An Exploration into Cellular Automata

Christina Snyder

Introduction

Cellular automata is a way to model systems using a grid of discrete cells. Each cell in a grid has a state and a neighborhood. The states of a particular cell will vary depending on the model, but they are typically discrete values. A neighborhood is a collection of adjacent cells within some radius. A cell's state will change over time by abiding to a strict ruleset. The ruleset will dictate the cell's state at time (t + 1) by analyzing the cell's neighborhood and current state at time (t). The most popular example of cellular automata is John von Neumann's "Game of Life". In his model, each cell had two states, 0 and 1 or "on" and "off". Depending on the cumulative states of a cell's immediate 8 neighbors, the cell's with either stay constant or change to the opposite state. There are 256 possible combinations of the neighbors and von Neumann was able to create the simulation using just a few rules.

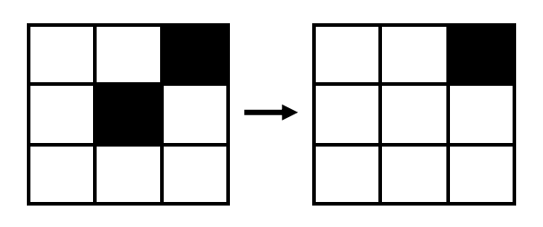
"Game of Life" is a two dimensional model, however, cellular automata can be applied to 1 or 3 dimensional grids as well. One popular application of a 3 dimensional model is image processing. One can think of a cell as a pixel and the cell's state as the color of that pixel. This could involve image blurring, noise reduction, and edge detection. There are also a number of image effects such as glow, fur, and impressionist. The two most relevant applications to the scientific community are noise reduction and edge detection.

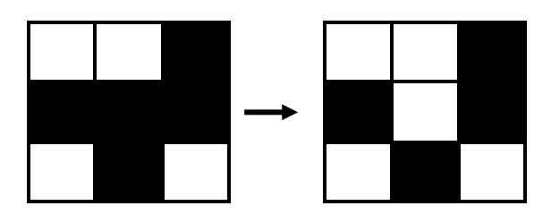
Game of Life

Von Neumann's Game of Life is dictated by three rules based on the states of the neighboring 8 cells. Each cell in this grid has one of two states, alive or dead (1 or 0 respectively). The rules are as follows:

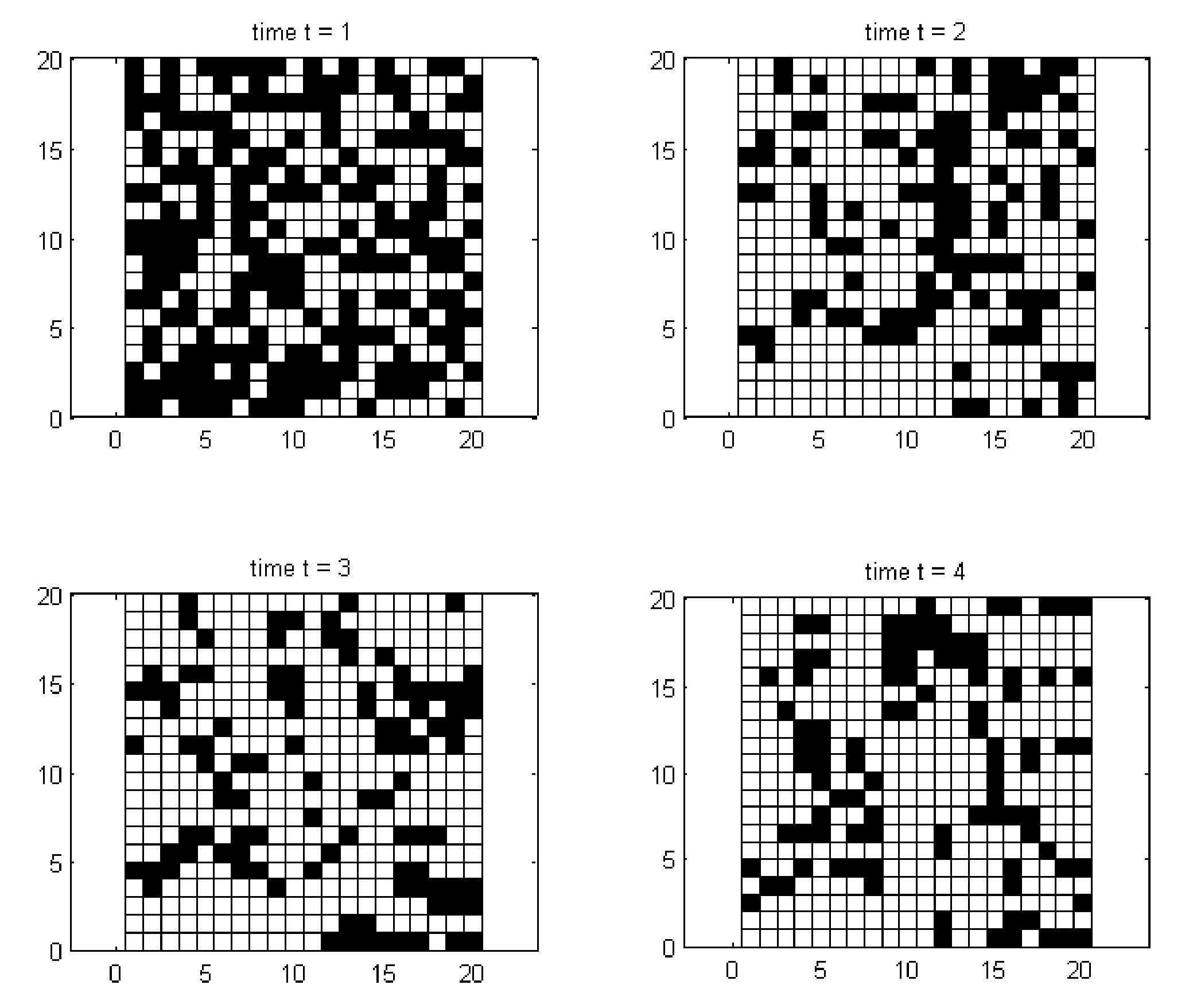
If a cell is alive,
- It will die if it has 4 or more alive neighbors.
- It will die if it has 1 or fewer alive neighbors.
- It will stay alive if it has exactly 2 or 3 neighbors.

If a cell is dead,
- It will come to life if it has exactly 3 alive neighbors.
- It will stay dead otherwise.


Death


Death

The image below shows 4 iterations of the game using a randomly generated 20x20 grid.



Literature

Goi, Hilton. "An Original Method of Edge Detection Based on Cellular Automata." (2003). Korea Advanced Institute of Science and Technology. Web. 13 Apr. 2015.

Shiffman, Daniel, and Shannon Fry. "Cellular Automata." *The Nature of Code*. Print.

Petric, Sinisa. "Image Effects with Cellular Automata." *Sigmapi Design* (2010). Print.

Popovici, Adriana, and Dan Popovici. "Cellular Automata in Image Processing." Web. <http://www3.nd.edu/~mtns/papers/17761_4.pdf>.

Qadir, Fasel, Peer M A, and Khan K A. "An Effective Image Noise Filtering Algorithm Using Cellular Automata." *ICCCI* (2012). Print.

Dalhoun, Abdel, Ibraheem Al-Dhamari, Alfonso Ortega, and Manuel Alfonseca. "Enhanced Cellular Automata for Image Noise Removal." Print.

Noise Reduction

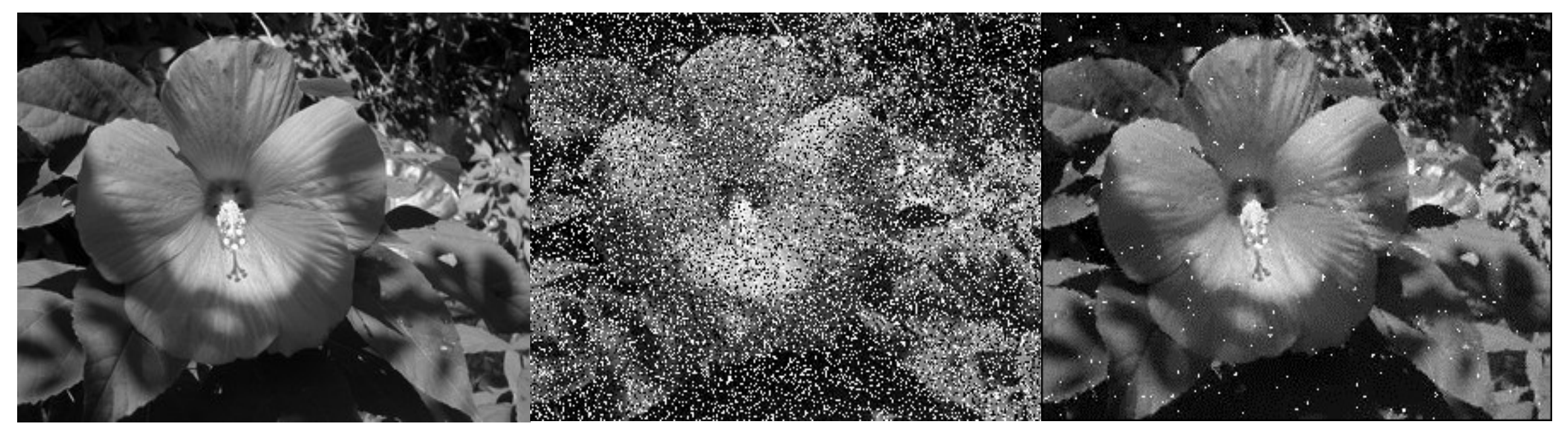
An interesting application of cellular automata is noise reduction. Given an image, there is some level of background noise, and the goal is to minimize that. This has many uses in biological settings, anything from DNA analysis to microscopic image processing.

To apply CA to noise reduction, images can be thought of as a grid where each cell is a pixel. MATLAB reads in an image as a 3 dimensional array where each layer contains the RGB values of a particular pixel. It is important to note that in a greyscale image, the RGB vector values are the same.

The following algorithm reduces "salt and pepper" noise in a greyscale image; pure white and black pixels have a 30% change to be randomly placed throughout the image.

Given a pixel value between 0 and 255 and it's neighboring 8 cells,
- It stays the same if it is between the minimum and maximum surrounding pixel values
- It becomes the median if it is between 0 and 255 exclusively
- It becomes the pixel value with the greatest difference to itself

The greyscale image below had salt and pepper noise added to it, then this algorithm reduced that noise.



The above algorithm is fairly easy to implement because it only involves one layer. Applying it to a color image is slightly more complex. The above algorithm requires that the minimum and maximum pixel value is known. In a color image, the pixel is made of 3 values, RGB. For example, in the greyscale image, 255 corresponds to white. The RGB vector [255, 255, 255] corresponds to white in a color image; the vector [255, 0, 0] corresponds to a pure red. Which of those would be the minimum or maximum? To account for this, the pixel$_{i,j}$ along with it's neighbors were converted to a grey pixel. This just requires averaging the RGB vector. Sorting the resulting set will return a set of indices that can be used to sort the initial set of RGB values. Once that step is completed, the greyscale algorithm can be applied to each layer of the color image.

The example below uses the greyscale algorithm modified for color images.

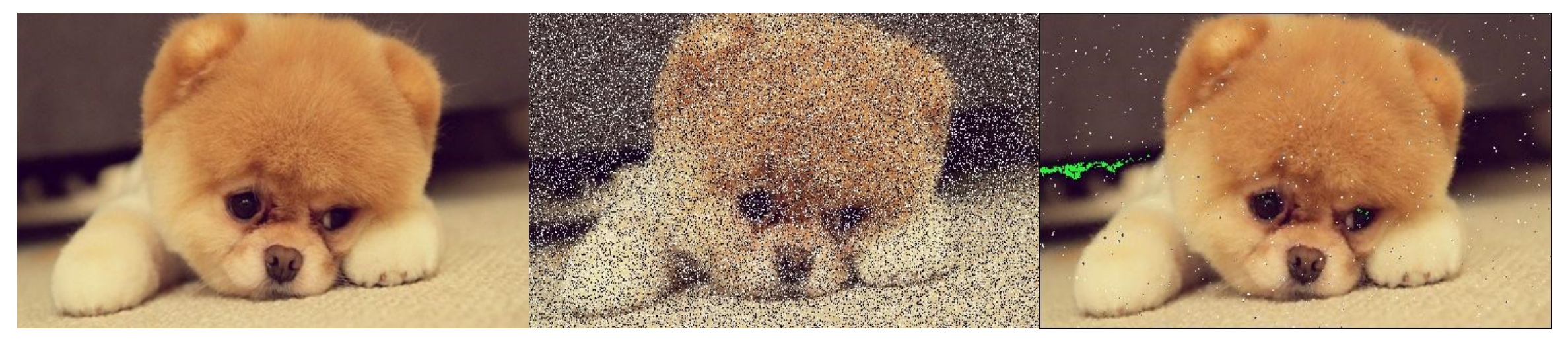
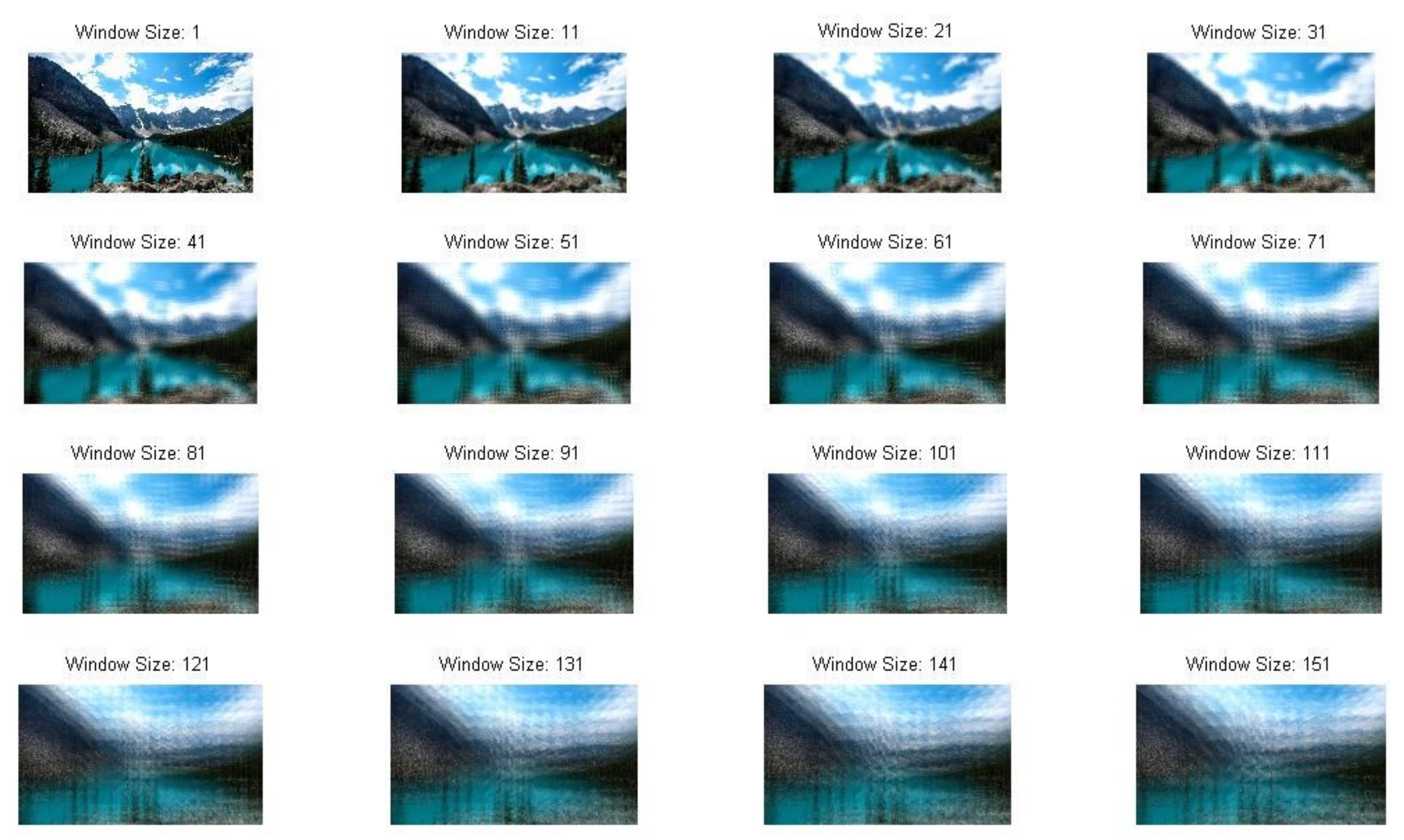


Image Blurring

Another application of cellular automata image processing is blurring. For a given pixel, it's next state is the average of the surrounding pixels within some radius. The example below shows image blurring with radii from 1 in steps of 10 to 151.



Edge Detection

Edge detection in images is important not only in the scientific community, but also in practical life as well. Edge detection can be applied to forensics, street regulations, etc. There are many methods to accomplish this, and one of the is using cellular automata.

The basic way to determine the instance of an edge is to measure the degree of pixel value change. A large change would indicate an edge.

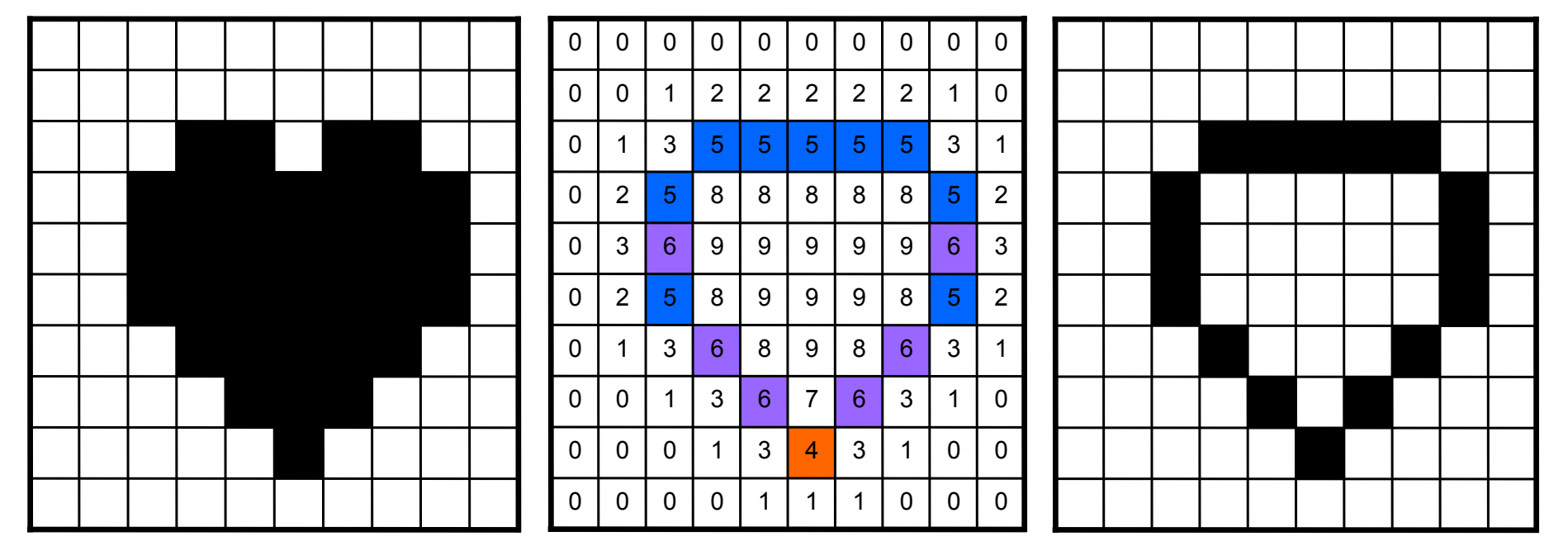
The simplest image on which to perform edge detection is a monochrome image. Given a monochrome image, applying a ruleset similar to the game of life would result in edge detection.

A cell will die if
- It has less than or equal to 3 neighbors
- It has 7 or more neighbors.

A cell will come to life if
- It has exactly 5 neighbors.

All other cells stay the same.

Applying this rule to the monochrome heart below yields a basic edge.



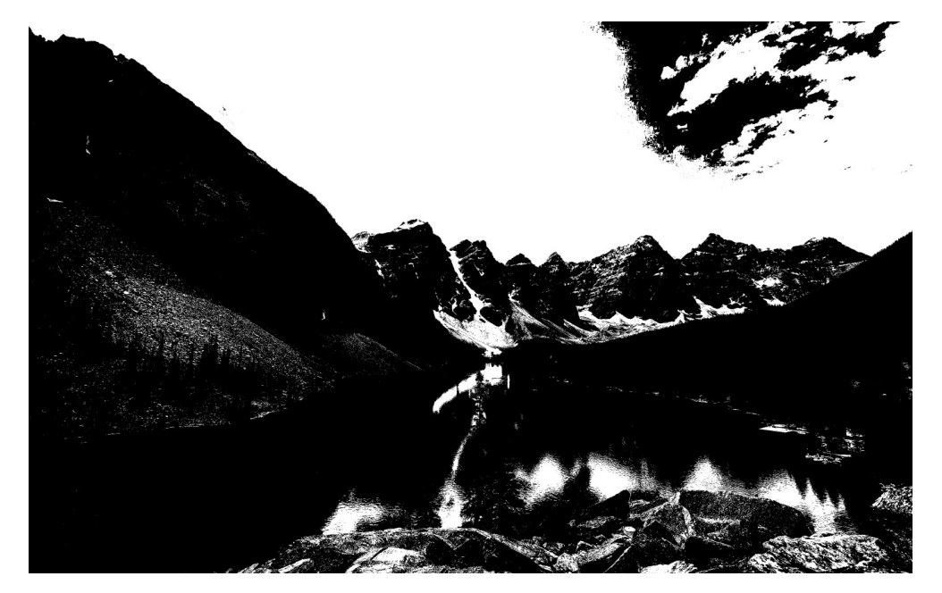
The number of possibilities is discrete (0-9). Therefore, they can be enumerated similarly to the one dimensional cellular automata. For the above case, the following ruleset was applied:

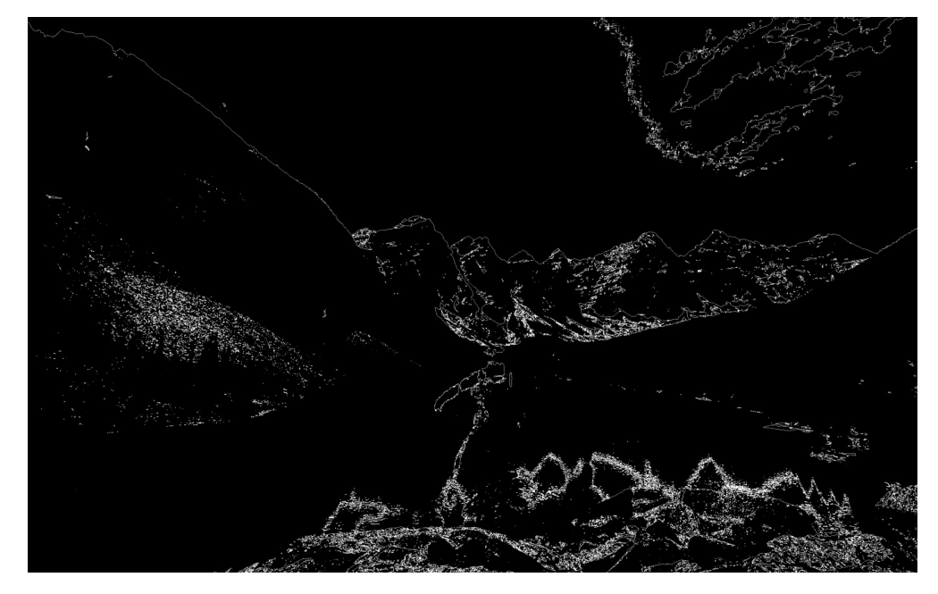
0	1	2	3	4	5	6	7	8	9
0	0	0	0	X	1	X	0	0	0

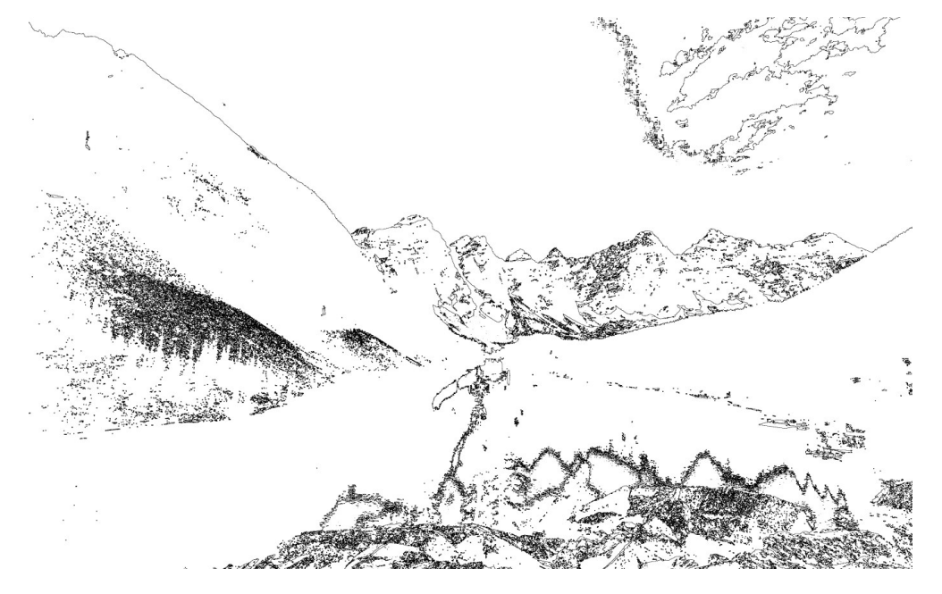
The X denotes that the state should not change. During implementation, X is equivalent to 1.

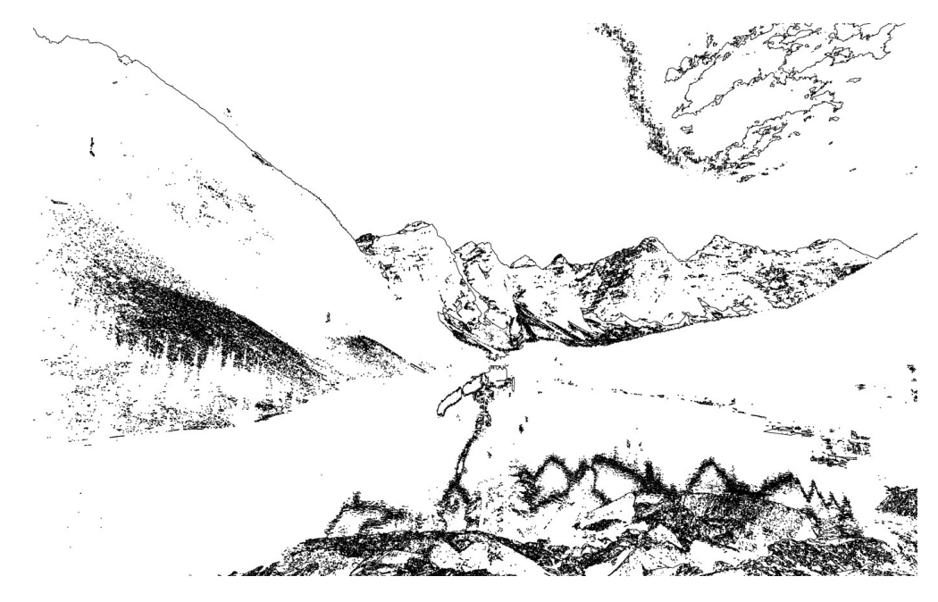
There are many rules that can be applied this way, and some work better than others for a particular picture.

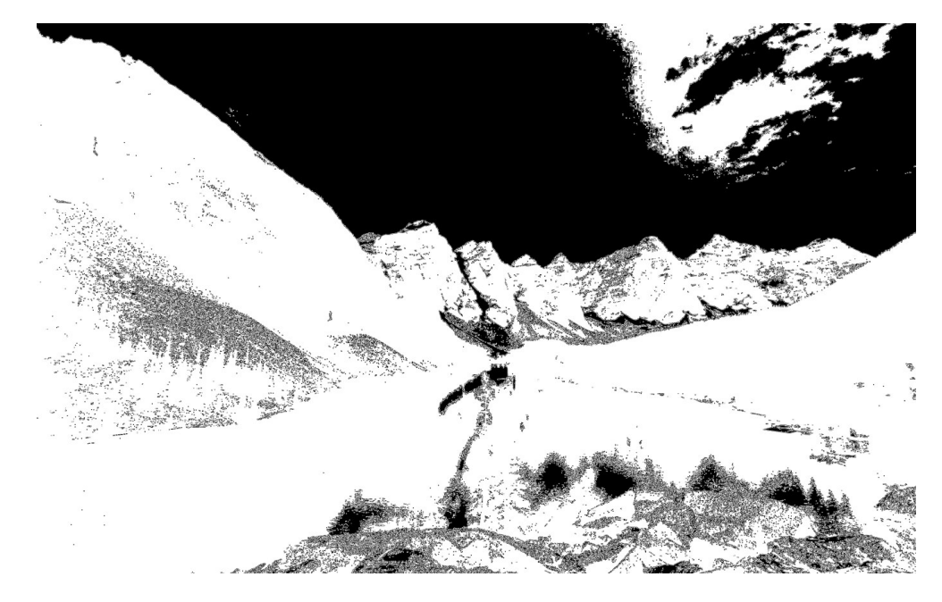
Original Image


Monochrome Image


Rule 56


Rule 543


Rule 515

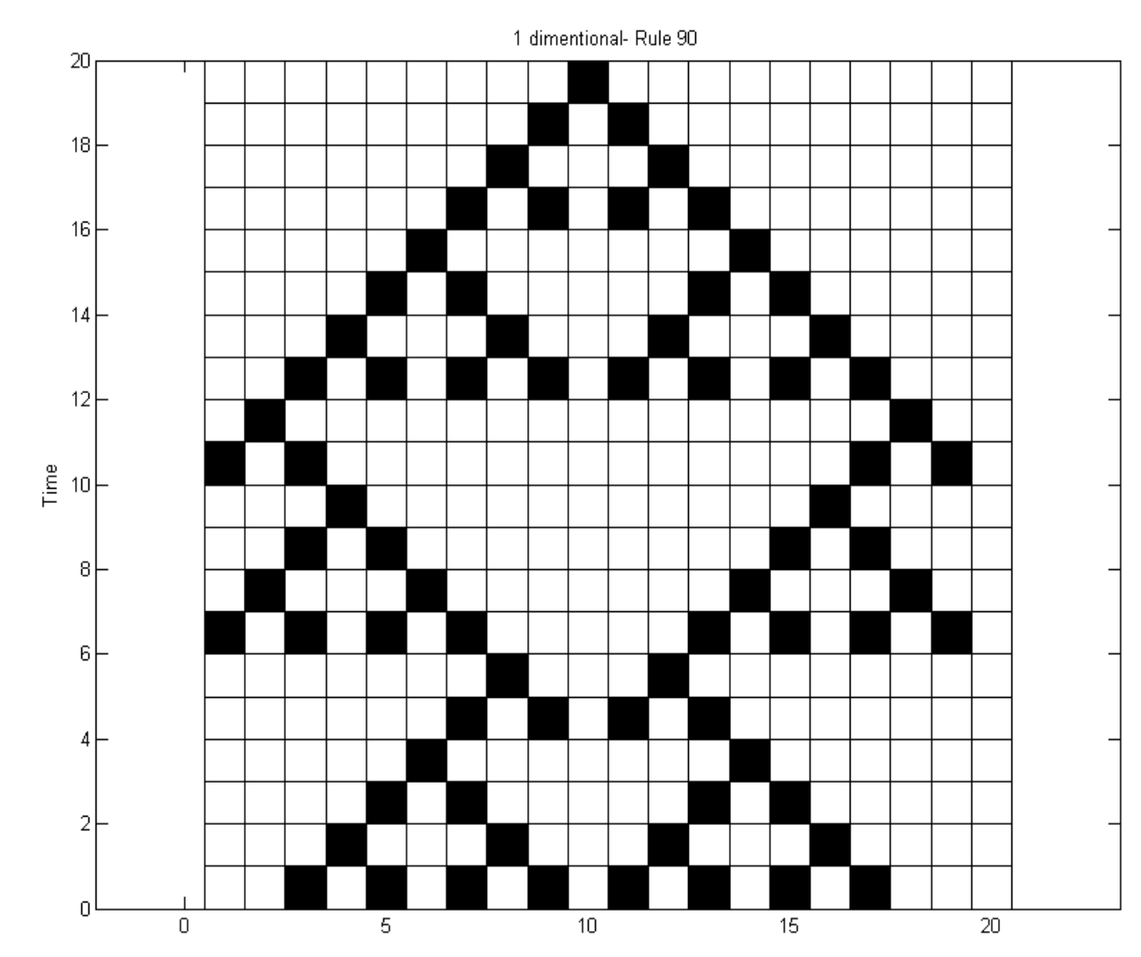

Rule 682

One Dimensional Exploration

One dimensional cellular automata is the most basic form. With a single row grid, each cell only has 2 neighbors. There are only 8 possible combinations of neighborhoods. With a neighborhood of this size, a rule for each instance is not unreasonable.

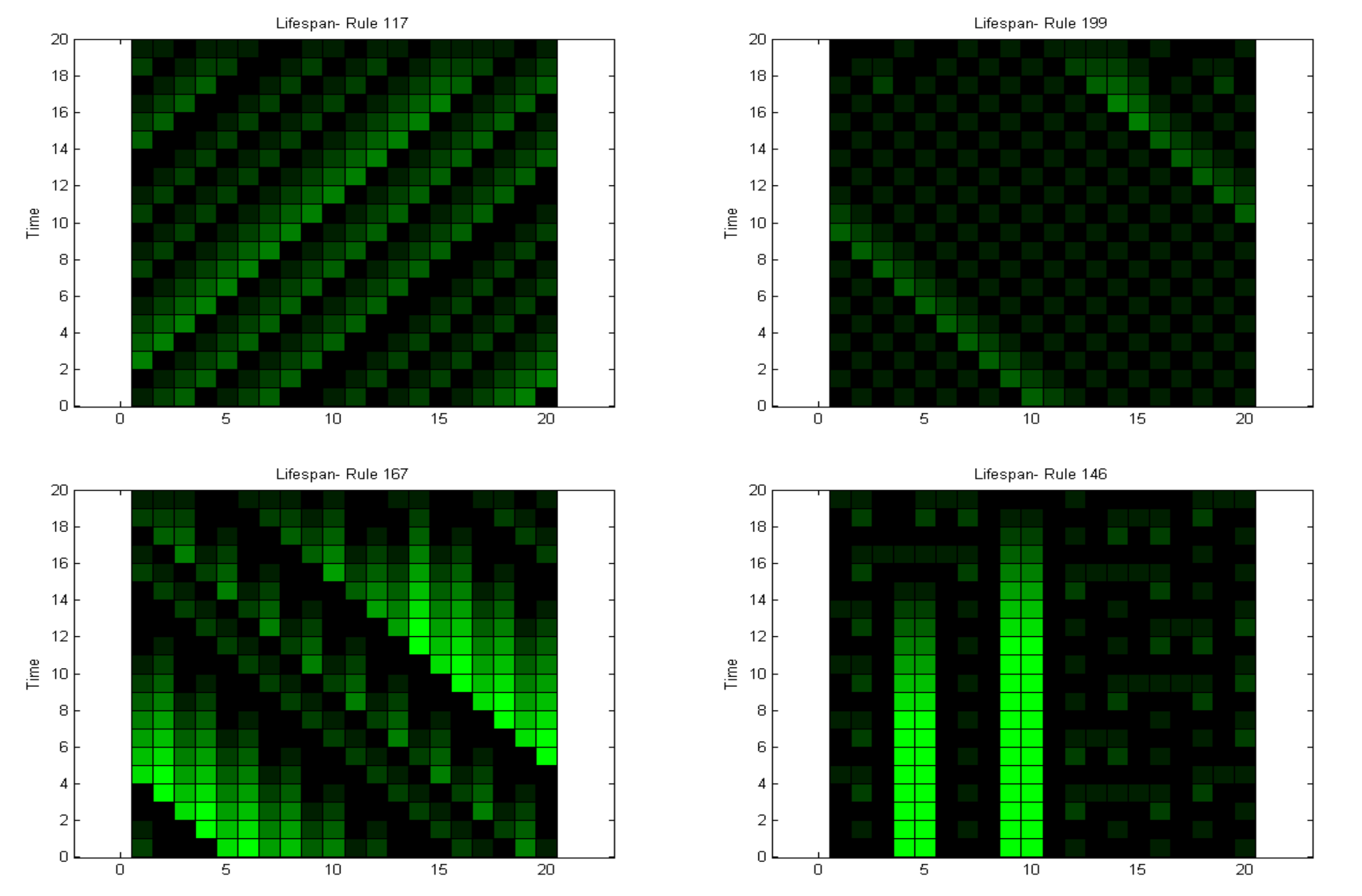
0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1
0			1			0			1			1			0			1			0		

The table above enumerates every possible instance and gives the value of the center cell's state at time t + 1. Using this ruleset over twenty time units, the image to the right is generated. The starting row is at the top of the image. With this particular set of starting conditions, the resulting image is the "Sierpinski triangle". The rule used to describe these states is rule 90, so named because when the future states are viewed as a binary number, it converts to the decimal number, 90. Using that logic in reverse, there are 256 possible rulesets for a 1 dimensional grid.



Sticking with the binary logic, we can use the combinations themselves to help with the generation of the next state. The '000' state corresponds with the 0 index, the '001' state with the 1 index, and so on. This is quite the simplification from the implementation standpoint because the need for 8 if statements have gone away.

One way to visualize this data is to see how long one cell can survive. This visualization would be useful when simulating an environment and focusing on animal lifespan. The following grids have random starting states and random rulesets.



Wolfram Classifications

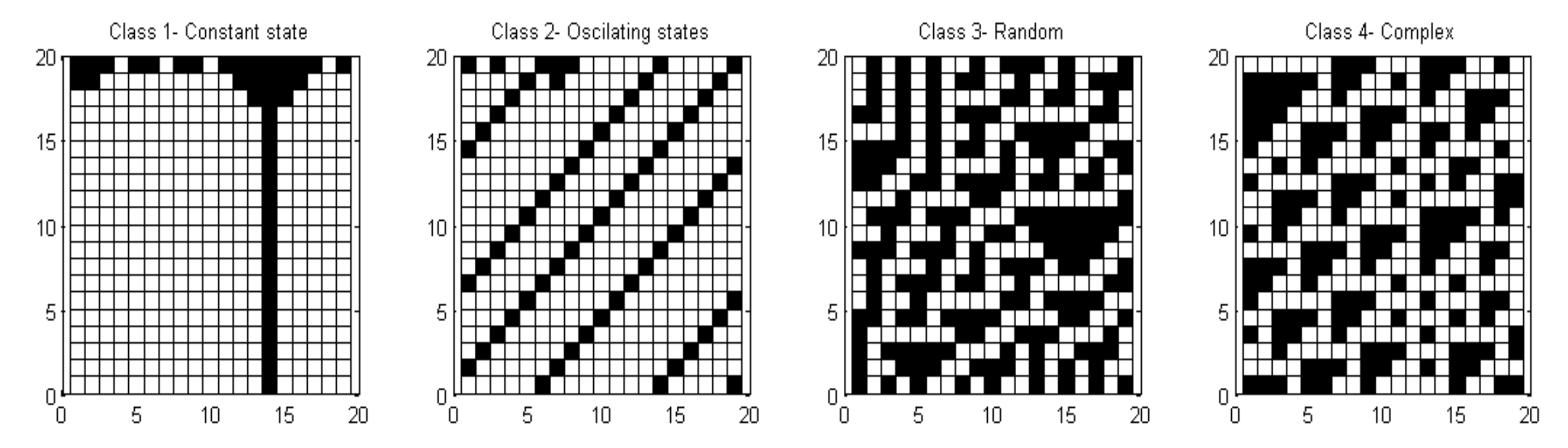
Stephen Wolfram worked with cellular automata and classified patterns observed during simulations into four categories. Each of the 255 rules fits into one of these categories. The placement of a rule into a category is independent of it's starting state.

Category 1: Uniform
- All cells will eventually find a steady state and remain unchanged.

Category 2: Repeating
- All cells will be stable, but their state will regularly change.

Category 3: Random
- All cells will appear to have random state changes.

Category 4: Complex


Class 1- Constant state
Class 2- Oscilating states
Class 3- Random
Class 4- Complex