parameter without any changes to cellular_automaton.py. This is useful in an educational setting. A student can design their own cellular automata by writing a single simple function, and still get the benefits of a more sophisticated program. Three example rules are defined: "seeds", "life", and "life_without_death." An instructor could, however, easily remove some or all of them and encourage students to devise them themselves.

The Python graphics library Pyglet is required to run the program. This allows for easy drawing to canvas and user-interaction. Drawing to the window is all done with OpenGL through Pyglet. Rather than drawing GL_QUADS or GL_TRIANGLE objects, the program uses GL_POINT objects, and scales the whole window via glScalef to make cells easily visible. This leads to much better performance than drawing polygons but means that text and other graphics drawn to the window would have to be scaled at the same scale as the cells. For this reason, that information is instead displayed in the window's title bar.

Cellular automata have been around for nearly as long as computer science itself. Despite their age, today's students are still able to learn from their implementation: both about cellular automaton themselves, and about the surrounding technologies. The program presented with this paper enables that learning and exploration and is helpful for instructors and students alike.

The companion program can be found:

- Submitted with this paper
- In my CSCI 474 folder on the shared drive
- On my github at https://github.com/carterpaul/CA-for-education

Bibliography

[1] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Reviews of Modern Physics*, pp. 601-644, July 1983.

[2] J. V. Neumann, Theory of Self-Replicating Automata, Urbana: University of Illinois at Urbana-Champaign, 1966.

[3] A. R. Smith, "Simple Computation-Universal Cellular Spaces," *Journal of the Association for Computing Machinery*, pp. 339-353, 1971.

[4] P. Chapman, "Life Universal Computer," 2002. [Online]. Available: http://www.igblan.free-online.co.uk/igblan/ca/. [Accessed 1 May 2018].

[5] PhiNotPi, E. Starman, K. Zhang, Muddyfish, K. Lithos, Mego and Quartata, "Build a Working Game of Tetris in Conway's Game of Life," 2017. [Online]. Available: https://codegolf.stackexchange.com/questions/11880/build-a-working-game-of-tetris-in-conways-game-of-life. [Accessed 1 May 2018].

[6] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Reviews of Modern Physics*, pp. 601-644, July 1983.

[7] S. Wolfram, A New Kind of Science, Champaign: Wolfram Media, 1959.

[8] L. B. Kier, P. G. Seybold and C.-K. Cheng, Modeling Chemical Systems using Cellular Automata, Springer, 2005.

[9] P. Guo, "Communications of the ACM," 7 July 2014. [Online]. Available: https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext. [Accessed 1 May 2018].