Wolfram's Nearest Neighbor Laws on 1-Dimensional Cellular Automata

Nick Caverly and Kevin Fretz

under the direction of
Dr. Michael Bardzell
Salisbury University

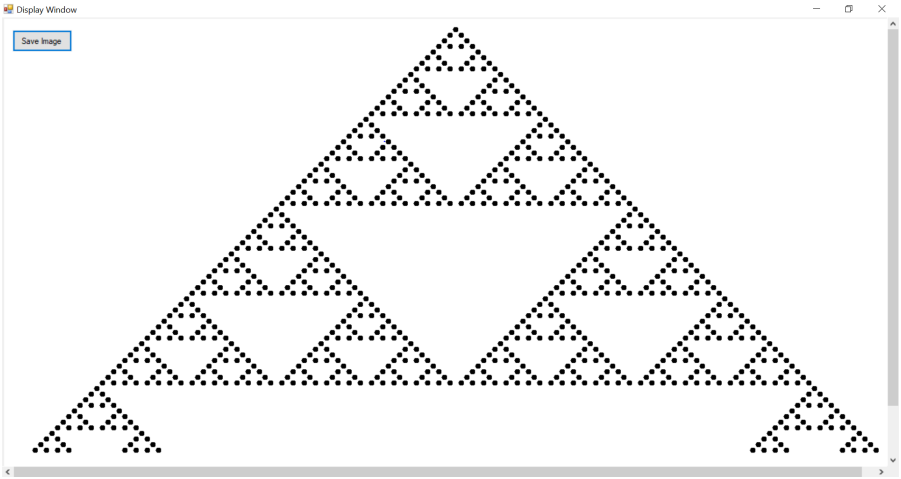
Fall 2019

Abstract

A 1-Dimensional Cellular automata is a system of cells arranged in order progressing through a series of generations. This project has been designed to show all of these generations in order, with the first generation at the top of the produced image and every generation following found directly underneath in the order they are produced. There are 256 nearest neighbor laws, based on the possible combinations available. The program has the ability to run laws on automatas of infinite size or a size input by the user. It also has the option to run executing the rules beginning with the leftmost cell and traveling to the right, going the opposite direction, or developing a random order and following that for the rest of the sequence. The user determines the length of the sequence as well as the starting states of the first generation.

Introduction

Wolfram's nearest neighbor laws refer to a system of rules for a cellular automata to follow. The laws can be understood by utilizing the binary system. Rule 90 is a commonly used example, and can be seen below:

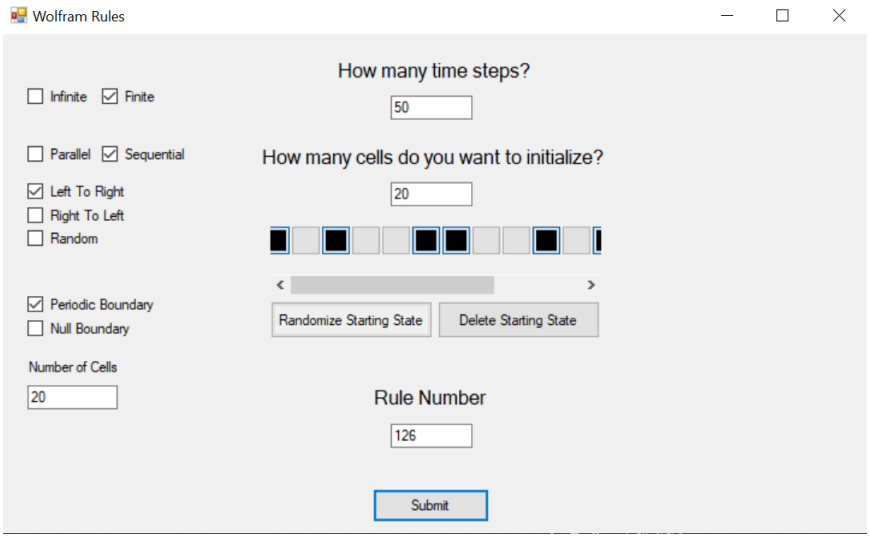


The reason this rule is known as rule 90 is illustrated above the pattern itself. The sets of three boxes on the top line indicate the previous generation's cells. These cells are what decide the next generation, each combination indicating what the cell in the middle will do for the next generation. This dependence on the

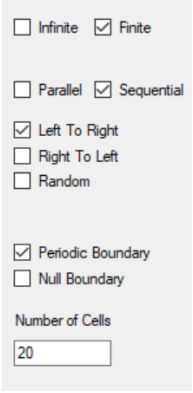
neighboring cells is where the name Nearest Neighbor Laws come from. Starting with the rightmost set of three boxes, you can see that all three are white, indicating these are *empty* cells. Underneath these cells, you can see a zero. This zero indicates that the resulting state for the cell in the middle is also empty. The rest of the numbers form the binary string $01011010_2 = 90_{10}$. The same sets of boxes are used in the same order to determine what each rule will do with any given combination, and the resulting cells are how the rule is determined, in the same way as rule 90 was shown.

These automata can either be infinite or finite in terms of the number of cells in each generation. This will affect the resulting pattern as finite automata can have null boundaries, meaning the cells outside the size of the automata will be represented by zeros or empty cells. Finite automata can also have periodic boundaries, meaning the first cell will look to the last cell in the generation previous to it to determine what its left neighbor's state is, and the last cell will look to the first cell in its generation. This looping is reversed for a sequence executing from right to left, meaning the last cell executes first and looks to the first cell in the previous generation, and the first cell in the generation will use the last cell as its neighbor.

Using the Application



To use the application, the user will have to check either the finite or infinite box which determines the number of cells in a generation. If the user selects finite, they will enter the desired size in the input box for array size at the bottom left. The user will then select the desired order of execution for the sequence by selecting one of the three options; left-to-right, right-to-left, or random.

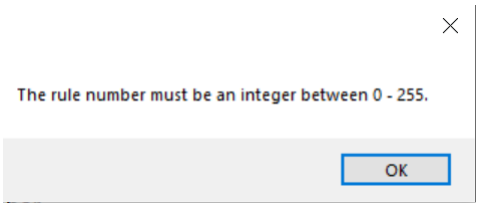


In the center of the application at the top there is a field for inputting the number of generations you would like to create. Some of these may not fit in the display field in the application, but will be shown in the full image if saved. Below the number of generations there is a field for the number of cells from the initial state you would like to change. These cells could be a fraction of the size of the system, and will be centered

in the middle. Below the space to enter the number changed there will be a corresponding number of check boxes. If you want a cell to be empty, leave the box unchecked, otherwise click the box. Alternatively, you could randomize the initial starting states of each cell with the randomize button below the check boxes. Below these is a field for entering a rule number between 0 and 255.

After all these fields have been selected there is a button at the bottom that will display as much of the image as possible. If you want to save it or view the full image there is a save image button in the top left of the display and it will save the jpeg to your Documents. There will also be a return button which will erase all the current data and return you to the menu to input new options.

There are also multiple error checks throughout the application. If the user enters in wrong information in certain areas or doesn't provide the required data, a message box will appear on the screen. This message will let the user know what went wrong or what information is missing. After closing the message, the user can re-enter data and submit.



Implementation

Implementing the generations and states was just storing "empty cells" as 0s in an array, and "full cells" as 1s. We only required two arrays, one as the current generation and one as the previous generation. After each current generation is determined it is output and then copied to the previous generation and the current generation array is emptied.

The way each generation was determined was dependent on the rule number, which was inputted by the user as a decimal number and then converted by the code into an array of 1s and 0s, similar to the rule 90 example. This array was then used to determine how each combination of 0s and 1s from the previous generation array would be treated. This process involved finding the value of each combination based on their binary representation, and then a switch would use that value to check what the resulting state was for that specific value and rule.