Universal Computation in the Prisoner's Dilemma Game

Brian Nakayama[1] and David Bahr[2]

[1] Department of Mathematics, Regis University, Denver, CO 80221, USA
[2] Department of Physics and Computational Science, Regis University, Denver, CO 80221, USA

Abstract. Previous studies of the Iterated Prisoner's Dilemma Game (IPDG) focus on the optimal strategies for accumulating points against another player or the evolution of cooperation. Instead, this paper expands upon the possible complexity in interactions by using a Cellular Automaton (CA) model to simulate large numbers of players competing within a limited space. Unlike previous works, we introduce a method for creating a wide variety of deterministic rules by mapping each possible interaction to a binary number. We then prove the computational universality of the resulting IPDG CA. An analysis of the number of interactions leads to the discovery of interesting properties when allowing only enough iterations for a strategy to use its "transient" instructions. The implications of universal computation (UC) are also discussed.

Keywords: Cellular Automata, Iterated Prisoner's Dilemma Game, Universal Computation, Universal Turing Machine.

1 Introduction

Studies about cooperation and optimal strategies abound for the Prisoner's Dilemma Game (PDG). As an approximation of interactions found in nature these studies impact our understanding of biology; however, to date few studies focus on the computational potential for the PDG when implemented in a cellular automaton (CA). Furthermore, none of the studies known presently provide a proof of universal computation in the aggregate interactions of players when situated in a lattice. In this paper, we introduce a memory based labeling scheme for strategies that allows 2^{15} different strategies to compete with each other. We report on the surprising effect of iterations on interactions and strategy. We then show interesting examples of self organization and a proof for Turing Universality in the iterated PDG CA.

The PDG [1] gets its name from the following scenario: Players A and B recently robbed a store, but the police do not quite have enough evidence to prove it. Players A and B know that they will both receive minor sentences of two months if they do not confess. Aware of this, the police put A and B in separate interrogation rooms and offer them each a deal. If A (or B) gives evidence on the other to the interrogators, A (or B) will get to leave without

O.H. Ibarra et al. (Eds.): UCNC 2014, LNCS 8553, pp. 293–304, 2014.
DOI: 10.1007/978-3-319-08123-6_24,

receiving any charges, but the person who does not 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 the rewards in Table 1.

Table 1. The Prisoner's Dilemma Game

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

In addition to the basic scenario, the PDG includes any interactions between two players creating Table 1 with the rewards T, R, P, and $S \in \mathbb{R}$ having the properties $T > R > P > S$ and $R > {}^{(S+T)}/_2$ [2].

Previous works have implemented multiple iteration PDG (IPDG) simulations in CA. CA allow for the study of simple nearest neighbor interactions in a lattice on a discrete set of states [3]. Studies of the IPDG CA use strategies for playing the game as states, while the choice to cooperate or defect, along with a replacement rule, allows the states to interact.

Axelrod [2] was the first to use CA to study the IPDG, finding that a strategy called Tit for Tat did not perform as well as other strategies in CA despite Tit for Tat's dominance in other computer tournaments he designed. Nowak et al. [4] have studied simulations with and without memory, where the agents either always defected or cooperated. Nakamaru et al. [5], Brauchli et al. [6],and Szabó et al. [7] performed similar experiments on IPDG CA. Newth and Cornforth [8] extended the work of Nowak et al. on an asynchronous grid. Tanimoto and Sagara [9] studied alternating reciprocity in infinite length simulations on 2×2 grids. Gelimson et al. [10] investigated the effects of mobility on the evolution of cooperation. Alonso-Sanz [11][12], implemented an IPDG with memory of each player's points on a Moore lattice; however, though he found gliders, he did not find universal computation in his simulations. Finally, Pereira and Martinez [13] studied the IPDG on a 1D lattice. They also found emergent complexity in their random simulations with objects such as gliders [14] and "fingers".

In Sect. 2 we introduce a scheme for memory and interaction in the IPDG CA, while Sect. 3 reveals interesting results of our scheme. The paper concludes with a proof of Turing Universality in Sect. 4 and a short discussion in Sect. 5.

2 Implementation of the Prisoner's Dilemma Game

For a detailed explanation of the CA, one can refer to the original thesis [15].

The IPDG CA used in this paper creates strategies that can remember the past three iterations of an opponent. A strategy is described using binary by assigning cooperation the value 1, and defection 0. Using binary, we can define a unique history of three iterations from least recent (left bit) to most recent

Table 2. 3 memory transient strategy for Tit for Tat

Opponent's History	111	110	101	100	011	010	001	000
Cooperate / Defect	1	0	1	0	1	0	1	0

(right bit). Table 2 uses a robust strategy called Tit for Tat [2] to demonstrate the numbering scheme.

Table 2 will work when it has played at least three iterations of the PDG with an opponent. Tit for Tat only considers the most recent move (the rightmost bit), copying it, but it ignores the previous two moves. This strategy lacks information for what to do for its first, second, and third iteration. After a strategy has played three or more iterations, it can use the past three moves of the opponent to determine what to do. Thus Table 2 describes the long term, *asymptotic* behavior. In order to define the initial behavior we need something that describes the *transient*. Table 3 includes the missing information.

Table 3 contains both the asymptotic, and the transient instructions. The leftmost 1's position defines to which iteration a strategy responds. The column with "0001", represents the first iteration. Two columns have "001x_1" to represent the second iteration, because the opponent will have one of two optional histories, $x_1 = 0$ or $x_1 = 1$. Similarly four columns have "01x_1x_2" for the third iteration (x_2 is the opponent's most recent move), and eight columns have "1$x_1x_2x_3$" for the eight possible histories for iterations four and beyond (x_3 is the most recent). The column with "0000" is always a "0" and isn't used. Rather, it lets us think of the strategies as 2^{n+1} bit numbers, where n is the longest sequence of iterations a player can remember.

Table 3. Complete 3 memory strategy 10818

Opponent's History	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000
Cooperate / Defect	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0

To get the strategy number, or *10818* in this example, we convert the bottom row of Table 3 from binary to base 10. We order our bits to get 0010101001000 $010_2 = 10818_{10}$. Using this method we represent each possible strategy with its own unique number. For graphic simulations we assign each strategy a unique 24 bit-color by using the strategy bits for the green and blue hue while giving initial defecting strategies a red hue. Using this method, the IPDG CA has a total of 32,768 different strategies.

The IPDG CA simulations implement a Moore neighborhood on a 2D lattice. Each cell, representing a player, must interact with the eight cells closest to it (the northwest cell, the north, the northeast, east, etc.). In addition to choosing a lattice, we must also have a simple rule that determines how a cell changes states between rounds. For this research, cells use a "Darwinian" rule [7][13], picking

the next state/strategy by changing a cell's current state to that of the neighbor with the highest points. If a neighbor does not have a higher point value, then the cell will not change states. Therefore, in the case of a tie between a cell and its neighbor, a cell will keep its own strategy. In order to keep our simulations deterministic and symmetrical around the square, the rule ignores all other ties. Ties do not arise often in simulations and have no noticeable effects.

In order to run the IPDG CA we need to define T, R, P, and S (Table 1) as well as the number of iterations. All simulations here will use the generic values $T = 5$, $R = 3$, $P = 1$, and $S = 0$ [4]. Further in this section we will determine the number of iterations.

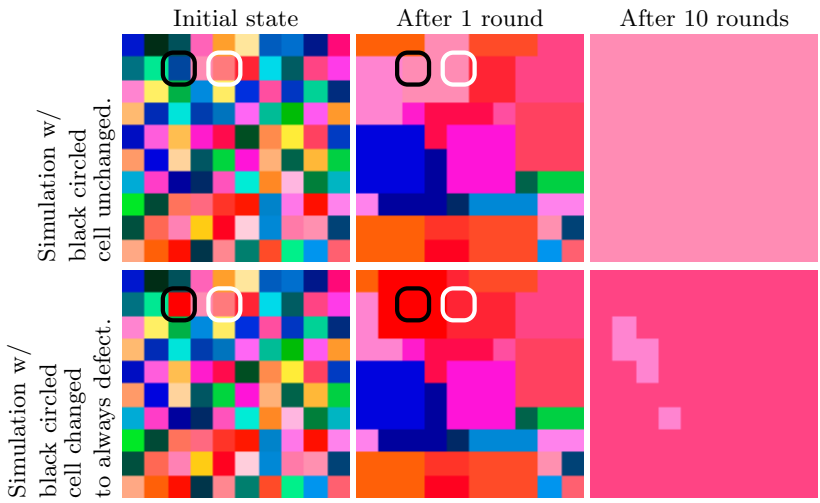

Fig. 1. The effects of the second neighbor

The Darwinian rule has the non-intuitive effect of creating second neighbor interactions. In a *round*, a cell plays the game with its eight neighbors for n iterations and sums up the points it receives. At the end of a round, it looks at all of its neighbors, and it picks the one with the most points and copies its strategy. Multiple rounds comprise a *simulation*. Suppose we have rules A (Fig. 1 circled in white), B, and C (Fig. 1 circled in black) where A is B's neighbor, 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. Thus, even though A is not C's neighbor, C affects whether or not A takes on B's strategy. Thus, *the neighbors of the neighbors of a cell, also known as a cell's second neighbors, affect its next state and strategy.* Figure 1 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 with its neighbors each round. In the bottom 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 affects 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 to the left, strategy *36020*. Strategy *36020* goes on to take over the whole lattice after just 10 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.

3 Initial Results

Data on the Hamming distance and strategy populations was gathered on random initial states (Fig. 2a) over the 2^{15} different strategies.

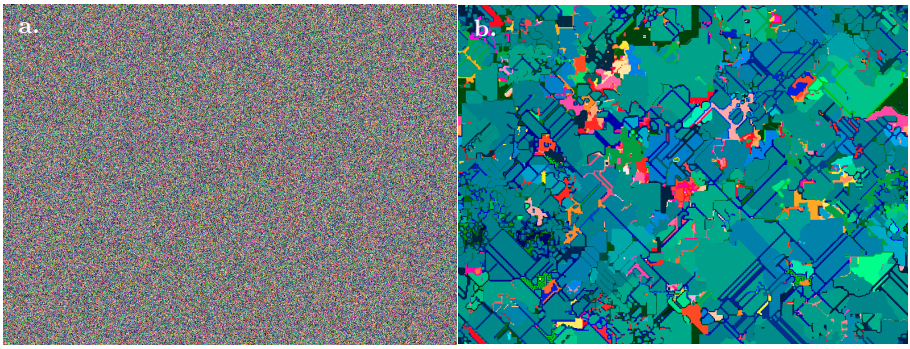

Fig. 2. a. A 800x600 random initial config, b. 4 iteration simulation after 750 rounds

The number of iterations for a simulation affected the outcome of the simulations by controlling how much a cell relied on either its asymptotic or transient memory. For a strategy with a memory of 3, a simulation must require the cells to do at least 4 iterations to take advantage of all of the transients: 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. Thus, when we set the number of iterations, we inadvertently affect the ratio of transient to asymptotic iterations.

First, examine the Hamming distance (HD) in Fig. 3, which measures how many cells switch strategies each round. The following graphs show the HD for different simulations run on an 800×600 random initial state (where each cell contains a randomly chosen strategy). The x-axis and the y-axis represent the time step and number of cells that have changed strategy, respectively.

We observe that the HD for playing the game with only 4 iterations converges at a much higher number of cells than the simulation with 9 iterations. After 750 rounds, the six simulations sampled for 4 iterations averaged a HD of 2538

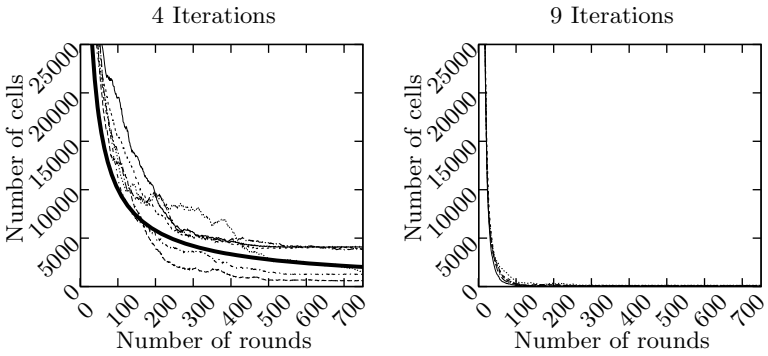

Fig. 3. The Hamming distance with a log-log plot approximation (the thick line)

or .0053% of the total number of cells. From 5 to 9 iterations, the simulations averaged a HD of 211, 295, 89, 63, and 77 after 750 rounds.

In all of the 4 iteration simulations, a log-log plot of the HD suggested that one can approximate the HD over rounds with the function, $f(x) = 393560 \cdot x^{-0.79645}$. When the HD decreases, one can interpret this as an increase of static neighborhoods in the system. Accordingly, we hypothesize that *if the IPDG CA contains a model of computation then the optimal number of iterations for finding UC in a random initial configuration exploits only the transient and none of the asymptotic*. Simulations with a high HD might represent more dynamic groups. Figure 2b shows a 4 iteration simulation converging to a steady state that coincidentally contains a glider gun [14]. The simulations that follow all use only 4 iterations unless specified otherwise.

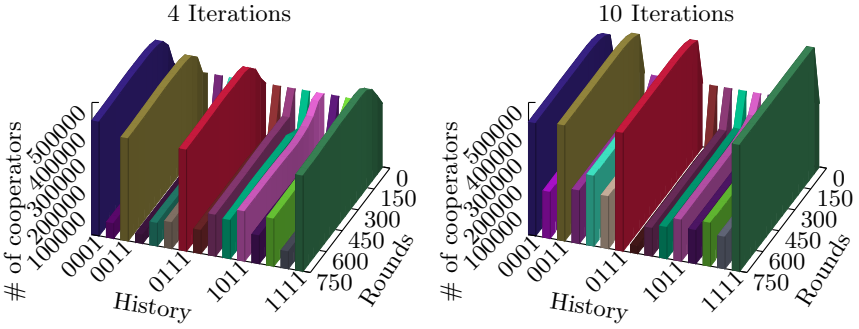

Fig. 4. Cooperating instructions over rounds

We next consider the number of strategies that cooperate or defect on a particular opponent's history. Figure 4 shows two graphs that trace the number of cells that cooperate for a particular history over time (rounds) on a 800×600

lattice. The vertical axis represents the number of cells cooperating for a certain history (the maximum is 480,000). The horizontal axis represents the history as defined in Sect. 2 earlier. The depth represents rounds, with the most recent round in front.

This method of gathering data on simulations provides insight into an ideal composite strategy made up of the most popular instructions for a lattice of cells. In both the 4 iteration and 10 iteration simulations *successful strategies almost always cooperate initially as well as whenever the strategy remembers that the opponent has always cooperated*; furthermore in both simulations, we should *always defect when the opponent has defected three times in a row.*

Contrasting the transient (4 iter.) to the more asymptotic (10 iter.) simulation reveals a difference in strategy. 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 create a potentially optimal (for points) *ad hoc* strategy that cooperates more initially, but still defects as long as it remembers an opponent defecting.

Looking at Fig. 2b, one can see that *simulations eventually reach a stable state where many different groups of strategies persist side by side.* In order to understand how groups form we must look carefully at the borders between two competing strategies. Figure 5a shows a stabilized group of strategies, and the points that they attain in grey-scale (b). The lighter the area is, the more points cells received.

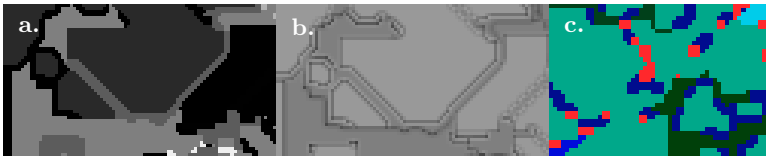

Fig. 5. a. A border, b. Point view of border, c. A natural "gun"

The cells with the lighter blue color in the middle of the trapezoidal shape use the strategy *38538* (A). The green colored cells have the strategy *41058* (B), and the darker blue color on the right border of that has strategy *33418* (C). The line of B cells attains less points than most of the A or C cells. However, the B cells drag down the points of surrounding A and C cells, preventing A or C from invading. *The inability of cells to invade results in many groups, despite high points between similar cells.*

When running random simulations, strategies sometimes self-organize into useful structures for computation. In Fig. 2 towards the bottom there is an interesting patch of red and navy blue dots on a teal background. Figure 5c zooms in on the patch to reveal a glider gun [14] and an arrow shaped glider. With enough space, interesting patterns emerge, similar to Conway's Game of Life [14].

Isolating certain strategies together increases the chances for formations 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 6 shows the initial setup on the left, and different time steps of the simulation as the triangle forms itself.

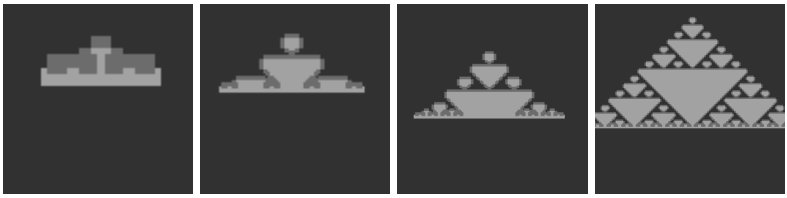

Fig. 6. An emergent Sierpinski triangle

Another example comes from the two-memory strategies *8748* (in red), *34954* (in blue), and *8746* (in black). This versatile set of strategies can make many patterns, the first of which can produce little "bullet" and "wave" like objects. Figure 7 shows these three strategies competing with only three iterations in order to avoid asymptotic behavior.

In both these simulations, the black-colored strategy separates the red and the blue creating borders and movement based on how one strategically places the cells. Figure 7a relies on making small walls of blue and black in order to produce a bullet. The borders keep in a spiral at the top that periodically shoots out a bullet, whose width the borders determine. Upon leaving the borders, they spread in all directions forming a wave. Figure 7b uses the same three strategies, but with a slightly different approach. Using the blue strategy (*34954*) for the background instead of the red one (*8748*), one can easily make something that looks more like a glider gun, repeatedly spitting out little arrow shaped gliders.

Fig. 7. Two different "guns" made with different config. of the same strategies.

4 Proof of Universal Computation

The glider guns observed while running random simulations prompted a search for a set of strategies capable of UC. We eventually found one set, strategies *10818*, *5218*, *3464*, and *33002*. The following proof of UC uses glider guns to

create a NAND gate and memory in a similar style to the proof of UC in Conway's Game of Life [14].

Figure 8 shows the basic configurations capable of UC. The setup of strategies in Fig. 8a to make a glider follows. Using strategy *10818* as a background, the slightly lighter *5218* forms a head that replaces *10818* each round. The pink strategy *3464* replaces *5218*, preventing it from traveling diagonally in both directions. Figure 8b shows the point view (lighter gray = more points). 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.

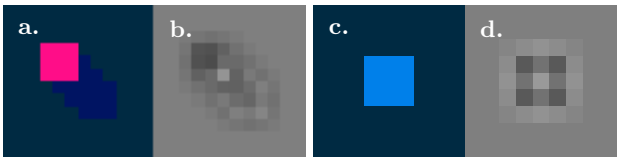

Fig. 8. *a.* A glider, b. Glider's point view, c. A pin, d. Pin's point view

The pin in Fig. 8c is made up of strategy *33002* which cooperates with itself. The center of the pin achieves a high number of points each round. Figure 8c shows a basic 3×3 square of *33002*. The point view of the same object (d) shows the high points of the center cell.

If a glider runs into two pins placed carefully, the background strategy *10818* will replace the glider due to a border of high points the background attains from strategy *33002*. Figure 9a shows a glider eater[14] in action. If two gliders collide as in Fig. 9b, they destroy each other. Finally, Fig. 9c shows a glider colliding with the edge of a pin. The pin pulls the glider around itself, making the glider swerve around the edges of the pin. The glider swerving around the pin then starts to produce another glider on every corner of the pin. This creates a glider gun, and with some strategic placing of more pins we can control its output.

Now that we know how the gliders interact and how to make a glider gun, we can start making logic gates. *In order to prove UC we only need to show one kind of logic gate, the NAND*[14]. Figure 10 shows the before (a) and after (b) of a NAND gate simulation. (The lines help in understanding the data streams and their direction, but are not part of the simulation.) The absence of a glider also conveys information. We assign a glider the binary value 1, and the absence of a glider a 0. Figure 10a feeds in a stream 0, 1, 0, 1 starting from right to left. The bottom row will feed in 1, 1, 0, 0.

The first glider gun creates gliders that 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 enter, 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 an AND gate. The second glider gun creates a NOT gate. The gliders

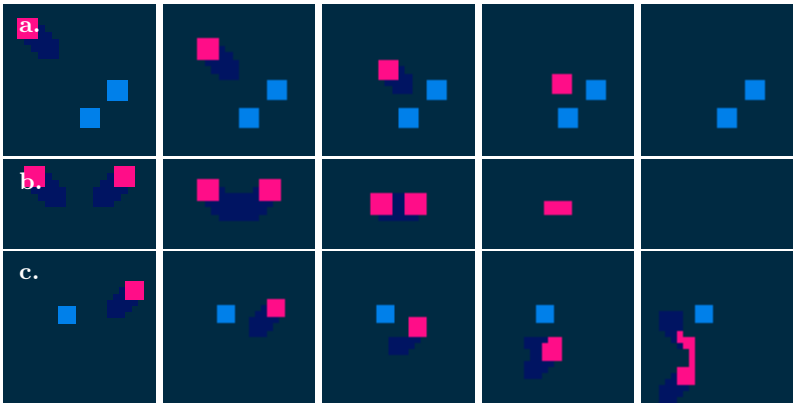

Fig. 9. a. A glider eater, b. A collision, c. A glider hooked on a pin

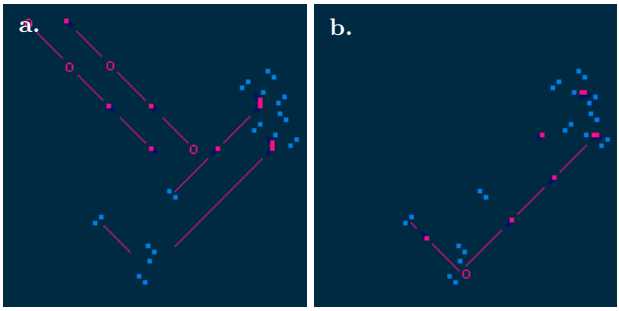

Fig. 10. A NAND gate made out of IPDG players

from this gun not only block the gliders input into it, but also create the output. If a glider enters, it will destroy a glider 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 have in Fig. 10b 1, 0, 1, 1.

Now we need to show that we can also store memory in this simulation. Figure 11 shows a D latch in CA form. One can create a register holding a unit of memory by placing two D latches together back to back. A register forms a safe way to store and retrieve bits with consistent results. However, half a register (a D latch) suffices for proving that the IPDG can hold memory.

The major gates and streams are labeled. The clock and the input each get split into two paths, with XOR gates crossing them. At the bottom we NOT the inputs and then AND them with the clock input. The clock has a cycle of 11 gliders "on" then 11 gliders "off". At the top, the input 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 same happens when a signal

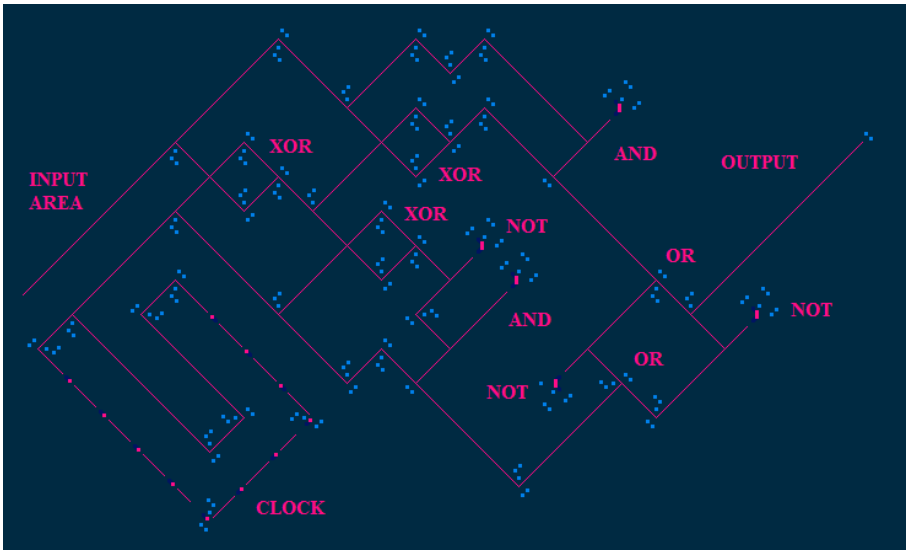

Fig. 11. D latch with a "0"

gets fed from the bottom, except that the opposite side of the loop will turn on. Figure 11 shows the D latch in its initial configuration. Eventually, since it has no input going into it, this latch will save a 0. To store a 1, instead put a stream of gliders on the input path. *Now that we have found a way to construct memory, we have proved that the IPDG CA has UC.*

5 Discussion

This proof shows the existence of UC in a subset of the strategies for playing the IPDG by the construction of a NAND gate and a D latch. The proof does not show all the possible ways to create a Universal Turing Machine (UC), as several sets of strategies seem to create similar gliders or other ways of transmitting information.

Furthermore, the IPDG CA introduced here contains persistent groupings of strategies that defect rather than cooperate in their interactions with each other and their surroundings. This model may help explain diversity in strategies present in nature. These defections were important for organizing larger structures within the CA, but perhaps most interesting is that this CA may also simulate a potential social interaction evident in human interactions [2]. Where other studies have focused on simpler strategies and the evolution of cooperation, this paper shows that complex players with history not only propagate themselves but also organize themselves through the adoption of strategies with higher points and through the imitation of more successful cells in a self-interested manner. Similar to Axelrod [2] which explains the "Evolution of

Cooperation", universal computation in IPDG cellular automata suggests the evolution of complexity and organization.

Acknowledgments. We thank James Seibert for advising the original thesis, Robyn Lutz for patiently editing, and Jack Lutz for advice on publishing. This work was supported by Regis University's Math-Science Scholarship, by RU's Math and Computer Science Departments, and by NSF grant 1247051.

References

1. Dresher, M.: Games of Strategy: Theory and Applications. Pren Hal. Appl. Math. Pren. Hal. (1961)
2. Axelrod, R.: The Evolution of Cooperation. Basic Books (1984)
3. Wolfram, S.: A New Kind of Science. Wolfram Media (2002)
4. Nowak, M.A., May, R.M., Sigmund, K.: The arithmetics of mutual help. Sci. Am. 272(6), 76 (1995)
5. Nakamaru, M., Matsuda, H., Iwasa, Y.: The evolution of cooperation in a lattice-structured population. J. of Theor. Bio. 184(1), 65–81 (1997)
6. Brauchli, K., Killingback, T., Doebeli, M.: Evolution of cooperation in spatially structured populations. J. Theor. Bio. 200(4), 405–417 (1999)
7. Szabó, G., Antal, T., Szabó, P., Droz, M.: Spatial evolutionary prisoners dilemma game with three strategies and external constraints. Phys. Rev. E 62(1), 1095 (2000)
8. Newth, D., Cornforth, D.: Asynchronous spatial evolutionary games. Biosystems 95(2), 120–129 (2009)
9. Tanimoto, J., Sagara, H.: A study on emergence of alternating reciprocity in a 2×2 game with 2-length memory strategy. Biosystems 90(3), 728–737 (2007)
10. Gelimson, A., Cremer, J., Frey, E.: Mobility, fitness collection, and the breakdown of cooperation. Phys. Rev. E 87(4), 042711 (2013)
11. Alonso-Sanz, R.: The historic prisoner's dilemma. Int. J. Bifurcat. Chaos 9(06), 1197–1210 (1999)
12. Alonso-Sanz, R.: Memory versus spatial disorder in the support of cooperation. Biosystems 97(2), 90–102 (2009)
13. Pereira, M.A., Martinez, A.S.: Pavlovian prisoner's dilemma-analytical results, the quasi-regular phase and spatio-temporal patterns. J. Theor. Biol. 265(3), 346–358 (2010)
14. Berlekamp, E., Conway, J., Guy, R.: Winning Ways for Your Mathematical Plays, vol. 4. A.K. Peters (2004)
15. Nakayama, B.: Universal computation in the prisoner's dilemma game. Undergraduate honors thesis, Regis U (2013)