2014

An Investigation of Complex Systems in 16 Dimensions

Jordon M. Huffman
Columbus State University, huffman_jordon@columbusstate.edu

AN INVESTIGATION OF COMPLEX SYSTEMS IN 16 DIMENSIONS

Jordon Miles Huffman

An Examination of Complex Systems in 16 Dimensions

by

Jordon Huffman

A Thesis Submitted in Partial Fulfillment of
Requirements of the CSU Honors Program
for Honors in the degree of
Bachelor of Science
in
Computer Science
TSYS School of Computer Science
Columbus State University

Thesis Advisor [illegible] Date 4/21/14

Dr. Rodrigo Obando

Committee Member [illegible] Date 4/21/14

Dr. Charles Turnitsa

Honors Committee Member [illegible] Date 4/21/14

Dr. Rylan Steele

Honors Program Director Cindy Ticknor Date 4/24/14

Dr. Cindy Ticknor

Table of Contents

An Examination of Complex Systems in 16 Dimensions

Description:

With the research of Dr. Rodrigo Obando, I have created an application to observe cellular automata rules spaces. The elementary rule space with two hundred and fifty-six rules proves easy enough to study, but the next rule space of 4,294,967,296 rules is much more of a challenge. Such an enormous jump makes studying each rule and its neighbors extremely time consuming. My program takes a rule from the 4 billion rule space and displays its neighbors in their appropriate cardinal directions.

Abstract:

Sir Isaac Newton studied the world around him. He observed unexplainable phenomena that the math of his time could not prove. With the help of Gottfried Leibniz, he created infinitesimal calculus to prove his theories. The new concepts he created revolutionized science, and opened new realms of science previously unthought-of. In 2002, Dr. Stephen Wolfram published *A New Kind of Science*. He argues that the processes of understanding cellular automata can be applied to other aspects of science. Dr. Rodrigo Obando of Columbus State University took Dr. Wolfram's work and dissected it. By breaking down the rules, he started seeing patterns emerge in their ordering. He created a system for organizing the rules of the elementary cellular automaton rule space with 256 rules. From there, he guided me to the next rule space of size 4,294,967,296. I used his procedures to examine that rule space. My program takes a rule and displays all of the neighboring rules in that rule space. Essentially, I am taking a node of a 16-dimensional hypercube and showing all of the neighboring nodes. This design has not been publicly created to my or Dr. Obando's knowledge. Understanding the patterns of cellular automata and what makes one rule different from a neighboring rule can be applied to other forms of science. This is what Dr. Stephen Wolfram stated in *A New Kind of Science*, and I firmly agree with his statement. Any exploration of the next rule space will bring us closer to seeing Dr. Wolfram's theory come true.

Introduction

Anytime a computer's power button is pressed or an email arrives in an inbox, anytime the internet is viewed or a digital game is played, no one ever asks the question, "How did this ever come to be"? Does any typical person even know how or why these machines were created? When John von Neumann's name is mentioned, does anyone other than scientists or mathematicians raise an eyebrow? We use technology directly influenced by him each and every day. Von Neumann's research in automata theory is the focus of this paper, specifically with regards to the correlation between the arrangement of the rules in this theory and the output produced in the system.

John von Neumann was born in 1903. A child prodigy, he became one of the most influential mathematicians of his time by his mid-twenties. Von Neumann is often referred to as the father of computer science, as his work directly led to the creation of the first computer. Some of his research areas included set theory, quantum theory, and automata theory. Von Neumann's automata theory consists of cellular automata and other abstract concepts. The Britannica Academic Edition describes cellular automata as the "simplest model of a spatially distributed process that can be used to simulate various real-world processes". In this model, cellular automata are represented as a two-dimensional array of blocks. The state of the current block's neighbors changes the state of the lower block. Cellular automata simulation capabilities present a wide range of uses for scientists and mathematicians alike. After the passing of von Neumann in 1957, cellular automata essentially fell off most scientists' radar. With only a few papers published on the topic from the 1960s to the 1980s, it seemed cellular automata would be forgotten.

In the 1980s, another child-genius-turned-scientist Dr. Stephen Wolfram awakened the realm of cellular automata once more. Throughout the decade, Dr. Wolfram published a collection of papers on the subject. For many years, he researched complexity and John von Neumann's discovery of cellular automata. In 2002, Dr. Wolfram finally finished and published *A New Kind of Science.* Stephen took the foundations of cellular automata laid out by John von Neumann and expounded upon them in this book. He argued that cellular-automata-based-computer models allow scientists to better understand the complexity of nature. In essence, he claimed that nature functions like a computer and is not random. Dr. Wolfram's work on cellular automata is ongoing, and with the development of his computer program, Mathematica, Dr. Wolfram has been able to solidify funding for his research. There exists a mutually beneficial relationship between the two. Researchers, teachers, and even students have access to Mathematica. In fact, all the experiments performed in this paper were done so using Mathematica.

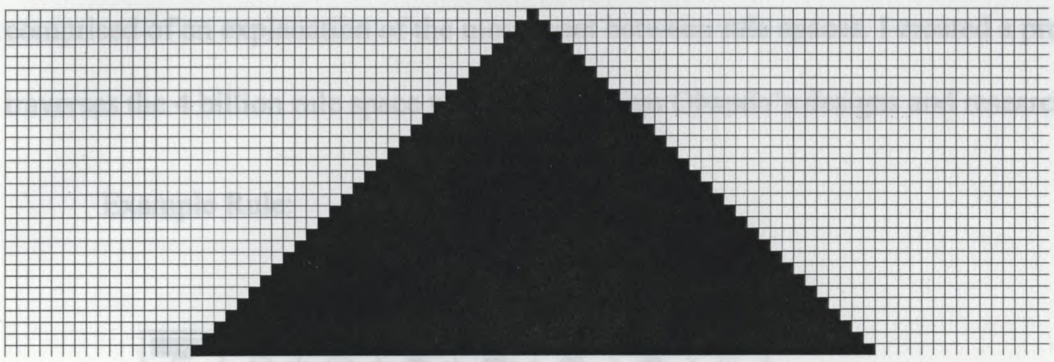
Cellular Automata

Cellular automata are models. We can imagine that there is a series of square blocks. We can say that each block is represented by a bit, one or zero, which represents the color of each block. Zero is white and one is black. The route taken at this point determines how the cellular automata react. The default choice for Dr. Wolfram was to take only one cell to the right and left of the target cell, so that only three cells are in the scope of each decision. The formula to represent the number of elementary cellular automata is $k^{k^{(2r+1)}}$ where k is the number of states and r is the number of cells to examine on both sides, the radius. The elementary cellular automata rule space uses $k = 2$ (binary: 0, 1) and $r = 1$ (three cells to use: central, left, and right). The formula for the elementary rule space is $2^{2^{(2(1)+1)}} = 256$. There are two hundred and fifty-six rules in the smallest cellular automata rule space. If we simply increase the number of cells used by one on each side, (2(2) + 1), our equation becomes $2^{2^5} = 4{,}294{,}967{,}296$. That means there are four billion two hundred ninety-four million nine hundred sixty-seven thousand two hundred and ninety-six rules!

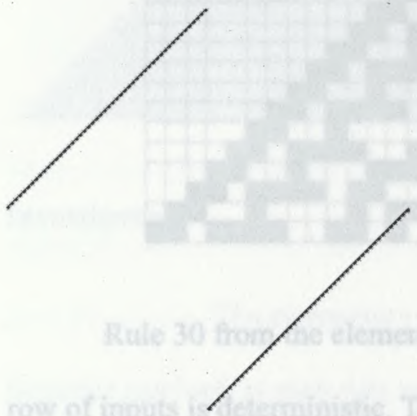
Each rule produces a distinct graph. Dr. Stephen Wolfram only used the elementary cellular automata for that reason. In his book, he listed and ordered the two hundred and fifty-six rules sequentially. With sequential ordering, no obvious patterns exist throughout the rule space. Dr. Rodrigo Obando undertook the task of discovering why there were no discernible patterns. The big picture is that the inputs to the bit string as well as the rules are deterministic; we know exactly what the bits and rules are before we apply the rules. However, the output seems random. This is a problem mathematically, scientifically, and philosophically. No deterministic system should produce random output. Dr. Stephen Wolfram accepted the fact that the distribution of behavior is random, but Dr. Obando could not let the thought slide so easily. Instead, he created a program that performs specifically designed algorithms on each rule which allows him to partially organize the rules.

Cellular automata rules are categorized into one of three classes: Class 1, Class 2, Class 3, and Class 4. Class 1 is uniform. There are no variations from cell to cell. Class 2 is semi-uniform. Here, the uniformity can be seen, but inconsistencies with neighboring cells make distinctly not uniform patterns. Class 3 is random. The randomness of Class 3 cellular automata creates more questions than answers. How can deterministic values produce seemingly random output? Below are examples of these classes:

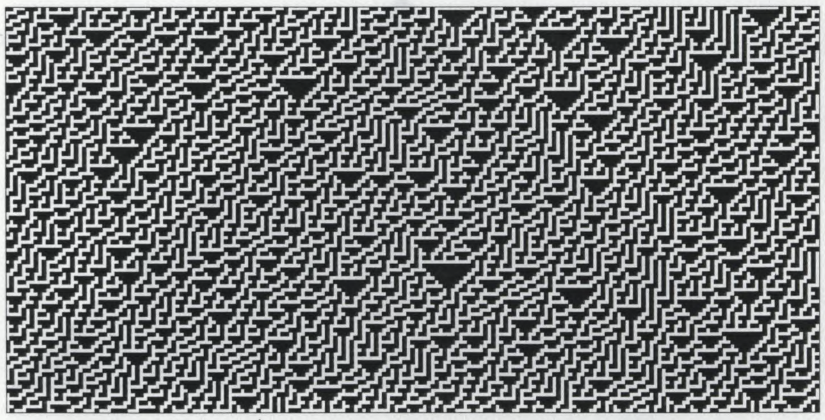
An example of Class 1: Rule 222



An example of Class 2: Rule 112

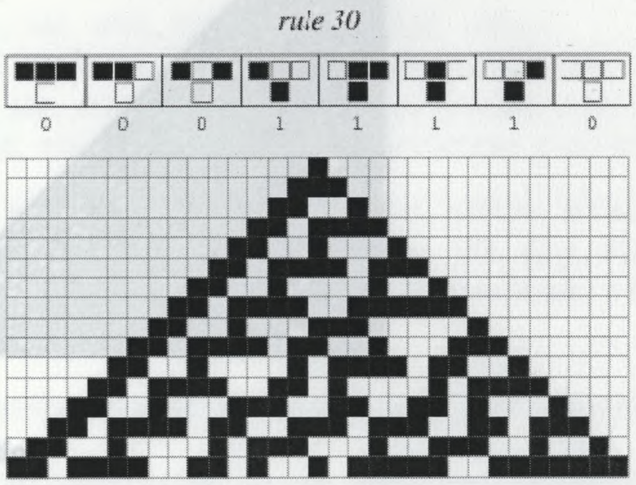


An example of Class 3: Rule 135

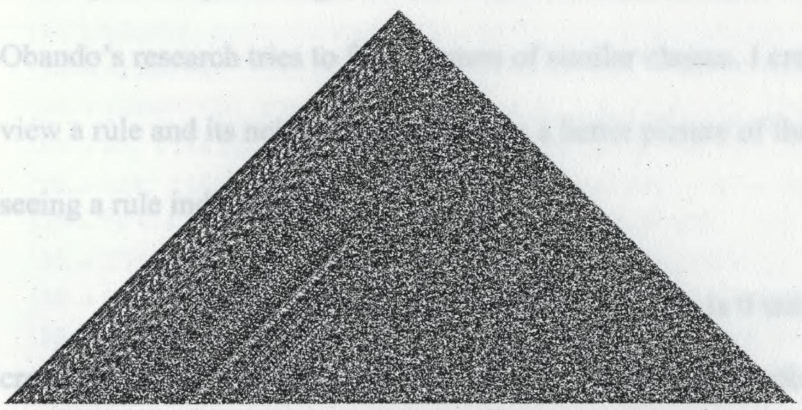


After using his program to organize the rules into a grid, Dr. Obando noticed something peculiar. Clusters of similar classes showed over the entire grid. After further investigation, there appeared to be a pattern when moving from one rule to the next on the grid. This paper is comprised of an explanation of the movement between rule spaces. I created a program that traverses the 4 billion rule space using Dr. Rodrigo Obando's designs and research.

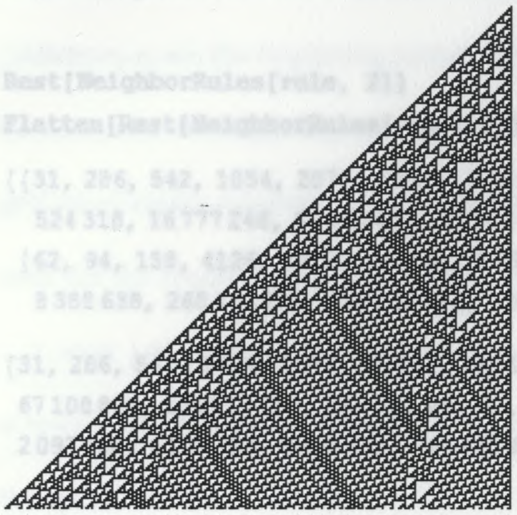
Example Rule:



Rule 30 from the elementary cellular automata rule space (256) is shown above. The first row of inputs is deterministic. The middle bit is flipped. Then, the rules are applied giving this seemingly random output. Below is the same rule, expanded.



The more a rule is expanded, the easier the patterns are to detect. Many computer scientists break down Class 3 into two parts, saying that Class 3 is completely random while Class 4 is complex (uniform parts and random parts). In Rule 110, we can see distribution of semi-uniform and random portions, so it is Class 4.



Investigation

The elementary rule space investigation of 256 rules proves easy enough. Scientist can look at each rule and its neighbors without too much trouble. However, the 4 billion rule space is not such an easy task. Viewing the entire space at once takes too much computer memory and requires huge screens. Most scientists condense the space to one rule at a time. Dr. Obando's research tries to find clusters of similar classes. I created a tool using Mathematica to view a rule and its neighbors. This paints a better picture of the surrounding rules rather than just seeing a rule individually.

First, I take a rule. By default, the rule is 0 unless specified by the user. Then, I create a list of the neighboring rules by using Rest[NeighborRules[rule, 2]]. The NeighborRules[*rule*, *radius*] function takes a rule and a radius. Then, it creates a list of lists with the rule and neighbors as subsets in each cardinal direction. The Rest[*expr*] function makes a new list out of all list components other than the first element. This leaves me with a list of lists of neighboring rules. The Flatten[*list*] function takes my list of lists and consolidates each inner list into a single complete list. The results are as follows:

```
Rest[NeighborRules[rule, 2]]
Flatten[Rest[NeighborRules[rule, 2]]]

{{31, 286, 542, 1054, 2078, 65566, 131102, 262174,
  524318, 16777246, 33554462, 67108894, 134217758}, {22, 26, 28},
 {62, 94, 158, 4126, 8222, 16414, 32798, 1048606, 2097182, 4194334,
  8388638, 268435486, 536870942, 1073741854, 2147483678}, {14}}

{31, 286, 542, 1054, 2078, 65566, 131102, 262174, 524318, 16777246, 33554462,
 67108894, 134217758, 22, 26, 28, 62, 94, 158, 4126, 8222, 16414, 32798, 1048606,
 2097182, 4194334, 8388638, 268435486, 536870942, 1073741854, 2147483678, 14}
```

Then, I pass that resulting list into the Map[*f*, *expr*] function as the expression along with my variables rule and str as the function. Map[] takes the rule and maps it to each piece of my list to give the following output:

```
Map[{rule → #, str} &, Flatten[Rest[NeighborRules[rule, 2]]]]

{{30 → 31, 2164260865}, {30 → 286, 2164260865}, {30 → 542, 2164260865},
 {30 → 1054, 2164260865}, {30 → 2078, 2164260865}, {30 → 65566, 2164260865},
 {30 → 131102, 2164260865}, {30 → 262174, 2164260865},
 {30 → 524318, 2164260865}, {30 → 16777246, 2164260865},
 {30 → 33554462, 2164260865}, {30 → 67108894, 2164260865},
 {30 → 134217758, 2164260865}, {30 → 22, 2164260865}, {30 → 26, 2164260865},
 {30 → 28, 2164260865}, {30 → 62, 2164260865}, {30 → 94, 2164260865},
 {30 → 158, 2164260865}, {30 → 4126, 2164260865}, {30 → 8222, 2164260865},
 {30 → 16414, 2164260865}, {30 → 32798, 2164260865}, {30 → 1048606, 2164260865},
 {30 → 2097182, 2164260865}, {30 → 4194334, 2164260865},
 {30 → 8388638, 2164260865}, {30 → 268435486, 2164260865},
 {30 → 536870942, 2164260865}, {30 → 1073741854, 2164260865},
 {30 → 2147483678, 2164260865}, {30 → 14, 2164260865}}
```

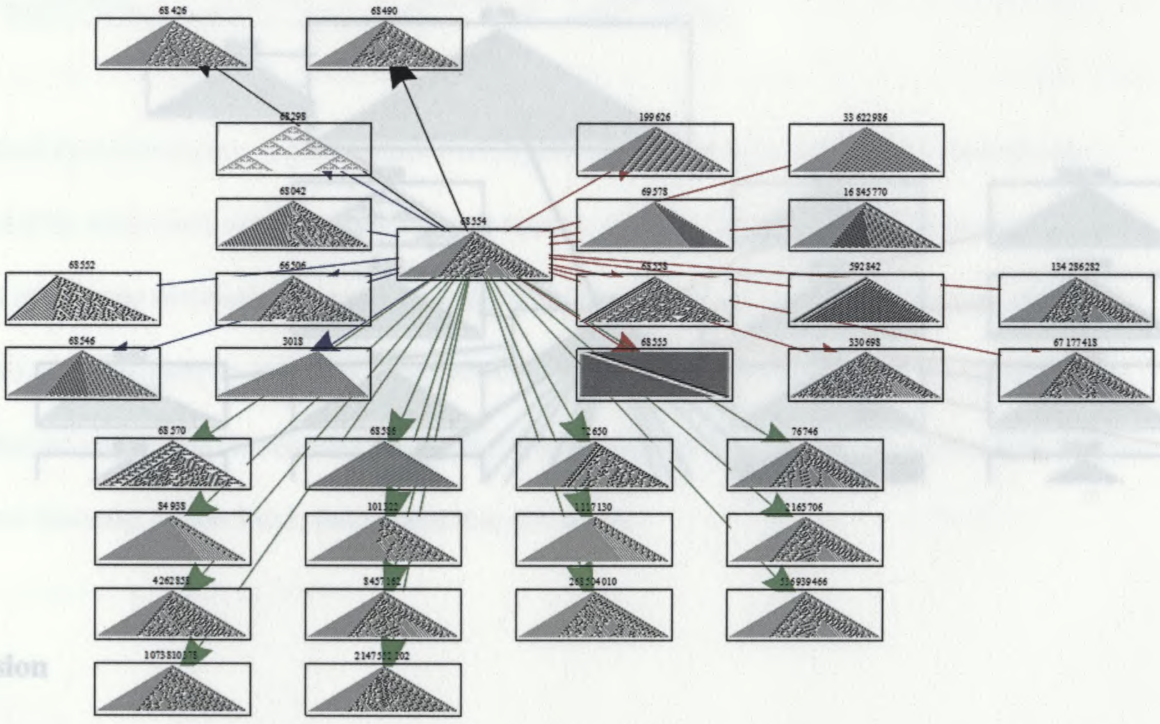
My mapped expression is now ready to be passed into the GraphPlot[{v_{i1} -> v_{j1}, v_{i2} -> v_{j2}, …}] function. A plot of the graph is generated in which vertex v_{i1} is connected to vertex v_{j1} etc. I also manipulate the GraphPlot options EdgeRenderingFunction, VertexRenderingFunction, ImageSize, MultiedgeStyle, and VertexCoordinates. EdgeRenderingFunction-> *g* specifies that each edge should be rendered with the graphics primitives given by g[{ r_i, …, r_j }, { v_i , v_j}, lbl_{ij}] , where r_i, r_j are the beginning and ending points of the edge, v_i , v_j are the beginning and ending vertices, and the lbl_{ij} is any label specified for the edge. For this, I pass Thick, getColorGraphicAttributes[#2[[2]], Rest [NeighborRules[rule, 2]]], and Arrow[#1]. I wrote the code for the function getColorGraphicAttributes[*rule*, *neighbor*]. It creates a Module[{*x*, *y*, …}, *expr*] of attribute colors red, blue, green, and black. These colors represent the color of the arrows in my program. Inside the module, I wrote a Do[*expr*, {*i*, i_{min} , i_{max}}] checking whether the specified rule is contained in the list of neighbors. If it is true, I return the color at that index. The VertexRenderingFunction -> *f* specifies that each vertex should be rendered with the graphics primitives given by $f[r_k,v_k]$, where r_k is the coordinate position where the vertex is being placed, and v_k is its name. I set the *f* to an Inset[*obj*, *pos*] where *obj* is the object to display and *pos* is the position to place it in the graphic.

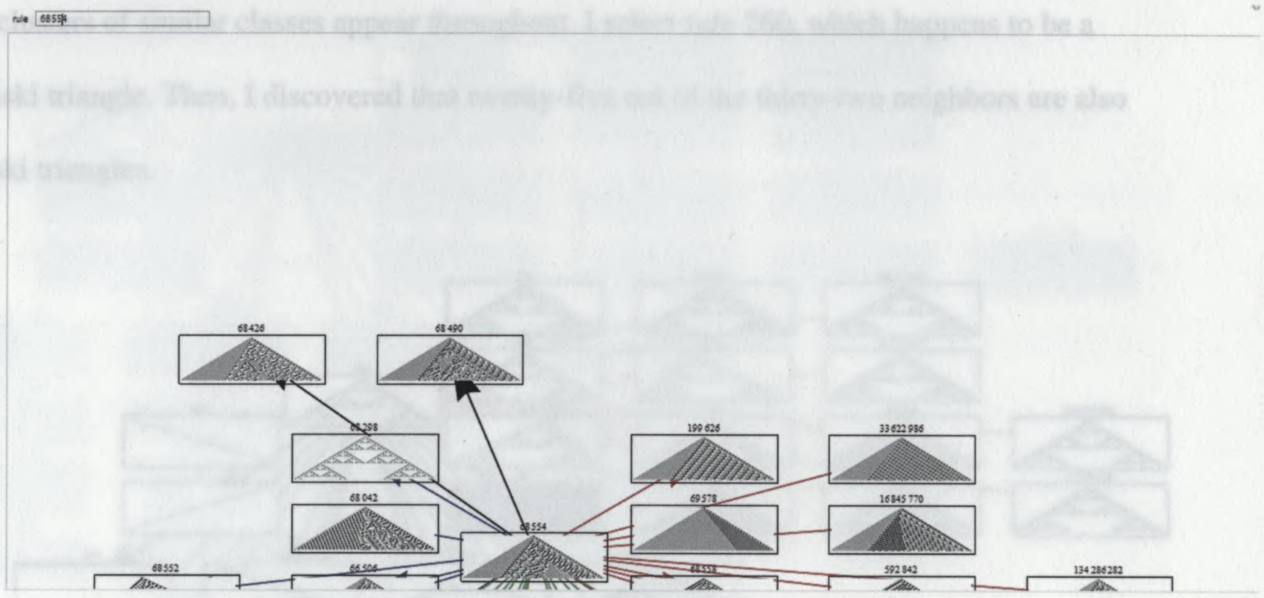
Because the Inset[] function is very intricate, it requires special attention, which is what I will discuss here. I started with the smallest portion of the code and worked outwards. I use the CellularAutomaton[*rule, init, t*] function passing the neighbor rules, one and zero for *init*, and 50 steps. This step creates an array of all the neighbors. I pass this array into the ArrayPlot[*array*] with a set aspect ratio of 1/3, a medium image size, and a label of the rule number above the image. The ArrayPlot gets passed as the *expr* of the Mouseover[*expr*, *over*] function. Mouseover represents an object that displays as *over* when the mouse pointer is over it, and as *expr* otherwise. For the *over* parameter, I pass a DynamicModule[{*x*, *y*, …}, *expr*] function. Here, I leave my first parameter empty but pass an EventHandler[*expr*, {"$event_1$" :> $action_1$, {"*event*$_2$" :> *action*$_2$,... }]. With EventHandler[]'s "MouseUp" attribute, I can capture mouse clicks and determine what they do. For the event, I want to draw an ArrayPlot[] of CellularAutomaton[] like before. Only, this time I will increase the image size. This combination creates a larger view when the mouse hovers over the cellular automaton. Since my neighboring rules are correctly drawn inside my Mouseover[], I pass that as the *obj* of my Inset[*obj*, *pos*] with my *pos* being #1, which represents the central rule.

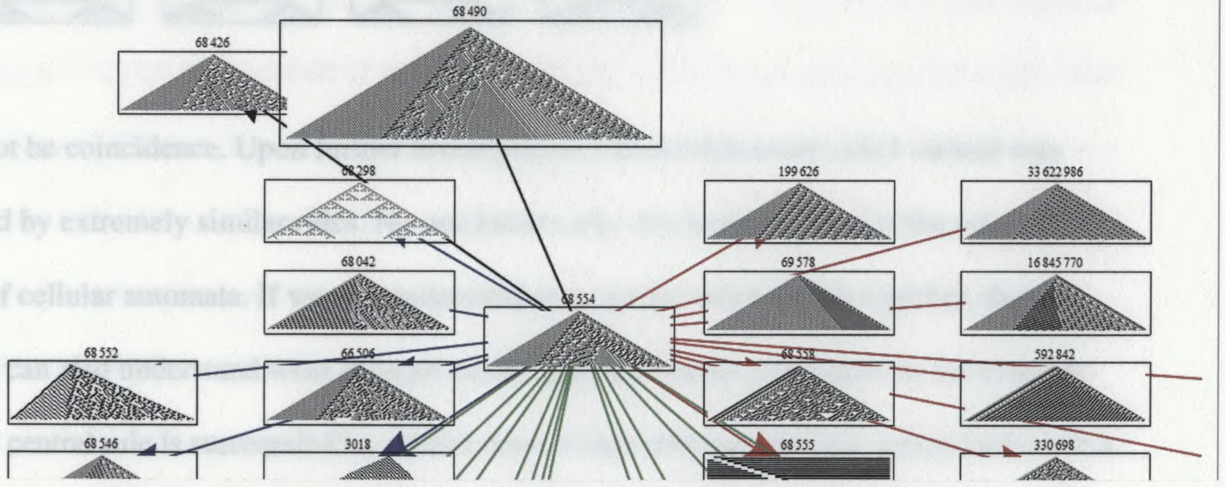
With no MultiEdgeStyling, my next task is to place the neighbors in their correct locations. As previously mentioned, Rest[NeighborRules[rule, 2]] produces a list of lists with each inner list representing the direction from which that neighbor appears. I pass my list of neighbors as well as my central rule to a functioned I created called placeNeighborsInDirection[*neighbors*, *rule*]. In this method, I created a Module[{*x*, *y*, ...}, *expr*] where the east, west, south, and north are my symbols {*x*, *y*, ...}. Each direction takes a similar algorithm to pass the correct coordinates for the neighbors.

For the x coordinate, I start at a predetermined value of six and add to that the xStep times the Floor[(Flatten[Position[neighbors[[1]], #]][[1]] − 1)/4]. The position of each east rule is flattened then subtracted by one to give you a value from zero to fifteen, since there is only a maximum of sixteen rules possible on any given side. This value is divided by four because I am making a possible four-by-four grid. The floor of that number is calculated and multiplied by my step value. For the y coordinate, I start at negative four and add the yStep times Mod[(Flatten[Position[neighbors[[]], #]][[1]] − 1), 4]. Similarly to the x coordinate algorithm, this algorithm takes the flattened position of the neighbor rules minus one and mods the difference by four. The result is multiplied by the yStep to populate cellular automata images upward in the output. To place coordinates in the west direction, I do the same process but negate the x value. To place coordinates in the south direction, I swap the algorithms placement, respectively. Also, I start my x coordinate at negative ten instead of negative four.

All of the steps taken up to this point have been to create a GraphPlot[] of cellular automata. Now, I can pass my GraphPlot[] as the *expr* of the Manipulate[*expr*, {*u*, …}, {*v*, …}, …] function. Here, I create my rule symbol with its initial value and a controlling mechanism for that symbol. I chose an InputField[*x*, Number] which represents an input field whose contents are taken to be a number. The number restriction feature works perfectly with my idea because no letters or punctuation can be typed except for a period and a negative sign. Cellular automaton rules cannot be negative and cannot have decimal points. To work around this, I made a Dynamic[] function to only accept the range from 0 to $2^{32} - 1$ by using a nested If[] statement.

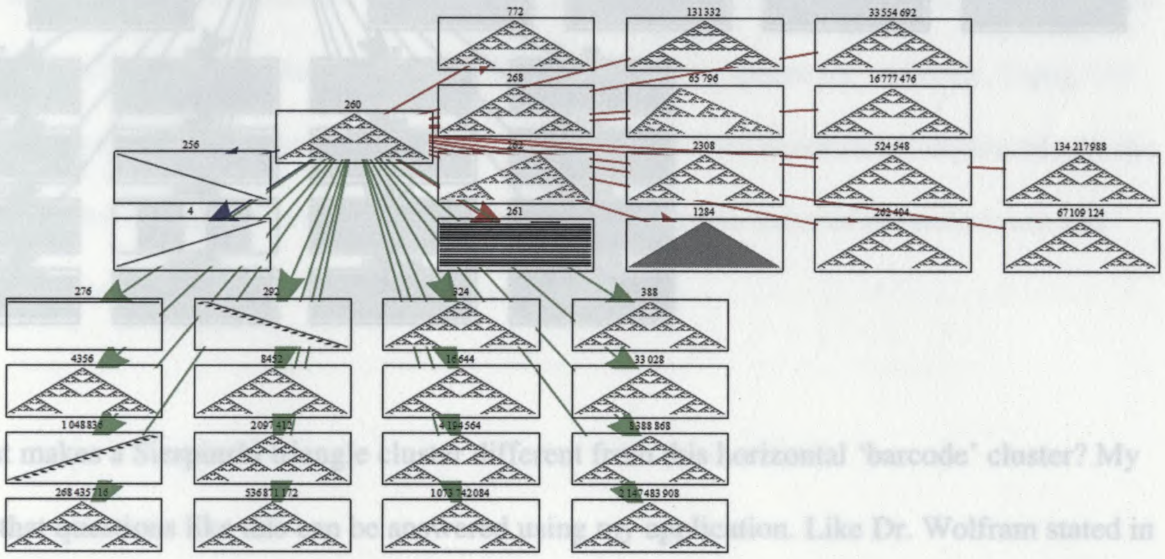




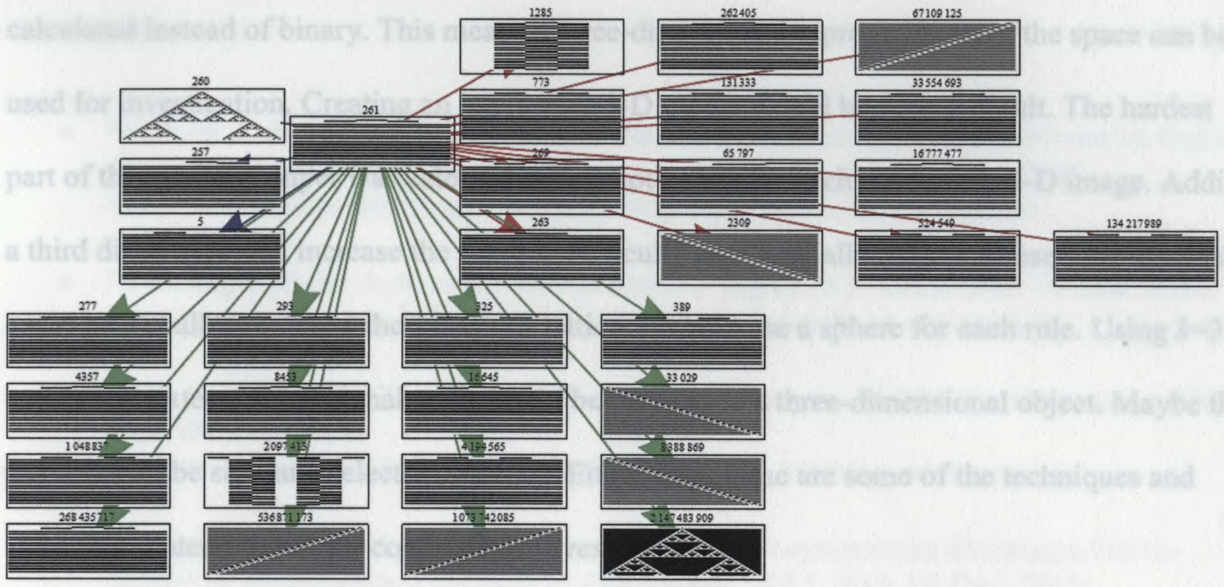


Conclusion

The application that I created can explore the radius 2 rule space in a unique and interactive way. No longer do scientists have to take one rule at a time in this rule space. With my application, the traversal of this space is quick and easy. Upon investigation through the space, clusters of similar classes appear throughout. I select rule 260, which happens to be a Sierpinski triangle. Then, I discovered that twenty-five out of the thirty-two neighbors are also Siepinski triangles.



This cannot be coincidence. Upon further investigation, I found that every rule I viewed was surrounded by extremely similar rules. No one knows why this happens. That is the magic question of cellular automata. If we can understand why similar rules cluster together, then maybe we can also understand what changes occur from one cluster to another. In the example above, the central rule is surrounded by similar rules. I then selected rule 261, which looks like a horizontal barcode. When I did, this image was the result.



So, what makes a Sierpinski triangle cluster different from this horizontal 'barcode' cluster? My hope is that questions like this can be answered using my application. Like Dr. Wolfram stated in *A New Kind of Science*, the methods of understanding cellular automata may be copied into other forms of sciences. What makes one cluster of biological cells muscle cells? What makes adjacent cells skin cells? The knowledge of cellular automata patterns can open doors to new types of sciences just like when Sir Isaac Newton and Gottfried Leibniz created calculus to explain physics. We may disprove randomness altogether. Deterministic systems like cellular automata should not produce random output. So, does this mean randomness can be calculated? Can randomness be predicted? If so, randomness is not truly randomness. Instead, it would be complex systems working in unison.

Future

I plan on continuing research in cellular automata for graduate work. Future improvements to this research should be in the *k*=3 area. Here, ternary output would be calculated instead of binary. This means a three-dimensional representation of the space can be used for investigation. Creating an interactive 3-D model would be very difficult. The hardest part of this research paper was calculating the plot points for each rule onto a 2-D image. Adding a third dimension will increase the plotting difficulty exponentially. Also, representing each rule could be a challenge. Since the space is a lattice, I would use a sphere for each rule. Using k=3 and r=2, a sixteen-dimensional space could be mapped to a three-dimensional object. Maybe the patterns will be similar to electron orbitals. Either way, these are some of the techniques and ideas that I intend to use for continuing my research.

References:

- "Cellular Automata (CA)." *Encyclopaedia Britannica*. Encyclopaedia Britannica Online Academic Edition. Encyclopædia Britannica Inc., 2013. Web. 09 Dec. 2013. <http://www.britannica.com/EBchecked/topic/862593/cellular-automata>.
- "Evolving Cellular Automata." *Everything is a Ghetto*. n.p. 2013. Web. 09 Dec. 2013. < http://www.thattommyhall.com/2013/07/04/evolving-cellular-automata/>.
- "File: Rule 30 2000 Generations." *PiAndWhippedCream*. WikiMedia Commons., 2012. Web. 09 Dec. 2013. <http://commons.wikimedia.org/wiki/File:Rule_30_2000Generations.png>.
- "John von Neumann." *Encyclopaedia Britannica*. Encyclopaedia Britannica Online Academic Edition. Encyclopædia Britannica Inc., 2013. Web. 09 Dec. 2013. <http://www.britannica.com/EBchecked/topic/632750/John-von-Neumann>.
- "Rule 110." *Wolfram MathWorld*. n.p. n.d. Web. 16 April 2014. <http://mathworld.wolfram.com/Rule110.html>
- "Rule 135." *Wolfrule.com*. n.p. n.d. Web. 09 Dec. 2013. <http://www.wolfrule.com/pages/rules/135.html>.
- "Stephen Wolfram." *Encyclopaedia Britannica*. Encyclopaedia Britannica Online Academic Edition. Encyclopædia Britannica Inc., 2013. Web. 09 Dec. 2013. <http://www.britannica.com/EBchecked/topic/862591/Stephen-Wolfram>.
- "The 256 Rules." *Stanford Encyclopedia of Philosophy*. Francesco Berto and Jacopo Tagliabue., 2012. Web. 09 Dec. 2013. < http://plato.stanford.edu/entries/cellular-automata/supplement.html>.
- "The Nature of Code." *Daniel Shiffman*. Daniel Shiffman., n.d. Web. 09 Dec. 2013. < http://natureofcode.com/book/chapter-7-cellular-automata/>.