

Regis University

ePublications at Regis University

Regis University Student Publications
(comprehensive collection)

Regis University Student Publications

Spring 2013

Universal Computation in the Prisoner's Dilemma Game

Brian Nakayama
Regis University

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Arts and Humanities Commons](#)

Recommended Citation

Nakayama, Brian, "Universal Computation in the Prisoner's Dilemma Game" (2013). *Regis University Student Publications (comprehensive collection)*. 596.
<https://epublications.regis.edu/theses/596>

This Thesis - Open Access is brought to you for free and open access by the Regis University Student Publications at ePublications at Regis University. It has been accepted for inclusion in Regis University Student Publications (comprehensive collection) by an authorized administrator of ePublications at Regis University. For more information, please contact epublications@regis.edu.

Regis University
Regis College
Honors Theses

Disclaimer

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

UNIVERSAL COMPUTATION IN THE PRISONER'S DILEMMA GAME

**A thesis submitted to
Regis College
The Honors Program
in partial fulfillment of the requirements
for Graduation with Honors**

by

Brian Nakayama

May 2013

Contents

1	The Iterated Prisoner's Dilemma Game	1
1.1	Introducing the Game	1
1.2	The Nash Equilibrium and Iterating the Game	4
2	Cellular Automata Theory	8
2.1	What is Cellular Automata?	8
2.2	Cool Cellular Automata	10
2.3	Universal Computation and Construction: Self-Replicating Machines	12
2.4	A Review on Literature for the IPDG CA	15
3	Playing the Prisoner's Dilemma Game with Memory	18
3.1	Giving Each Player "Instructions"	18
3.2	Constructing the Lattice	22
3.3	One Interesting Consequence	25
4	Patterns, Fractals, and Grouping	27
4.1	Creating Order Out of the Randomness	27
4.2	How Groups Form	34
4.3	Interesting Results	35
5	Proof of Universal Computation	39
5.1	Creating a NAND gate	39
5.2	Making a Computer	43
5.3	What this Proof Means	46
6	Social and Biological Relevance	48
6.1	Possible Implications	48

6.2 Conclusion	49
7 Further Ideas for Research	52
8 Bibliography	54

List of Figures

1	Axelrod's Inequalities for the PDG	3
2	Equations for the IPDG	6
3	Rumor Spreading in a Classroom	8
4	Different Wolfram Rules	10
5	Von Neumann's Universal Constructor	12
6	A Glider Gun and Gliders in the Game of Life	14
7	Composition of Strategy Types	19
8	Converting to Base 10 to Make a Strategy Number	22
9	A Few Different types of Neighborhoods	23
10	Odds of Having a Single Strategy in a Random Initial State	25
11	The Effects of the Second Neighbor	26
12	A 800x600 Random Initial Configuration	27
13	The Hamming Distance	29
14	4 Iteration Simulation After 750 Rounds	30
15	9 Iteration Simulation After 750 Rounds	31
16	Cooperating Instructions over Iterations	32
17	Point View	34
18	A Natural Gun	35
19	An Emergent Sierpinski Triangle	36
20	The Wave Gun	37
21	The Three Iteration Gun	38
22	Randomness Capable of Universal Computation	39
23	A Glider	39
24	A "Pin"	40

25	A Glider Eater	40
26	Collision Between Two Gliders	41
27	Glider Hooked on a Pin	41
28	A NAND Gate made out of IPDG Players	42
29	OR and XOR Gates	43
30	D Latch with a “0”	45
31	D Latch with a “1”	46
32	A Glider Gun without pins	47

Picture Credits

- Figure 3: Audience in classroom listening intently to speaker during meeting. Harless Todd, U.S. Fish and Wildlife Service. Web 08 Aug, 2012. <http://www.public-domain-image.com/people-public-domain-images-pictures/crowd-public-domain-images-pictures/audience-in-classroom-listening-intently-to-speaker-during-meeting.jpg.html>
- Figure 4(Left): Weisstein, Eric W. “Rule 90.” From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Rule90.html>
- Figure 4(Middle): Weisstein, Eric W. “Rule 30.” From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Rule30.html>
- Figure 4(Right): A Textile Cone Snail. 2005. Richard Ling. Web 08 Aug 2012. http://en.wikipedia.org/wiki/File:Textile_cone.JPG
- Figure 5: The Nobili-Pesavento Self-Replicator. 2008. Original automaton: Renato Nobili and Umberto Pesavento. Tape: Tim Hutton; Image: Golly screenshot. Web 22 Sept 2012. http://en.wikipedia.org/wiki/File:Nobili_Pesavento_2reps.png
- Figure 6: Created by Brian Nakayama using David Bahr’s Cellular Automaton Explorer 5.0
- All other images created by Brian Nakayama

List of Tables

1	The Prisoner's Dilemma Game in months	2
2	The Prisoner's Dilemma Game	2
3	Cleaning the Dishes PDG	4
4	3 Memory Transient Strategy for Tit for Tat	20
5	3 Memory Asymptotic and Transient Strategy	21
6	Average Percentage of Initial Cooperators	30
7	Average Percentage of Population Points	32
8	The NAND Gate	42
9	Properties of XOR	44

Acknowledgements

A Special Thank you to Dr. Bahr, Dr. Seibert, and Dr. Caulk for helping me through the long and arduous task of writing a thesis.

1 The Iterated Prisoner's Dilemma Game

1.1 Introducing the Game

Throughout our daily lives, we must make decisions on whether or not to cooperate with other people. In some situations it makes sense to cooperate no matter what. For example, when a policeman asks a person to pull over her car, complying helps her avoid a high speed chase; however, many situations also give us great incentives to not cooperate or defect. Whenever the dirty dishes pile up, a person can try to not clean them in the hope that his roommate will clean them instead. The latter example resembles the Iterated Prisoner's Dilemma game, and when run in a social simulation, such as one provided by a cellular automaton, many complex patterns emerge. When one assumes players act with memory of their opponents decision to cooperate or defect, these patterns include everything from fractals to Universal Computation, and the number of iterations of the game has great effect on the outcome of a simulation.

Created by Merrill Flood and Melvin Dresher in 1950 [3], the Prisoner's Dilemma game gets its name from the following scenario: Bob and Terry recently robbed a 7-11, but the police don't quite have enough evidence to prove it. Bob and Terry know that they'll both receive minor sentences of a few months if they don't confess. Aware of this, the police put Bob and Terry in separate interrogation rooms and offer them each a deal. If Bob or Terry gives evidence on the other to the interrogators, they'll get to leave without receiving any charges, but the person who doesn't confess will receive a harsher sentence of five months. If both provide evidence on each other, both will spend at least four months in prison. This scenario creates Table 1.

To read Bob and Terry's table choose either cooperate or defect for each. Trace the row and column to the appropriate cell. Bob's sentence will be to the left of the slash, and Terry's sentence will be on the right. Notice that one can switch Bob and Terry's place on

		Terry	
		cooperate	defect
Bob	cooperate	2 months/2 months	5 months/0 months
	defect	0 months/5 months	4 months/4 months

Table 1: The Prisoner's Dilemma Game in months

the table, and it still gives the same points for defecting or cooperating.

Now the example with Bob and Terry is a bit more specific than the general Prisoner's Dilemma Game(also referred to as PDG). In order to derive the general game, examine the previous example in terms of opportunity costs(we will use the same notation as Axelrod [3]). Let R be the reward for cooperating. The worst possible scenario costs Bob or Terry 5 months in jail. Using this we can assign R as the worst case imprisonment minus the cooperation imprisonment. This gives us $5 - 2 = 3$, and $R = 3$. Let T be the temptation to cooperate, P the lesser reward for mutual defection, and S be the suckers payoff. Similar to R , we can derive $T = 5 - 0 = 5$, $P = 5 - 4 = 1$, and $S = 5 - 5 = 0$. Now lets examine the latter in Table 2:

		Player 2	
		cooperate	defect
Player 1	cooperate	$R = 3/R = 3$	$S = 0/T = 5$
	defect	$T = 5/S = 0$	$P = 1/P = 1$

Table 2: The Prisoner's Dilemma Game

The numbers 5, 3, 1, and 0 may seem strict. Many events don't follow a ratio of 5, 3, 1, and 0, and this might lead one to think that many things won't reduce to the Prisoner's Dilemma game. Luckily, one doesn't need to use the numbers 5, 3, 1, and 0, rather one can use any numbers for T , R , P , and S that follow the following rules set by Axelrod in his paper on Cooperation [3]. Examine the equations in Figure 1.

Equation (1) shows that the temptation to defect outweighs the benefits of both cooperating; furthermore, both cooperating outweighs the punishment for both players de-

$$T > R > P > S \quad (1)$$

$$R > (S+T)/2 \quad (2)$$

Figure 1: Axelrod's Inequalities for the PDG

fecting, but if only one cooperates and the other defects, cooperating will return the lowest possible amount of points. Notice that (1) implies that as a group, both players cooperating gives more points than both players defecting: $2R > 2P$. Equation (2) ensures that as a group, cooperating has a higher combined return than one cooperating and the other defecting: $2R > S + T$.

Using these generalizations, one can represent any situation that matches the requirements of (1) and (2) with the PDG. In fact the great breadth of situations representable by the PDG might surprise the reader. Sometimes Ribonucleic acid conforms to the Prisoner's Dilemma Game when two viruses try to infect the same cell [11]. Vampiric bats can choose whether or not to regurgitate food for unfed companions, instead of keeping it all to itself [17]. In the U.S. Senate, cooperating to help other's pass bills competes with one's own self interest, also creating the Prisoner's Dilemma Game [15]. One could also argue that trench warfare also created the Prisoner's Dilemma Game, [4][2]; Axelrod shows that whether or not members from one trench kill members in another trench satisfy the equation $T > R > P > S$ or (1). Lymphoma also has to balance it's desire to propagate with keeping it's host alive in order to propagate to other organisms[13][14]. In economics, interactions amongst oligopolies also fit the PDG [10]. Many of the previous examples are also examples of the Iterated Prisoner's Dilemma.

Finding everyday situations that fit the prisoner's dilemma can add a fun way to apply the results found in this paper to one's own life. For example, every now and then the dishes in my sink will stack up to the ceiling, at which point both my roommate and I have the option to cooperate, and clean the dishes together, or we can wait to see if the other will

tackle the miserable task on her own. If we both clean the dishes, we'll both be moderately content. If one of us cleans all the dishes, the cleaner will feel under-appreciated while the roommate will have gotten his dishes done without contributing any work. If neither of us clean, we'll still have a pile of dirty dishes, but neither of us will feel under-appreciated. The Table 3 fits the equations (1) and (2) from Figure 1 on page 3.

		Roommate	
		clean dishes	relax
Author	clean dishes	5 utility/5 utility	-1 utility/8 utility
	relax	8 utility/-1 utility	1 utility/1 utility

Table 3: Cleaning the Dishes PDG

To avoid confusion, the values 5, 3, 1, and 0 shall be used in place of the variables T , R , P , and S throughout the rest of the paper.

1.2 The Nash Equilibrium and Iterating the Game

So why do game theorists bother with the PDG at all; what makes it interesting? Here we need to examine the Nash equilibrium, but first lets play with this. Examining Table 2 on page 2 breaks down two independent player's decisions. Assuming that both of these players wish to maximize their points, with no knowledge of what the opponent will do, each must choose to cooperate or defect. If player 1 cooperates, she can earn at most 3 points and at worst no points. If player 1 defects, she can potentially earn 5 points (the most points attainable by a single entity in the game) and at worst she will score 1 point. If player 2 chose to cooperate or defect based off a coin toss, player 1 would gain an average of 1.5 points whenever she cooperated. For defecting, player 1 would gain an average of 3 points (the maximum attainable from cooperating). If player 1 chose to defect every time she played the game, she could potentially win twice as many points.

The Nash Equilibrium enforces the belief that a player will always defect. The Nash

Equilibrium is the stable interaction that occurs when both players choose the dominant strategy. For one round of the Prisoner's Dilemma Game a player can make 3 or 0 points for cooperating or 5 or 1 point(s) for defecting. Thus, defecting is the obvious dominant strategy. When both players defect, we call it the Nash Equilibrium.

Up to this point we've examined the PDG and what a player should do if playing the game just *once*. If two entities play the game against each other more than once (retaining their points from each round), what a player should do changes significantly. We call this multi-round version of the PDG the Iterated Prisoner's Dilemma Game (IPDG for short). At first glance, one might assume that we should use the Nash Equilibrium. The Nash Equilibrium will lead to both players always defecting, similar to what one would do in a single round. The proof for this goes like this:

Assume that two players play the Iterated Prisoner's Dilemma Game for N rounds. If we assume they always defect, like the Nash Equilibrium for one round, we're done. Instead, let us assume that both players cooperate initially. On the N th round, or the last round, one of the players will defect in the hopes of maximizing his points. The other, equally aware of this dominant strategy will do the same. Now look at the $N - 1$ round. One of the players will defect on this round, again hoping to maximize his points. The other does the same. This repeats for the $N - 2$ round, $N - 3$, $N - 4$... and 1st round. Recursively, each player defects every round they assume the opponent might cooperate. Thus we have shown that the Nash Equilibrium for the Iterated Prisoner's Dilemma Game is to defect every round.

At this point let's examine mathematical expressions that describe the possible amount of points a player can get. Let a and b be the moves made by two players in the game, and $g(a, b) \in \{5, 3, 1, 0\}$ be the points a earned from one game. Examine Figure 2. The function in equation (3) represents the possible points that a single player gets playing this game. If $g(n)$ is a constant, $c \in \{5, 3, 1, 0\}$, then $\sum_{i=0}^n g(n)$ simplifies to $c \cdot n$. Using this

we can derive equation (4), the points gained if both players always cooperate, and equation (5), the points gained if both players always defect. Notice that the cooperative function (4) gives triple the amount of points for both players than the one achieved through the Nash Equilibrium, (5). The last two equations here represent the maximum (6), and minimum (7) points that one can achieve by playing a special rule called Tit for Tat.

$$f(a, b, n) = \sum_{i=0}^n g(a, b) \quad (3)$$

$$R(n) = 3 \cdot n \quad (4)$$

$$P(n) = n \quad (5)$$

$$T_1(n) = 3 \cdot (n - 1) + 5 \quad (6)$$

$$T_2(n) = 5 + (n - 1) \quad (7)$$

Figure 2: Equations for the IPDG

Tit for Tat is a simple, yet powerful rule for playing the Iterated Prisoner's Dilemma Game. The strategy of Tit for Tat always cooperates on the first round. After this, Tit for Tat always copies the previous move of the opponent. We can see this clearly in equations (6) and (7). Equation (6) shows the points gained if one cooperates every single round, and then defects. Interestingly enough, this strategy gives the most points, but it does not follow the Nash Equilibrium. Equation (7) shows the points a player will receive if she uses the Nash Equilibrium against Tit for Tat. The first defection has a high reward (5 points), but Tit for Tat essentially punishes the player by defecting in response to every round that the opponent defected. In this way, playing against Tit for Tat encourages cooperation.

Robert Axelrod made Tit for Tat practically famous in his book and papers on the *Evolution of Cooperation* [3][4]. Axelrod created several different computer tournaments that pitted different strategies for the IPDG against each other, and in almost every different environment he used to pit the strategies against each other, Tit for Tat prevailed as the clear victor. Axelrod writes: "What can be said for the empirical success of TIT FOR TAT is

that it is a very robust rule: it does well over a wide range of environments. Part of its success might be that other rules anticipate its presence and are designed to do well with it. Doing well with TIT FOR TAT requires cooperating with it, and this in turn helps TIT FOR TAT.” (53) In only one environment did Tit for Tat fail or perform worse than other strategies. We call this environment, based off of competing neighborhoods and territory, Cellular Automata.

2 Cellular Automata Theory

2.1 What is Cellular Automata?

Now imagine yourself in a classroom. Everyone sits facing the teacher, spaced about uniformly apart. All of the sudden student *A* hears a rumor about the professor and decides to tell his nearest neighbors, students *B*, *C*, and *D*. These three students then tell all of their neighbors, and so on. Pretty soon everyone has heard the rumor, including all of the students sitting across the room (Figure 3). If we examine this example more closely we can define how rumors spread. First we have a lattice of students, and we know that each student can only whisper to a student close by. We call these students within the whisper range *neighbors*. Next, we can define two states for the students: those who know and those who don't. We also know that once a student learns a rumor she will tell all her neighbors, creating a rule for how each student updates their state. We have formalized this classroom; we could now program a spatial simulation for how rumors spread in a classroom on a computer, also known as a *Cellular Automaton*.



Figure 3: Rumor Spreading in a Classroom

In order to be a cellular automaton(abbreviated as *CA*) a simulation has distinct states like a 1 or a 0 and *know* or *don't know*. The states need not be finite, one could have an infinite number of discrete states. It must also have a uniform rule for updating states based off of neighbors or a combination of one's neighbors and oneself. Some other rules generally apply such as synchronicity, where all cells or nodes must update at the same time. Some Cellular Automata do not follow all of these rules. For example, certain simulations may benefit from asynchronous updating or different rules for updating states based off of location [19].

Using the previous classroom example lets try to create our own cellular automaton! First we can define each student as a node with the states 1 for knowing the rumor or 0 for not knowing the rumor. Each node has one or more links to another node within whispering distance. Every time cycle, we update each node in the network with a simple rule: if a node with state 0 is linked to a node with state 1, it will update its status to a 1. If the node is already in state 1, do nothing. And we're done! Program this into a computer, add some fancy graphics for your students if you want, and you'll have your own cellular automaton.

If we wanted to make this even simpler we can assume that our classroom exists on a grid or a lattice instead of a network. Often in kindergarten through 12th grade most students spend their time in a setting similar to a grid, where each has her or his own desk spaced uniformly apart from all the other students. This makes our simulation even easier to visualize and program since it creates a uniform neighborhood for each of our nodes. For example, we can assume that each student can only whisper to a student to the north, south, east, or west of them. Now our students look and interact more like square cells.

This example highlights the meaning in the words Cellular Automata. The *cellular* in Cellular Automata stands for the study of simulations made up of either nodes and networks or *cells* and lattices. The *automata* part describes how each cell acts locally, as if it were its own small processor.

2.2 Cool Cellular Automata

The simulations of Cellular Automata span many subjects both useful and philosophical. Wolfram first pioneered the subject with his work on one dimensional, two state cellular automata[19][21]. Based off of all the possible states a cell could take on according to its previous state and its two closest neighbors Wolfram separated the different one dimensional CA into 256 different “rules”. Examine Figure 4 below. In each of these simulations, the time steps are shown in descending order from top to bottom, so that the top line will be $t = 0$ and the n th line will be $t = n$.

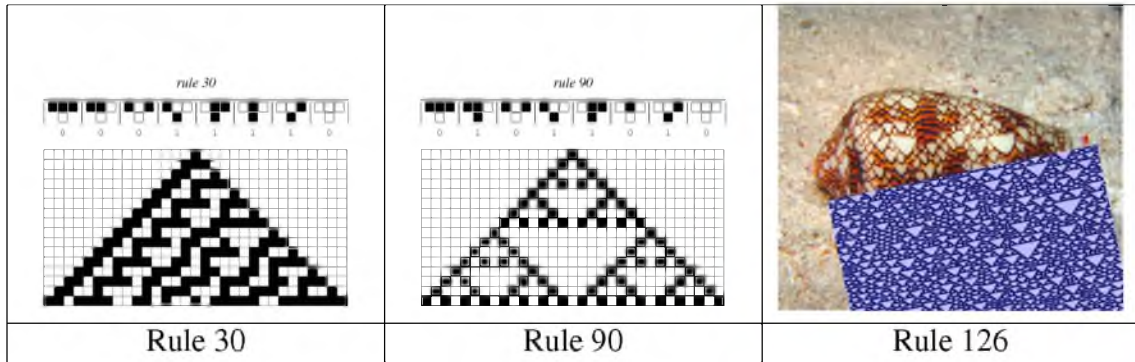


Figure 4: Different Wolfram Rules

Although Wolfram’s rules may seem simple at first glance, an in depth analysis of their behavior could fill thousands of pages. For now, take in the depth and breadth of knowledge that Wolfram’s rules cover. On the left, Wolfram’s rule 30 can simulate random numbers. This means that one can use rule 30 or a different CA for cryptography or applications that simulate probabilities. In the middle, rule 90 creates a famous fractal known as the Sierpinski Triangle. Rule 90 also represents the odd numbers in the Pascal Triangle with an “alive” or black cell(evens are white). Finally on the right, rule 126 emulates the colors made by the proteins of the shell shown in the picture. Although its not shown here, rules 110, 124, 137, and 193 exhibit Universal Computation. Universal Computation means that these rules have all of the properties of a Universal Turing Machine, a mathe-

mathematical model for a computer. We will examine how simple interactions amongst squares can do everything a computer can do in the next section.

Other CA are made for the purpose of simulating social interactions. Ofcourse, Axelrod used CA to test strategies of the IPDG against each other; however, this represents just one focus of research amongst many within CA theory. A much simpler cellular automaton called Majority Rules can also simulate how people change and react based on their neighbors. In fact Majority Rules can be implemented in many different ways to show what a person will do based on what her neighbors will do. As the title of the simulation implies, a cell usually changes its state to whatever a majority of its neighbors are, but one can also insert random changes in state, barriers to change, and other variables into the simulation. Using Majority Rules, one can see how things like political opinions [6][7], obesity [5], or happiness [8] spread amongst a population.

Just as one can find many situations which fit the Iterated Prisoner's Dilemma Game, one can also find countless examples of things in nature that can be simplified or understood through a Cellular Automaton. One can use CA to simulate snowflakes, fluid flow, neural activity, waves, ecosystems, diffusion, and more [19]. The simulations presented in this thesis focus intensely on only the IPDG. If one wants to learn more about Cellular Automata in general, I suggest reading *Cellular Automata: A Discrete View of the World* by Joel L. Schiff.

Finally there's also some CA that leave the realm of immediate usefulness, but tell us a lot about the possible origin of life and perhaps thought. John Conway's Life, Brian Silverman's Wireworld, and John Von Neumann's Self-Replicating Machine exhibit these philosophical properties, and one must understand these concepts in order to analyze the computation within the Iterated Prisoner's Dilemma Cellular Automaton.

2.3 Universal Computation and Construction: Self-Replicating Machines

John Von Neumann was the first to use CA to create a model for a self-replicating machine [19]. Now the important question, what does it mean to be a self-replicating machine? Imagine a machine which can create a copy of itself by piecing together resources found around it. This machine contains not only the tools to build itself, but a method to copy instructions for creating itself into its copy. In many ways, self-replication resembles the process of reproduction, a key aspect of living organisms. This definition of self-replication may even sound similar to the process of mitosis in some basic cells. Figure 5 shows Von Neumann’s Universal Constructor reproducing itself.



Figure 5: Von Neumann’s Universal Constructor

Yet self-replication by itself fails to account for some of the other aspects of organic life. For example, many organisms move, gather resources, have a metabolism, and do a wide array of things outside of just existing and replicating. Thus Neumann’s CA also had another special feature, Universal Computation. Something with Universal Computation, also known as a Universal Turing Machine, can compute any algorithm. An algorithm of course is a “step-by-step procedure for reaching a clinical decision or diagnosis” [1]. Some examples of algorithms are simple things like breathing, eating, and hunting in addition to running a computer. Each of these actions requires a procedure that has the potential to terminate or reach a decision. In other words, each action has a potential halting state, even if each action does not always definitively halt. Thus with the ability to run any algorithm or

to make any decision, a self-replicating machine can do more complex things that resemble organic life. It can even evolve if it has programmed a mechanism for doing so, like meiosis.

A Universal Turing Machine can simulate other things that exhibit Universal Computation, including self-replicating machines. For this reason, Universal Turing Machines are the holy grail of CA theory. As one might imagine, they can be hard to understand and even harder to find, but the philosophical consequences make them well worth the search. If we look at Neumann's CA as a metaphor for the complexity of the human brain we can see how a large network of small neurons can create "complex behavior" [20] intrinsically without any other unknown variable or force.

Unfortunately, John Von Neumann's CA has a total of twenty nine states making it a little cumbersome to understand. John Conway's Game of Life on the other hand has only two states, *alive* and *dead*, and it also exhibits Universal Computation. The Game of Life works on a two-dimensional, square lattice, and it has the following simple rules [9]:

- Rule 1: If a cell is dead at time t it becomes alive at $t + 1$ if exactly three of its neighbors were alive at t . Furthermore, it stays alive at $t + 2$ if and only if it has two or three alive neighbors.
- Rule 2: If a cell is not alive, and it does not have three alive neighbors at t , it will remain dead at $t + 1$. Also, if a live cell has less than two or more than three neighbors alive it will die due to exposure or overcrowding respectively.

Upon an initial read it may seem that this Cellular Automaton doesn't do very much; however, with many cells the Game of Life can achieve marvelous patterns on a holistic level. One of the most important of these patterns is the Glider Gun in Figure 6 below.

The Glider Gun periodically creates a small formation of alive cells called a Glider every 30 time steps. These gliders can travel in any diagonal direction, and in Figure 6

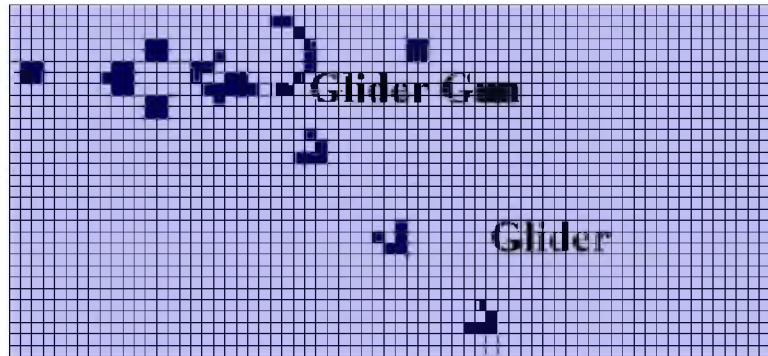


Figure 6: A Glider Gun and Gliders in the Game of Life

these Gliders travel south east. What makes these gliders so special is that they can convey information by their presence or absence. If we count a Glider as a binary 1, we can create strings of information in the form of Gliders spaced out carefully. This means that the Glider Gun spits out a constant stream of 1's.

Even more interestingly, when two Gliders collide together in just the right way, they erase each other turning into dead cells. This allows one to use a Glider gun to invert a string of gliders that represents binary information, creating a *NOT* gate. Using similar methods, one can also create a boolean *AND* and an *OR* gate. Using these logic gates in conjunction with the special properties of Glider Guns and Gliders themselves, one can also create registers or *memory*. All we need to compute any binary algorithm is *memory* and the *AND*, *OR*, and *NOT* gates. Since we can compute any algorithm we know that Life has Universal Computation.

Conway has also shown that one can create and destroy Glider Guns along with other interesting and useful patterns. So, one can potentially construct a machine with a high level of complexity, as the collision of gliders seems capable of making an innumerable number of patterns. Though it may not seem immediately apparent, one can even simulate Von Neumann's CA.

Amazingly enough, with only two states and relatively simple rules, one can create

patterns that act like self replicating “animals”. Obviously, with infinite time steps and infinite space, these organisms are bound to appear naturally, albeit very far and few between. The implications this model has for organic life are immense, as Conway says:

“Analogies with real life processes are impossible to resist. If a primordial broth of amino-acids is large enough, and there is sufficient time, self-replicating moving automata may result from transition rules built into the structure of matter and the laws of nature.” [9]

For Cellular Automata enthusiasts, the Game of Life almost represents a metaphorical proof for the origin of life or complex reproducing entities. The game of Life most certainly proves this point by example through its various holistic configurations. Wolfram also shows the potentiality of self-replicating Machines in Class 4 CA[21].

In addition to the CA already mentioned, there exists CA aesthetically closer to a computer than a biological entity. A good example of this is Wire World [19]. Wire World focuses on creating cells that move like electronic signals on a wire; however, the cellular automaton lacks any way to place new wire cells, and in that way it can only simulate replication as an algorithm inside itself. In other words, a Turing machine made in Wire World can compute any algorithm, but just like modern computers, it lacks a method for building glider guns or other important patterns onto itself.

Now that we have an understanding of this powerful tool and the Iterated Prisoner’s Dilemma Game, we need an elegant way to combine these two theories to make our own cool Cellular Automata simulation.

2.4 A Review on Literature for the IPDG CA

A couple of recent studies (in addition to Axelrod’s old one) have created their own PDG CA experiments. First Nowak and May studied simulations without memory, where the

agents either always defected or cooperated [16]. Similar to this thesis, Nowak and May created interesting results by tweaking a few of the variables in the Prisoner's Dilemma Game. Specifically they set $R = 1$, $T = b$ ($b > 1$), and $S = P = 0$. This does not necessarily violate the definition of the PDG, $T > R > P > S$. It assumes that $P = \varepsilon$ where ε is so small it essentially acts the same way as 0. Then by experimentally setting b to different values, Nowak and May found spacial chaos with a variety of different objects similar to the Game of Life: rotators, gliders, blinkers, and eaters.

In their paper they implemented a network of cooperators and defectors in a square lattice similar to the one used in this thesis. They tried hexagonal lattices, but it didn't necessarily add to the experiment (everything one could find in a hexagonal lattice, one could probably find in the square lattice). Similarly in this paper, one lattice will create robust results.

Next, Nowak, May and Sigmund created another simulation closer to the one's we will explore and Axelrod's original computer tournaments [17]. This time around they focused on the rules Always Cooperate, Always Defect, Tit-for-Tat, and Pavlov. The Pavlov rule switches from cooperation to defection or defection to cooperation only when it makes a poor amount of points the turn before (either S or P). Pavlov created some semi-interesting results; however, unlike Tit-for-Tat, its strategy leads to cooperating every other move with a rule that always defects, essentially halving its own points while rewarding defectors. Overall, their research concluded that cooperation persists as a viable strategy in a simulation focused on spatial structures; furthermore, it can exist side by side with other strategies that defect, creating diversity.

Recently Pereira and Martinez [18] also investigated the Prisoner's Dilemma game using a similar method as Nowak and May, except Pereira and Martinez focused on a Pavlovian Evolutionary Strategy transition rule, where a cell will change from a cooperator to a defector or vice versa similar to the Pavlov rule mentioned in the previous paragraph. This

is in contrast to what they call a Darwinian Evolutionary Strategy that works essentially the same way as Nowak and Mays' and mine. (The rule for transitions is in the next section.) Again, by changing the value of T , the temptation for defection, Pereira and Martinez also found emergent complexity in their random simulations with objects such as gliders and "fingers". They describe the point at which complexity emerges as the quasi-regular phase.

Unlike the previous research, this thesis focuses on the number of iterations per time step on a lattice of cells that all have memory. In fact, my research more closely resembles one of the rules in Axelrod's original computer tournaments called NYDEGGER [4]. This rule used a look up table of the opponent's previous three moves in order to pick a response for the next move. One can extend the look up table concept to create a method for defining all kinds of cells with different strategies as a number.

3 Playing the Prisoner's Dilemma Game with Memory

3.1 Giving Each Player “Instructions”

What if one wants to implement the Iterated Prisoner's Dilemma Game (IPDG) in a Cellular Automata (CA) environment without individually coding hundreds, maybe thousands of strategies for playing the game? Individually coding thousands of rules is not pretty; put succinctly, no one wants to see that mess. Luckily, I devised a general method that has agents playing the IPDG using different strategies based off of the opponent's memory.

First, I must introduce some rules that act on the players:

- Rule 1: Players only remember opponent's moves, not their own. They lack introspectiveness.
- Rule 2: Players can remember more than one move (called a history).
- Rule 3: Players have a defined action for every possibly history of opponent's moves.

These rules allow us to create agents that deterministically react to any history of an opponent's moves. By deterministic, I mean that any set of actions will cause a predictable reaction. So if our agent cooperated with an opponent who decides to defect, cooperate, and then defect again, I would expect that agent to always cooperate with any opponent who decides to defect, cooperate, and then defect. Why don't our players react to their own moves? In many social situations with a low amount of iterations, players may pay more attention to the opponent's actions than their own. As an example, Tit for Tat does not take into account its own moves, and it doesn't need to. Intrinsically, it follows the Golden Rule; however, other strategies may benefit from knowing their own moves. A strategy that defects and essentially “pisses off” another strategy may benefit from remembering its own moves, but for simplicity's sake, this simulation leaves out this device.

Say our agent could also remember its own moves; if so, it could also use every deterministic strategy possible for two agents in the IPDG. If we modify rule 1 (Make it this: players can remember all moves), this becomes clear. By rule 3, let $f(H) : H \rightarrow \{cooperate, defect\}^*$ be a function that maps a history to the decision to cooperate or defect. Every deterministic strategy will make the same move, either cooperate or defect, each time for a specific history (no matter how complex). Thus we can convert every deterministic strategy into a well defined function, $f(H)$.

At this point we can look at groups of strategies playing the IPDG as sets. First we have the set of all un-deterministic strategies, U , that don't rely on memory, like the random strategy. Second we have the set of all deterministic strategies, D , that do rely on memory. Together they make the set of all strategies, A , and $U \cup D = A$. The set of all strategies that remember only the opponent's moves, D' , is a subset of D . Furthermore, all strategies that remember only $n \in \mathbb{N}$ moves, D'' , is a subset of D' . All together, this forms the composition $D'' \subset D' \subset D \subset A$. The Venn diagram in Figure 7 below shows this composition with $n = 3$.

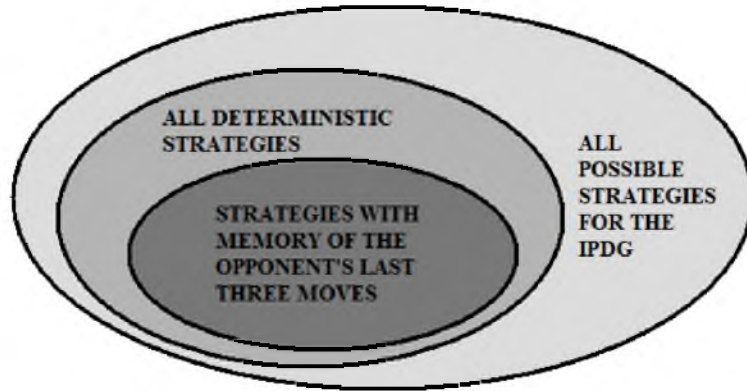


Figure 7: Composition of Strategy Types

In our CA simulations we'll focus on strategies that can remember the past three moves of the opponent. Using a number convention similar to those used to define simple

2D CA by Wolfram [21], one can describe a strategy using binary by assigning cooperation the value 1, and defection 0. Table 4 uses Tit for Tat to demonstrate the numbering scheme.

















Tit For Tat	
Opponent's History	C / D
00 	
00 	
01 	
01 	
10 	
10 	
11 	
11 	

Table 4: 3 Memory Transient Strategy for Tit for Tat

Table 4 will work perfectly when it has played at least three iterations of the PDG with an opponent. Notice that Tit for Tat only cares about the most recent move, copying it, but it ignores the previous two moves highlighted in yellow. This means that Tit for Tat has a memory of 1 iteration, which causes a repeating pattern of 1, 0, 1, 0, etc. Notice that this strategy lacks information for what to do after only two iterations, one iteration, and if it should cooperate or defect for its first round. After a strategy has played more than three iterations (four iterations, five, etc.), it can always use the past three moves of the opponent to determine what to do, thus Table 4 describes the long term,

asymptotic behavior. In order to define the initial behavior we need something that describes the *transient*.

Transient or initial instructions are only used during the beginning and then never used again. However, one should not disregard the transient interactions, because they deterministically influence the rest of moves that two strategies will make. The instructions that take over for all moves past 3 are the asymptotic. Similar to the word for asymptote, the asymptotic controls the long term trends of a particular strategy. Table 5 contains both the asymptotic, and the transient.

Strategy 10818

Opp. H.	C / D
0000	0
000 1	1
00 1 1	0
00 1 1	0
0 1 1 1	0
0 1 1 1	0
0 1 1 1	1
0 1 1 1	0
1 1 1 1	0
1 1 1 1	1
1 1 1 1	0
1 1 1 1	1
1 1 1 1	0
1 1 1 1	1
1 1 1 1	0
1 1 1 1	0

Table 5: 3 Memory Asymptotic and Transient Strategy

Table 5 defines what to do for the first, second , and third moves, as well as what to do for every move after that. The yellow bit’s position defines what iteration the green bit responds to. Notice that only one row has “000 1”, since this row represents the first move. Two rows start with “00 1”, since the opponent has two optional histories for the second move highlighted in blue. Similarly four rows start with “0 1” for the third move, and eight rows start with “1” for the eight possible histories for moves four and beyond. The first row in Figure 5 remains a “0”, and this bit isn’t used. The green bits, by instructing a cell what to do in certain situations(cooperate or defect), creates “instructions” for how the cell should act.

To get the strategy number, or *10818* in this example, convert the left column of Table 5 from binary to base 10 (0000 → 0, 0001 → 1, 0010 → 2, ..., 1111 → 15). The number in the left column represents the position of the bits that define the strategy from rightmost (0) to leftmost (15). Once we order our bits we get 0010101001000010. Convert this number to base 10 (Figure 8) in order to get a strategy number. Using this method we can represent each possible strategy with its own unique number.

Creating a strategy number creates an easy way to implement many different deterministic strategies, and it has advantages when displaying different strategies in the CA simulation. When creating a random simulation, the computer can pick a random number that maps to a strategy, and then the computer can also assign each strategy a unique color using a one-to-one function that maps numbers to colors. The simulations analyzed below will use many different colors to display the vast

number of strategies.

$$\begin{aligned}
& 0 \cdot 2^{15} + 0 \cdot 2^{14} + 1 \cdot 2^{13} + 0 \cdot 2^{12} + 1 \cdot 2^{11} + 0 \cdot 2^{10} + 1 \cdot 2^9 + 0 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot \\
& \quad 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
& \quad = 10818
\end{aligned}$$

Figure 8: Converting to Base 10 to Make a Strategy Number

Now lets test our instructions for strategy *10818* against *Tit for Tat* or strategy *43690*. Refer to Equation (3) in Figure 2 on page 6 for a description of the functional notations. Let $a = \text{strategy } 10818$ and $b = \text{strategy } 43690$. Both a and b cooperate during the first round, so $g(a, b) = g(b, a) = 3$, and both a and b get 3 points. During the second iteration b (*Tit for Tat*) will cooperate again since a cooperated; however, this turn a will defect (notice that a always defects on the second iteration). On the third iteration, b will defect since a defected, and a , referring to its two memory asymptotic instructions for the history 11, will also defect. For the fourth iteration, b will defect again, and a will refer to its transient instructions for the history 110, and it will defect again. Every iteration after this, both a and b will defect. Notice, in this example $f(a, b, n) = 8 + (n - 2), n > 1$. This is very close to the results of Equation (7), meaning that strategy *10818* barely makes more than the smallest payoff possible from *Tit for Tat*.

So how many different strategies can one make with a memory of three moves? Ignoring the first row(since this row doesn't hold any information), we have 15 rows that contain either defect, 0, or cooperate, 1. Since each row has two options, we have 2^{15} different possible strategies or 32,768. Due to the method of creating a strategy number, strategies will have all the even numbers from 0 to 65,536.

3.2 Constructing the Lattice

Axelrod first implemented the IPDG in a CA simulation with the goal of testing *Tit for Tat*'s robustness [4]. Interestingly enough, he found that *Tit for Tat* didn't excel in the CA

simulations as well as it did in other simulations, and more often a rule called NYDEGGER often excelled. NYDEGGER used an algorithm based off of the opponent’s previous three moves that sounds similar if not identical to some of the strategies created by our simulation. In order to understand why some rules thrive, one must first understand the exact environment in which a rule fights for survival.

The simulations used in the following chapters implement a Moore Neighborhood on a 2D lattice. Figure 9 shows some of the different kinds of neighborhoods that one may use for a cellular automaton. Keep in mind that Figure 9 represents just 4 common neighborhoods, whereas the number of lattices and neighborhoods are theoretically infinite. To an extent, a scientist can arbitrate the choice of what type of neighborhood to use. Each cell, representing a player, must interact with multiple cells; furthermore, a neighborhood must restrict a cell’s interactions such that it can only interact with cells nearby. Martinez and Pereira attained similar results when finding an ideal “phase” in their paper “Pavlovian Prisoner’s Dilemma-Analytical Results, the Quasi-Regular Phase and Spatio-Temporal Patterns” [18], and they used a 1-D lattice instead of the 2-D lattice we will choose here.

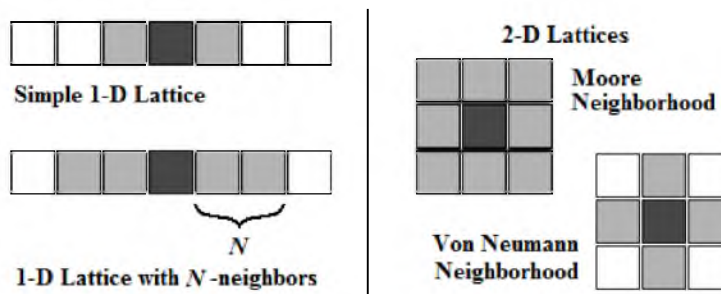


Figure 9: A Few Different types of Neighborhoods

In addition to choosing a lattice, we must also have a simple rule that determines how a cell changes states. Referring again to Axelrod’s initial setup, our rule picks the next state/strategy by changing a cell’s current state to that of the neighbor with the highest points [4]. If a neighbor does not have a higher point value, then the cell will not want to

change states. Therefore, in the case of a tie between a cell and its neighbor, a cell will keep its own rule. In order to keep our simulations deterministic and symmetrical around the square, the rule ignores all other ties. In the case of a tie between two neighboring cells, a cell will retain its own rule. Fortunately, this circumstance does not arise often in simulations, and has no noticeable effects; furthermore, keeping the rule for state changes deterministic offers more control in determining the outcome of a particular initial setup of cells.

From this point on, a random initial state refers to a $n \times m$ lattice of cells that have one of the 2^{15} deterministic strategies. One must now wonder, how do we determine that a lattice has at least one cell for each strategy? The lower the odds of representing a certain cell, the more inconsistencies one should expect between the results of different random initial states. Thus, one needs a function that will compute the odds of a single particular strategy existing in a random initial state. Figure 10 contains this function, equation (10). First, examine the case where our lattice has one cell. This one cell will randomly take one of our 2^{15} strategies, and the odds of having any one strategy is equation (8). Next, examine the case where our lattice has two cells. Take the odds of having any one strategy from the previous equation, and add to it the odds that a chosen strategy does not occur in the first cell *times* the likelihood that it will appear in the next cell, or equation (9). By extending these two examples to the odds of a single strategy occurring in a lattice with j cells, we derive the geometric series in equation (10).

Many random initial states used in the future consist of an 800×600 lattice. This lattice has 480,000 cells, and equation (11) shows that the odds of having at least one of each strategy are exceptionally high. In most simulations, one can assume that each strategy has had a chance to compete based on its existence in the initial state.

$$\frac{1}{2^{15}} = 3.05 \cdot 10^{-5} \quad (8)$$

$$\frac{1}{2^{15}} + \frac{2^{15} - 1}{(2^{15})^2} = 6.10 \cdot 10^{-5} \quad (9)$$

$$\mathbb{P}(j) = \sum_{i=1}^j \frac{(2^{15} - 1)^{i-1}}{(2^{15})^i} \quad (10)$$

$$\mathbb{P}(480,000) = 0.99999957 \quad (11)$$

Figure 10: Odds of Having a Single Strategy in a Random Initial State

3.3 One Interesting Consequence

Two crucial rules from the previous section have some non-intuitive effects on simulations, and these effects will also affect how strategies create barriers between each other. First, a strategy sums up all the points it receives from its Moore neighborhood. Next, it looks at all of its neighbors, and it picks the one with the most points and copies its strategy. Imagine we have rules A , B , and C where A is B 's neighbor (and vice versa), B is C 's neighbor, but A is not C 's neighbor. C always defects, which brings down B 's score. When A compares scores surrounding it, it will see that B has a lower score, and A will pick a different strategy for its next state. This example shows that even though A is not C 's neighbor, C affects whether or not A takes on B 's strategy; furthermore the neighbors of the neighbors of A , also known as A 's second neighbors, affect its next state and strategy. Figure 11 demonstrates this case example.

Both of the initial configurations for this simulation are the same except for the cell circled in black. Both start with a 10×10 lattice, and each cell plays 10 iterations of the game with its neighbors each round. In the second simulation, the cell circled in black has the strategy 0 instead of strategy 18078 . The cell with strategy 0 always defects, bringing down the score of the cells next to it. This effects the next state of the cell circled in white. In the first simulation, the cell circled in white takes on the strategy of its neighbor

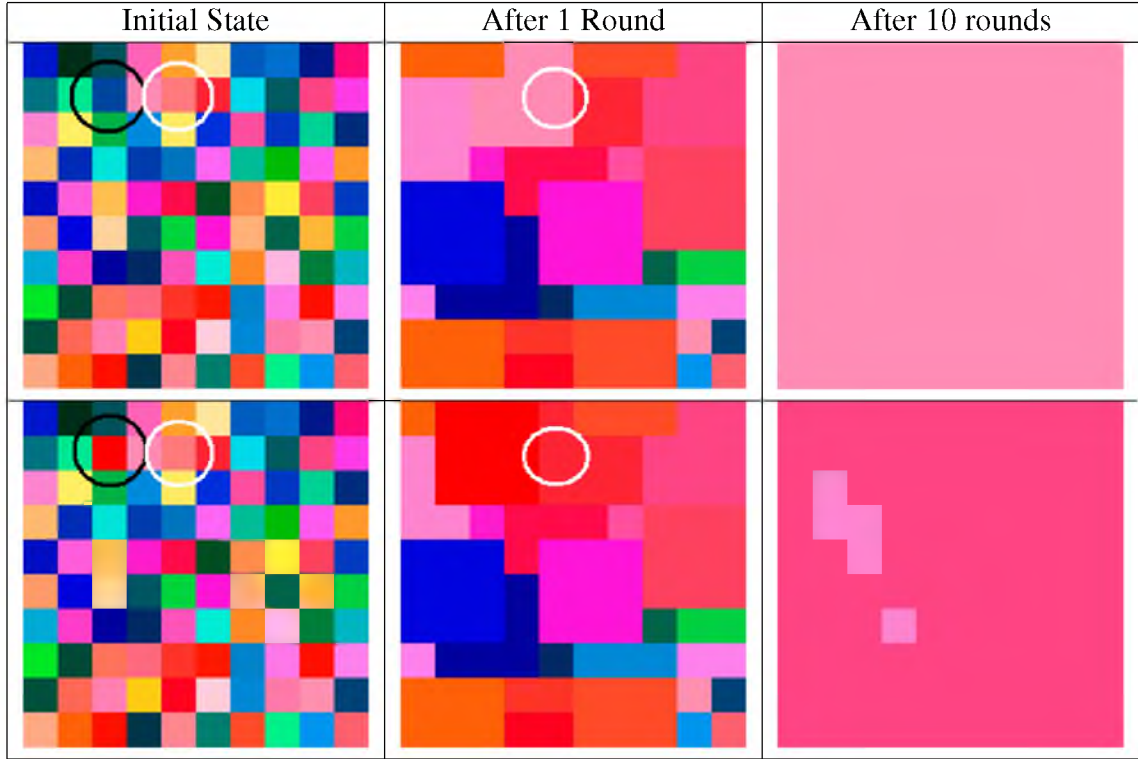


Figure 11: The Effects of the Second Neighbor

to the left, strategy *36020*. Strategy *36020* goes on to take over the whole lattice after just ten rounds. In the second simulation, strategy *0* brings down the score of strategy *36020*, which leads our cell circled in white to take on the strategy *9268* instead. After ten rounds, strategy *17540* takes over the lattice instead.

The actual strategy numbers do not matter as much in this example, as the fact that small initial changes affect second neighbors and the final state of the lattice. In future (more interesting) simulations, we will see how second neighbors create a wave or invisible force field of point densities around cells.

4 Patterns, Fractals, and Grouping

4.1 Creating Order Out of the Randomness



Figure 12: A 800x600 Random Initial Configuration

When beginning a simulation from a random initial state(Figure 12), one must define many variables. To start, one must define T , R , P , and S (See Table 2). Many experiments focus on tweaking the values of either T , R , P , or S to create complexity. In Martin A. Nowak and Robert M. May’s *Evolutionary Games and Spacial Chaos*, they achieved both static and dynamic patterns such as their “evolutionary kaleidoscope” by setting $2 > T > 1.8$, $R = 1$, $S = 0$ and P extremely close to S [16]. Marcelo Alves Pereira and Alexandre Souto Martinez found spatio-temporal patterns in 1-dimensional

cellular automata by also tweaking T , R , P , and S [18]. Despite this interesting research, all simulations here will use the generic values $T = 5$, $R = 3$, $S = 1$, and $P = 0$.

However, wanting to ensure interesting results one must tweak a variable that will allow at least a little bit of randomness in our IPDG simulations. Luckily, our strategies have memory, and when dealing with strategies that have a memory another important variable affects how the cells interact. The number of iterations each cell plays between each round controls how much a cell relies on either its asymptotic or transient memory. At minimum for a strategy with a memory of 3, a simulation must require the cells to do 4 iterations to take advantage of all of the transient: 1 iteration for the first move followed by 3 iterations to use the transient memory. After 4 iterations, the cell relies on its asymptotic memory for the past three moves. So when we set the number of rounds, what we really care about is the ratio of transient rounds to asymptotic. What ratio will give us interesting results?

Here comes the science. Empirically running many simulations and gathering data creates a way to find a good ratio, and there exists a few good indicators of whether or not a simulation will converge to a homogenous (boring) or mixed (ideal) steady state.

First, examine the *Hamming Distance* in Figure 13, which measures how many cells switch strategies each round. The following graphs show the Hamming Distance for different simulations ran from a 800×600 random initial state. The x-axis and the y-axis represents the time step and population of cells that have changed strategy respectively.

Immediately one can see that the Hamming Distance for playing the game with only 4 iterations converges at a much higher number than the simulations that had 5 iterations or more. After 750 time steps the six simulations sampled for 4 iterations averaged a Hamming Distance of 2538 or .0053% of the total population. Even though this may seem like a very small percentage of the population, compared to the more asymptotic simulations it's huge! From 5 to 9 iterations, the simulations averaged a Hamming Distance of 211, 295,

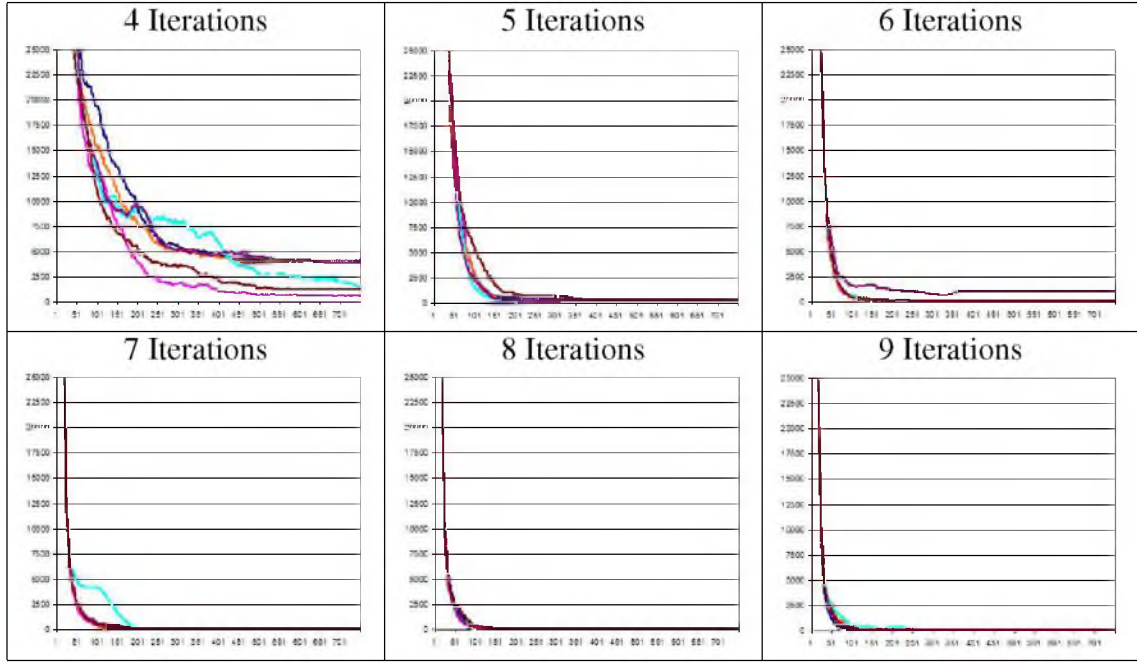


Figure 13: The Hamming Distance

89, 63, and 77 after 750 time steps.

Also notice how in all of the simulations, the Hamming Distance initially starts out high, and then eventually it slopes out sharply. A log-log plot of the 4 iteration simulations reveals strong evidence that one can approximate the hamming distance over time-steps with the function, $f(x) = 393560 \cdot x^{-0.79645}$ ($R^2 = 0.8986$). (In contrast, a log-linear plot shows that the data does not take the form $f(x) = ca^x$ where $c, a \in \mathbb{R}$.) One can look at Hamming Distance as the amount of energy or movement in a system. When Hamming Distance decreases, one can interpret this as an increase of static neighborhoods in the system. From here we can form a hypothesis that the optimal number of iterations exploits only the transient and none of the asymptotic, since simulations with a low Hamming Distance might represent more homogenous groups. Examining the simulations individually seems to confirm this.

In Figure 14 one can see many complex groupings made of several different rules from one of the 4 iteration simulations. It has a good amount of diversity and randomness.

Upon close inspection, if one examines the southern area of the simulation, there's a persistent formation of gliders bouncing, colliding, and producing more gliders. Gliders could lead to a proof that the Iterated Prisoner's Dilemma CA has Computational Universality.

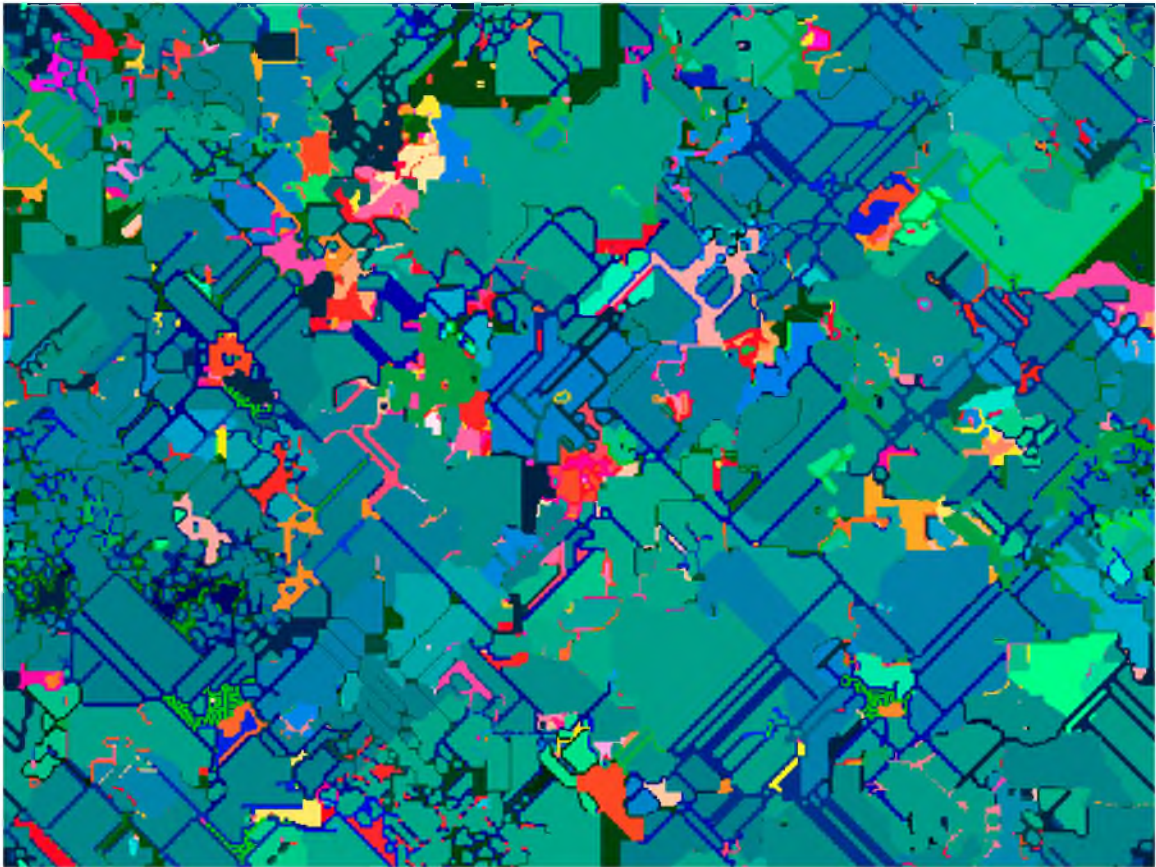


Figure 14: 4 Iteration Simulation After 750 Rounds

Now lets examine our simulations by the number of initial cooperators versus the number of initial defectors at each time step. Initially one might think this would also test for diversity; however, the simulations created some rather unexpected results. Table 6 shows the average results for 6 simulations ran for 4, 5, ..., 10 iterations.

Iterations	4	5	6	7	8	9	10
Percentage	91.8%	96.1%	94.4%	93.4%	89.5%	87.4%	83.0%

Table 6: Average Percentage of Initial Cooperators

Contrary to what one would expect according to the Hamming Distance results, as the number of iterations increases the number of initial cooperators actually decreases and the number of defectors increases. Though simulations with higher iterations have a more static, less diverse population, it seems that strategies that initially defect can take advantage of initial cooperators early in the game and eventually group together creating borders that cooperative rules cannot invade into. Figure 15 shows some of these groups of defectors. The initial defectors have a red or orange tint to them while the initial cooperators have a blue or green tint.

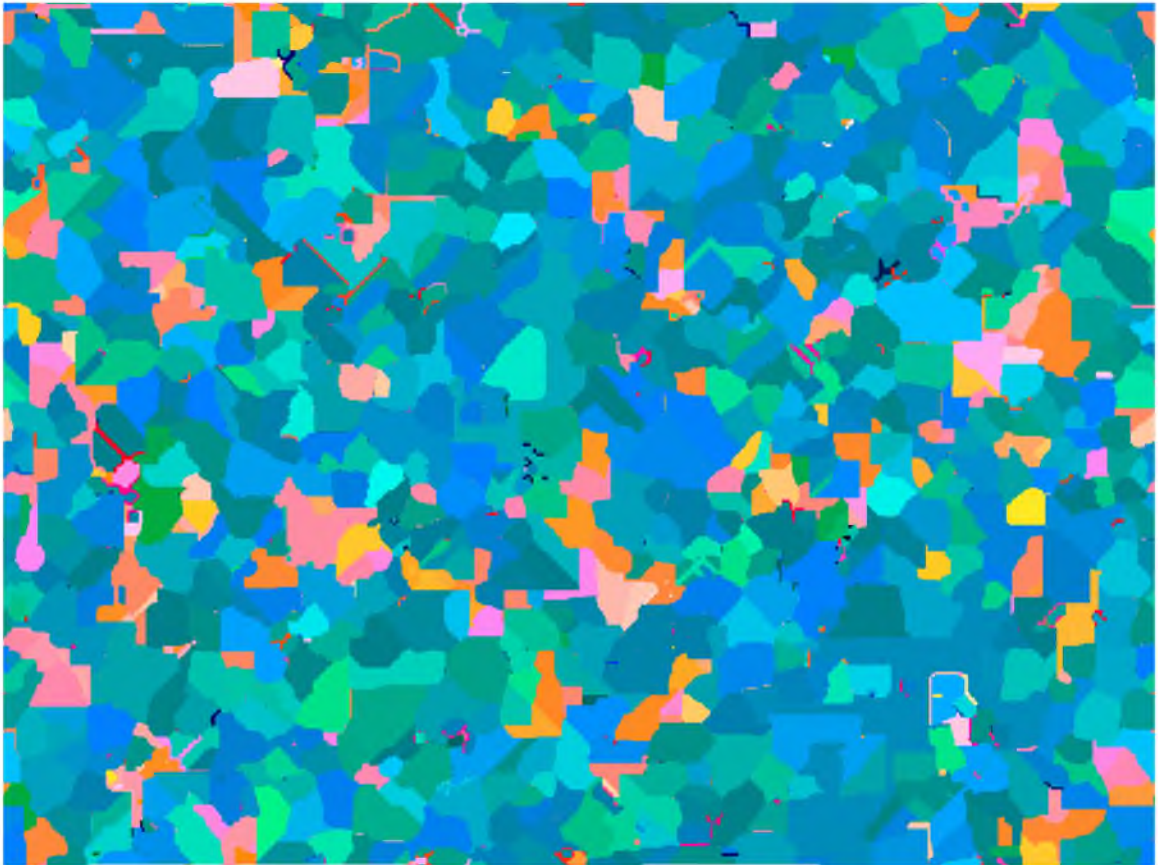


Figure 15: 9 Iteration Simulation After 750 Rounds

Of course, more defectors may also mean a lower total percentage of the highest possible amount of points for the population. Remember that the highest amount of points

that two players can get collectively in a round is 6 if both choose to cooperate. This means the highest amount of points in a simulation equals $3 \cdot 8 \cdot p \cdot i$ where p is the number of players or cells and i is the number of iterations (remember that each cell plays 8 times, once with each of its 8 neighbors). Also keep in mind that this number looks at how the population does as a whole and not at individuals who can make up to $5 \cdot 8 \cdot i$ points. Table 7 shows the average percentage of total points that simulations achieved at different iterations.

Iterations	4	5	6	7	8	9	10
Percentage	92.6%	97.9%	98.0%	98.3%	98.1%	97.8%	97.6%

Table 7: Average Percentage of Population Points

Strangely enough, the percentage of possible points seems to increase until it reaches 7 or 8 iterations, and then it slowly starts to recede. However, for the most part the total percentage of possible points stays fairly high for all simulations, which may mean that the only initial defecting strategies that can survive have very cooperative asymptotic properties. This also makes initial defection a feasible strategy. If one assumes this simulation emulates daily interactions, this validates a strategy whereby people can indeed defect whether through rudeness or another social force to people they have only initially met, so long as they cooperate later.

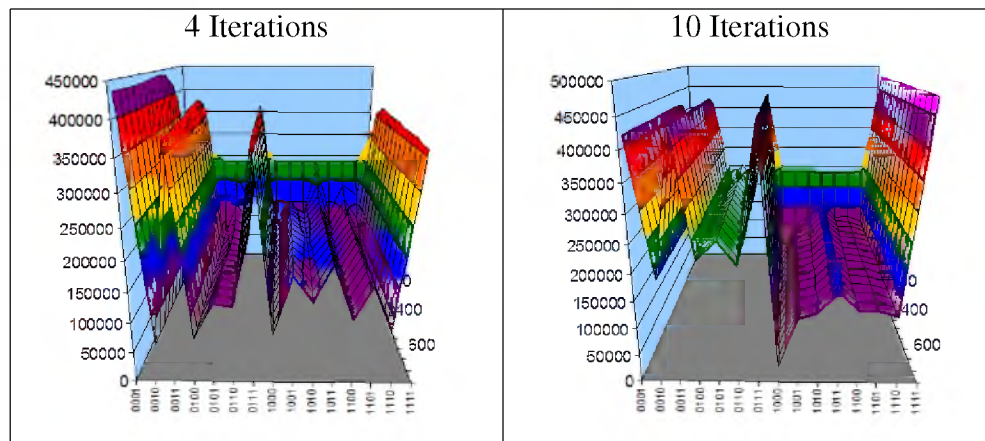


Figure 16: Cooperating Instructions over Iterations

Finally, one can also look at the fitness of cooperating or defecting on a particular instruction. Figure 16 shows two graphs that trace the popularity for a strategy to cooperate for a particular history over time (time steps) on a 800×600 lattice. The left axis represents the number of strategies cooperating for a certain history (note that the maximum number of strategies cooperating for a given history is 480,000). The bottom axis represents the history as defined in Section 3 earlier. The right axis represents the time, with the most recent time step in the front.

This method of gathering data on the simulations can help us intuit an ideal composite strategy made up of the most popular instructions for a given population. By comparing simulations with different iterations per time step, we can also see how the popularity of cooperating for one history changes when we shift emphasis to the asymptotic memory. In both the 4 iteration and 10 iteration simulations successful strategies almost always cooperate initially as well as when the opponent has always cooperated; furthermore, in both we should always defect when the opponent has defected three times in a row.

The main difference between the transient simulation, and the more asymptotic simulation reveal some interesting patterns. As the number of iterations increases, a strategy needs to cooperate more in the transient stage in order to survive; however, it should also cooperate less when the opponent has not cooperated historically for the past three moves in the asymptotic memory. Using this information, we can try to create *ad hoc* a strategy that I'd call Unforgiving Tit for Tat. As long as Unforgiving Tit for Tat remembers an opponent defecting, it will defect, otherwise it cooperates. Unfortunately, this interesting result does not belong within the scope of this paper (see Section 7).

Overall, these evaluations have led to the conclusion that the transient parts of an IPDG strategy lead to complexity and perhaps Universal Computation. The simulations that follow all use only 4 iterations unless specified otherwise.

4.2 How Groups Form

Looking at Figure 14, one can see that simulations eventually reach a mostly stable state where many different groups of strategies live side by side. One might wonder why doesn't one strategy eventually end up dominating the whole simulation? In order to understand this we must look carefully at the borders between two competing strategies. Figure 17 shows a stabilized group of strategies on the top, and the points that they receive in grey-scale on the bottom. The lighter the area is, the more points players received.

The cells with the lighter blue color in the middle of the trapezoidal shape use the strategy 38538. The green colored cells have the strategy 41058, and the darker bluish color on the right border of that has strategy 33418. To make this easier, let strategy $A = 38538$, $B = 41058$, and $C = 33418$. Most of the A cells make 96 points every time step, except for the ones right along the border. On the right side of the grouping of A , a border of strategy B keeps A and C separated; however the line created by B only makes 92 points and 80 points based on if it interacts with A or C .



Figure 17: Point View

However, when B interacts with A or C it lowers their points so much, that on the next time step it will choose itself as the best strategy. Using equation (3) from figure 2 on page 6, we can understand this as a set of equalities.

For simplicity's sake, these inequalities ignore the number of neighbors each round,

but it still follows the general idea. First, B excels relatively whenever it faces A or C , so $f(A, B, 4) < f(B, A, 4)$ and $f(C, B, 4) < f(B, C, 4)$; however, A and C do better when matched with their own kind than with B , so $f(B, A, 4) < f(A, A, 4)$ and $f(B, C, 4) < f(C, C, 4)$. B defects in such a way that it can persist by lowering the scores of opposing rules, but it can't invade either A or C since both A and C exert a force field of points over their domain. Thus borders and groups form.

Say we had strategies D and E , such that $f(D, E, 4) < f(E, E, 4)$ and $f(D, D, 4) < f(E, D, 4)$. Here, strategy E will push and invade into D no matter what, and a stable grouping of D will never persist; however, what if we introduce a new strategy F ? This D can invade F , but F has a strategy that invades E , creating a rock-paper-scissors effect. With the addition of F we can create more complex moving objects in a simulation. All we have to do is initialize our lattice with a specific formation of strategies.

4.3 Interesting Results

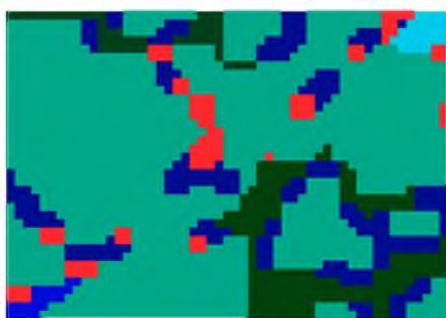


Figure 18: A Natural Gun

When running random simulations, interesting formations of strategies will often develop from the randomness. If one looks closely at Figure 14 on page 30 towards the bottom there's an interesting patch of red and navy blue dots on a teal background. Figure 18 zooms in on this peculiar formation.

This actually happens to be a glider gun, and the little worm looking thing on the upper-right hand side is a glider. Just like Conway's Life, with enough space interesting objects emerge; however, though this hints that the Iterated Prisoner's Dilemma CA can perform Universal

Computation, proving it will take a lot more work.

Isolating certain strategies together increases the chances for interesting behavior like the Glider Gun. By combining strategies *11196*, *36866*, and *1088* in a specific initial configuration one can make a “program” that recursively builds the Sierpinski triangle. Figure 19 shows the initial setup on the left, and different time steps of the simulation as the triangle forms itself. The bottom represents how many points each cell playing the IPDG makes, forming the triangle.

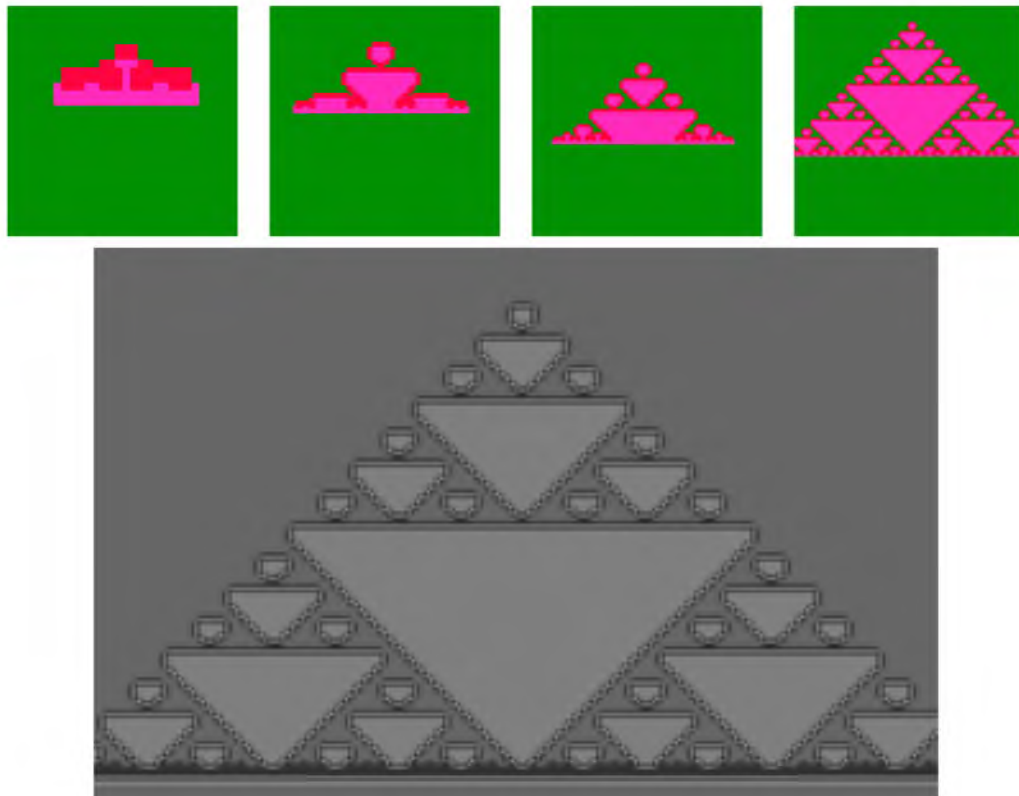


Figure 19: An Emergent Sierpinski Triangle

Each of the triangles in this simulation create a point which upholds a wall going to the right and to the left of it. This wall has the pink, *11196*, filling underneath it, which then invades the green rule *36866*. As *1088* invades from the sides, creating a triangle, it cuts off smaller walls which create even more triangles. The whole process creates quite a

mesmerizing visual.

The ability to tame the IPDG CA simulation to create a fractal builds upon a hypothesis that perhaps we can create controllable initial configurations in the IPDG capable of Universal Computation. However, in order to create a proof for Universal Computation, one needs more than a pretty fractal since many CA capable of creating fractals can not compute everything. Specifically, one needs a way to transmit and store information, as well as a way to make logic gates.

An interesting case example comes from the two-memory strategies 8748 (in red), 34954 (in blue), and 8746 (in black). This versatile set of rules can make many interesting patterns, the first of which can produce little bullet and wave like objects. Figures 20 and 21 show these three rules competing with only three iterations in order to avoid transient behavior.

In both these simulations, the black colored strategy separates the red and the blue creating walls, and it also creates movement based on how one strategically places it. Figure 20 relies on making small walls of blue and black in order to produce a bullet. For the most part, combinations of the strategies existing in the lattice do not have the power to penetrate into these walls. The walls keep in a spiral at the top that periodically shoots out a bullet, whose width the walls also determine. Unfortunately these bullets do not destroy themselves upon collision like the gliders in the game of Life. Also, when they leave the walls, they spread in all directions forming a wave.



Figure 20: The Wave Gun

Figure 21 uses the same three rules, but with a slightly different approach. Using the blue strategy for the background instead of the red one, one can easily make something

that looks more like a glider gun. From top-left to bottom right, Figure 21 shows each time step of the creation of a glider.

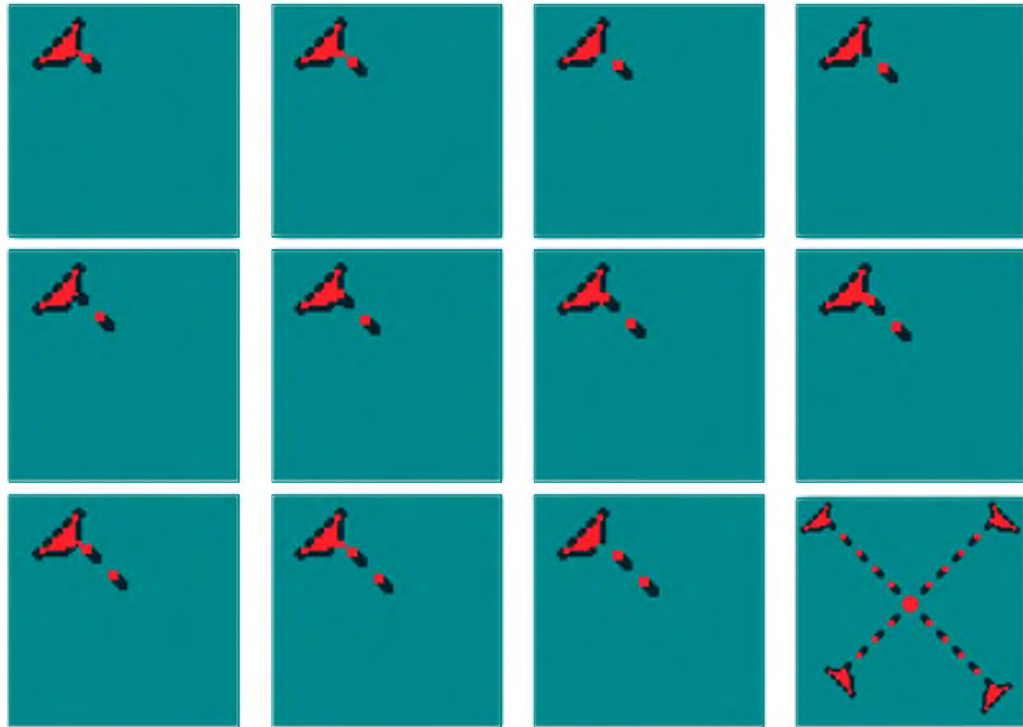


Figure 21: The Three Iteration Gun

Unfortunately, upon collision these gliders also persist, and often the collisions create a great amount of unpredictable randomness. The bottom-right image of Figure 21 shows the only way to destroy a glider, by colliding four going in opposite directions all at the same time. So though the simulation can produce gliders and transmit information, it can't create the logic gates fundamental to Universal Computation.

Luckily, a few sets of three-memory rules interacting with four iterations can create a special glider which does destroy itself upon impact with another glider. But as this section shows, one can find many different combinations of strategies which have interesting properties. One can also create strategies ad hoc, in order to create specific interactions. In this way our IPDG CA excels in versatility.

5 Proof of Universal Computation

5.1 Creating a NAND gate

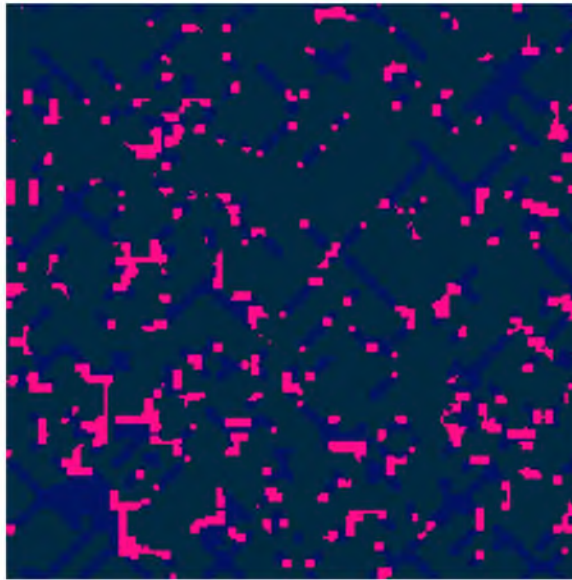


Figure 22: Randomness Capable of Universal Computation

While running several different random simulations, an interesting group of strategies appeared. Figure 22 shows how three of these strategies interact with each other. In the random initial setup shown, many gliders appear. These gliders hold the key to Universal Computation.



Figure 23: A Glider

Figure 23 shows one of these gliders in action. The basic setup of strategies to make a glider follows. Using strategy *10818* as a background, the slightly darker *5218* forms a head that invades into *10818*. The pink looking strategy *3464* eats up *5218*, preventing it from traveling diagonally in both directions. On the right of figure 23 the point view shows a cell with a lot of points(light grey) which advances the pink square each time step. If examined closely, the head of the glider moves by lowering the points of the

background strategy in front of it, while giving the lower-right most cell of the head just slightly more points than the other cells around it.



Figure 24: A “Pin”

Another object, though not shown in Figure 22, can eat and redirect gliders. Figure 24 shows a “pin” made up of strategy *33002*. *33002* cooperates with itself, which allows it to achieve a high amount of points each time step if its neighbors have the same strategy. On the left we have a basic 3×3 square of *33002*. On the right, the point view of the same object shows the high amount of points a cell makes in the middle of the square.

Now if a glider runs into two of these pins placed very carefully, the glider will die due to a force field of points made from the background off of the light blue strategy *33002*. Figure 25 shows one of these glider eaters in action. Notice that the glider begins to die two cells before coming into contact with the eater. This may seem strange at first, but remember that the pins create a force field of high points on the background, making it impossible for a rule to invade or even touch the pins.



Figure 25: A Glider Eater

In order to finding Universal Computation, the gliders must interact either with each other or with the environment in such a way that one can create logic gates. Figure 26 shows what happens when two gliders collide orthogonal to each other. Conveniently enough, these gliders take each other out when they collide in a manner similar to the game of Life. From here a proof of Universal Computation follows similarly to that of Life, if we can create a “glider gun”.



Figure 26: Collision Between Two Gliders

An interesting phenomenon happens when we have a glider collide with one pin at just the right angle. Figure 27 shows a glider colliding on the edge of a pin. The pin actually pulls the glider into itself, making the glider swerve around the pin; furthermore, the glider swerving around the pin then starts to produce more gliders in every direction! Eureka! This is a glider gun, and with some strategic placing of more pins we can control its output.



Figure 27: Glider Hooked on a Pin

Now that we know how the gliders interact and now that we know how to make a glider gun, we can now start making logic gates. In order to prove Universal Computation we actually only need to make one kind of logic gate, the NAND. The NAND gate combines a NOT gate with an AND gate. Table 8 shows how the NAND gate works (Notice that \odot means NAND here). Notice that one can create the boolean AND, OR, and NOT gate as well. This means that all we need to find to display the logic gate requirement of Universal Computation is an example of a NAND gate.

Figure 28 shows the before and after of a NAND gate simulation. The pink lines should help the reader understand the data stream and their direction, but they are not part of the simulation. On the top row, starting from the lower-right hand side, we have two gliders in the same stream. Notice that the absence of a glider also conveys information.

x	y	$x \odot y$	$(x \odot y) \odot (x \odot y)$	$(x \odot x) \odot (y \odot y)$	$x \odot x$
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	1	0
			= AND	= OR	= NOT

Table 8: The NAND Gate

We can assign a glider the binary value 1, and the absence of a glider a 0. So the top row will feed in starting from the bottom a 0, 1, 0, and finally a 1. The bottom row will feed in a 1, 1, 0, 0.

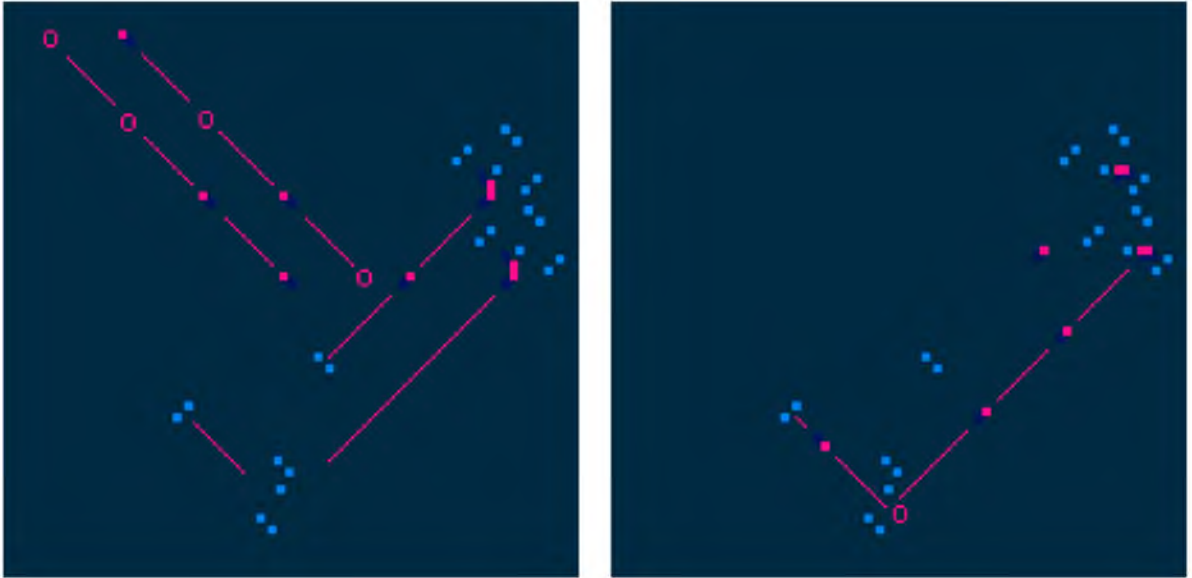


Figure 28: A NAND Gate made out of IPDG Players

The first glider gun creates gliders which will collide with the incoming streams. If only one glider enters, the glider gun will block it, making the output a 0. Similarly if no gliders go in, none will come out. In the special case when two gliders try to enter at the same time, the glider gun will only block one of them letting the other slip through. Thus we must have two 1's entering to get a 1 out. This is our AND gate.

The second glider gun creates our NOT gate. This time the gliders from the gun

not only block the gliders input into it, but it also creates the output. If a glider enters, it'll destroy one from the gun. Thus a 1 will create a 0, and if nothing enters it outputs a 1.

After going through the AND and the NOT gate, we should get a 1, 0, 1, and a 1. On the right of Figure 28 we have just that. Now all we need to show is that we can also store memory in this simulation, and luckily all this requires is some everyday knowledge of Digital Design.

5.2 Making a Computer

Today's computers work with a very smart device called a transistor. Similar to our gliders, transistors act as gates to manipulate voltages. Since we can reduce the input and outputs made with transistors to logic gates, we should be able to produce anything a transistor can make with our gliders and glider guns. So lets make some cool stuff!

First, though we can make every gate using just the NAND gate, the difficulty of making basic gates with a NAND makes finding smaller substitutes a fruitful activity. Figure 29 contains both an OR gate(top) and an XOR gate(bottom). The OR gate receives two gliders going in the same direction, and combines them into one stream by using pins to rotate one 90 degrees into the same path as the other. Remarkably, when two gliders collide while rotating, they simply combine instead of dying. This

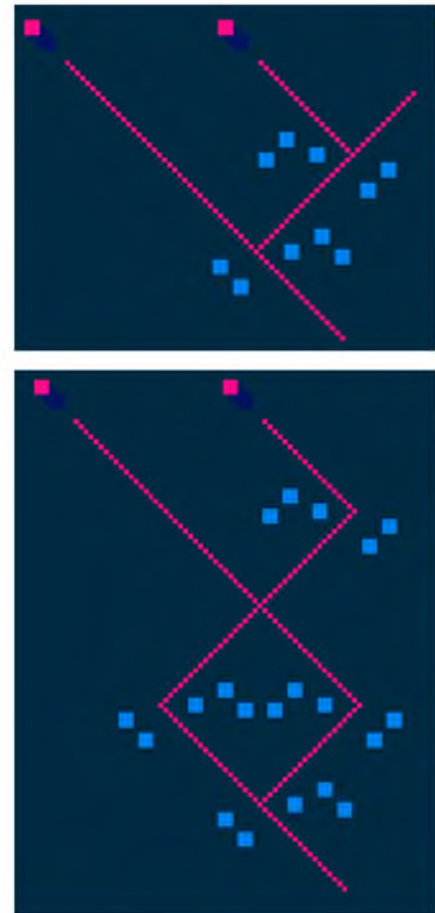


Figure 29: OR and XOR Gates

makes creating an XOR gate very easy. First a rotator takes gliders from one stream and purposefully collides that glider with another. Then three more rotators combine the two path ways into one.

The XOR will come in handy since in both digital design and here we need the ability to cross paths. In digital design we can simply make one wire or pathway go over or overlap another. Our two-dimensional environment will push us to be more creative. Table 9 shows how to use XOR(\oplus means XOR here) to cross signals. First create a shared stream, $x \oplus y$, then using a copy of the y signal derive x by XORing the result of the shared stream.

x	y	$x \oplus y$	$(x \oplus y) \oplus x$	$(x \oplus y) \oplus y$
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

Table 9: Properties of XOR

Now that we have these tools, lets create the missing piece to our proof for Universal Computation, *memory*. Figure 30 shows a D latch in Cellular Automata form. Usually in a computer, one can create a register by placing two D latches together back to back. A register forms a safe way to store and retrieve bits getting consistent results; however, half a register works just fine for proving that the IPDG can hold memory.

I have labeled many of the major components in pink. First notice that the clock and the D input both get split into two paths, with the XORs crossing them. Then at the bottom we NOT the D and AND it with the clock input. The clock has a cycle of 11 gliders “on” then 11 gliders “off”. At the top, the D and the clock get ANDed together as well. This guarantees that the leftover latch will receive only one signal at a time. If it receives a signal from the top, the latch ORs it and then NOTs it, saving the value “1” in a loop. The

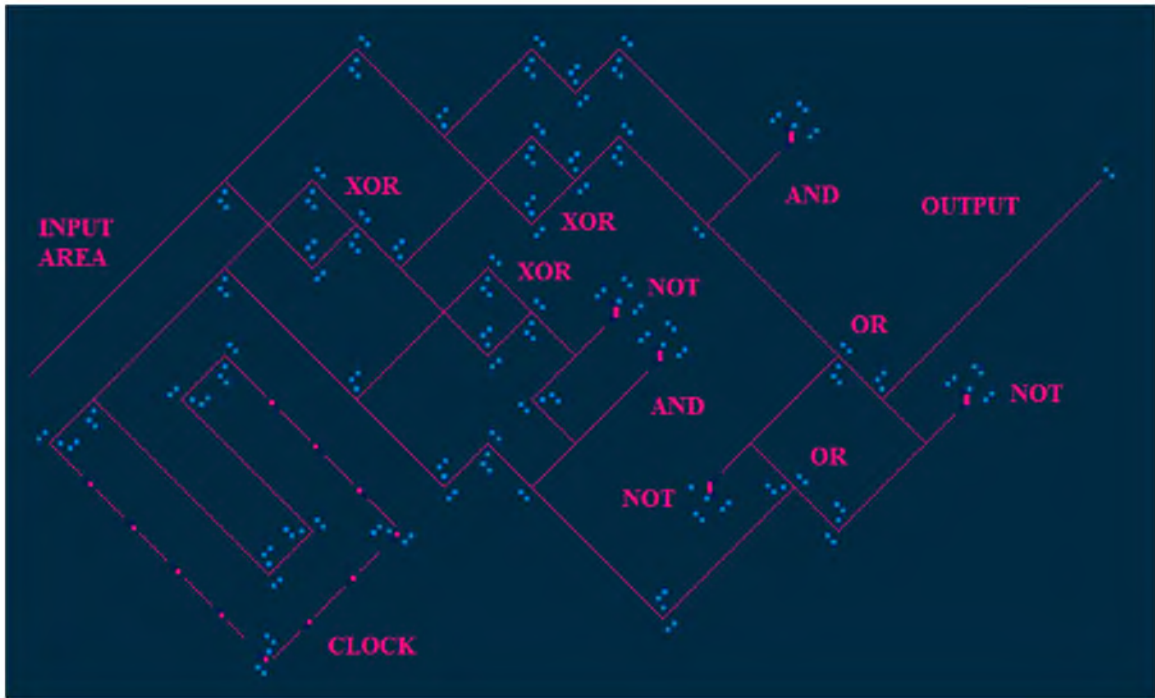


Figure 30: D Latch with a “0”

same happens when a signal gets fed to the bottom; except the opposite side of the loop will turn on. Figure 30 shows the D latch in its initial configuration. Eventually, since it has no input going into it, this latch will save a “0”. Figure 31 shows the same simulation, but with a glider gun feeding an input of “1” into the input path. When both D and the clock output “1” at the same time, the value gets saved into the latch. As we’d expect, the simulation with gliders in the D input, also has gliders in the output.

Now that we have a case example for how to construct memory, we have proven that the Iterated Prisoner’s Dilemma Game has Universal Computation on a Cellular Automata Lattice. □ So what does this mean?

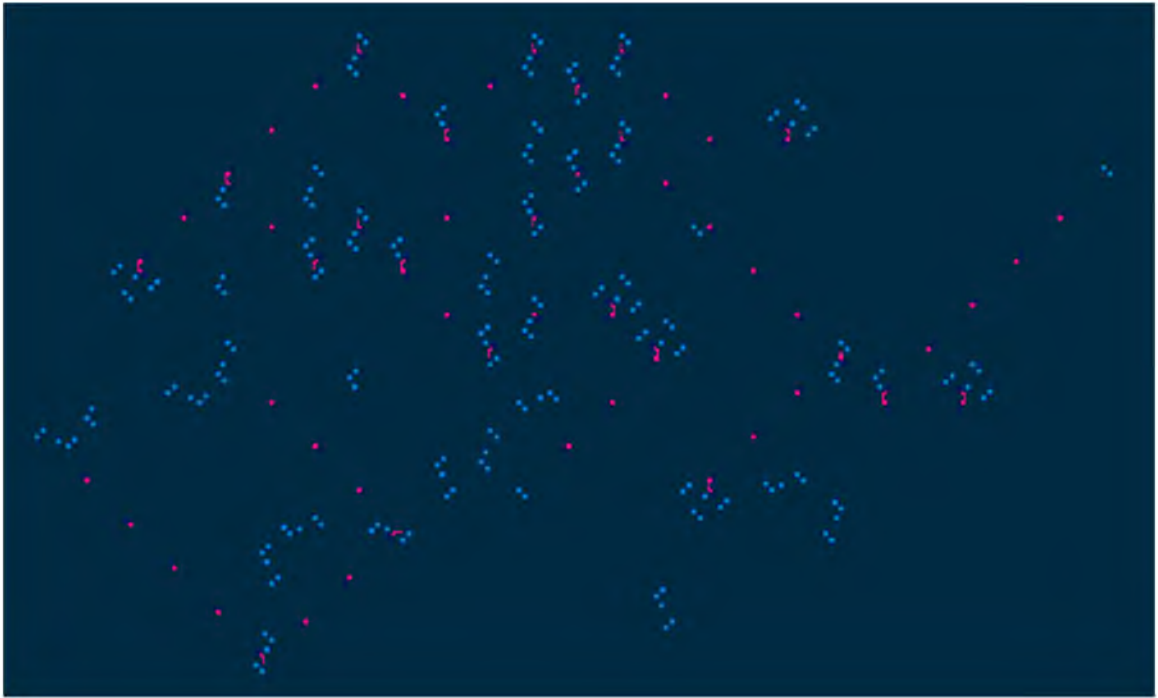


Figure 31: D Latch with a “1”

5.3 What this Proof Means

This proof shows partially by construction the existence of Universal Computation in a subset of the strategies for playing the Prisoner’s Dilemma Game. With infinite pins and infinite space one could definitely compute any algorithm, and this includes algorithms that self-replicate. Due to the static placement of pins, this Cellular Automata seems similar to Wireworld in its strict placement of paths for the gliders; however, this proof has similarities with the proof for Universal Computation in Conway’s Life(see Section 2.3).

This proof does not show all the possible ways to create a Universal Turing Machine. In fact several sets of strategies seem to create similar gliders or other ways of transmitting information. The possible number of combinations of strategies that one could try is inexhaustible; however, often in the small (800×600) random initial configurations combinations of strategies that look like a Universal Computer usually emerge, and sometimes

persist after 700 times steps. Remember the natural glider gun in Figure 18 on page 18? The following gun in Figure 32, the glider, and eaters are made up of all the same stuff. The gliders made from this gun destroy themselves upon collision; therefore, a proof for its Universal Computation will work similarly to the proof for the set of strategies above. Who knows how many other interesting computers might exist among the combinations of strategies!

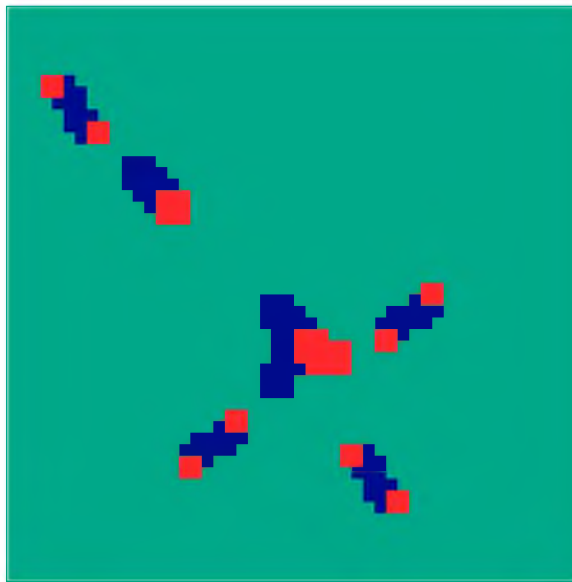


Figure 32: A Glider Gun without pins

This proof for Universal Computation, like the other ones (review Section 2.3), has quite the philosophical impact. With an infinite amount of cells filled with random strategies we know that somewhere groupings of strategies would create a configuration capable of Universal Computation.

6 Social and Biological Relevance

6.1 Possible Implications

Jonathan D. Victor, a Neurologist at Cornell University, had this to say about the implications of Neumann's Cellular Automata in the context of brains:

The observation that the cerebral cortex is composed of a large number of local neural assemblies that are iterated throughout its extent, by itself, is not an existence proof that complex behavior may result from a network of simple elements. Von Neumann's construction is necessary to show that such a structure is indeed capable of complex behavior, *without the need to invoke* region-to-region variability, long-range interactions, stochastic components, or mysticism. [20]

Similar to Victor's interpretation of Neumann's CA, a proof of Universal Computation using the Iterated Prisoner's Dilemma Game shows that people can convey information across a large network through the simple act of defecting or cooperating. As another way to put it, there might exist something in the whole of interactions of human kind that the individual may not have awareness of. Quite unconsciously, we can transmit information to people we don't even know. Like a colony of ants, this simulation should blur the line between what exists only as an individual and what exists only as part of a whole.

In Stephen Jay Gould's book, "The Flamingo's Smile", he talks about the continuity and ambiguity in defining a single organism, and a collection of organisms [12]. In particular, he talks about Siphonophores, a creature made up of many individual organisms that essentially cooperate together. The individual organisms can specialize to move the colony, capture food, and even reproduce for the entire being!

Axelrod's original research shows how organisms can benefit from cooperation, even in situations that create strong incentives to defect. In the context of Siphonophores, one can conceive how cooperation leads to creating a colony of beings, which slowly forms into its own organism. Research on both Siphonophores and cooperation bridges the gap between the single cell and the multi-celled, the organism and the superorganism.

One could posit that this research - that focuses on the complex holistic interactions, showing nearly limitless potential interactions, Universal Computation as well as an explanation of the persistence of variance and different strategies - helps us understand complexity and differentiation in individual beings. It can add to our understanding of how a single celled individual through simple interactions can evolve into the multicellular. Universal Computation makes this simulation a compelling metaphor for a multiple leveled, composite view of nature.

Since we have essentially found a machine that can compute any algorithm in our simulation of the IPDG, we can essentially create a machine that plays the IPDG. In this way it acts like a larger organism, composed of many smaller entities all with the same goal. Then of course, we can create larger machines out of that, and go on recursively *ad infinitum*.

Furthermore, the results found in this paper extend to all interactions reducible to the Iterated Prisoner's Dilemma Game. From things as small and intricate as RNA to things as subtle as washing dishes, this paper adds to the literature finding computation in many aspects of nature.

6.2 Conclusion

This paper has shown many of the holistic effects emergent in a social CA simulation from players with different deterministic strategies. First, for any number of iterations during a

time step, the CA creates groupings of strategies that persist without invading each other due to a point density effect. Furthermore, this effect can also create movement within the lattice, as strategies invade and take over each other. This can create interesting patterns algorithmically, like the Sierpinski Triangle.

By treating the movement of strategies as bits, this paper constructed a proof for Universal Computation similar to that for Conway's Life. By setting up particular formations of players, the simulation can create gliders and glider guns. These gliders work similarly to signals along electrical circuits, and one can easily create any sort of digital logic. One can also create a register or memory, central to a proof for Universal Computation.

The interesting part of this thesis comes from the fact that this CA simulates a potential social interaction evident not only in human interactions, but also in nature. Where other studies have focused on simpler interactions, with simpler players, this paper shows that complex players with history don't only propagate themselves, but they can also organize themselves through the adoption of strategies with higher points and through the imitation of more successful cells in a self-interested manner. If Axelrod showed us the "Evolution of Cooperation", surely this would be the "Evolution of Complexity and Organization."

This paper has also found an ideal number of iterations per time-step for players that have a memory of three moves. Simulations with a number of iterations that focus on the transient properties of a strategy will create the most complexity in a randomly configured lattice. This implies that a person must remember all of an opponent's history for interesting results.

Finally, this paper has created evidence for the grand possibilities that human interactions (and other interactions) might have. It should bring into question whether or not we convey information that we don't intend to, and whether or not we can create a higher

consciousness through interaction.

Of course, besides just the philosophical, creating Universal Computation with any set of rules can be a fun and intellectually rigorous activity. I hope those who read this paper had as much fun as I did.

7 Further Ideas for Research

Some might read this paper skeptically identifying many of the weaknesses of this interpretation of Cellular Automata. For example, human interactions don't always necessarily follow the IPDG or any other game in game theory. Also, human interactions do not happen in conveniently isolated neighborhoods with just 8 opponents in synchronized time steps. In many ways real life contains a higher level of complexity and unpredictability that may not always seem present in this cellular automaton.

One person's doubts and apprehensions are someone else's research. Further studies that show similar results to the ones found here in different lattices, including ones where each player has a unique variable neighborhood more akin to real life, would demonstrate the robustness of this theory. Especially if this research found both evidence in nature and in further simulations that complexity emerges when players have a memory of all its opponents moves, one can apply this theory as a supplement to the many others about evolution. Furthermore, a statistical mechanical approach as used by Bahr and Passerini can approximate information about groups of people without "delving into the detailed social processes of each individual" [7]. Thus, though a cellular automaton might not faithfully adhere to the exact interactions of individuals, it may still accurately represent interactions of the whole [6].

Also, many of the empirical findings in Section 4.1 have implications that one could verify on real people under a controlled experiment. For example, are people more likely to initially defect when they know they will interact with a person longer? Further computer tournaments could also investigate the survivability of Unforgiving Tit for Tat. One could even use the results of Section 4.1 to test if spatio-temporal systems accurately predict real people's behavior, assuming people transition between strategies in order to maximize their utility. If such evidence exists, the IPDG CA will have potent predicting powers, and it will

be the stepping stone for a lot more research with direct impact on several related subjects.

Hopefully this should also encourage research that looks for more holistic traits in people's interactions. The search for which strategies make the most points or for which strategies are the most robust, loses sight of the possible complexity inherent in interactions. Theories that just support Tit for Tat fail to account for why a person would pick a less than optimal strategy or act less logically. All research on the Iterated Prisoner's Dilemma could benefit by emphasizing the interactions and locality. Great places to start would be social networking websites. Something as small as whether or not a person sends or receives messages could create a profound pattern.

8 Bibliography

References

- [1] “Algorithm, n. Med.” The Oxford English Dictionary. 2nd ed. 1989. OED Online. Oxford University Press. Web. 15 November 2009. <<http://dictionary.oed.com/>>.
- [2] Ashworth, Tony. *Trench Warfare, 1914-1918: The Live and Let Live System*. New York: Holmes and Meier, 1980. Print.
- [3] Axelrod, Robert, and William Hamilton. “The Evolution of Cooperation.” *Science*. 211. (1981): 1390-1396. Print.
- [4] Axelrod, Robert *The Evolution of Cooperation*. New York: Basic, 1984. Print.
- [5] Bahr, David B., R.C. Browning, H.R. Wyatt, and J.O. Hill. “Exploiting Social Networks to Mitigate the Obesity Epidemic.” *Obesity*. 17.4. (2009):723-728. doi:10.1038/oby.2008.615. Web. 6 January 2013.
- [6] Bahr, David B. and Eve Passerini. “Statistical Mechanics of Collective Behavior: Macro Sociology.” *Journal of Mathematical Sociology*. 23.1. (1998):29-49. Print.
- [7] Bahr, David B. and Eve Passerini. “Statistical Mechanics of Collective Behavior: Micro Sociology.” *Journal of Mathematical Sociology*. 23.1. (1998):1-27. Print.
- [8] Fowler, James H. and Nicholas A Christakis. “Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study.” *BMJ*. 307. (2008).doi:10.1136/bmj.a2338. Web. 13 January 2013.
- [9] Berlekamp, Elwyn R., John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*. Volume 4, 2nd ed. Wellesley: A K Peters Ltd, 2004. Print.

- [10] Blue, Stanley L., and Campbell R. McConnell. *Economics: Principles, Problems, and Policies 17th ed.* New York: McGraw-Hill/Irwin, 2008. 453-460. Print.
- [11] Chao, Lin, and Paul E. Turner. "Prisoner's dilemma in an RNA virus." *letters to nature*. 398. (1999): 441-443. Print.
- [12] Gould, Stephen J. "A Most Ingenious Paradox." *The Flamingo's Smile* Ontario: Penguin Books Canada Ltd., 1985. 78-95. Print.
- [13] Hamilton, William D. "The Evolution of Altruistic Behavior." *American Naturalist*. 97. (1963):354-56. Print.
- [14] Henle, Werner, Gertrude Henle, and Evelyne T. Lenette. "The Epstein-Barr Virus." *Scientific American*. 241:1. (1979): 48-59. Print.
- [15] Matthews, Donald *U.S. Senators and Their World*. Chapel Hill: University of North Carolina, 1960. Print.
- [16] Nowak, Martin, and Robert May. "Evolutionary Games and Spatial Chaos." *Nature*. 359. (1992): 826-829. Print.
- [17] Nowak, Martin, Robert May, and Karl Sigmund. "The Arithmetics of Mutual Help." *Scientific American*. (1995): 76-81. Print.
- [18] Pereira, Marcelo, and Alexandre Martinez. "Pavlovian Prisoner's Dilemma- Analytical Results, the Quasi-Regular Phase and Spatio-Temporal Patterns." *Journal of Theoretical Biology*. 265. (2010): 346-358. Print.
- [19] Schiff, Joel L. *Cellular Automata: A Discrete View of the World*. Hoboken: John Wiley & Sons, 2008. Print.

[20] Victor, Jonathan. "What can Automaton Theory Tell Us About The Brain." *Physica D*. 45. (1990): 205-207. Print.

[21] Wolfram, Stephen. *A New Kind of Science*. Champaign: Wolfram Media, 2002. Print.