# ALTERED PAYOFF VALUES AND THE EFFECT ON A POPULATION OF ITERATED PRISONER'S DILEMMA PLAYERS

By

Michael Clark Weeks
B.E.S., University of Louisville, 1993

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed Scientific School
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Engineering Mathematics and Computer Science

May 1994

ALTERED PAYOFF VALUES AND THE EFFECT ON A POPULATION OF ITERATED PRISONER'S DILEMMA PLAYERS

Submitted by: ____________________________________
Michael Clark Weeks

A Thesis Approved on

(Date)

by the Following Reading and Examination Committee:

Dr. R. K. Ragade, Thesis Director,
Engineering Mathematics and Computer Science

Dr. M. J. Maron,
Engineering Mathematics and Computer Science

Dr. T. G. Cleaver,
Electrical Engineering

## ACKNOWLEDGMENTS

The author would like to thank Dr. R. K. Ragade for his support, encouragement, and ideas for this thesis. The author appreciates Dr. M. J. Maron and Dr. T. G. Cleaver for serving on the thesis committee.

The following people were also helpful in making this project possible. Georg Fuellen at MIT, who provided a list of initial references. David B. Fogel of the ORINCON Corporation, who was kind enough to send a copy of one of his papers, which the author referenced extensively. Leigh Testfatsion also provided a copy of an important paper. Mark Atkinson, Gregory Seront from the Free University of Brussels, Belgium, and I. M. Ikram from Rhodes University, South Africa, who helped by clarifying some of the biological terminology. Finally, the author appreciates the Internet correspondence of everyone posting to the comp.ai.genetic bulletin board. The people of this newsgroup were helpful in providing references and insight through personal correspondence.

## TABLE OF CONTENTS

## ABSTRACT

Game theory and evolutionary programming are used to model social interactions, and simulate aspects of nature. Scientists often use the prisoner's dilemma game with Genetic Algorithms for this purpose. The prisoner's dilemma gives each player a choice between the move best for both players, if they can trust each other (cooperation), or the selfish but safer choice (defection). The combined choices result in a payoff of utilities to each player. This thesis examines the effect of varying payoff values on populations of prisoner's dilemma players. For example, will a selfish player thrive under certain payoffs? As the saying goes, "every man has his price." This thesis asks, "if we alter the risks for selfishness, would the population become more or less selfish?"

The prisoner's dilemma game is significant because it readily models conflict in cooperative situations. Scientists also use it to simulate evolutionary biology.

Altering payoff values for the supergame influences the success of the individual players. The author found that the payoff values do affect how the population evolves, if cooperation evolves at all. How the players interact also changes the evolution of cooperation. Random interaction provides a more realistic simulation of several kinds of natural behavior. Finally, counting the number of cooperative

outputs of the finite state machine proves to be another way to measure cooperation.

## NOMENCLATURE

FSM = Finite State Machine

GA = Genetic Algorithm

PGA = Parallel Genetic Algorithm

PD = Prisoner's Dilemma

IPD = Iterated Prisoner's Dilemma

XPD = Extended Prisoner's Dilemma

C = Cooperate

D = Defect

DC = payoff value for a player that defects against a cooperator

DD = payoff value for a mutual defection

CC = payoff value for a mutual cooperation

CD = payoff value for a player who cooperates with a defector

## LIST OF FIGURES

## LIST OF TABLES

# CHAPTER I: INTRODUCTION

Scientists often use the prisoner's dilemma (PD) game and the Genetic Algorithm to model and simulate people's actions. The prisoner's dilemma gives each player a choice between the move best for both players, if they can trust each other (cooperation), or the selfish but safer choice (defection). The combined choices result in a payoff of utilities (points) to each player. The prisoner's dilemma models conflict situations well.

In the original prisoner's dilemma, two people are in prison for a minor crime. A detective suspects that they are guilty of a much more serious crime, but lacks enough evidence to convict either of them. He goes to each prisoner privately. Each prisoner can "fink", where he tells on the other prisoner. Or, a prisoner can "stonewall", and not say anything to the detective. If one prisoner tells the detective about the other prisoner, and the other stonewalls, then the finker gets a reduced sentence, while the stonewaller gets hung. If both prisoners fink, then both receive a long prison sentence (the equivalent of the Defect-Defect outcome). If both prisoners stonewall (mutual cooperation), then they will only serve their current sentence. Clearly, both prisoners do well when they mutually cooperate. But cooperation leaves them vulnerable for the other prisoner to take advantage of the situation.

The prisoner's dilemma occurs in life in more subtle ways. For example, suppose a woman named Dawn loses her purse. While looking for it, she finds Bob's wallet. She could take the wallet to the police, and Bob would get it back. Or she could take the money and throw the wallet away. To keep things simple, though hypothetical, Bob happens to find Dawn's purse on the other side of town. Now he faces the same choice. If he turns in Dawn's purse, and she takes his money, then she comes out ahead, while he gets nothing. If both people take the money, then they at least get a little compensation. If both turn the items in, then both of them get their money, plus all of their possessions back, such as identification and family pictures. Dawn and Bob will both do well if they cooperate, though each can try to take advantage of the other. Their situation parallels the prisoner's dilemma game. Previous studies have found that the best thing to do when faced with this choice is to defect, to keep the other person from taking advantage. However, Dawn and Bob live in the same town, and are likely to run into each other again. Repeating (iterating) the game adds a new dimension to it. Cooperation emerges as a better strategy.

Varying payoff values on a population of prisoner's dilemma players produces a different future population. If Bob always carries more money than Dawn, then she might become a rich woman if she always defects every time she gets Bob's wallet. In game theory, the payoffs to the players need to be quantified. Instead of Dawn getting her purse back and Bob's wallet, she would receive a number of utility points. The prisoner's dilemma game is thus seen to be non-zero-sum, because the points Dawn wins does not have to equal what Bob loses.

Also, the players are not allowed to communicate with each other. Communication adds a dimension to the game that goes further into psychology than suits this experiment. The players know their opponent, though, which allows their past encounters to affect how they will play. The simple example of Bob and Dawn shows how two people could be faced with a prisoner's dilemma situation.

This kind of situation comes up all the time in life, where one person has the opportunity to take advantage of another. It also applies when a person can choose between being selfish or doing what is best for the community. Thus, scientists model social interactions with the prisoner's dilemma (PD) game. Figure 1 shows an example payoff matrix, used extensively in these studies. The payoffs are arranged as Player 1's payoff, followed by Player 2's payoff.

| | | Player 2 | |
|---|---|---|---|
| | | Cooperate | Defect |
| Player 1 | Cooperate | 3,3 | 0,5 |
| | Defect | 5,0 | 1,1 |

Figure 1 - Payoffs for the 2 player PD game [Axelrod, 1980]

By changing payoff values, the results of iterated prisoner's dilemma (IPD) games produce different rankings of players. For example, a sports magazine might compile a list of the 10 best college basketball players. If the three-point shot were taken away, the magazine would probably want to change its list. In the prisoner's dilemma context, a player that always defects (All-D) might do well in a population of players that always cooperate (All-C) using the payoffs given in Figure 1. However, would the All-C players do better if the payoffs changed? Let DC be the payoff for a defector when the other player cooperates, the exploiter's payoff. CC represents the payoff to a player for mutual cooperation. DD represents the payoff to a player for a mutual defection. Finally, let CD be the "sucker's payoff", the amount given to a player that is exploited. When changing the payoff values, then values must satisfy DC > CC > DD > CD in order to remain a prisoner's dilemma game.

The prisoner's dilemma can also be a three player game, as Figure 2 illustrates. The author does not work with the three player case in this thesis, but does refer to it in later chapters.

```
                 Player 2 ( Column )

                   C       D              C       D

             C     1,1,1   0,3,0          0,0,3   -2,2,2
Player 1
( Row )      D     3,0,0   2,2,-2         2,-2,2  -1,-1,-1

                        C                       D
                        |                       |
                        +------ Player 3 ------+
                                ( Box )
```

Figure 2 - Three player PD payoffs [Rapoport, 1970, page 80]

In this experiment, the author uses a tournament of PD players to produce rankings of the players, from best to worst. Finite State Machines (FSMs) represent the Iterated Prisoner's Dilemma (IPD) players. When these payoff values are altered, the same FSMs do not necessarily prosper. This thesis will look at this effect on evolving cooperation.

Random interactions are studied, as a more natural way to conduct a tournament of the players. Previous simulations have used a round-robin tournament, where every FSM plays every other FSM a set number of times. Giving the players "space", or coordinates relative to other players, also has been shown to affect the simulation, as the paper by Nowak and May [16] demonstrates. This experiment allows the players to move about randomly in a world. When two players meet on the same square, they play the PD game, after "remembering" the last encounter with their opponent. After the number of games played by the entire population exceeds a set average, the Genetic Algorithm evolves the population. This method of conducting a tournament is more realistic than the round-robin approach. The author ran experiments on other initial populations, and found that the results were consistent.

Due to the nature of this experiment, relevant computer science and biological terms will need to be explained before proceeding.

## A. Definitions of Terms

**All-C :** A prisoner's dilemma player that always cooperates.

**All-D :** A prisoner's dilemma player that always defects.

**Alleles :** The values of genes. "A gene for eye color...might contain the allele for blue pigmentation or the allele for brown." [Levy, page 162] A binary gene has an allele of either 0 or 1.

**a priori :** Deduced from theory, instead of experience.

**Artificial Morality :** "Artificial morality is a method for providing a fundamental justification for moral constraint." [Danielson, page 19].

**Chicken Game :** Also called "Hawk/Dove", this game is similar to the Prisoner's Dilemma, except for the payoff values. See Figure 3.

| | | Player 2 | |
|---|---|---|---|
| | | C | D |
| Player 1 | C | 3,3 | 1,4 |
| | D | 4,1 | 0,0 |

Figure 3 - Chicken (also known as Hawk/Dove) Payoff Matrix [Danielson, page 164]

**Chromosome :** A structure of genes, like a chain of DNA in cell biology. Often, this appears as an array of bits.

**Demes :** Possibly from "demesne", territory. An autonomous group in a population. Individuals mate only with members of their group.

**Epistemic :** Cognitive.

**Evolutionary Stable Strategy :** Maynard Smith coined this term in 1974 to describe phenotypes (for example, PD players) that resist being invaded by other phenotypes. If a single All-D player were to suddenly appear in a population of All-C players, the All-D player would thrive and reproduce. If it were to suddenly appear in a population of TFT (see definition later in this listing) players, the All-D player would not thrive. TFT is an evolutionary stable strategy.

**Extended Prisoner's Dilemma (XPD) :** A variation of the PD game, where the second player can react to the first player's move.

**Finite State Machine (FSM) :** A type of automata. The FSM has an initial state, an initial output (move), and an array of values that contain next state and next output information.

**Fitness :** Measurement of the success of an individual. A Genetic Algorithm uses a fitness function to evaluate all potential solutions, to rate them from best to worst.

**Game Theory :** Game theory uses mathematics to deal with cases where the best policy varies. Economists, the military, control theorists, and social scientists use game theory.

**Gene :** The basic unit of the chromosome, which holds one parameter's value.

**Genetic Algorithm (GA) :** A procedure developed by John Holland. It views a problem as a fitness function, and mimics natural evolution to make a solution. Reproduction, Crossover, and Mutation operators work on a "population" of potential solutions. The population evolves until it satisfies some stopping criteria.

**Genome :** "The entire collection of genes (and hence chromosomes) possessed by an organism." [Joerg Heitkoetter, 1993]

**Genotype :** How an individual is composed genetically. The instructions needed for building an organism.

**Hill-climbing :** "Hill climbing is named for the strategy that might be used by an eager, but blind mountain climber: go uphill along the steepest possible path until you can go no farther" [Luger and Stubblefield, 1993]. Look at the children of the current search-tree node, and select the best one to expand. This process continues until the children of the current node are no better than the node itself.

**Iterated :** The case where a game is played repeatedly. Technically, game theorists call an iterated game a super-game. Iterated prisoner's dilemma (IPD) is commonly studied for repeat behaviors.

**Panmictic :** A population where an individual can potentially mate with any other member, regardless of subdivisions. See also demes.

**Phenotype :** The organism itself. "The informational basis of the organism is known as the genotype. The expression of those genes results in a physical organism known as the phenotype." [Levy, page 161]

**Prisoner's Dilemma (PD) :** A non-zero-sum, non-communication game that is commonly played with two players. The players can either cooperate or defect.

**Schema** (plural **schemata**) **:** A template that describes gene values. It includes wildcard states, marked by *'s, that function as place-holders. For example, the schema *01* describes 1010, 1011, 0010 and 0011, but not 0110. The number of wildcards is also called the order of the schema. Holland explained the way a GA acts with schemata theory. "A schema describes sets of chromosomes that have similarities at defined bit positions." [Galletly, page 27]

**Stackelberg Game :** A game where each player moves in order, with certain information available, specified by the particular game. Player 1 might be a company, which "moves" by setting a production level. Player 2, another company, would observe player 1's output level before making its move.

**Sociobiology :** "Sociobiology is a specific application of Darwinian theory to animal behavior..." [Danielson, page 40].

**Super-game :** see iterated.

**Tit-For-Tat (TFT) :** A strategy developed by Anatol Rapoport. This strategy won two IPD contests held by Robert Axelrod. It cooperates on its first move. After that, it simply repeats its opponent's last move.

**Utilities (Utils) :** Utility points of a game. They can be thought of as "dollars", something of equal value to both players. The importance is the difference in utils that each player receives.

## B. Objective of this Thesis

This thesis will answer the following two questions. Does the payoff matrix affect how quickly the players reach cooperation, if they do reach it? Does a mobile population, where players interact randomly, affect the evolution of cooperation, as compared to a round-robin tournament? This thesis explores these questions by a series of simulations. The simulation modules will verify previous experiments, demonstrate the effects of space on the prisoner's dilemma players, provide PD payoff values that give unexpected results, and examine how these payoff values affect evolving cooperation. The computations generated by the simulations are interesting from a computer science point of view. Examining the PD game gives insight into the nature of complexities. Lessons learned apply not only to game theory, but also to human behavior. The author's contributes to the study of the iterated prisoner's dilemma by looking at a new type of tournament, where the players move randomly. The assumptions of this experiment follow.

## C. Assumptions

The 2 player IPD games will be considered. Any payoff matrix that violates the relative positions DC > CC > DD > CD will be rejected. The payoff values will be integers with a range of 0 to 255. The payoff matrix values will be the same for each player. For example, if both players defect, then both will receive the same score. The experiment will be done with a set of FSMs, each representing a player. The program initializes the FSMs randomly, but the initial population will be the same for every run of the "Wandering Players" program, due to the pseudo-random number generator. Thus, the same initial population produces different results, depending only on the payoff values. The players are not allowed to "know" the number of games to be played, so no end-game strategies are possible. As far as the players know, the game will repeat indefinitely. The players will be able to "remember" previous encounters with other players, by storing an array of states and outputs. During the reproduction and crossover phase of "Wandering Players", any player will be able to mate with any other player, depending only on fitness.

Allowing negative values for the payoff values might affect human players psychologically. Relative aspects would be the same, however. This thesis does not examine negative payoff values. The programs used for this study, though, do not prohibit the user from using negative values.

Previous studies have found that an increased population size does not yield better (or worse) results. [Fogel, 1993] The author found that Genetic Algorithm researchers prefer a minimum population size between 20 and 50 individuals, depending on the problem. A population size smaller than 50 should produce consistent results, and could be used in future studies to make a quicker experiment. However, this experiment uses 50 individuals. The author chose 500 as the number of generations after some initial tests. The average score of the populations tended to level off by 250 generations, if not earlier. The author extended the experiments another 250 generations, in order to make sure that the averages had reached a plateau. The author acknowledges the possibility that the average scores could change later if the experiment were to run indefinitely. The populations that did not reach mutual cooperation after 500 generations could possibly do so in future generations.

# CHAPTER II: LITERATURE REVIEW

The literature review covers the following topics.

A) Evolving IPD players,
B) The prisoner's dilemma used to study morality and rationality,
C) Evolutionary programming and Genetic Algorithms,
D) Genetic Algorithms and IPD players,
E) The parallel Genetic Algorithm and Darwin's continental cycle,
F) Cellular automata and the prisoner's dilemma,
G) The validity of cellular automata experiments,
H) Payoff values and their effect on evolving players.

Each topic is explained in detail in the sections below.

## A. Evolving IPD players

Axelrod gathered 14 PD playing programs from experts in differing fields, and had the programs play each other [1]. A fifteenth player, Random, who cooperated stochastically about half the time, also played. This was done to find out how to play the game well. Axelrod notes that a program's effectiveness depends on the other programs' strategies as well. The history of the game must also be taken into account for the programs to be successful.

All 15 programs played every other program, including a copy of themselves. There were 200 moves per game. The payoff matrix appears in Figure 1. If both players cooperated with each other every time, they would get 600 points. The average scores for the 15 programs ranged from 276 (Random) to 504

(Tit-for-Tat).

Tit-for-Tat was submitted by Anatol Rapoport, the author of "Two Person Game Theory" [20]. Tit-for-Tat simply cooperates on the first move, then copies the last move of the opponent. Many of the entries were variations of this idea, though they did not do as well. Some programs took advantage of the fact that they knew when the game was going to end, and altered tactics in the end-game. However, this did not account for much improvement. Axelrod noted this and changed the number of games played in the second tournament.

The programs fell into two groups. Axelrod [1] attributes the placement in each group with whether or not the program was "nice". He defines nice as meaning the program will not be the first to defect, at least before the end-game. All of the top eight entries were nice programs.

To play well against the Random entry, its opponent must give up on it early in the game. However, this presents a paradox, since the programs that tended to do well against Random did not necessarily do well with other programs. Axelrod defines "forgiveness" as the likelihood of cooperating after the opponent defects. Tit-for-Tat will defect once after an opponent defects, but only once. Other programs would try to teach the defector a lesson. Axelrod notes that the programs that were not nice tended to be unforgiving as well, and this lead to bad performance. Axelrod concludes that a program must be nice and "relatively forgiving" in order to be effective.

Tit-for-Tat demonstrated that reciprocity is a very good rule for the individual. Many programs that were variations of Tit-for-Tat did not take echoes of defections into account. This is important to look at "in an environment of mutual power". Niceness was a pleasant surprise to find as a rule which characterized the successful programs. Forgiveness played a part in keeping the opponents defections from echoing, and thereby improving the effectiveness. Some players served as "king-makers", affecting the rankings of the top players. The performance of a program was decided not only on its own strategy, but also by the other programs in its environment.

In order to better understand how to effectively play the iterated Prisoner's Dilemma game, Axelrod held a second round of competition between programs. This time, there were 62 entries, including a Random program. Each program was paired with every other program, including a copy of itself. The games were played an uncertain number of times in this second round, to keep the programs from using an end-game strategy. The score for cooperation by both sides throughout the game equals 453.

Axelrod defines tournament approach as "each player is trying to do well for himself and each player knows in advance that the other players are intelligent and are also trying to do well for themselves." [2] This second round promised to be much more sophisticated, since each person entering a program had the results of the first tournament. Some people entered programs based on the improvements that Axelrod noted at the end of the first tournament. Once again, Tit-For-Tat was submitted by Anatol Rapoport, and won first place. Many of the entries were variations of this idea, like the first round, but could not do better. Axelrod notes that no personal attribute of the contestants, such as nationality or programming language used, had anything to do with the performance of their program. The two properties of "niceness" and "forgiveness", established in the first tournament, accounted for how well programs did in the second round. Axelrod examined the results in comparison to 5 of the 63 programs. These 5 representative programs accounted for 96% of the variance of the tournament scores. Thus, these 5 could be used to predict the fitness of a new program, if it were introduced into the population.

There was only one program in the top 15 which was not nice. A third generalization of the successful programs emerged in this round. "Provocability" is the trait where a program responds to an "uncalled for" defection by the opponent program. "Uncalled for" is loosely defined as a defection where the opponent is trying to take advantage of an easy-going player. The lack of this trait led to the downfall of Tit-For-Two-Tats. Another program could (and did) exploit Tit-For-Two-Tats by defecting every other turn. Since two defections in a row would not happen, Tit-For-Two-Tats would not be provoked out of cooperating. Axelrod examined the robustness of the entries. Robustness is how good a rule rates in different environments. Tit-For-Tat still won 5 out of 6 of simulated tournaments, showing that it is very robust.

Axelrod looks at the tournament from an ecological aspect, leading to a genetic-algorithm-like process. He compares the rules to animals that interact with each other by cooperating or defecting. The average payoff would be an indication of the success of the animal in reproducing for the next round. The top ranking rules would become a bigger part of the population of animals in the next round. The average score would be weighted according to the relative success of other animals. The expected payoffs between the animal types is already determined, so the animal's future success can be easily simulated. The good animals become more numerous, while the bad animals eventually take on an insignificant percentage of the population. This survival-of-the-fittest simulation is not a Genetic Algorithm because the animals do not change, only the amount of them changes. Thus, no new animal types evolve. Success demands that the animal types perform well with other successful animal types. Axelrod found that most of the successful round 2 rules also did well in a survival-of-the-fittest simulation. Tit-For-Tat stayed the best, and was still growing by 0.05% per generation at the end of 1000 generations.

The final question is whether Tit-For-Tat is the best Prisoner's Dilemma rule, or just the best one seen in this experiment. The author cites three reasons why Tit-For-Tat may not be the ultimate Prisoner's Dilemma rule. First, if the Random program could be identified by another rule, this rule could outscore Tit-For-Tat. When TFT and Random play, both do poorly. A program that always defects against Random, yet plays like TFT with the other programs, would beat TFT. Rules have tried ways of identifying the Random player, but not with much success. Second, he notes that Tit-For-Tat would have come in fourth if only the top 31 rules had been entered. Third, environment always plays a role in determining the best rule. Tit-For-Tat is robust, though, and no rule can exploit it. The properties of niceness, forgiveness, and provocability are all parts of Tit-For-Tat that make it robust.

Next, Axelrod looked to evolve good iterated prisoner's dilemma (IPD) players [3]. Though cooperating players of the IPD do well, if one were to change to a defector, it would do better. Axelrod defines an iterated prisoner's dilemma as one where the individual recognizes other players, and remembers some of their previous plays. A strategy is defined as a rule deciding whether or not to cooperate based on the past. The Genetic Algorithm evolves the strategies.

The Genetic Algorithm (GA) was invented by John Holland. It is a subset of artificial intelligence. In the GA, the strategies are represented by a string of numbers called a chromosome. Changing the chromosome alters the strategy. The chromosome consists of the initial history of three previous plays, followed by the play to make for each of 64 possible past histories. The two-player outcome makes up each history. The total number of C or D's represented by this chromosome is 70 genes, also called loci. There are $1.2 * 10^{21}$ strategies that can be represented this way, a number much too large to try every one. The Genetic Algorithm provides a quick way to search this space for good strategies.

The Genetic Algorithm for this type of simulation has 5 steps. First, a population of chromosomes is initialized, often randomly. Second, each individual, represented by the chromosome, plays the iterated prisoner's dilemma with other individuals. The average score from the tournament is the individual's fitness. A more generalized GA would have a fitness function in the place of this tournament. Third, the individuals with above average fitness values are selected for reproduction. Fourth, the selected individuals pair with a random partner, and produce two offspring. Crossover and mutation occur during the mating of individuals in this step. Finally, the offspring replace the parents, keeping population size constant.

Specifically for Axelrod's simulation, 20 individuals comprised the population. Crossover occurs about once for every chromosome. A mutation happens about once every other chromosome. The players move 151 times during the game.

Instead of the individuals playing each other, they played against 8 representative strategies from Axelrod's earlier tournament [2]. 50 generations made up a single run, and the experiment carried out 40 runs.

Five behaviors emerged that gave some individuals an edge:

1 Don't defect if the last three plays are mutual cooperations.
2 If the other player suddenly defects, then defect too.
3 If the other player "apologizes" by cooperating again, then cooperate as well.
4 Continue to cooperate in the above case.
5 Continue to defect if the last three plays are mutual defections.

These behaviors are like the Tit-For-Tat strategy, and the individuals displaying these performed about as well as Tit-For-Tat. The test strategy called "Adjuster" changes its defection rate in order to exploit other players, and succeeded 95% in exploiting the rules displaying these behaviors. Adjuster also exploits Tit-For-Tat.

Other strategies evolved that were able to do better than Tit-For-Tat. These discriminated among the other players, and exploited them when possible. They were able to exploit the vulnerable players without making enough mistakes to lower their score. They were not "nice", breaking the rule that Axelrod established in his first tournament [1]. They would start by defecting, but would "apologize" to gain mutual cooperation with the non-vulnerable players. Though they did better than Tit-For-Tat about 25% of the time, Axelrod quickly points out that they are not robust, and would not be able to pull this off in other environments. He notes that the GA evolves specialists in specific environments.

Axelrod concludes with a few points about Genetic Algorithms. First, the Genetic Algorithm searches large spaces efficiently. Second, the search process is helped by sexual (two parent) reproduction. Third, he found that convergence can happen arbitrarily. For example, an ineffective gene can ride the coat-tails of an effective gene, and become prevalent. Also, he noticed that the initial play and initial state of the chromosomes rapidly become fixed, though these initial values do not seem to matter. Fourth, flexibility and specialization have a reciprocal relationship, though both are valued. A specialized individual might score highly. Giving the individual many offsprings would produce more successful individuals, but the variety of the population would suffer. This variety might be necessary in the long run.

Simulating nature in experiments like this leave a lot to be desired. The simulations of today are highly abstract, with small populations and few generations. The reproductive process is very simplified, with no actual sexes, and two children are usually produced. Despite these shortcomings, complex strategies can be evolved. This gives us a glimpse of what the future might have to offer. Evolution can be "played with", as a design problem. With the advances that microbiologists are making, we should be able to simulate genetic systems to answer our questions.

**B. Prisoner's Dilemma used to Study Morality and Rationality**

Danielson [5] examines the subjects of morality and rationality. He defines morality as the choice good for society, whereas rationality is what's good for the individual. Are these two fundamentally opposed? He looks at three problems, each like the prisoner's dilemma. First, the commuting problem gives each individual a choice between traveling on the bus (public transportation) and taking a private car. Second, the greenhouse problem consists of burning more or less fuel. Third, the hacker's dilemma presents the case of a person proficient with computers. Does he use his knowledge beneficially to all, like producing a shareware tool, or does he do whatever makes him happy, such as the person who created the Internet worm? Artificial Morality is morality derived from rationality.

Games make good models for this problem. The Extended Prisoner's Dilemma (XPD) varies from the prisoner's dilemma in that the second player has the opportunity to react to the first player's move. This represents a social contract, otherwise called the compliance problem. Danielson gives an example of two farmers, one who plans to move. If the farmers agree that both will harvest farmer 1's crops this week, and farmer 2's crops next week, then should farmer 1 help farmer 2? Since one farmer will leave after the harvest, the sucker farmer will not be able to reciprocate (as in the iterated case). Games are used, but not necessarily game theory. Danielson does not assume that both players have common knowledge of subjective information. Game theory, he points out, assumes that both player are rational, which adds a psychological element to the players. For further discussion of the notion of rationality as used in game theory, the reader is referred to Rapoport. One cannot assume that people think through every decision. He goes on to separate preferences and interests. Though a person would prefer to stay warm, she will be interested in going outside into the cold to get food. Humans have levels of needs that depend on the environment, and complicate their behavior.

Sociobiology has been used to show kin and reciprocal altruism, but cannot explain restraint. "Sociobiology is a specific application of Darwinian theory to animal behavior..." [Danielson, page 40]. Agents (organisms) who recognize others in their environment, and who are likely to interact with these agents in the future, can be modeled with the iterated prisoner's dilemma (IPD) game. For example, predators have been observed to seldom fight each other. The IPD game reveals that they gain no advantage in the long run by fighting other predators. Danielson claims the IPD strategy Tit-For-Tat is a maximizing strategy which gives the best outcome. Therefore, Tit-For-Tat is not moral, and shows that an IPD player does not need moral constraint to succeed. Also, some social situations are not modeled well by the IPD [Danielson, page 50]. As a result, Danielson eliminates iteration from his work.

Danielson introduces XPD players, ones that are flexible, and ones that learn. A coordination problem exists when two players, who do not have matching principles, fail to co-operate with each other. A strategy adapts by changing its principles, which can lead to the above problem. Learning can be risky, especially in conflict situations, such as a chess game. An opponent may intentionally mislead. Danielson's game players are not in a totally friendly environment, but are also not in pure conflict. Thus, he deems learning as beneficial to his players. One learning strategy he calls Copycoop, which tends to adopt the principles of other strategies that have co-operated with it. Learning lets the strategies get around the coordination problem. Also, the learners can adopt moral goals. Finally, the learners provide a good population to conduct a tournament, where the initial strategies can improve themselves.

The moral agents (players) exchange information with other players. Each player makes its principles known, and responds to the principles of the other players. The author studies the importance of information exchange between players, and the costs to the players of making their principles known. Some straightforward maximizers (SM's, such as Tit-For-Tat) use the public information to cooperate with constrained players. Thus, it is possible for a non-moral player to do well in the XPD game. Also, co-operation can exist in this game without mutual constraint. Because of the costs of information exchange, it pays for the players to be moral to different degrees. For the two player XPD game, Danielson finds reciprocal cooperators to do the best. He argues that this strategy exhibits rationality and morality, thus solving the compliance problem.

The game of Chicken might provide a better model for artificial morality. Other people call Chicken the Hawk/Dove game. Please see Figure 3, in the Definitions section. It differs from the prisoner's dilemma because both players suffer most when both defect. To reinforce his conclusions, Danielson looks at the Chicken game. He claims that the Chicken game makes a good model because of its aggressiveness. Each player has the choice to concede to the threatener (the rational choice), or to resist the threat (the moral choice). The conceding choice is abbreviated "C", and the resisting choice is abbreviated by "D". It costs a player more to resist than to concede. Since a player can get a "free ride", this game better approximates social interactions. Once again, the author examines the extended version of the game, where the second player can react to the first player's choice.

Danielson develops several new strategies to play this game. The rational strategy, he points out, is the Less Broadly Cooperator (LBC). It has the properties of discriminating in the concessions, and resists threats of amoral players (straightforward maximizers). Therefore, the LBC strategy is both rational and moral to a point.

The book ends by arguing for the validity of this study. Although this model may be too simple to apply to people, it can apply to organizations. The moral players have roots in straightforward maximizers, and can revert back to a SM if they are not performing well. Therefore, the moral players do just as well, if not better than, the strategic players. Danielson demonstrates that the XPD and the extended Chicken games simulate natural interactions well.

Similar to the XPD game, Shinkai [10] looks at an oligopolistic market with random demand to see if a three-person Stackelberg game has reduced advantages for the first two players. Stackelberg games often model business firms. The two-player case shows that it is more profitable for the first player to tell the second player his private information than to let the second player guess at it. The three player case, however, shows no gain for the first player to form such a coalition. Under some conditions, the second player ends up with the least amount of profit. Assuming equilibrium, the second player has an incentive to share his information with the third player. Player 1 does not have an incentive, so a coalition about private information will not be formed. For the two-player case, both players do better to cooperate. In the Stackelberg game, revealing private information can be thought of as the cooperating choice.

## C. Evolutionary Programming and Genetic Algorithms

Andersen et al [8] apply evolutionary programming (EP) to successfully minimize the objective function of a system identification model. EP works for Single-Input-Single-Output (SISO) systems, and Andersen shows that it works for Single-Input-Multi-Output (SIMO) systems also. Evolutionary Programming uses evolution to search for a multi-parameter solution. Note that EP does not use crossover like the Genetic Algorithm. The EP algorithm appears below.

```
1. An initial population is created.
2. Find the fitness of every individual.
3. Each individual gets mutated, where the more
   successful individuals are less likely to mutate.
4. Find the fitness of every individual, again.
5. Have the individuals compete amongst each other.
6. Rank the individuals, based on the competition.
7. Using only the high ranked individuals, Goto step 3.
```

Figure 4 - Evolutionary Programming Algorithm [17]

The algorithm applies to the SIMO system by finding the system matrix, and the input matrix. Andersen examined two mutation strategies. The first mutated according to the fitness value, as described above. The second method mutated according to the iteration number, and performance improved. The objective function takes noise into account, and determines the cost linked to each parameter. The costs are dimensionless, and are based on the confidence. A noise-free parameter will contribute largely to the cost.

Andersen experimented with a critically damped second-order spring system. The system identification took about 300 iterations. When the order of a linear SIMO system is known, EP can identify the system's parameters. MIMO systems will be identified in the near future. Andersen's experiment was not conclusive, for a few reasons. The model did not take process noise into account. Also, the order of the system must be known. Despite these shortcomings, EP promises to become a powerful tool for solving adaptive control systems.

Both EP and GAs are subsets of artificial intelligence. Thus, the GA is a powerful tool for solving widely varied problems, like its cousin, evolutionary programming. Homaifar et al [4] list the following differences between Genetic Algorithms (GAs) and more standard search techniques. First, GAs need a set of parameters to be optimized. These are represented as a string. Second, a population of strings is used at the same time, so solutions develop in parallel. Third, instead of using deterministic rules, GAs depend on probability to guide the search. Finally, GAs evaluate each string with a fitness function. An initial population is generated randomly, and the GA operators evolve the population until the fitness function detects a solution.

The operators of a Genetic Algorithm are reproduction, crossover, and mutation. Reproduction is similar to a "roulette wheel", where the strings that have the better fitness values also have the better chances of reproduction. Order-based crossover takes parts from each of two parents, and creates two new strings from re-combining these parts. Order-based mutation selects one gene at random, and switches the bit value. A Genetic Algorithm can have other forms of operators, but they are outside the scope of this thesis.

## D. Genetic Algorithms and IPD Players

Stanley et al [9] use the Iterated Prisoner's Dilemma (IPD), where player's scores are accumulated. The number of possible players is fixed, but the players can play many different other players during an iteration. The only information that the players have about other players is the history of payoffs from games with that player. The payoffs for this experiment are the same as in Figure 1. During each iteration, each player can choose which of the other players are tolerable game players. When the expected payoff for playing an opponent falls below the tolerance level, then the player refuses to play the opponent.

Each iteration has the following stages:

1) Choice stage. Each player offers to play up to K others.
2) Refusal stage. For every received offer, reject the ones where the expected payoff is too low.
3) Play stage. Play out the PD games.
4) Cleanup. Calculate payoffs for everyone.
5) Update. Update expected payoffs.

Players are represented by automata. There are six types studied by Stanley et al:

| | | |
|---|---|---|
| 1) | Always Defect | (AllD) |
| 2) | Ripoff-Artist | (Rip) |
| 3) | Gentle Ripoff | (GRip) |
| 4) | Tit-For-Tat | (TFT) |
| 5) | Tit-For-2-Tats | (TFTT) |
| 6) | Always cooperate | (AllC) |

Whether a player chooses to cooperate or defect depends on its state. States are changed according to the opponents last move. After each tournament, the automata are replaced with a new population. The worst are discarded, and the rest are retained for reproduction (and crossover). Bit values describe each finite state machine (FSM). There are 16 possible states, requiring 4 bits (0000 to 1111) to represent. The first bit is the initial move (Cooperate of Defect). The next 32 * 4 bits are arrows, pointing to the next state. Finally, 32 additional bits are needed to label the arrows.

The experimenters found that including choice/refusal in the IPD supergame results in new features of evolved players. Choice and refusal brings mutual cooperation into evolved populations earlier than IPD supergames without it. The player

All-D eventually becomes intolerable to All-C players. Strategies also shun All-D, Rip, and GRip, depending on their tolerance and memory. Payoff bands emerge, where a group makes stable average payoffs, despite the fact that the members are diverse. Choice and refusal allows a broad range of new interactions among players.

## E. The Parallel Genetic Algorithm and Darwin's Continental Cycle

Darwin's continental cycle theory says that production of new forms of life is most likely to happen on a broken-up large continent. This theory about the population structure on evolution is examined in Mühlenbein's paper, using a parallel Genetic Algorithm (PGA), and the Prisoner's Dilemma.

Darwin said, "I conclude that a large continental area, which will probably undergo many oscillations of level, and which consequently will exist for long periods in a broken condition, will be the most favorable for the production of many new forms of life, likely to endure long and spread widely." [12] In order to avoid over-specialization, and yet allow new life forms to emerge, Darwin imagined a situation where a large continent breaks up into small islands, then re-combines to form a continent again. He saw evolution as a result of the way a group of organisms was put together, instead of an individual property.

Reproduction, selection, and variance are the three mechanisms of evolution, and simulating evolution on a computer is not a new idea. John von Neumann invented automata theory and was able to make an automaton that could reproduce itself [18]. Though von Neumann did not go on to include the other two evolution mechanisms in his theory of automata, the Genetic Algorithm does take them into account. Mühlenbein decides to use a parallel Genetic Algorithm for a variety of reasons. One reason is that they run with maximal efficiency on parallel computers.

There are a few differences between a PGA and a GA. First, a PGA has a spatially structured population. Second, the individuals choose their mates. Third, the parent is replaced by the child. Finally, the individuals can improve their fitness. Mühlenbein argues that the PGA is more realistic when it models natural evolution. Many mathematical models simply ignore the idea that an individual would take up space.

A spatial population structure is accepted by biological researchers to have more variety than a panmictic one, where any two members can mate. "It is a well-known fact that the GA suffers from the problem of premature convergence", says Mühlenbein [p 463]. Many GA's implement diversification by rejecting offspring that are not sufficiently different from the rest of the population. Mühlenbein's PGA diversifies the population via the spatial structure. An individual's fitness is determined by its neighborhood. It chooses a mate from within that neighborhood as well. This PGA stays with the biological paradigm.

Two models are examined. One is the isolation-by-distance model, where individuals have a finite home range. The other is the stepping-stone model, where individuals can migrate between adjacent islands only. Higher fitness peaks are found when migrants mate with individuals in the foreign population. Mühlenbein credits the static fitness function with the success of the stepping-stone model. Real evolution, he says, cannot be modeled with a static fitness function. Thus, he turns to the iterated prisoner's dilemma (IPD) game to provide a fitness evaluation. The strategies of the individual players are coded into genes, and the PGA changes these genes in every generation. The programmer must first represent the problem in genetic terms. The algorithm for the PGA appears below.

```
1)  Make a structure for the population, and initialize
    the population.
2)  Have each individual do local hill-climbing.
3)  Have each individual select a mate within the
    neighborhood.
4)  Use crossover between the parents to create an
    offspring.
5)  Have the offspring do local hill-climbing.
    If it does better than (acceptance criterion),
    Then have the offspring replace the parent.
6)  Goto step 3, unless done.
```

Figure 5 - The Parallel Genetic Algorithm
[Mühlenbein p 465]

The individuals select mates based on their desirability, presumably their fitness. An offspring usually has half of its genes from each parent. According to Mühlenbein's earlier works, this PGA has done much better than the GA on optimization problems. Mühlenbein's interest here is in the evolution of the whole population, instead of finding one optimal strategy. He uses a look-up table to represent strategies instead of a FSM. The look-up table has unused portions within it. For example, the table would store the play that an All-C strategy would use if the last game was Defect-Defect. This play would be irrelevant, since Defect-Defect will never be the outcome of a game played by an All-C strategy. He argues that this is a feature, modeling real-life evolution. This problem would also exist in a finite automaton representation, since a FSM would be likely not to use every possible state. He notes that the crossover operator does not compare to real evolution, though.

Mühlenbein simulates the continent cycle by periodically breaking the continent of IPD players into islands, then recombining them. He notes that the variation of strategies is reduced during the continent phase. When compared to a couple panmictic populations, the continent cycle takes longer to reach cooperation. But the continent cycle's average payoff does not dip below 2.00, as do the panmictic populations. Overall, the continent cycle produces a better population. The ring population, where each individual has 4 neighbors, oscillates because the PGA mate selection is too "soft" for this structure. Mühlenbein also finds that an island of All-D players cannot successfully invade a population of 9 All-C players. A group of All-D players successfully invades a large panmictic population, however. Also, a continent cycle is the first of the populations studied that evolves to a cooperating population from a group of All-D players. These results seem to confirm Darwin's theory. Note that the winning cooperative strategies tended to look more like Tit-For-Tat than All-C.

## F. Cellular Automata and the Prisoner's Dilemma

Nowak and May consider only two types of PD players in their simulation. These players either always defect or always cooperate. Also, the players do not keep track of previous encounters. Every player on the matrix plays PD with its neighbors. The highest scorer gets the square. The spatial patterns that emerge from this game usually fluctuate back and forth. Physics and biology are cited as two fields that can benefit from the implications of this game.

Each square on the n x n lattice holds a cooperating or defecting player. The owner of the square plays with its neighbors, and the square is taken over by the best scorer immediately around. The payoffs for this game are CC=1, DD=0, (the defect when the opponent cooperates) DC=b, and CD=0. "b" is the only parameter. Nowak and May found that making the DD payoff slightly larger than 0 had no effect, so that violating DC > CC > DD > CD was not a problem. Also, the results are the same whether or not the boundary squares are connected to the opposite boundary squares. The value of b was found to be most interesting between 1.8 and 2.0. When $b>1.8$, clusters of D players grow, whereas they shrink when $b<1.8$. When $b>2$, clusters of C grow, but they do not grow when $b<2$. Thus, when $1.8 < b < 2$, the patterns that are produced vary wildly. The number of cooperators stays around 32% of the population for this range of b. This would be expected for the large, symmetrical matrices. It also works for the chaotic, irregular ones, though the authors cannot explain this.

Each player can have one of four colors. The cooperators are blue if they were cooperators in the last iteration, or green otherwise. The defectors are red if they were defectors previously, or yellow if they were not. They generate an "evolutionary kaleidoscope", examples of which appeared in Nowak and May's article. Though the results look like Conway's "Game of Life", the fate of every cell depends on 25 other cells, instead of just 9. Every cell in question changes depending on its 8 neighbors. Each of these neighboring cells depend on their neighbors to determine their success. Thus, every cell depends on the two layers of cells around it. The authors point out that the patterns generated by their program are both complex and regular in a way unlike any other cellular automaton.

"We believe that deterministically generated spatial structure within populations may often be crucial for the evolution of cooperation, whether it be among molecules, cells or organisms." [16] [Nowak and May, page 829] This paper is important because it shows the validity of cooperation in a territorial environment, even when there is no memory of past encounters.

The strategies of Nowak and May's experiment fluctuate depending upon the payoff values. The moral of the Nowak and May article is "territoriality favors cooperation" [Sigmund] [15]. Sigmund points out a few real-life examples to back this up. "Cellular automata provide a natural framework for evolutionary models with spatial structure." [15] J. H. Conway's "Game of Life" is a classic example of this. Nowak and May combined evolutionary game theory with cellular automata in their paper, and their experiment's results can be unpredictable.

## G. The Validity of Cellular Automata Experiments

Huberman et al [14] discuss the paper by Nowak and May [16], where the number of cooperators evolve to the same percentage of the population, given certain parameter values, but regardless of initial conditions. Thus, there could be a universal constant for 2 dimensional prisoner's dilemma simulations. Cellular automata can produce some interesting patterns, but their behaviors might not map to anything in the "real world".

A computer simulation and a real-world experiment are not equivalent. Sometimes, the differences can be enough to invalidate the social implications. Huberman and Glance analyze the differences between computer simulations and real-world systems. Then, they try the same simulation in synchronous and asynchronous modes in order to show that the results are not always the same.

A cellular automata simulation is only a model of the dynamics of a natural process. The simulation uses discrete time increments, with all players updated at the same time. This can be defined by a finite difference equation. To a process in nature, time is relative. Usually the elements get their updates with occasionally flawed information at different times. Differential equations describe this asynchronous environment. The two types of equations are not guaranteed to have the same solutions. Therefore, a computer simulation of a real-world system should have asynchronous and continuous time increments.

A square lattice of prisoner's dilemma players was simulated synchronously [14]. A defecting player occupied the center position at the beginning. Cooperators made up the rest of the "world". After 217 generations, a symmetrical pattern appears, looking a lot like a black and white fractal image. Huberman and Glance carry out the same experiment, but with asynchronous time, updating only one player per time increment. The pattern that appears after approximately 100 generations is a completely white image, where all of the players are defecting.

When real-world systems have delays, it could be argued that synchronous simulations are valid. The scores would depend on player's past and current performances. But the authors found that asynchronous updating of delayed scores effected the evolution of cooperation, depending on the delay.

Computer simulations that study real-world systems should be done carefully, so that the conclusions are valid. The systems that people see in nature need to be modeled with continuity and asynchronous time.

Fogel's paper [17] looks at conditions that result in behavior patterns in a clear-cut, but evolutionarily complex world. He reminds the reader at the end that these conditions are not necessarily what is to be found in the "real world."

## H. Payoff Values and their effect on Evolving Players

In the IPD supergame, two rational players stand to gain when they can learn to "trust" their opponents. Fogel's paper attempts to evolve a stable strategy for the prisoner's dilemma. He quotes two rules from Rapoport in 1966, "DC > CC > DD > CD", and "2*CC > CD + DC" [17].

Axelrod held a tournament in 1979 that used the payoff values of Figure 1, where Tit-For-Tat won [1]. Later studies have tried to find a more stable strategy by using a Genetic Algorithm. Many of the resulting strategies of Axelrod's second tournament [2] acted very similar to Tit-for-Tat. In another experiment, the evolving rules played each other. Effective rules propagated, while ineffective rules were replaced. A trend of reciprocating cooperation eventually took over the population.

The FSMs used to represent strategies have finite input and output alphabets, and a limited number of possible states. The finite (and fairly small) number of bits needed to represent a FSM make it a good candidate for evolutionary programming. The number of possible states for this experiment was set to 8. The output alphabet consists of {C, D}, and the input alphabet appears as (previous move, opponent's previous move), or { (C,C), (C,D), (D,C), (D,D) }.

Fogel's first experiment merely confirmed that cooperative behavior would evolve. "In each evolution, there was an initial tendency for mutual defection, but after a few generations (e.g., 5 to 10), machines evolved that would cooperate if they recognized other cooperative machines." [17] Eventually, the defectors were driven almost to extinction, as expected.

Fogel performed a second experiment to see the effect of population size on rates of cooperative evolution. He found that a smaller population size does not mean quicker cooperation. In fact, there is little evidence to show that 1000 parents make a better sample than 50 parents. Fogel notes that cooperation is usually seen by the tenth generation. Also, "all of the best evolved FSMs are initially cooperative." [17] When these best evolved machines play against themselves, cooperation on both sides happens quickly. This could be due to machine recognition of other machines that reciprocate cooperation.

Fogel's third experiment was to see how changing the payoff values affects cooperation evolution. If Rapoport's second rule were violated, would a high DC value preclude cooperation? This experiment revealed that most parent FSMs scored a mean of approximately 3.0, when DC payoff values ranged between 5 and 6. The FSMs would have a mean score of 3.0 if they mutually cooperated every time, however this is not the case. Many of the FSMs were found not to be cooperative when DC > 6. Fogel speculates that the best of the parent machines may have tended to produce gullible offspring that the parents could exploit. He concludes that mutual cooperation tends not to evolve when DC > 6. Besides the parents taking advantage of their offspring, alternating cooperate and defection in this case leads to an average payoff greater than 2 cooperations.

The payoff matrix has much to do with whether or not mutual cooperation will evolve. Fogel identified three critical regions:

$2 * CC > CD + DC$
$2 * CC = CD + DC$
$2 * CC < CD + DC$

This boils down to how good 2 mutual cooperations compare to an alternating cooperation, defection. Cooperative behavior will evolve when it is "profitable" for the machines. If 2*CC = CD+DC, then behavior will be stochastic, depending only on the initial FSMs.

In the following chapters, the author presents a sequence of simulations to study cellular automata, a population of mobile iterated prisoner's dilemma players, the effects of payoff values on IPD players, and the evolution of cooperation. First, the author discusses the design and approach of the experiments. Next, he discusses the simulation modules. Common elements of the programs are then detailed. After the procedure chapter, the author presents the results and analysis. A discussion of the results follows. Conclusions and recommendations are drawn from the discussion. Finally, the future study chapter talks about experiments that could be done in continuation of this research.

# CHAPTER III: DESIGN AND APPROACH

The author constructed a series of programs to study the effects of payoff values on evolution of prisoner's dilemma players. Each of these programs had a separate goal, and produced separate results. However, each program used pieces of the program before it, in order to cut down on the time to completion. The fourth and final program combined elements and ideas from the previous three. In a sense, the final program evolved from its predecessors.

The first program recreated the simulation that Nowak and May [16] described. Some additional features were added, including "walls", which ties in Darwin's continent theory. The program's goal was to show the effects of cellular automata that Nowak and May described, and to examine how including barriers affects the players.

The second program kept the idea that the players would take up a spatial location, but the players moved randomly about, playing only the other players that come into contact. The keyboard controlled one player, allowing the human user to interact with the players. The user moves where he/she wants, and can switch between cooperation or defection. The other players either always cooperate or always defect. The author designed this program to be like a video game. The user moves about, trying to stay alive. Each player started with an average of 3.0, as if it had experienced a few mutual cooperations. However, when a player's average falls below a 1.75, the player dies. For a human to play this simulation successfully, he/she must make his/her player strong enough to withstand the mutual defections that surely will occur when the cooperators die out. A player becomes strong by bumping into cooperators. The more experienced a player, the greater chance it has for survival.

The third program diverged from the themes of the earlier two. The author suspected that all prisoner's dilemma games are not created the same. The payoff values would affect how well a player performs. A player, such as All-C, might have 10 DC payoffs and 10 CC payoffs. Another player might acquire 10 DC payoffs and no CC payoffs, and have a higher score because of this. Now suppose the payoff value for CC increases. All-C might gain the higher score because of this increase. The third program examined the effect of different payoff values on the rankings of finite state machine players who played each other a total of 50 times.

The final program combines several elements from the above programs. The FSM players wander about a simulated world, playing the PD game every time it encounters another player. The random interactions allow for delays like Huberman and Glance [14] mention, making the simulation more realistic. When a FSM bumps into another FSM, they play the PD game, and their scores are affected by the payoff matrix that is the "law" of the land, set at the beginning by the user. Walls exist, randomly placed, and serve as barriers to the players. The author keeps the population size fixed, in order to keep computation time to a minimum. Since the players are FSMs, each player has memory of previous encounters. The players start out randomly initialized, but the Genetic Algorithm evolves the FSMs. Thus, better players emerge. The final program shows the effect of the payoff values on the evolution of the inhabitants. The algorithm for the final program appears on the following page.

```
1) Create a class for prisoner's dilemma players as
   FSMs

2) Create a "world" where the players can exist.
   The world has boundaries (walls) and a payoff matrix
   that the user can specify.

3) Initialize the FSMs randomly. Distribute the players
   throughout the world.

4) Reset the player's scores, memories, and other
   variables.

5) Allow the players to wander randomly, playing the PD
   game with every other player that they bump into.

   a) Each player should remember the encounter.

   b) When the players, as a group, have played a
      certain amount of games, move to the genetic
      algorithm phase.

6) Use the Genetic Algorithm to evolve the population.

   a) Reproduction:
      i) Evaluate each player's fitness by finding their
         average score.
      ii) Determine the average score for the group.
      iii) Eliminate all below-average players.
      iv) Give each player a chance to reproduce based
          on fitness.

   b) Crossover:
      i) Pick 2 parents randomly.
      ii) Make a new player for each dead one by 2-point
          crossover of the parent FSMs.

   c) Mutation:
      i) Determine which players will have a mutation.
      ii) If so, randomly change one bit of the FSM
          code.

7) Continue steps 4,5 and 6 for 500 generations.

8) Generate a report about the players.
```

Figure 6 - The Algorithm of "Wandering Players".

Several C++ classes are included in the final two programs. These classes are the finite state machine, the player class, the people class, the barriers class, and the pay_matrix class. Each will be explained in the following text.

The finite state machine was easily represented as an array of 17 bytes. The first FSM byte represents the initial play and the initial state of the FSM. The number 0 represents the Defect play, and 1 represents the Cooperate play. These are the only two possibilities, so one bit represents the play. There are 8 possible states, numbered 0 to 7, or 000 to 111 in binary. To represent a combination of play and state, only 4 bits are needed. Therefore, the first 4 bits of the FSM are not used, but the lower 4 bits represent the initial play and initial state. The play/state combination can be thought of as an arrow, pointing to a new state. The second and third bytes hold the 4 arrows for state 0, one for each input DD, DC, CD, and CC, respectively. Likewise, the 4 arrows leaving state 1 are held in the fourth and fifth bytes. Pairs of bytes represent the arrows for the rest of the states. See Figure 7 for a close up of how to encode state 0 of a FSM. Figure 8 shows a drawing of a complete FSM.

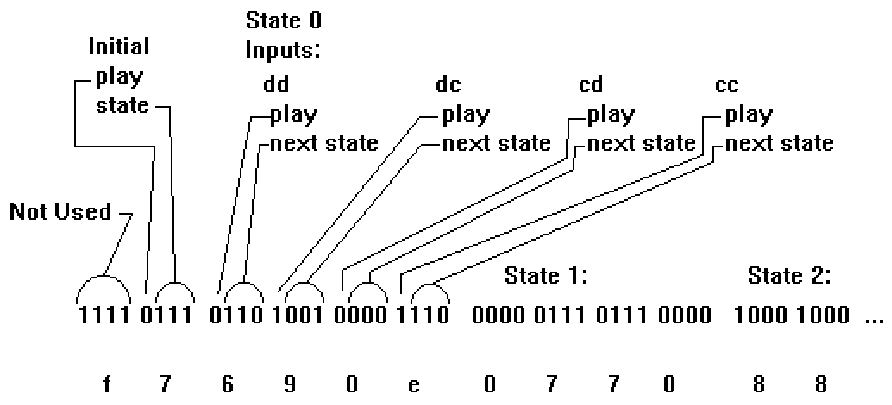
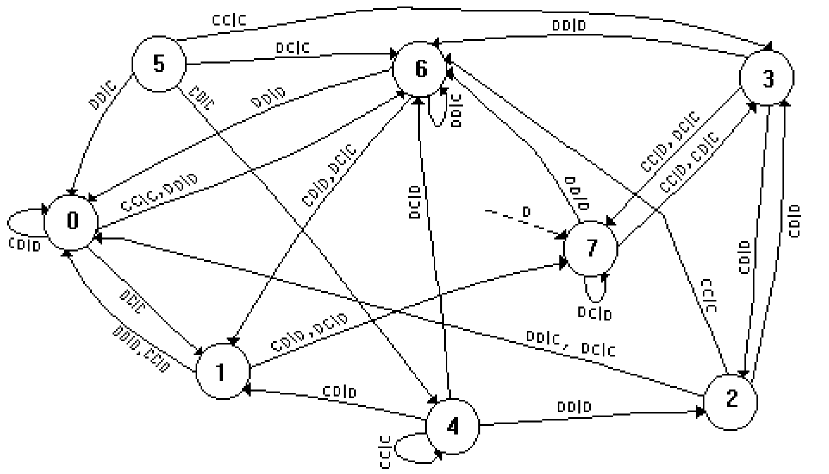


Figure 7 - How the FSM is encoded



**This FSM was ranked "best" after evolving the population with payoffs 5, 3, 1, 0.**
**The code : f7690e0770883e0b27c81f964be13867b3**

Figure 8 - Drawing of a FSM

The player class includes the FSM array, and a history array. One byte for every player comprises the history array. The history array stores the last play between the player and every possible opponent. During initialization, the program copies the first byte of the FSM to every position of the history array. The player class has several variables that each player needs. These include the score, count of games played, counts of each type of PD outcome, X and Y coordinates, color, name, and fitness. The program calculates the fitness by dividing the score by the count of games played. Not all of these variables are in use by both programs. For example, "Payoffs and Rankings" do not use the X and Y coordinates, nor the color. Figure 9 shows the matrices class for "Payoffs and Rankings". The player class appears in Figure 10, as well as the people class. The barrier class, and the matrices class used in "Wandering Players" appears in Figure 11.

The people class includes the player class. A player object can be thought of as an individual, while the people object would be the society as a whole. The people class includes primarily functions for the players. The genetic operators reproduce, crossover, and mutate appear in this class. There are functions reporting information about the players, such as report, print, who_is_avg, and coop_count. Report generates a detailed report of each player, whereas print(player number) outputs the FSM for the player number

```
/****** Matrices class  ******/
class matrices
{
  class pay_matrix
  {
    public:
      int delta1;
      int delta2;
      int delta3;
      int cc;
      int cd;
      int dc;
      int dd;
      float rank[num_players];
      int sorted[num_players];
      int sorted2[num_players];
      // An array of how the players ranked.
      pay_matrix()  { init(); };
      void init();
      // overload the = operator
      pay_matrix & operator = (pay_matrix & pm)
      {  int i;
         this->delta1=pm.delta1;
         this->delta2=pm.delta2;
         this->delta3=pm.delta3;
         this->cc=pm.cc;
         this->cd=pm.cd;
         this->dc=pm.dc;
         this->dd=pm.dd;
         for (i=0; i<num_players; i++)
         {   this->sorted[i]=pm.sorted[i];
             this->sorted2[i]=pm.sorted2[i];
             this->rank[i]=pm.rank[i];
         }
         return *this;
      };
  };
  public:
    pay_matrix payoffs[pop_size];
    pay_matrix temp_po;
    void sort(int opt);
    void graph();
    void three_d();
    void table();
    void count();
    void find_best();
};
```

Figure 9 - The Matrices Class for "Payoffs and Rankings"

```
/******  Group of people  ******/
class people
{    int ii,jj;
   public:   // class inside a class
      class  player {  private:  int ii;
                      public:
                       unsigned char FSM[gene_size];
                       unsigned char history[num_players];
                       boolean  dead;
                       unsigned char      state;
                       unsigned char      output;
                       long     score, count;
                       int      cc_count, dc_count;
                       int      cd_count, dd_count;
                       int      init_play;
                       int      X,Y;        // Coordinates
                       unsigned long color;
                       char     name[7];
                       float    fitness;
                       void     init();
                       player()  { init(); };
                       // overload the = operator
                       player & operator = (player & eq_to)
                       {  int i;
                          this->state = eq_to.state;
                          this->X = eq_to.X;
                          this->Y = eq_to.Y;
                          for (i=0; i<gene_size; i++)
                          { this->FSM[i]=eq_to.FSM[i]; }
                          return *this;
                       };
                   };
             float reproduce();  // Returns scaled fitness
             void  crossover();
             int   mutate();     // Returns the # killed
             int   coop_count(); // Amount of cooperation
             void  report();
             void  who_is_avg();
             void  play_a_game(int player1, int player2);
 void  play_a_game(int player1, int player2, int player3);
 void  tournament(Display *display, Window window, GC gc);
             void  old_tournament();
             void  init();
             long  total_s, total_c;
             player member[num_players]; // Array of people
             player tempg;
             people()  {  init(); };
};
```

Figure 10 - The People Class

```
/****** barrier class  ******/
class barriers
{
   public :  int X, Y;
     barriers()
      {
       X = 0; // rndd(0,MaxXCoord);
       Y = 0; // rndd(0,MaxYCoord);
      };
};



/******  Matrices class  ******/
class pay_matrix
{
  public:
    int cc;
    int cd;
    int dc;
    int dd;
    pay_matrix()  { init(); };
    void init();
};
```

Figure 11 - Barriers class and Matrices class for "Wandering Players"

passed. The function who_is_avg figures out what the "average" FSM looks like. Coop_count returns the average number of cooperative branches within the population, which can give the user an idea of the degree of cooperation. An array of players is created here, so that the programmer only needs to realize the people object. The other functions of the people class deal with tournament and IPD playing.

Playing a round of the iterated prisoner's dilemma is done by calling the play_a_game function. For the two player case, two player numbers are passed with the function call. The people object keeps an array of players, numbered from 0 to the constant num_players. These numbers are unique to each player, and are easy for the programmer to deal with, similar to human social security numbers. When the program calls the play_a_game function, each player resets their FSM to the state and next play found in the history array, using the other player's number as the array offset. The players recognize each other, and remember what they "thought" at the last encounter. Thus, they make their next play. The player's scores are updated, and their FSMs are moved according to the outcome. These next moves are stored back in the history array, for whenever these two players meet again. The tournament function controls how the players meet.

Two tournament functions are provided, old_tournament, and tournament. Old_tournament calls a round-robin function, where each player plays everyone else a set number of times.

The papers in the literature review use this type of tournament. The author contributes to the IPD study by introducing the other function, called simply "tournament". It implements the idea of randomly moving the players. First, it chooses a player to move. Next, it randomly attempts to move the player in one of the 8 directions. The function checks the boundaries, and then checks to see if a wall occupies the new spot. If the player cannot move in the direction chosen, then the coordinates are not updated. Finally, the coordinates are checked against the other players. If another player occupies the same coordinates, then the play_a_game function takes over. When the programmer desires a round-robin tournament, one call to "old_tournament" will suffice. However, the mobile tournament function was made to be a part of the Xwindows event loop. Therefore, the main program calls "tournament" within a loop, until a minimum number of games have been played by the population as a whole.

One difference between "Payoffs and Rankings" and "Wandering Players" is that the former uses the old_tournament, and does not move the players. Since the players of "Payoffs and Rankings" do not need to occupy space, the program does not use the barriers class. The barriers class simply stores X and Y coordinates, and relies on a function external to the class to place the barriers (give the X and Y coordinates a value besides 0). Figure 11 includes the barrier class. "Wandering Players" uses an array of barriers to represent walls. The pay_matrix class also differentiates the two programs, as Figures 9 and 11 show. The latter program only needed values for DC, CC, DD, and CD, which the user sets at the beginning of the program. The "Payoffs and Rankings" program, however, used an array of 900 matrices. Each of these matrices was initialized to random values, with the constraint DC > CC > DD > CD. The second constraint of 2*CC > CD+DC appeared as a software controlled switch, which the user controls. The "Payoffs and Rankings" program shows how these different payoff matrices affect the rankings of the players. It uses a function to graph the ranking of the player versus the payoff matrices. Also, the payoff matrices can be sorted, but no trends emerged from sorting the payoffs. Another graphing function allows the user to make a three column table with any three variables, for use with a three-dimensional plotting program. Though the two final programs had different goals, the main classes are essentially the same.

# CHAPTER IV: SIMULATION MODULES

This chapter describes the simulation segments of the last section in greater detail. Specific implementation issues are discussed here, on an individual basis. The segments are:

A) A module on cellular automata

B) A module about randomly interacting (mobile) PD players

C) A module examining payoff values and ranking orders

D) A module exploring an evolving population of mobile IPD players

The details of the four programs are described below.

## A. Program 1 : Cellular Automata

The author wrote "Cellular Automata" in Turbo C on a 486-based personal computer. The program requires the VGA graphics driver provided by Borland. Memory requirements are less than 640K.

The author based the first program on the paper by Nowak and May [16]. In the paper, they talk of a population of PD players. These are not true IPD players, since they do not keep track of previous games. Therefore, only the All-D and All-C player types can be represented, each player either cooperates or defects. The players play the PD game with each of their neighbors. The players play a copy of themselves as well, which does not make a difference, according to the paper, though it is convenient to program. Looking at the immediate neighboring squares, the square under consideration is taken over by the neighbor with the highest average score, if it beats the original owner's average score. The program uses the average score to give the players on the edges of the population no disadvantage. Nowak and May used four colors, red for defectors, blue for cooperators, green for a square that was a defector, but becomes a cooperator, and yellow for a square that was a cooperator, but becomes a defector. The author introduced two more colors, pink for a defector's square that is taken over by another defector, and grey for a cooperator's square that is taken over by another cooperator. This produces colorful displays of patterns, that also indicate unstable areas. The program does allow the user to change the colors. A user could restrict all the defectors to red and all the cooperators to blue, for example. The default payoff values are DC=1.9, CC=1.0, DD=0.0, CD=0.0, like those used by Nowak and May.

The program allows these values to be changed by the user as well. The program places no restrictions on the payoff values, except what the C compiler accepts as a floating point number. The author chose these values to correspond to Nowak and May's paper. The DC value was not explicitly given in the paper, just a range from 1.8 to 2.0.

With these values, the author observed the same effects as the paper described. The population consisted of 900 individuals, a size of 30 x 30. Some features of the program included step by step control of the evolution. The program has an edit mode, and a run mode. In the run mode, the population evolves, and the squares reflect the evolution by changing colors. The evolution is automatic, and continues until the user presses "e", for edit mode. The program starts in edit mode.

The population starts as a random field of red and blue squares. The "i" button allows to user to initialize the population at any time, for example, to start over with a new population. The Enter key takes the players through one evolutionary step every time it is pressed. When in edit mode, a cursor appears over a player. The cursor can be moved around with the arrow keys. Pressing the space bar changes the strategy of the player. This way, the user can change the individuals around. A player can also become a "wall" at the user's discretion. A wall does not play the PD game, and it does not give up its square. The walls serve to separate the players from each other, and has some interesting effects. For example, a diagonal line of walls can support two diagonally adjacent cooperators, who will actually prosper. The key "c" allows the user to change player's colors, as mentioned above. The "y" key allows the payoff values to change.

The program keeps track of the number of cooperators and defectors, and the percentage of the population that are cooperators.

**B. Program 2 : The Prisoner's Dilemma World**

The author also wrote the second program in Turbo C on a 486-based personal computer. Again, it requires the VGA graphics driver provided by Borland. This program used the code from "Cellular Automata", adding and deleting functions as needed. As a result, the graphics routines and the user interface are very similar. Recycling the previous program allowed the author to produce a prototype within a short amount of time. All programs were written with modularity in mind, because the author planned to incorporate all previous work together in the last program.

In the "Prisoner's Dilemma World" simulation, a group of 99 PD players move about randomly on a 30 x 30 square area. As in the "Cellular Automata" program, the players were not IPD players since they did not remember past games. One additional player appears as a white square. The keyboard controls its movement, as well as its current prisoner's dilemma strategy. The other players only have one strategy, either always cooperate or always defect. Every time a player bumps into another player, the two play a game. The computer tracks the score, number of games played, and average for each player. When a player's average falls below 1.75, that player dies. The simulation lasts until the user-controlled player dies. If this condition were removed, then the simulation would continue until all but one player dies, or the last two defectors kill each other. This is not likely, but if the last two defectors did kill each other, then whatever remaining cooperators would thrive. The computer displays the number of players left, and how many of them are cooperators. Also, the user's player's statistics are shown. A special key, "x", shrinks the boundaries of the world every time it is pressed. This feature speeds up the game, since the players have less room to move around.

## C. Program 3 : Payoffs and Rankings

This program was written in C++, on an HP Unix system. The Unix system provided greater speed and memory capabilities than the personal computer the author had used to this point. Before starting this thesis, the author experimented with FSMs, Genetic Algorithms, and object-oriented programming. The switch to the Unix system allowed the author to incorporate these early attempts. Though the Genetic Algorithm code did not need to be present for this program, the author realized that it would be included in the final program.

"Payoffs and Rankings" examined the effects of altering payoff values on a population of PD playing FSM's. It generates a set of FSMs that play each other a total of 50 times. The program compares the outcomes of the FSMs according to different payoff values, and tallies when the ranking order of players, from best to worst, changes. The players do not move, but rely on the above tournament for interactions with other players. The author created this program as a tool to discover trends among payoff matrices. This program provides graphs and statistics to analyze the data. Two groups of players tested this program. A set of 5 players provided average payoff values that generate rankings. A group of 50 players were used for other data analysis.

**D. Program 4 : Wandering Players**

The final program also used C++ on the HP Unix system. First, the author copied sections of code from "The Prisoner's Dilemma World" into a copy of "Payoffs and Rankings". Though C++ will compile and run C program lines, the author took the time to convert all input and output commands to C++. Structures from "The Prisoner's Dilemma World" became classes in the new program. The old graphics routines had to be removed, since the Turbo C graphics commands did not have corresponding commands under the Unix C library. Later, the author added a small Xwindows program for graphics, and "Wandering Players" was born.

This program simulates a world where players randomly interact. The author kept the world size to a 10 x 10 area, because of time constraints. Walls take up some of the space that the players have to move around. The number of walls depends on the world size, defined as "length*2+width*2+c". With a small world, the programmer set the constant c=1. The people of this simulated world start out as random FSMs, and the population evolves for 500 generations. The author conducted several runs, each with a different payoff matrix. The dominant player types are examined when the simulation finishes. The author shows that different payoff values lead to different evolutionary traits. Special runs of this program use the round-robin tournament as did Fogel [17] and Axelrod [3], except with differing payoff values. This allows the author to examine the effects of a spatial, mobile population, as opposed to the more traditional round-robin tournament. During the Genetic Algorithm sequence, the program kills the below average players. The program informs the user when almost all of the players are killed, or if only a few die. Either condition indicates that the players have evolved to a local maximum, since most are about the same genetically. The author kept an eye on this, but the mutation operator took care of any population becoming stagnant.

This program has graphics capabilities. Xwindow routines are embedded within the program, and turned on by a constant in the "#define" section. The author decided to put this switch as a constant in the program instead of a variable. The program runs much faster without the graphics turned on, and the graphics cannot be displayed on the VT100 terminal that the author often used. The program has to be re-compiled after the Xwindow switch changes state. The graphics are interesting to watch as the players move around. The program does not overwrite the player's former positions, simply from an aesthetic point of view. Instead of colored blocks moving about on a black background, the players leave a colorful trail. The walls appear as white, and, of course, do not move.

The program also features a graphing switch, again defined as a constant. When the programmer sets this switch to make a graph, the program prints only the generation number and the average fitness score. The cooperative branch count, as explained later, can be chosen to print in place of the fitness. When the user runs the program, the results can be piped to a file, such as a graphing data file. Note that the payoff values are included in the first 3 lines, for the user to comment out, or delete, according to the graphing package used.

As a final feature, the program generates a report of the players in their final state at the end of the simulation, unless the graph option prohibits this report. The program examines every bit of every FSM, and computes whether the bit is set a majority of the time. The program creates an "average" FSM in this manner. The report contains this average FSM, as well as the information about the number of cooperative branches for each FSM. The author constructed this program as a simulation tool with speed in mind. A version can easily be developed for the common user, though it will be slower.

# CHAPTER V: PROCEDURE

The programs described in the previous chapter contain concepts common to both the literature surveyed, and the programs themselves. These central ideas are explained here.

A) The Finite State Machine

B) The Players

C) The Population

D) The Genetic Operators

E) The Data Structures

## A. The Finite State Machine

The players of the two-player iterated prisoner's dilemma game have a built-in finite state machine, similar to an organism's DNA. Each FSM is a string of 17 bytes, numbered from 0 to 16. These bytes encode how the player will react to other players. Byte 0 contains the initial play and the initial state. When one player encounters another for the first time, this byte tells how it will play. The first 4 bits of this byte are not used. The next bit represents the initial play, 0 for defect and 1 for cooperate. The final three bits represent the initial state of the FSM. The FSM's are limited to a total of 8 states, since there are 8 different combinations of 0's and 1's that a three bit word can have. The rest of the string should be thought of as "what if" cases. Each state uses two bytes, broken up into 4 bit sections. The first 4 bits represent the next move and next state of the FSM if DD is the outcome of the game. Like above, the left-most bit represents the move, and the other three represent the state. The next 4 bits represent the next move and next state for the DC case. Note that the outcomes are relative to the player. For example, if player 1 has the outcome of DC, then player 2 sees it as CD. The second byte tells the FSM what to do for the case of CD and CC, respectively. This pattern continues for the other states. Each FSM has 136 bits, with 4 bits not used at this time, allowing $2^{132}$ possible representations, or $5.44 * 10^{39}$ possible strategies.

## B. The Players

The players are more than just the FSM, just as organisms are more than their genetic pattern. Each player also includes a score, totalling all points accumulated, and a count of the number of games played. With these two variables, the program computes an average which serves as the fitness of the player. The players have position, kept as a pair of coordinates. When a pair of players meet, each FSM will move from one state to another. The next state and next play becomes a history, or memory of the encounter. These histories are stored as a list for the next time the players run into each other.

## C. The Population

Since the language used, C++, supports object oriented programming, the players have a variety of functions attached. The individual players only have an initialization routine. However, the player class nests within a population class. The population, as the name implies, is the collection (array) of players. Thus, each player has an identifying number. The constant "pop_size" defines the size of the population. The most used function of the population plays the prisoner's dilemma game between two players. The "play_a_game" function looks at the player's histories together, and updates the FSM's accordingly. The author overloaded this function to play the three player prisoner's dilemma as well. Though not used for this thesis, the three player mode can be invoked simply by passing three numbers to the play_a_game function. A tournament function plays every member of the population against each other for the purposes of examining the payoff values' effect on stationary players. Thus, the "Wandering Players" program, that randomly moves the players, does not normally invoke this tournament. The programmer can run this program with the old tournament function, when desired.

**D. The Genetic Operators**

The population object contains genetic operators. These operators are reproduce, crossover, and mutate. The population of players evolves as described in the literature review. As mentioned above, a player's score divided by the number of encounters provides the fitness function. Two players are needed for the sexual reproduction function, and mate according to their fitness. Note that sexual reproduction means that there are two parents, though the players do not have gender. The crossover function randomly picks two crossover points, and fuses sections of the parent FSMs to make a new FSM. The crossover only produces one offspring, but the successful player has many chances to reproduce. Mutation occurs approximately once every 20 genes. The new players replace the least successful ones, and all memories are erased during the genetic step. The genetic step happens after the average number of moves exceeds the number of players multiplied by a constant of 500. In preliminary tests, the author found that the average fitness scores would level off after 250 generations. The cycle continues for 500 generations, to be relatively sure that the average scores will not drastically change.

**E. Data Structures**

The programs used arrays and objects as the primary data structures. An array keeps the finite state machine code, as well as the history of past games with other players. Though the programmer used C with the first two programs, he made the code as modular as possible, to make the transition to C++ as easy as possible. The C structures were placed with the functions altering them. Later, C++ classes took the place of former structures. The programmer used arrays with objects as well. The author includes more detail about the classes in the Design and Approach chapter.

# CHAPTER VI: RESULTS AND ANALYSIS

The author wrote both "Cellular Automata" and "Prisoner's Dilemma World" as tools to examine prisoner's dilemma populations and concepts. The programs do not supply any statistical data or graphs. Therefore, the observations gleaned from these programs will be included in the discussion chapter.

For the "Payoffs and Rankings" program, a fixed-size population of players play the IPD game a given number of times. The player's scores are evaluated according to 900 different payoff values. Five players are enough to demonstrate that the payoff values affect each player's score, though 50 players also provided results. The program makes every player randomly. Each player plays the IPD with each other player 4 times, including a copy of itself. This environment produces a set number of mutual cooperations, mutual defections, etc, for different payoffs. 900 random payoff matrices are studied. The matrices conform to the rule DC > CC > DD > CD. The program allows the user to decide whether to use the rule 2*CC > DC + CD. When the user chooses the second rule, then the program only considers the 406 remaining payoff matrices. When the payoffs change, the rankings change, as seen below. "Payoffs and Rankings" produced the below results, finding payoff matrices that produces different player ranking orders.

| Player Ranks | | Frequency | | Explanation |
|---|---|---|---|---|
| 1 4 2 5 3 | occurred | 66 times. | 7% | |
| 1 4 3 5 2 | occurred | 98 times. | 10% | 2, 3 switch |
| 1 5 2 4 3 | occurred | 18 times. | 2% | 4, 5 switch |
| 1 5 3 4 2 | occurred | 587 times. | 65% | 4, 5; 2, 3 switch |
| 2 3 1 5 4 | occurred | 9 times. | 1% | 1, 2; 3, 4, 5 switch |
| 2 4 1 5 3 | occurred | 121 times. | 13% | 1, 2 switch |
| 2 5 1 4 3 | occurred | 1 times. | 0% | 1, 2; 4, 5 switch |
| | Total = | 900 | | |

Table 1 - Initial Results from "Payoffs and Rankings"

How successful the IPD players are depends on the values of the payoff matrix. The first player was the best most of the time (received the rank of 1), 84% of the time. The second player ranked worst 67% of the time. The third player got 3rd place 75% of the time. The fourth player got 4th place 67% of the time. The fifth player came in 2nd 75% of the time.

The ranking 2 3 1 5 4 is the most interesting, since the players really switch around, compared to 1 4 2 5 3. The other cases can be explained by one or two players doing better under the payoff values than the player immediately in front of it.

The players themselves are not that important. What is important is how many DC's, DD's, CC's, and CD's they get when they play others. For the above, the players had the following game outcomes:

|  | Number of times that these outcomes occurred: |  |  |  | Total Games Played: |
|---|---|---|---|---|---|
|  | DC | CC | DD | CD |  |
| Player A | 18 | 8 | 22 | 2 | 50 |
| Player B | 0 | 26 | 4 | 20 | 50 |
| Player C | 6 | 26 | 12 | 6 | 50 |
| Player D | 4 | 14 | 20 | 12 | 50 |
| Player E | 16 | 8 | 22 | 4 | 50 |

Table 2 - PD Outcomes for 5 Players

D.B. Fogel quoted Rapoport (from 1966) about the second PD rule, which the above chart does not take into account. This rule states "2*CC > CD + DC", which restricts the payoff values even further. Here are results when this rule is taken into account.

```
 Only cases where 2*CC > CD + DC will be printed.

1 4 2 5 3  occurred   66 times.  16%
1 4 3 5 2  occurred   53 times.  13%
1 5 2 4 3  occurred   18 times.  4%
1 5 3 4 2  occurred  138 times.  33%
2 3 1 5 4  occurred    9 times.  2%
2 4 1 5 3  occurred  121 times.  29%
2 5 1 4 3  occurred    1 times.  0%
 Total =  406

2 CC > CD + DC  happened 406 times.
2 CC = CD + DC  happened 13 times.
2 CC < CD + DC  happened 481 times.
```

Table 3 - Initial Results with 2*CC > CD + DC

Notice how 1 4 3 5 2 and 1 5 3 4 2 are the only two outcomes that are affected. Outcome 1 5 3 4 2 dominated previously, but with this rule in effect, it is not much more dominant than 2 4 1 5 3. Because the second rule restricts the payoff values to a prisoner's dilemma game, the rest of the paper will focus on the case that uses the rule.

The results of the first run with 5 players appear below. The first column is the ranking order, followed by the number of occurrences. The author calls differences between payoff values "delta" values, and refers to them as d1, d2, and d3. The d1, d2, and d3 values are the average differences DC-CC, CC-DD, and DD-CD, respectively. The last 4 columns show the average payoff values that result in the ranking pattern. The results show that the 5 players will be ranked differently, depending upon the payoff values.

```
 Only cases where 2*CC > CD + DC will be printed.

pattern    # occurrences       d1  d2  d3     dc  cc  dd  cd
1 4 2 5 3     66 times   16%  34  58  13 |  124  90  31  18
1 4 3 5 2     53 times   13%  45  47  10 |  118  73  25  15
1 5 2 4 3     18 times    4%   6  27  43 |   96  90  62  18
1 5 3 4 2    138 times   33%  29  21  43 |  115  86  65  21
2 3 1 5 4      9 times    2%   6 119   4 |  161 155  35  30
2 4 1 5 3    121 times   29%  14  77  14 |  124 110  32  18
2 5 1 4 3      1 times    0%   6  56  63 |  136 130  74  11
Total =  406
```

Table 4 - Average payoff values that affect rankings.

This chart shows that the delta values affect how well a player will do. To generalize, a fairly small d1 and d3, but large d2 produces perhaps the most interesting rankings of 2 3 1 5 4. A small d1 or d3 value seems to point to an unstable set of rankings. Consider 1 5 3 4 2 to be "normal", since it is the dominant ranking order. Its delta values are fairly even, probably much like the payoffs DC = 5, CC = 3, DD = 1, CD = 0 that Axelrod used. Aside from a big d2 value, the other differences from "normal" have a d1=6, or a d3 <= 14. The variances from "normal" are usually when one player overtakes another in the ranking order. The change in delta values can be subtle to allow one player to suddenly out-score another.

The program ran with 50 players, and made a table of the ranking of player 4, d1, and d3. The difference DC-CC and DD-CD stood out as the delta values most likely to alter the player's ranking order. The author chose player 4 at random, though the other 49 players gave almost identical results. A three dimensional plot of these values shows a striking trend, as seen in Figure 12. In this figure, each triangle represents player 4's rank. Spikes appear below the triangles to give the viewer better perception. The lower the d1 and d3 values, the more likely this player would be to score worse in the population. Of course, when player 4 scores worse for a set of payoff values, then another player scores better. Player 4 ranks better when the delta values are large. When d1 or d3 is low, player 4's rank could seriously be affected.

"Payoffs and Rankings" executed with 10 randomly-initialized groups of FSMs. This showed that the changes in the ranking of players were not confined to one group of FSMs. The author found one case in the eight run, where there was no difference in the rankings. This run shows that the FSMs of the eighth run were not affected enough to cause a change in

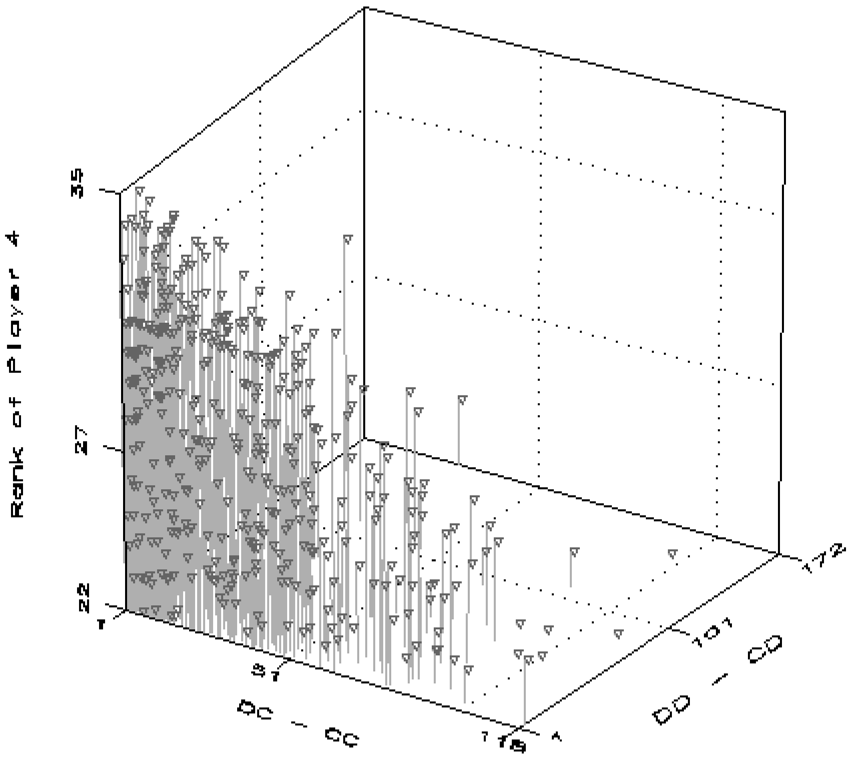


Figure 12 - Delta Values and Ranking Changes

rankings. This program ran for a population size of 50 players, and confirmed that the rankings changed for a different sized population as well. If a changing payoff values affect a small population, it only goes to reason that a larger population will be similarly affected. Figure 13 shows the rankings for player 1, out of 50 players. Note that the ranks are numbered 0 for best, to 49 for worst. This player usually does not have a good fitness rating, with an average of 46. However, under certain payoff values, the ranking shoots up 10 places. The higher the rank, the more likely this player would reproduce in an evolving population, or at least survive until the next generation. The ranking difference supports the idea that the population will evolve partly according to the payoff values. The final program explores this.

The "Wandering Players" program used the payoff values from table 4, which produced different ranking orders. These payoff values were chosen because they had already proved to give different rankings of IPD players. Now they would be put to evolving populations, to see how they effected the evolution of cooperation.

The author overcame the problem of comparing evolution of populations by scaling the fitness scores. For example, how can one compare a fitness of 2.6 (with a payoff of 3 for mutual cooperation), with a fitness of 46.3 (with the mutual cooperation payoff of 90)? Scaling the fitnesses to a range of

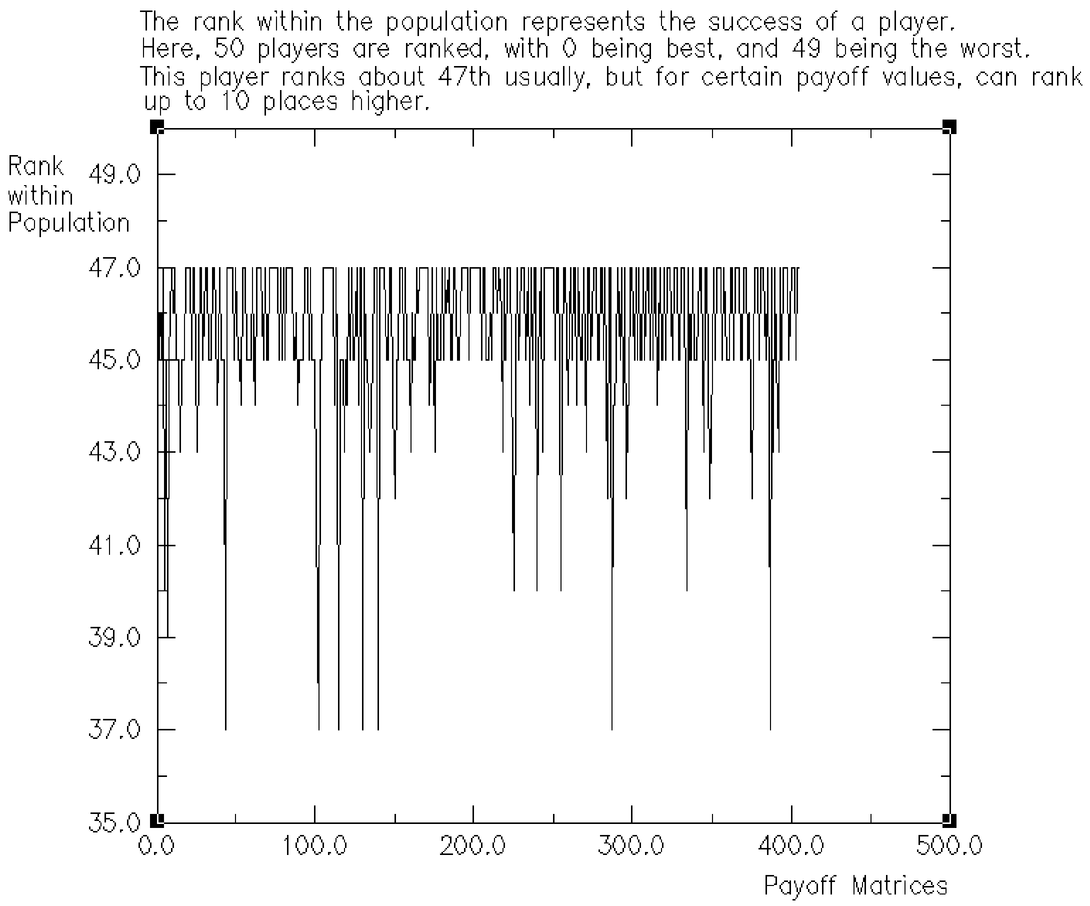


Figure 13 - How Payoff Values Affected Player #1

0 to 99 provided a solution. No player can achieve a score less than CD, nor higher than DC, so these values became the lower and upper bounds of the scale. Mutual defection became the value of 33, and mutual cooperation mapped to the value 66. The program first checks to see which range the value-to-be-scaled fits. Then the program interpolates this value to find the scaled score.

The author examined another form of measuring the amount of cooperation within a population. Each FSM has 8 states, with 4 arrows per state. Each arrow has an output of C or D, and points to another state. 8 x 4 = 32, plus the initial arrow equals 33 possible cooperative branches. A player with 33 cooperative branches would be All-C, and a player with 0 cooperative branches would be All-D. A Tit-For-Tat player would be somewhere in between, probably around 17 cooperative branches. A player could be All-C, with only 5 cooperative branches, though. If the initial arrow points to a state that always points back to itself, then only 5 arrows would be used for this FSM. However, a single mutation could make this player an All-D. Therefore, this method of measuring cooperation of a population could suffer on an individual scale, but would give a good general indication when measuring the population's cooperativeness. This method corresponded well with the graphs of the scaled average score. See Figures 14-17 on the following pages.

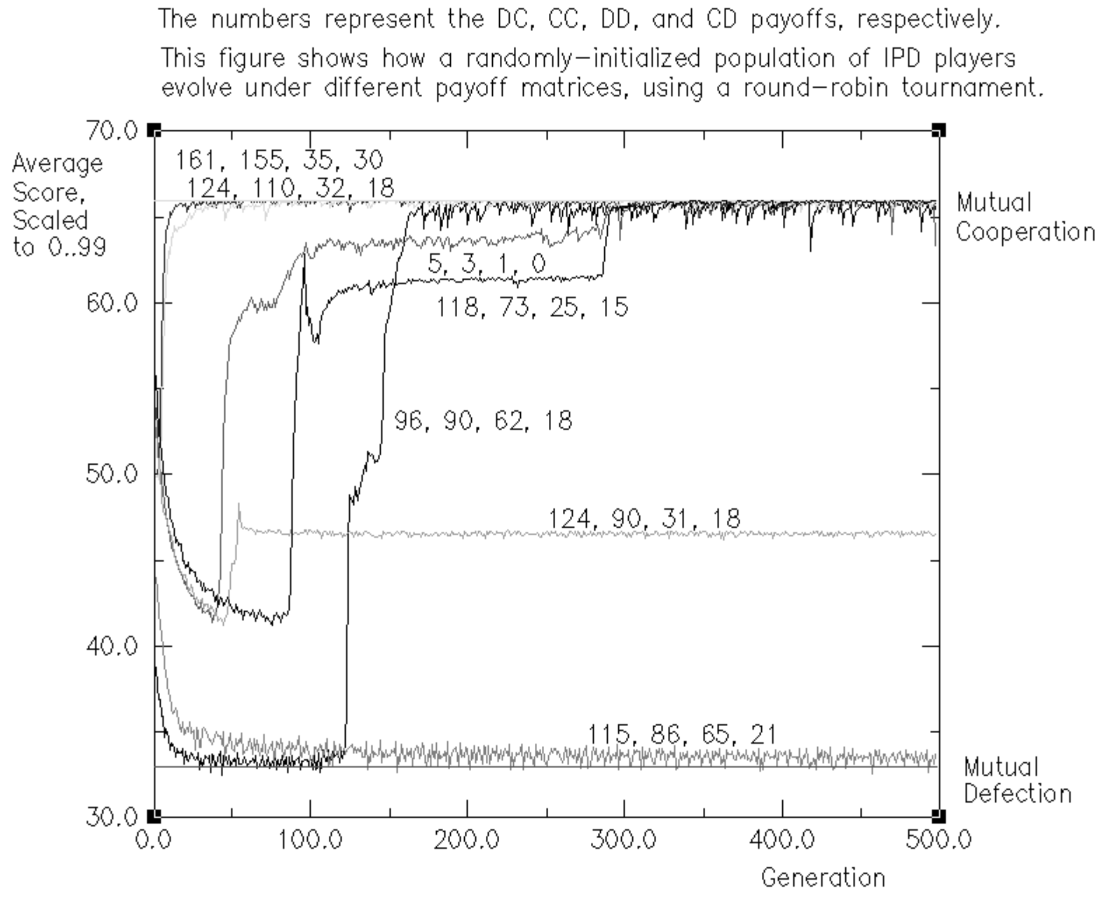


Figure 14 - Round-Robin Population Evolution

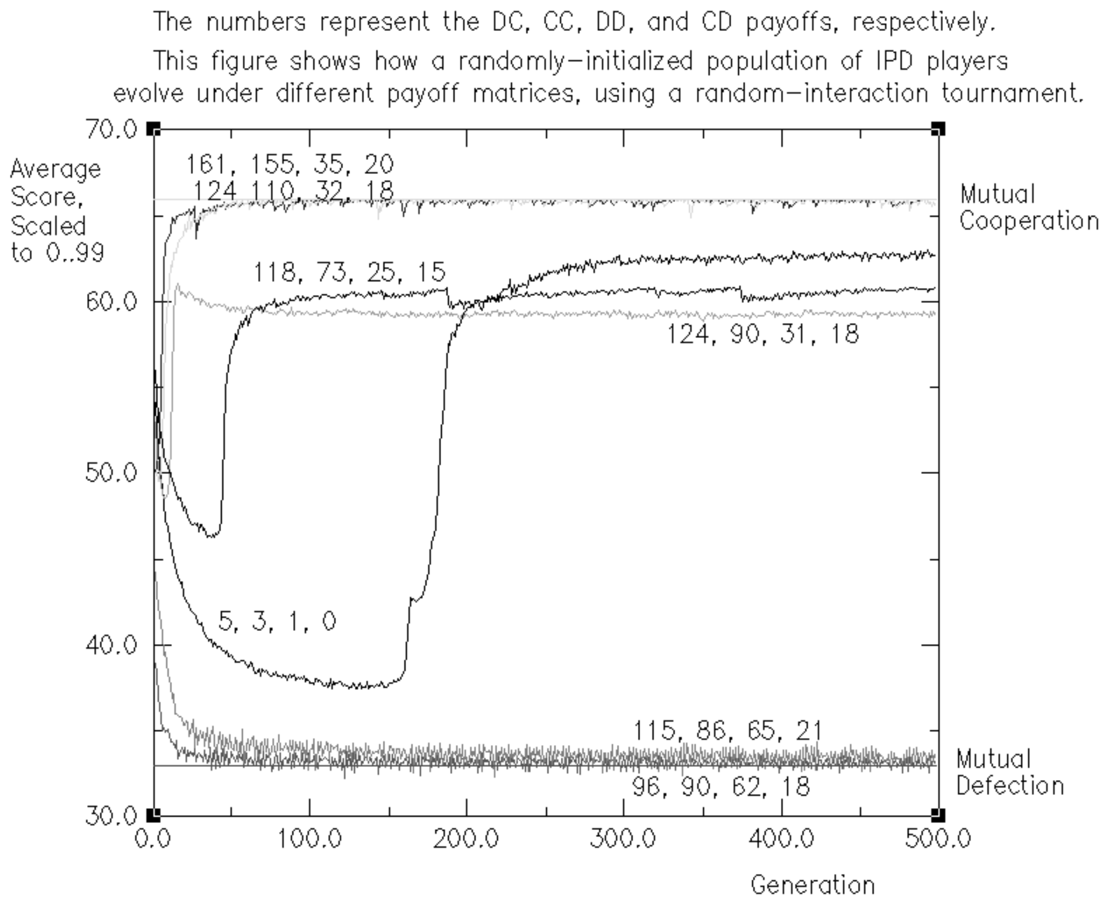


Figure 15 - Mobile Population Evolution

The average number of FSM branches with Cooperative output is another way measure the amount of cooperation within the population. Since an 8 state FSM is used, with an initial arrow, this number ranges from 0 to 33.

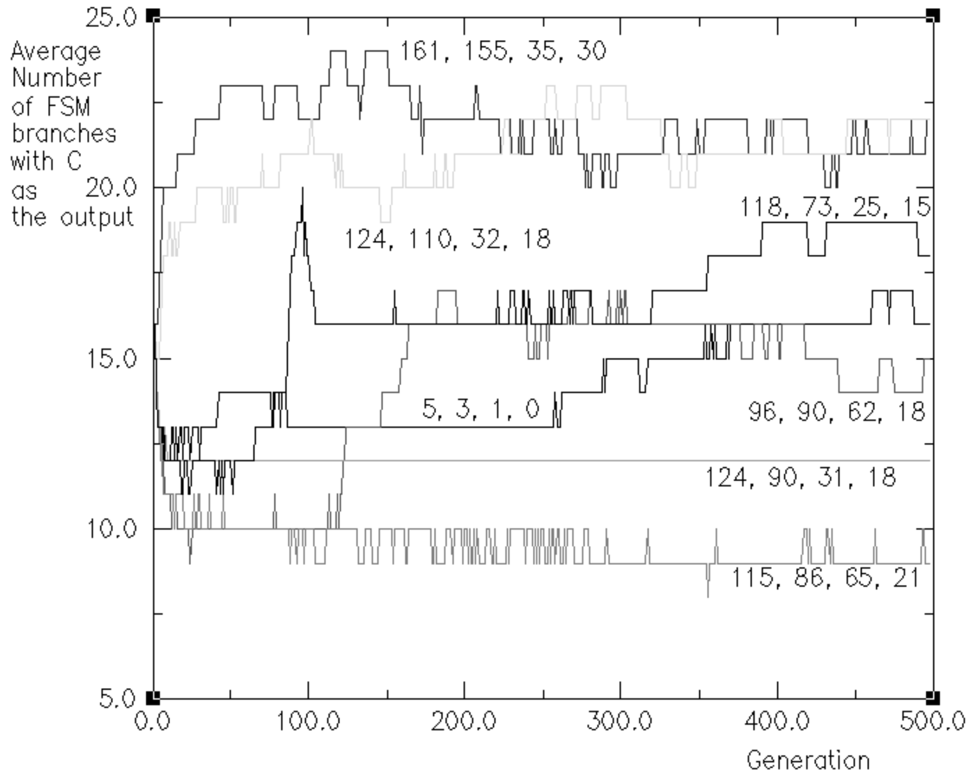


Figure 16 - Round-Robin Population Cooperative Branch Count

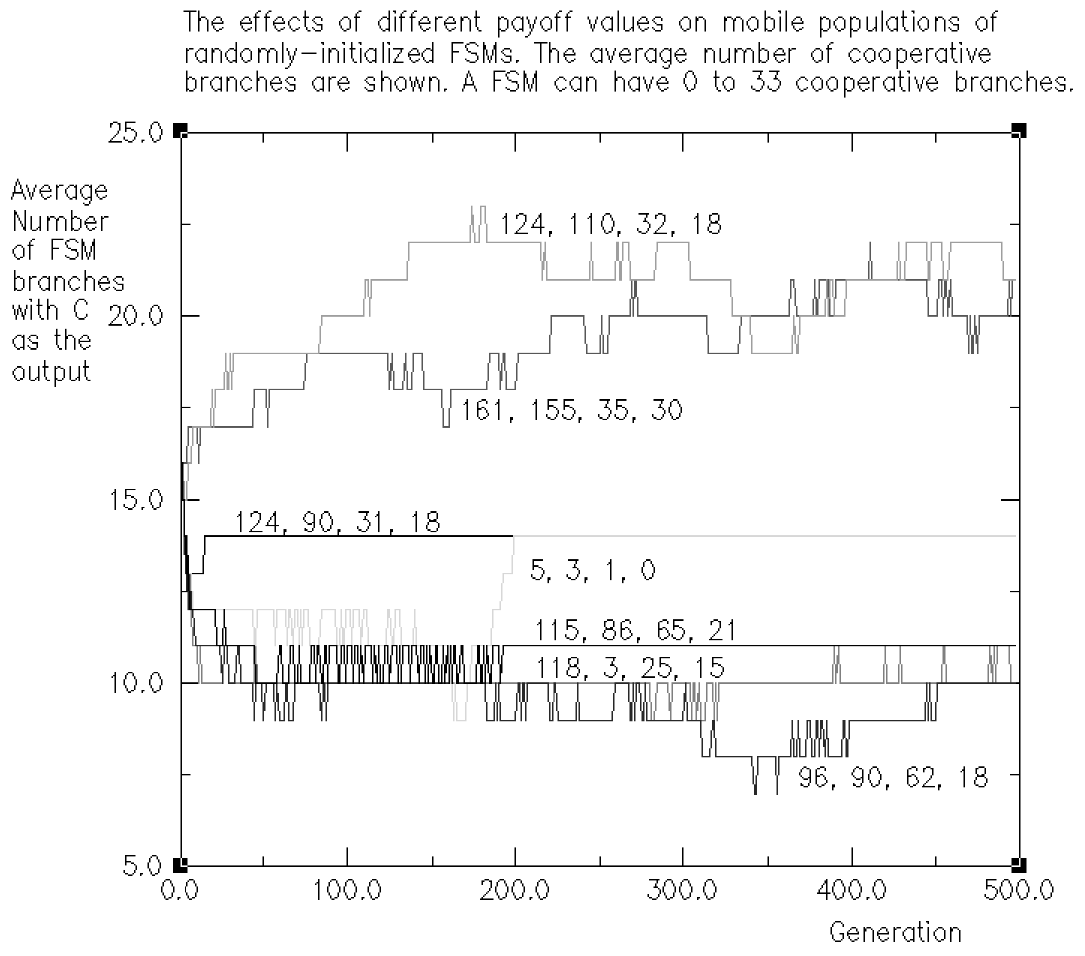


Figure 17 - Mobile Population Cooperative Branch Count

The amount of cooperation within a population depends on the payoff values, as the "Wandering Players" program shows. First, the population of 100 players play a round-robin tournament of 20 games for each pair of players. This experiment repeats for the sample payoff values chosen above. Figure 14 shows this evolution of cooperation for the 6 cases chosen. The seventh pattern only made a difference in 1 case, so it was not used. Though the graphs include the fifth pattern, it is very similar to the sixth pattern. These graphs clearly show that the payoff values affect the evolution of cooperation. Next, the players wander about randomly, replacing the round-robin tournament. When two players meet, they each load in the histories of the last game against each other, and play another round of the IPD game. The player remembers each experience from his point of view. If one player remembers a CD outcome, the other player will remember a DC outcome. The wandering continues until the population has counted a total of 500*number-of-players games. Naturally, some players will not have as much experience as others. A player could conceivably be trapped within walls, and not play the game at all. This situation does not happen, but limiting the tournament by counting the games played by everyone allows for this condition without hurting the experiment. Figure 15 shows the effect of different payoff values on the scaled fitness scores of a mobile population of players. Next, Figure

16 shows the evolution of cooperation according to the count of cooperative arrows of the FSMs. Figure 17 also shows the effects on the evolution of cooperation, except it graphs the average number cooperative branches for the mobile case. Cooperation is precluded for one case in the round-robin experiments, and for two cases in the mobile experiments. These appear as the graphs that quickly fall to mutual defection payoffs, and do not rise within the 500 generations studied. The FSMs for every run of the mobile population are averaged, and the results appear below. As predicted, the FSM codes are different for each run.

| Payoff Values | | | | | States | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DC | CC | DD | CD | init | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 5 | 3 | 1 | 0 | f7: | 690e | 0770 | 883e | 0b27 | c81f | 964b | e138 | 67b3 |
| 124 | 90 | 31 | 18 | a8: | 6718 | 0734 | 9d66 | 1214 | 6799 | cec2 | 079e | 66b6 |
| 96 | 90 | 62 | 18 | f7: | c34c | 3580 | 1148 | 7eda | 5a53 | 3523 | 4699 | 7721 |
| 115 | 86 | 65 | 21 | a1: | 5107 | 71f4 | bd44 | a0b5 | 4122 | 0524 | 1b19 | 77a1 |
| 161 | 155 | 35 | 30 | 8: | 6f88 | 5cac | e93b | ff9b | 5f9c | 61ad | e589 | 1372 |
| 124 | 110 | 32 | 18 | 48: | 7c0b | fa2b | f8ed | b35b | 7931 | fe8b | eb1b | 9e3a |
| 118 | 73 | 25 | 15 | 96: | 603d | 5c42 | 6077 | 3963 | 7a31 | 82c6 | 4a32 | c9a3 |

Table 5 - The "Average" Evolved FSMs.

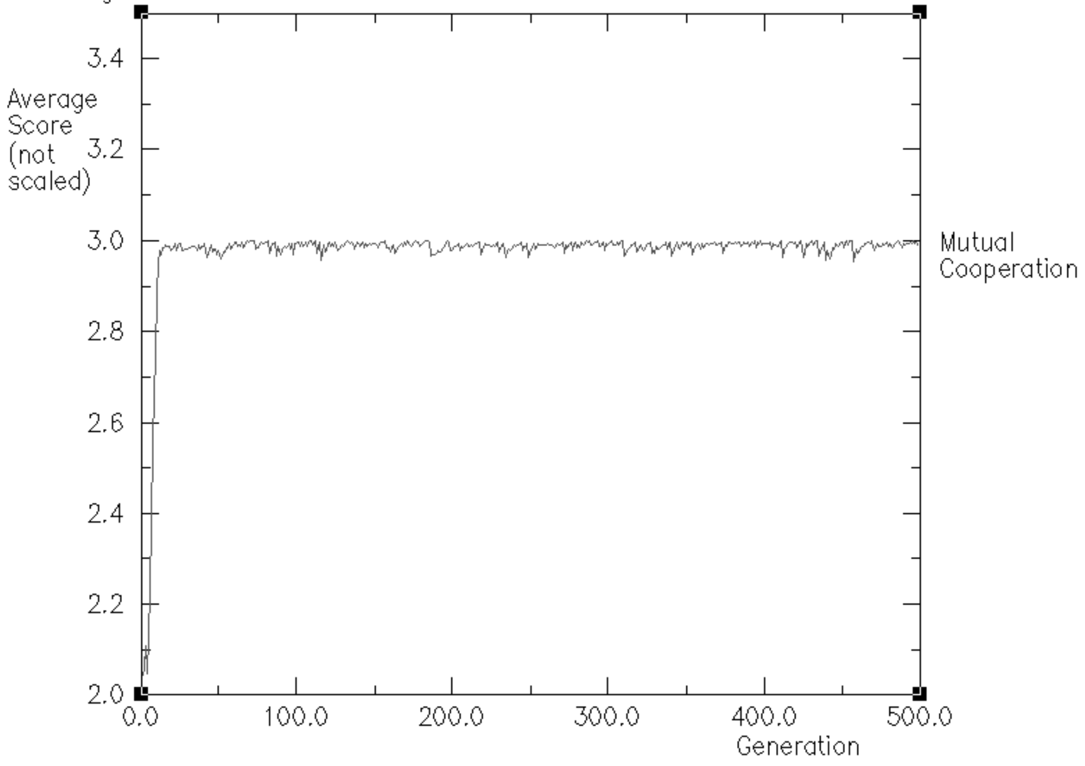
The next figure shows the best FSM for the run, after 500 generations of a mobile population. Again, these FSMs are different from each other. These should be close to the corresponding average FSMs in most cases, since the best genes tend to propagate through the population.

| Payoff Values | | | | | | States | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DC | CC | DD | CD | init | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 5 | 3 | 1 | 0 | f7 | : | 690e | 0770 | 883e | 6f27 | 261c | 8ecb | e918 | 67b3 |
| 124 | 90 | 31 | 18 | a8 | : | 6718 | 0734 | 9566 | 1298 | 7b99 | 1ac1 | 98cc | b2b6 |
| 96 | 90 | 62 | 18 | f7 | : | 7dcd | 0734 | 0d23 | 45da | 5a1f | 50a5 | 1031 | 7721 |
| 115 | 86 | 65 | 21 | 3f | : | 2818 | 66c5 | 3e57 | 52d6 | d4c2 | 5022 | 0d36 | ba28 |
| 161 | 155 | 35 | 30 | 08 | : | 6788 | 5c0b | e93b | bfbf | 5f9c | 51bd | 2189 | 136b |
| 124 | 110 | 32 | 18 | c8 | : | f418 | da2a | faed | b35b | 7930 | de0a | eb1b | 963a |
| 118 | 73 | 25 | 15 | 96 | : | 603c | 5843 | 6477 | 3963 | 7a31 | 82c2 | ca77 | c9a2 |

Table 6 - The "Best" Evolved FSMs.

The game of Chicken appears repeatedly in studies similar to this. The author includes a special run of the final program showing the evolution of cooperation among players of the iterated Chicken game. See Figure 18, on the following page. The payoff values used are the same as Danielson [5]. The main difference between Chicken and the prisoner's dilemma are the payoff values. Chicken punishes mutual defection the most. The graph shows that the Chicken players evolve mutual cooperation very rapidly, and keep the level for all 500 generations. Danielson's point that this game better models some human interactions means that cooperation would evolve more rapidly than expected from an IPD simulation.

This graph shows how a population of FSMs evolve to average mutual cooperation for the game of CHICKEN. When the payoff values are changed to a non-prisoner's dilemma game, the WANDERING PLAYERS program simply prints a warning and continues the simulation.



DC=4, CC=3, DD=0, CD=1, so cooperation evolves quickly.

Figure 18 - The Chicken Game

# CHAPTER VII: DISCUSSION

In the "Cellular Automata" program, a problem arose which Nowak and May [16] did not mention. The generated patterns were not always symmetric, as expected. For example:

```
                          Next iteration
Initial pattern           Expected             Actual
***.                      ***.                 ***.
***.                      **..                 ***.
**..                      **..                 **..
***.                      **..                 **..
***.                      ***.                 ***.

 * = cooperators
 . = defectors
```

Figure 19 - Unexpected Cellular Automata Pattern

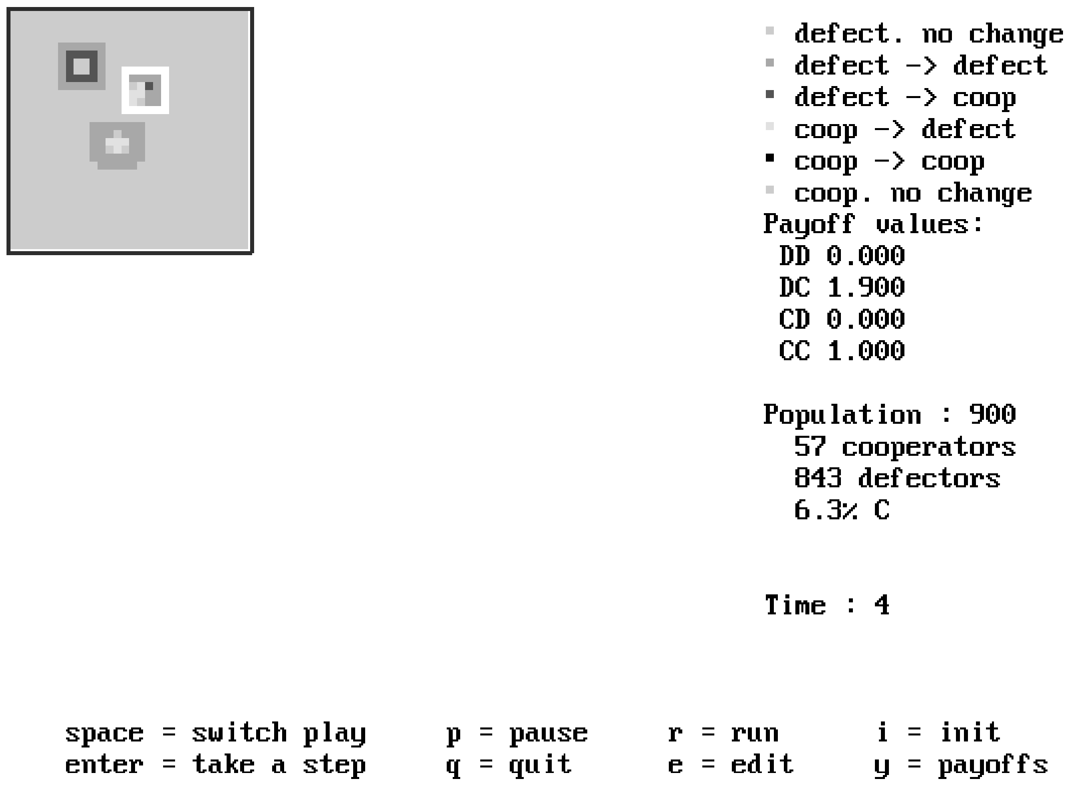
Why would the defectors "convert" a square to the south, but not the one to the north as well? The problem existed in the function that updates the map after the PD games have been played. A square would be taken over by the first neighbor with a better score. If another neighbor had a score even higher than the first conquering neighbor, then this new neighbor would take over the square. The surrounding neighbors are examined from the row above to the row below it, and from left to right. What occurred in the above case was that a square would have two neighbors with the same score. These squares would be taken over by the first neighbor examined with this score, but the second neighbor would not be considered. The author changed the program to look for this case. When this occurs now, the square will be taken over by the neighbor that plays the same. If a cooperator has two neighbors, and both neighbors have the highest score around, but no more than one is a defector, then the square will remain a cooperator.

The "Cellular Automata" program confirmed the results that Nowak and May [16] described. The addition of walls revealed that 2 cooperators can survive in a world of defectors. Without the walls, fewer than 4 cooperators would die out. The author observed patterns that would evolve for hundreds of generations before settling down, or killing off the cooperators. The Cellular Automata program allows for editing of the pattern, so the user can interactively experiment with patterns.

The program performed like the one described, including the groups of cooperators that formed "spinners" and "travelers". The boundaries of the world often helped support the defectors. For example, a square group of cooperators in a world of defectors would grow, eventually taking over the world except for an outer ring of defectors. Adding horizontal and/or vertical walls produced a similar effect - the defectors would be able to retain a line of spaces at the boundary. The additional colors of this simulation allowed the author to observe that even in what appeared to be a stable situation, some border cells actually were being taken over by the same type of player every time. The user could place and remove walls at will, isolating sections and then re-combining them. The process takes time, however, and the walls occupied space, meaning that to place a wall on a cell, the original occupant would be lost. The author discovered that a diagonal wall supports two cooperators, and can actually allow them to prosper. A diagonal down the middle of the world, for example, might start with only two cooperators, but they would quickly take over other cells along the diagonal. An example screen from this program appears in Figure 20, on the following page.

The "Prisoner's Dilemma World" was presented to the public at Speed School's 1994 Engineers' Days. Every year, the Student Council of Speed Scientific School, at the University of Louisville, holds this exhibition featuring current research. Students are encouraged to present projects about the engineering field that they study. The student chapter of the Association for Computing Machinery at the University of Louisville requested that the author present this project on their behalf. The author tailored this program to show the public aspects of the prisoner's dilemma.

Groups of people would come to the display, with varied levels of understanding. A group of computer science engineering professors might be followed by a class of school children. To deal with this situation, the author explained the prisoner's dilemma game, then had two volunteers sit at a table. Each person held two cards, one marked "Defect" and the other marked "Cooperate". At the count of three, both people



```
▪ defect. no change
▪ defect -> defect
▪ defect -> coop
▪ coop -> defect
▪ coop -> coop
▪ coop. no change
Payoff values:
 DD 0.000
 DC 1.900
 CD 0.000
 CC 1.000

Population : 900
  57 cooperators
  843 defectors
  6.3% C


Time : 4


space = switch play     p = pause     r = run      i = init
enter = take a step     q = quit      e = edit     y = payoffs
```

Figure 20 - Sample Screen from "Cellular Automata"

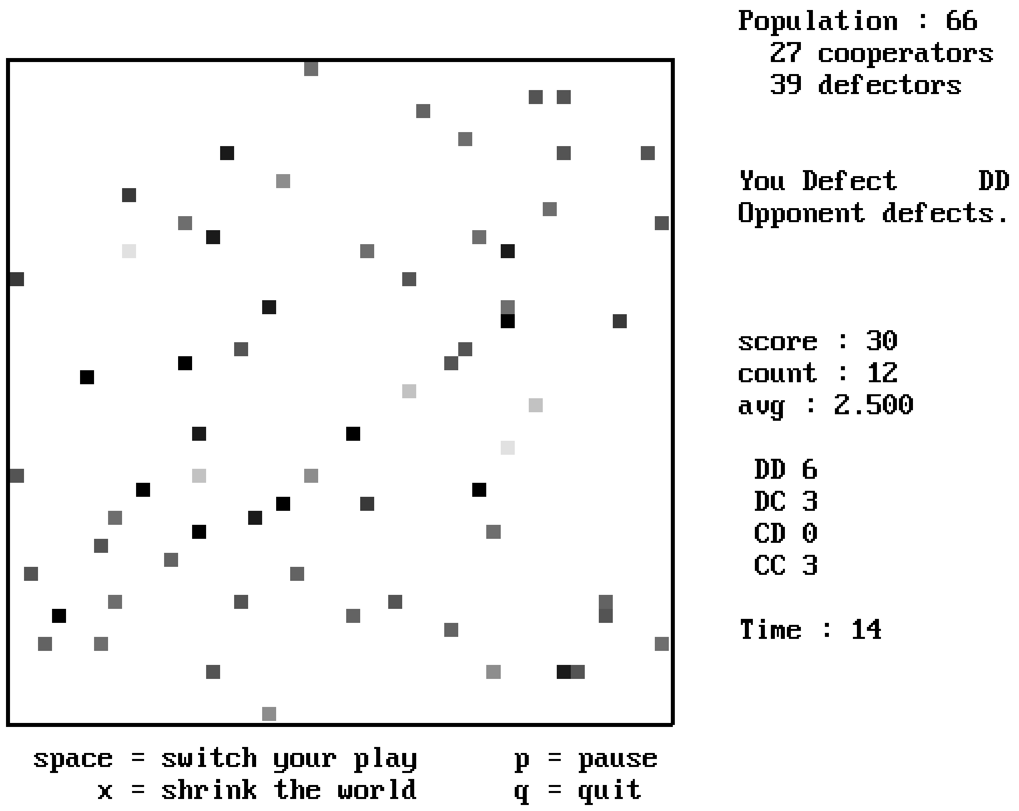
played a card. The exhibitor kept the scores on a chalk board, using the payoff values of Figure 1. The game repeated between 3 and 7 times, at the exhibitor's discretion. He would tell the players, before they played their final card, that this round would be the last. Next, he tallied the results, finding the average payoff per round. Though this demonstration was not supposed to be a controlled experiment, he noted some observances. Many players tended to defect after hearing that they were about to play the last round. An example of how the demonstration went appears below in Figure 21.

| Iteration | Player 1 | Player 2 |
|---|---|---|
| 1 | C | C |
| 2 | D | D |
| 3 | C | D |
| 4 | C | C |
| 5 | D | D |
| Average | 8/5 | 13/5 |

Figure 21 - An Example Game from the Demonstration

The exhibitor also observed that a player rarely got an average score of >= 3. Of course, he could always hold a few extra rounds to make the point that gains made from early defection(s) would not pay off in the long run. No couple of players managed to cooperate every time.

After the demonstration, the exhibitor explained the "Prisoner's Dilemma World" simulation. Children especially liked it. Often, these young players would set their strategy to "defect", and go off in search of cooperators to exploit. They tended to hunt the cooperators into extinction. Then, realizing that any encounter with another player would lower his/her average, the child would stay away from all players. The programmer likened the situation to a city where people are afraid to leave their homes. One high school student caught on to the idea quickly, saying that it would be best to find a cooperator, then follow it around, cooperating with it. Many defectors would kill each other off, meanwhile the user's player would become strong. When a player with little experience gets the sucker payoff, the result might be bad enough to kill it. However, a player with 1000 mutual cooperation games played would not suffer much from one encounter with a defector. The number of cooperators would fall much more rapidly than the defectors. Once the cooperators population was decimated, the defectors would start dying in great numbers, since they depended on the cooperators to survive. In a few interesting cases, the cooperators managed to survive well into the game, even outnumbering the defectors. The cooperators must have been either well grouped, or very lucky with respect to whom they bumped into. A sample screen from this program appears on Figure 22.



```
Population : 66
  27 cooperators
  39 defectors


You Defect     DD
Opponent defects.



score : 30
count : 12
avg : 2.500

 DD 6
 DC 3
 CD 0
 CC 3

Time : 14

space = switch your play      p = pause
    x = shrink the world      q = quit
```

Figure 22 - An Example Screen from "Prisoner's Dilemma World"

The simulation allowed the players to develop a strategy much like Tit-For-Tat. The successful player would cooperate with the cooperators, while either avoiding or defecting against the defectors. The exhibit showed the basics of the Prisoner's Dilemma game, as well as game theory concepts. One teacher suggested trying to have the children develop a simulation like the Prisoner's Dilemma World. The exhibitor commented that the class might enjoy programming a version of Rock-Paper-Scissors World instead. With the help of some junior Rock-Paper-Scissors experts, the payoff matrix appears below.

|  | Rock | Paper | Scissors |
|---|---|---|---|
| Rock | 0,0 | -1,1 | 1,-1 |
| Paper | 1,-1 | 0,0 | -1,1 |
| Scissors | -1,1 | 1,-1 | 0,0 |

Figure 23 - Rock-Paper-Scissors Payoff Matrix

"The Prisoner's Dilemma World" program has only a few possible outcomes. Either the defectors die out, and the cooperators live on together, or the cooperators die out, and then the defectors die out. The chance of all the defectors dying is slim, since the last two defectors would have to kill each other. If only one defector survives, it will eventually kill all of the cooperators. With the addition of a keyboard controlled player, the user can switch strategies to kill off the defectors, while making the cooperators strong. However, the best the user can really hope for is to survive. The simulation starts with an approximately equal amount of All-D and All-C players. In the observed runs, players of both types start dying, but the cooperators die roughly twice as fast. The author observed people who successfully moved their player to survive until the end adopted a strategy similar to Tit-For-Tat. They would cooperate with the All-C players, and defect against the All-D players. The author observed that people tended to defect first against an unknown player. The literature review supports these observations.

The "Payoffs and Rankings" program generated some interesting results. First, the payoff matrices that affect one population of players might not effect another. The average payoff values that did have an affect revealed that the differences DC-CC and DD-CD had the most impact. The second delta value, CC-DD, did not make as much difference to the players. The three dimensional graph illustrates that a player's rank stabilizes when the d1 and d3 values are large. The rank of a player can mean whether or not the player's FSM code survives in the population. A small difference in the rank means a great deal to the player.

Genetic Algorithms find solutions in large search spaces in a short amount of time. The finite state machines evolve into optimal IPD players in most cases. From a computer science point of view, FSMs make good candidates for Genetic Algorithms because they readily break down into bit patterns. Object Oriented Programming, such as C++, allows the programmer flexibility in adding classes and class functions from previous work. This practice cuts down the time needed to engineer software. If the previous work was well designed, then the pieces will be modular enough to be removed and recombined with other software. The author found this to be the case as he evolved the final program from a variety of parent projects.

A graphics demonstration of Xwindow functions was one such project that the author mated with the project. Xwindow libraries provide a machine independent graphics capability. However, the author found that the machine independent concept did not extend to his version of Turbo C. In the future of software engineering, the author expects more to be made of the Xwindows functions. Also, the software engineering paradigm of evolving programs will become more important. This can be done explicitly by having a Genetic Algorithm evolve programs, also know as "soft" computing. More implicitly, software authors are evolving programs by using pieces of their own work over and over again. Consider the personally built libraries of functions as chromosomes, and the biological analogy can be taken to a new level.

As previously mentioned, the "Wandering Players" program overloads the PD playing function. If a programmer used

"Wandering Players" as an include file, she could explore the 3 player IPD game by simply adding a third player number as a parameter to the play_a_game function. For example, the call "play_a_game(11,43);" has players 11 and 43 play one round of the prisoner's dilemma. The call "play_a_game(11,43,28);" would have players 11, 43, and 28 play one round of the 3 player prisoner's dilemma. The internal workings of the play_a_game function are hidden to this programmer, but she would not need to see the details. Thus, the "Wandering Players" program should be seen as a software tool, made of building blocks, rather than as a homogenous whole. Future study by the author will likely use this program as a starting point.

The "Wandering Players" program showed that populations of IPD players evolved differently, depending on whether the programmer used the round-robin or the mobile tournament. These differences can be seen on Figures 24-27. Figure 24 graphs the evolution of cooperation for the population with payoff values of DC=96, CC=90, DD=62, and CD=18. The mobile population does not evolve cooperation. In Figure 25, which uses different payoff values, the mobile population evolves cooperation quickly, and stays at a higher level than the round-robin population. Figure 26 shows that the mobile population evolves faster for one set of payoffs, but the round-robin population attains mutual cooperation. The author includes Figure 27 to show that the evolutions are again different for round-robin and mobile tournaments. These figures show that the mobile tournament does not evolve a population better than the round-robin tournament. However, selecting the mobile tournament causes the population to develop differently than it might with the round-robin tournament. The other payoff values that this thesis studies show that the evolution can be very similar for mobile and round-robin tournaments.

To determine if the outcomes were sensitive to initial conditions, the author ran the "Wandering Players" program using several different initial populations. He achieved this by changing the random number generator's seed value. The populations evolved differently for each seed value, as expected. The initial players form an environment, which has impact on the evolution of cooperation. For the rest of the experiment, the author used a seed value of 95. The consistent seed value assures that the initial populations are the same for each run. Thus, the evolution of cooperation with varied payoff values show that the payoff values themselves make the difference in how a population evolves. The evolution with different seeds appears in Figure 28.

The graphs in Figure 28 show that outcomes are sensitive to intial conditions. The Genetic Algorithm converges the population to a local maxima, without reaching mutual cooperation. The author discusses this problem in the next chapter.

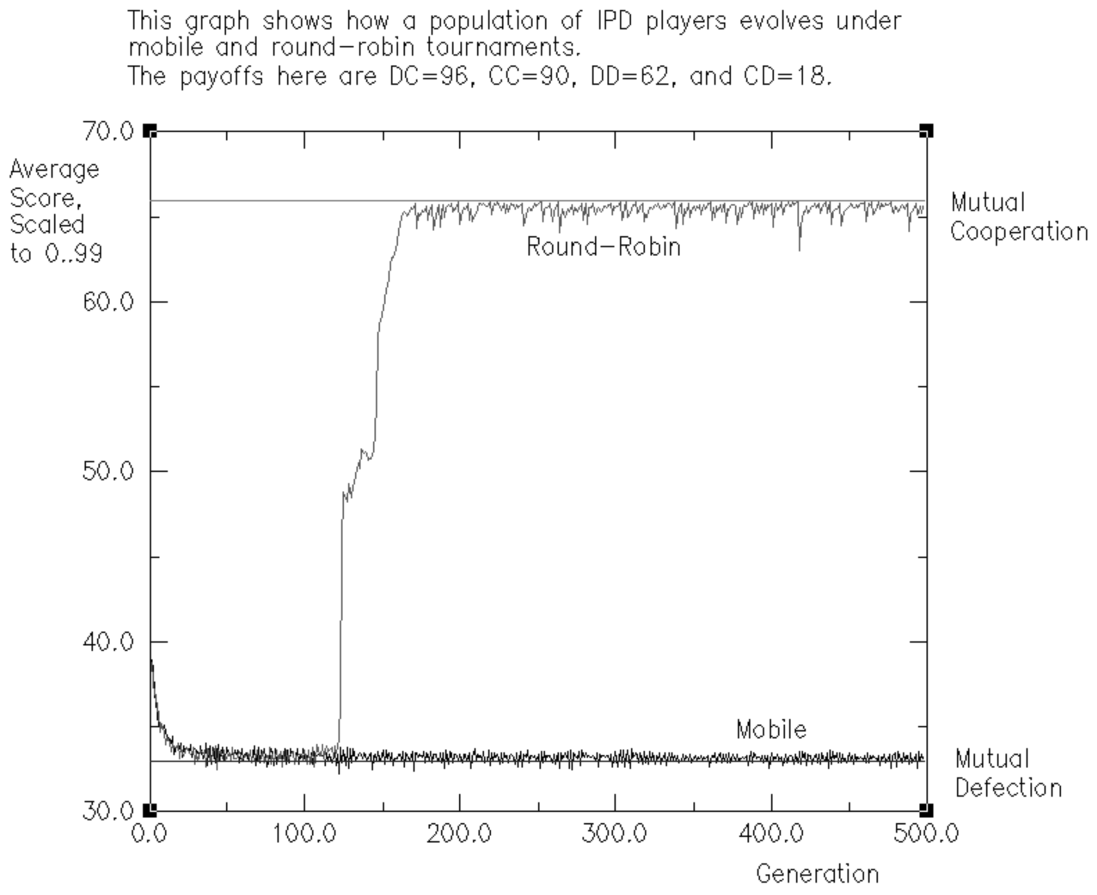


Figure 24 - Mobile and Round-Robin Evolution #1

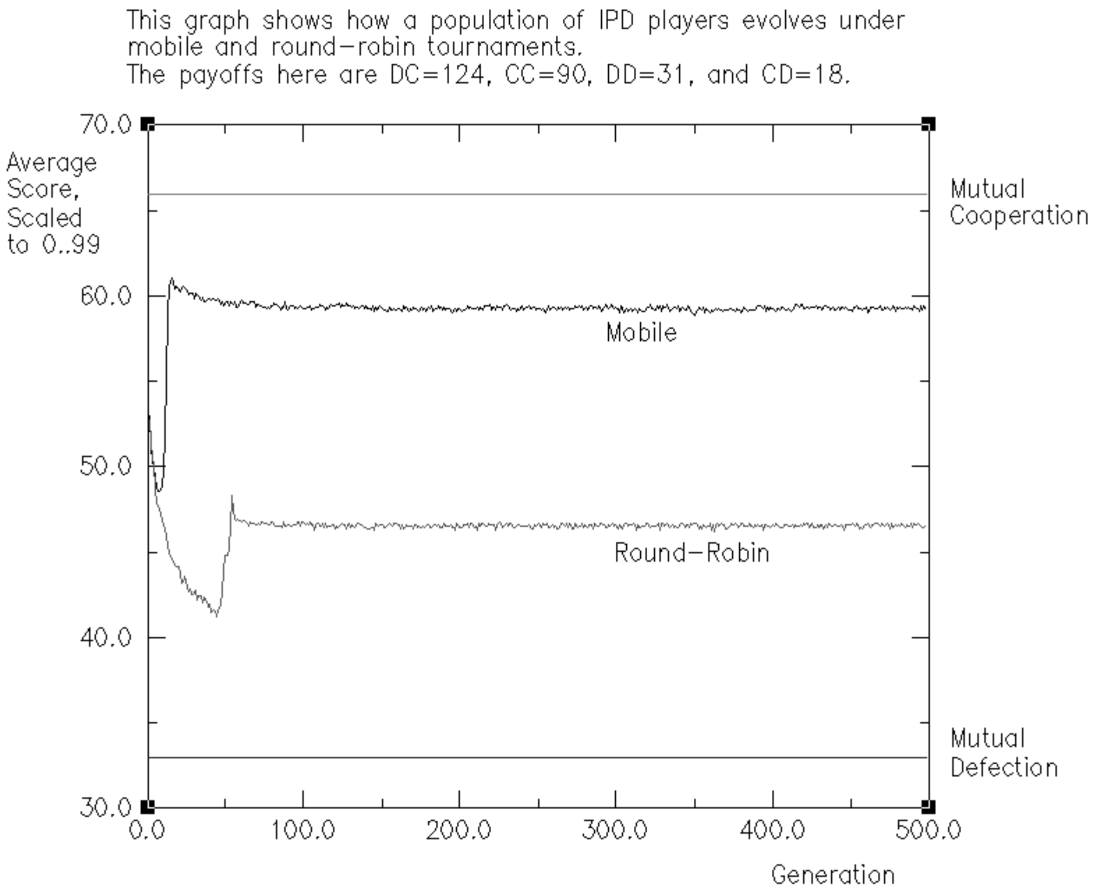


Figure 25 - Mobile and Round-Robin Evolution #2

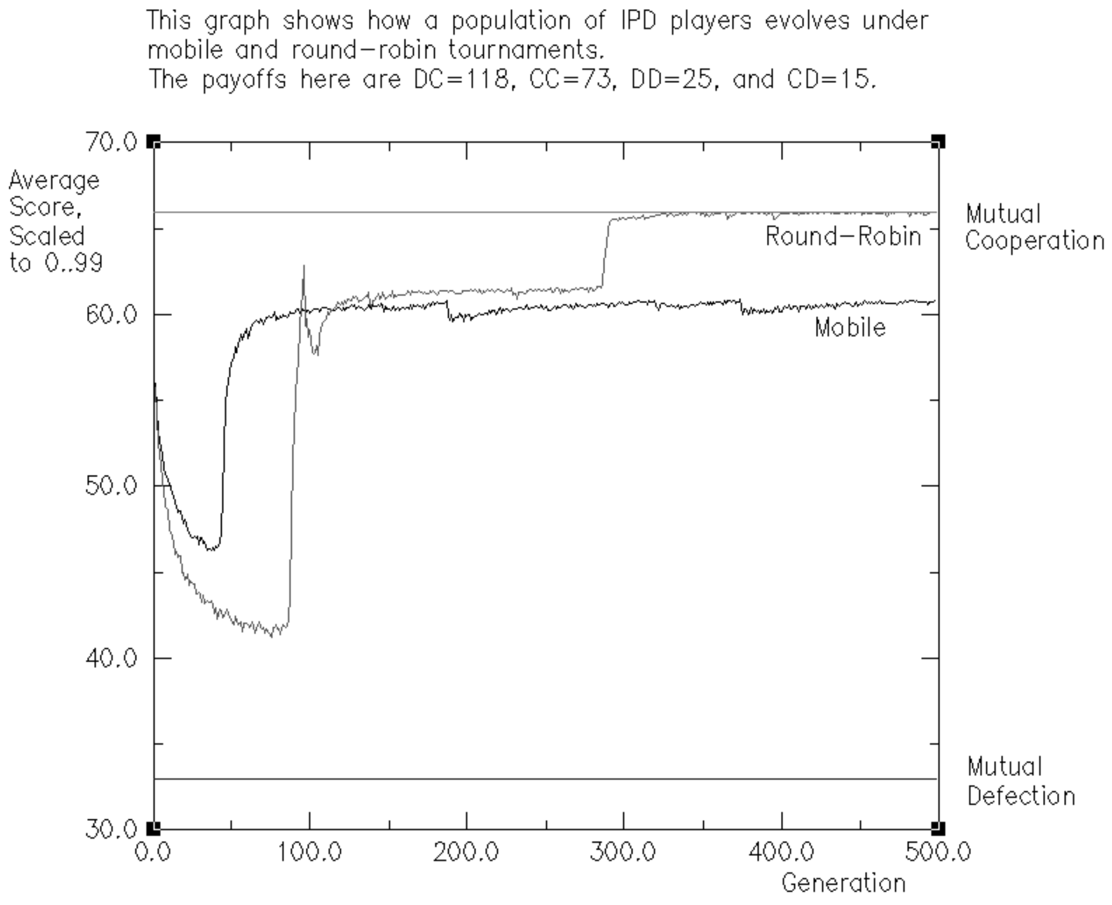


Figure 26 - Mobile and Round-Robin Evolution #3

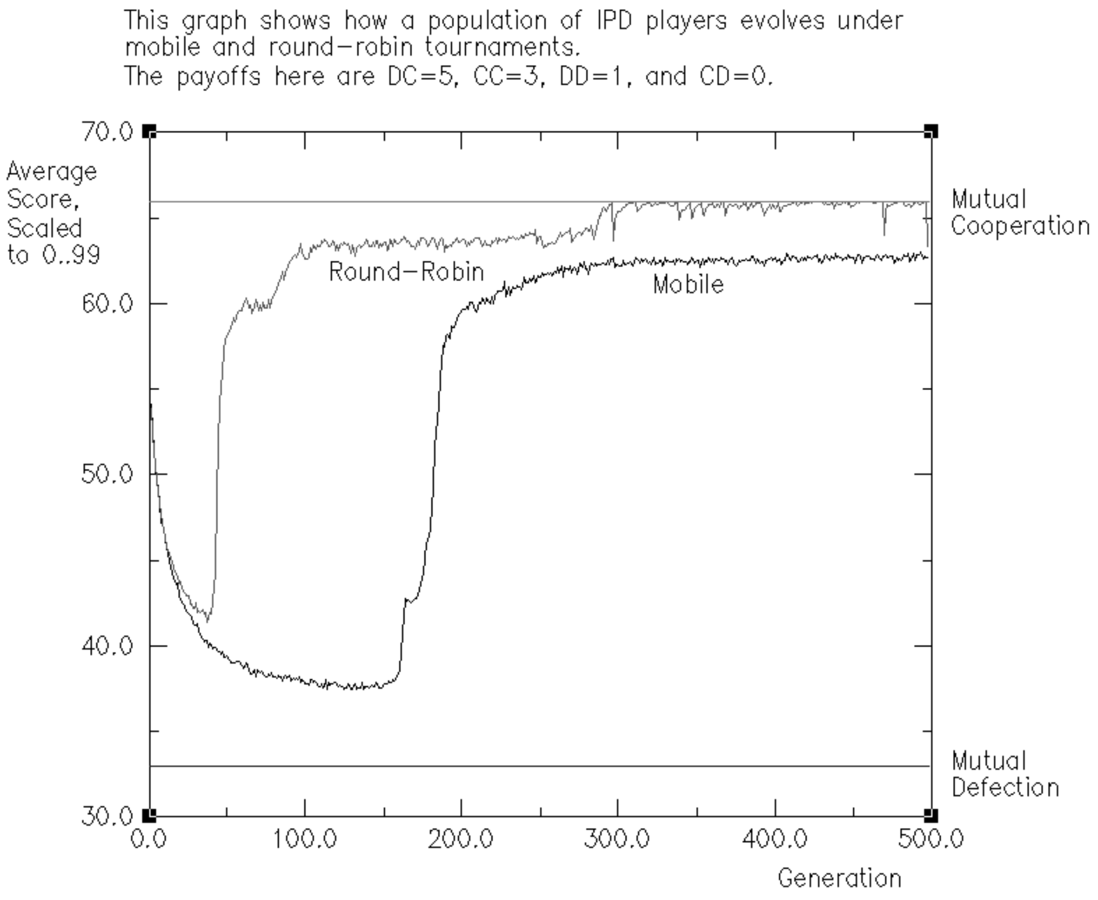


Figure 27 - Mobile and Round-Robin Evolution #4

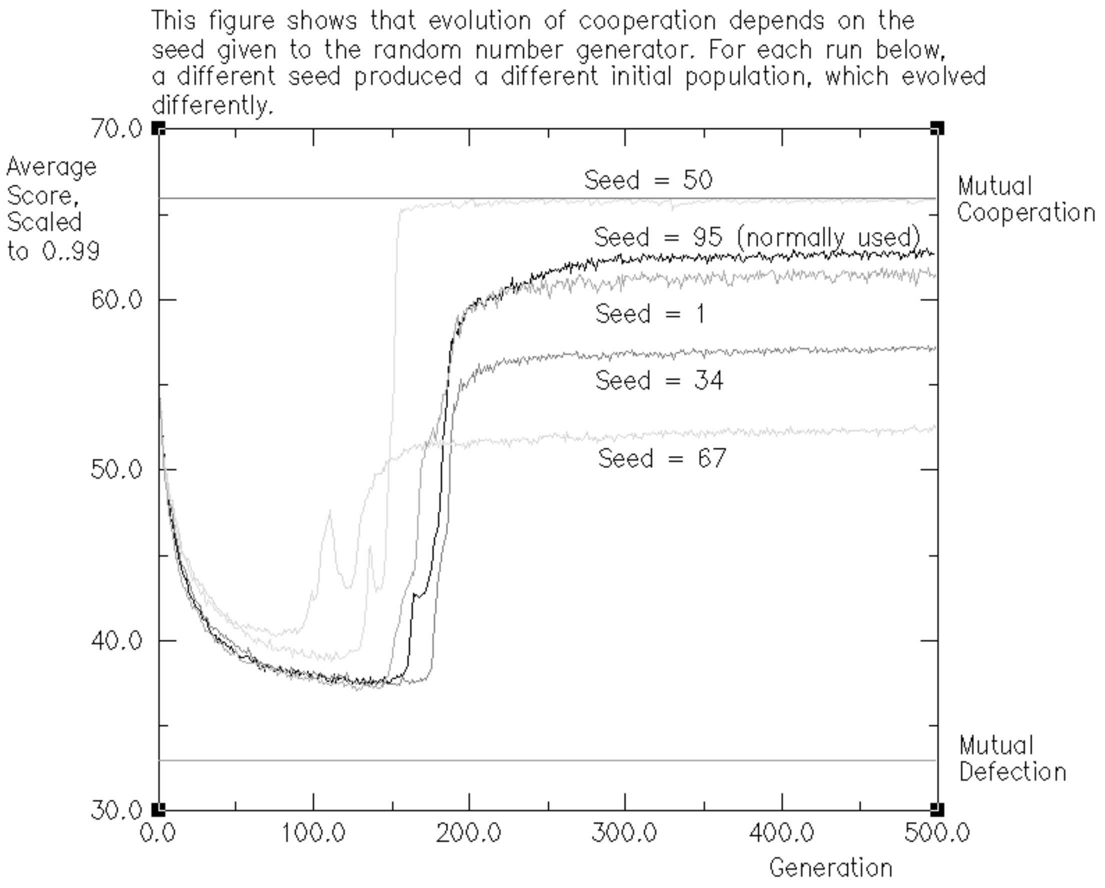


Figure 28 - Evolution with Different Random Number Seeds

# CHAPTER VIII: CONCLUSIONS AND RECOMMENDATIONS

This thesis covers several points. First, payoff values for the IPD game must be carefully chosen, since different values affect how the population evolves. In an extreme case, the payoff values could prohibit cooperation from evolving. Second, a randomly mobile population of IPD players is superior to the round-robin tournament when simulating natural interactions. The resulting evolution of the population might be different, but also more realistic. Third, counting the cooperative branches of the FSMs gives a good idea of how cooperation evolves within the population. The iterated prisoner's dilemma game, combined with a Genetic Algorithm, allows scientists a new method of simulation. "We will be able to approach genetics and evolution as a theoretical design problem." [Axelrod 1987, page 41]. Finally, the evolution of cooperation is sensitive to initial conditions.

Note that the author used the random number generator provided with the Turbo C and Unix C++ packages. The seed for the final two programs were kept the same, to produce the same initial population consistently. Therefore, the results show that the payoff values can have a drastic impact on the evolution of cooperation within a population. Also, the random number generator affects the evolution of cooperation since it changes the initial population.

Cooperation does evolve in real world circumstances.

However, the simulations used to show this are far from complete. This work should be included in future studies of evolving cooperation.

Allowing the players to randomly move about in a world presents a few points to keep in mind. First, the size of the world may have an effect on how the population evolves. Groups of defectors may be able to hold out on the perimeters of society, as the "Cellular Automata" program demonstrates. The defectors influence other squares around them, creating a "discrete chaos", where a square always gets taken over by one of its neighbors. Second, barriers are a natural occurrence in the real world, so they should not be neglected in the simulated worlds. Separation of peoples allows the evolution that Darwin and Mühlenbein describe. Walls can be used to achieve a continent cycle simulation.

Finally, avoidance plays a part in the natural world, as the paper by Stanley et al describes. This could be done by giving two players that meet a chance to escape, if one finds the other player undesirable. For example, a simulation of different species might include a predator/prey relationship. A rabbit will run from a fox, and probably will escape a certain percentage of the time. A simulation where the players have the capability of avoidance would be a new step forward in making the simulated world a more accurate reflection of the natural world. The classic fox/rabbit simulation could be repeated as a design problem, instead of relying on differential equations.

A group project provides another example of a real-life prisoner's dilemma situation. Suppose that two students get together to do a project for a class. Each will receive the same grade, regardless of how much work the individual does. If one student does all the work, both will benefit. If both work, the result should be the best possible grade for both student.

The idea that people in a simulated world will not interact uniformly can be implemented in a less time-consuming fashion. For a large world, the time to simulate players randomly moving about may be prohibitive. In this case, the programmer could generate tables of the likelihood and frequency of two players meeting, based on the player's initial locations. Tables such as these would greatly reduce the simulation time. On the other hand, the simulation with randomly moving players would be very interesting if the players could only mate locally. For example, during the reproduction and crossover steps, the parents would be chosen by only looking at a section of the world. Local mating brings the simulation one step closer to real life. The simulation of wandering players has a panmictic population, but only because of the small world size. The next stage of this simulated world would include a larger area for the players to roam, and allow only mating with other players in the same neighborhood.

The resulting players of each region should be robust, since they are the survivors. However, as Darwin noted, breeding can lead to over-specialization. An interesting question would be whether the resultant players are evolutionary stable strategies (ESS). The author suspects that populations that do not reach full mutual cooperation are not ESS. A Tit-For-Tat player should be able to invade these populations and dominate the gene pool. Within a small number of generations, the Tit-For-Tat players should overtake the population and bring the average fitness to the mutual cooperation level. These ideas lead to future experiments that could have interesting results. In the next section, the author discusses one such future study in detail.

Figure 28 shows that the populations do not always converge to the same average fitness value within 500 generations. This non-convergence is probably due to the Genetic Algorithm, which attained a local maximum instead of evolving the population to an absolute maximum. This problem exists because of reduced variety in the population. The reproduction operator could be adjusted to kill fewer players, or the mutation rate can be increased. Either way allows for a more diverse gene pool, which should solve the problem.

Due to the sensitivity of the experiment to initial conditions, the comparisons of tournament types should be taken as example outcomes. Investigating chaos theory, and its effect on the outcome of this experiment, is the first step in expanding this study.

# CHAPTER IX: FUTURE STUDY

One example of an application of the IPD game are distributed computing networks. For example, a simulation running on a distributed network of computers would depend on time-stamped messages sent between the nodes. If a computer receives a message that has a lower time stamp that other messages already processed, then the computer would need to backtrack, and redo the work done on any messages timed after the late message. From a selfish computer's standpoint, it would be advantageous to spend less time on the simulation than the other computers on the network. Any late messages would then be the problem of the other computers, and this one would not waste time (though it may waste the time of the others on the network by sending late messages). This is only one example of how the IPD game applies to the real world.

The "walls" are a good way to implement a network topology. It would suffer a few disadvantages, however. One disadvantage is that the paths from one point to another would be constrained to 2 dimensions. If two paths crossed, it would be like a street intersection, not like two insulated wires overlapping each other. An alternate way to have a network without using "walls" would be to have nodes that the players would occupy. The players could randomly travel the paths to jump from one node to another, without concern for other players on the path. A simulation with this type of architecture might be more like the Nowak and May simulation; where every node either defects of cooperates, and is taken over in the next round by the strongest neighbor. A defector moving through this network would spread just like a virus on a computer network. "Say you have any LAN topology, such as star, or token ring or whatever. A PD type interaction will have impact on connectivity and reliability of the system. In the Urban environment, companies on a sewer network would cooperate or not with companies downstream or upstream. Again, it has a system impact. A hypothetical simulation along such networks will show the applicability of the PD model for the two areas." [Dr Ragade, 1994, private communication]. The network topology of walls is outside the scope of this thesis, however, the author includes this discussion for future study. The ideas presented herein make a good starting point for network simulations.

This study was limited by the size of the world studied. The time constraints are prohibitive when the programmer expands the world to a greater size. The author had a city construct in mind for the study, with walls making buildings. Outside the city, there would be fewer walls, but also fewer people. It would be interesting to examine evolution in terms of a city and countryside. The programs can be modified to make such a simulation, but the computers would take a long period of time to process such a simulation. Memory constraints may also be a problem. The data structures, however, would only need to have the constants governing array-size made larger. The functions would need work, though, to make the simulation run faster.

## REFERENCES

1. Axelrod, Robert; "Effective Choice in the Prisoner's Dilemma", Journal of Conflict Resolution, Vol. 24, No. 1, March 1980, pp 3-25.

2. Axelrod, Robert; "More Effective Choice in the Prisoner's Dilemma", Journal of Conflict Resolution, Vol. 24, No. 3, September 1980, pp 379-403.

3. Axelrod, Robert; "The Evolution of Strategies in the Iterated Prisoner's Dilemma", Genetic Algorithms and Simulated Annealing, L. Davis (Ed.), 1987, pages 32-41.

4. Homaifar, Abdollah; Turner, Joseph; Ali, Samia; "The N-Queens Problem and Genetic Algorithms", Proceedings of the 1992 IEEE Southeast Conference, Vol. 1, pages 262-267, 1992.

5. Danielson, Peter; "Artificial Morality  Virtuous Robots for Virtual Games", published by Routledge, New York, NY, 1992.

6. Bozma, H. Isil; Duncan, James S.; "Modular system for image analysis using a game-theoretic framework", Image and Vision Computing, Vol. 10, Iss. 6, July-Aug 1992, pages 431-43.

7. Levy, Steven; Artificial Life, Pantheon Books, New York, NY, 1992.

8. Andersen, B.L.; Page, W.C.; McDonnell J.R.; "Multi-Output System Indentification using Evolutionary Programming", Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers, Nov 4-6, 1991, pages 546-550.

9. Stanley, E. Ann; Asklock, Dan; Tesfatsion, Leigh. "Iterated Prisoner's Dilemma with Choice and Refusal of Partners", Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity, ed. by Chris Langton, Addison-Wesley, Vol. 16, 1993.

10. Shinkai, Kyoto T.;  "Three-Person Stackelberg Game With Private Information", Methods of Operations Research, Vol. 64, 1991, pages 323-333.

11. Mühlenbein, Heinz; "Darwin's Continent Cycle Theory and Its Simulation by the Prisoner's Dilemma", Complex Systems, Vol. 5, No. 5, 1992, pages 459-78.

12. Darwin, Charles; On the Origin of Species by Means of Natural Selection, Published by John Murray, London, 1859.

13. Galletly, John E.; "An Overview of Genetic Algorithms", Kybernetes, Vol. 21, No. 6, 1992, p26-30.

14. Huberman, Bernardo A.; Glance, Natalie S.; "Evolutionary Games and Computer Simulations", Proc. Natl. Acad. Sciences (USA), In press. Dynamics of Computation Group, Xerox Palo Alto Research Center, Palo Alto, CA 94304

15. Sigmund, Karl; "On prisoners and cells", Nature, October 29, 1992, Vol. 359, page 774.

16. Nowak, Martin A.; May, Robert M. "Evolutionary games and spatial chaos" Nature, Vol. 359, October 29, 1992, pages 826-829.

17. Fogel, David B.; "Evolving Behaviors in the Iterated Prisoner's Dilemma", Evolutionary Computation Vol. 1, No. 1, pages 77-97.

18. von Neumann, John; Theory of Self-Reproducing Automata, Urbana and London, University of Illinois Press, 1966.

## BIBLIOGRAPHY

1. Axelrod, Robert; "Effective Choice in the Prisoner's Dilemma", Journal of Conflict Resolution, Vol. 24, No. 1, March 1980, pp 3-25.

2. Axelrod, Robert; "More Effective Choice in the Prisoner's Dilemma", Journal of Conflict Resolution, Vol. 24, No. 3, September 1980, pp 379-403.

3. Axelrod, Robert; "The Evolution of Strategies in the Iterated Prisoner's Dilemma", Genetic Algorithms and Simulated Annealing, L. Davis (Ed.), 1987, pages 32-41.

4. Homaifar, Abdollah; Turner, Joseph; Ali, Samia; "The N-Queens Problem and Genetic Algorithms", Proceedings of the 1992 IEEE Southeast Conference, Vol. 1, pages 262-267, 1992.

5. Danielson, Peter; "Artificial Morality  Virtuous Robots for Virtual Games", published by Routledge, New York, NY, 1992.

6. Bozma, H. Isil; Duncan, James S.; "Modular system for image analysis using a game-theoretic framework", Image and Vision Computing, Vol. 10, Iss. 6, July-Aug 1992, pages 431-43.

7. Levy, Steven; Artificial Life, Pantheon Books, New York, NY, 1992.

8. Andersen, B.L.; Page, W.C.; McDonnell J.R.; "Multi-Output System Indentification using Evolutionary Programming", Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers, Nov 4-6, 1991, pages 546-550.

9. Stanley, E. Ann; Asklock, Dan; Tesfatsion, Leigh. "Iterated Prisoner's Dilemma with Choice and Refusal of Partners", Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity, ed. by Chris Langton, Addison-Wesley, Vol. 16, 1993.

10. Shinkai, Kyoto T.; "Three-Person Stackelberg Game With Private Information", Methods of Operations Research, Vol. 64, 1991, pages 323-333.

11. Mühlenbein, Heinz; "Darwin's Continent Cycle Theory and Its Simulation by the Prisoner's Dilemma", Complex Systems, Vol. 5, No. 5, 1992, pages 459-78.

12. Darwin, Charles; On the Origin of Species by Means of Natural Selection, Published by John Murray, London, 1859.

13. Galletly, John E.; "An Overview of Genetic Algorithms", Kybernetes, Vol. 21, No. 6, 1992, p26-30.

14. Huberman, Bernardo A.; Glance, Natalie S.; "Evolutionary Games and Computer Simulations", Proc. Natl. Acad. Sciences (USA), In press. Dynamics of Computation Group, Xerox Palo Alto Research Center, Palo Alto, CA 94304

15. Sigmund, Karl; "On prisoners and cells", Nature, October 29, 1992, Vol. 359, page 774.

16. Nowak, Martin A.; May, Robert M. "Evolutionary games and spatial chaos" Nature, Vol. 359, October 29, 1992, pages 826-829.

17. Fogel, David B.; "Evolving Behaviors in the Iterated Prisoner's Dilemma", Evolutionary Computation Vol. 1, No. 1, pages 77-97.

18. von Neumann, John; Theory of Self-Reproducing Automata, Urbana and London, University of Illinois Press, 1966.

19. Lugar, George F.; Stubblefield, William A.; Artificial Intelligence Structure and Strategies for Complex Problem Solving Second Edition, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1993.

20. Rapoport, Anatol; Two-Person Game Theory The Essential Ideas, Fourth Printing, Ann Arbor Science, 1973.

21. Rapoport, Anatol; N-Person Game Theory Concepts and Applications, Ann Arbor Science, 1970.

22. Rapoport, Anatol; "Applications of Game-Theoretic Concepts in Biology", Bulletin of Mathematical Biology Vol. 47, No. 2, 1985, pages 161-192.

23. Rosenthal, Edward C.; "Coalition Formation under Limited Communication", Games and Economic Behavior, Vol. 4, 1992, pages 402-421.

24. Davis, Lawrence; "Bit-Climbing, Representational Bias, and Test Suite Design", Proceedings of the Fourth International Conference on Genetic Algorithms, July 13-16, 1991, pages 18-23.

25. Horner, Andrew; Goldberg, David E.; "Genetic Algorithms and Computer-Assisted Music Composition", Proceedings of the Fourth International Conference on Genetic Algorithms, July 13-16, 1991, pages 437-441.

26. Koza, John R.; "Evolving a Computer Program to Generate

Random Numbers Using the Genetic Programming Paradigm", Proceedings of the Fourth International Conference on Genetic Algorithms, July 13-16, 1991, pages 37-44.

27. Grefenstette, John J.; "Incorporating Porblem Specific Knowledge into Genetic Algorithms", Genetic Algorithms and Simulated Annealing, L. Davis (Editor), 1987, pages 42-61.

28. Goldberg, David E.; "Simple Genetic Algorithms and the Minimal, Deceptive Problem", Genetic Algorithms and Simulated Annealing, L. Davis (Editor), 1987, pages 42-61.

29. Lindgren, Kristian; "Evolutionary Phenomena in Simple Dynamics", Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, edited by C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Addison-Wesley, 1991, pages 295-312.

30. Barnett, Claude C.; Stewart, David J.; Amlaner, Charles J, Jr.; "'Deercare' A Deer Population and Hunting Simulation Model version 1.0", Proceedings of the 1990 Symposium on Applied Computing, April 5-6, 1990, pages 304-307.

31. Mühlenbein, Heinz; "Asynchronous Parallel Search by the Parallel Genetic Algorithm", Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, December 2-5, 1991, pages 526-533.

32. Pham, D. T.; Yang, Y.; "Optimization of Multi-modal Discrete Functions using Genetic Algorithms", Proceedings of the Institution of Mechanical Engineers, Part D, Journal of Automobile Engineering, Vol. 207, No. 1, 1993, pages 53-59.

33. Ragade, Rammohan K.; "Fuzzy Games in the Analysis of Options", Journal of Cybernetics, Volume 6, 1976, pages 213-221.

34. Musselman, Kenneth J.; "Conducting a Successful Simulation Project", Proceedings of the 1992 Winter Simulation Conference, edited by J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, pages 115-125.

35. Rozenblit, Jerzy W.; Jankowski, Piotr L.; "An Integrated Framework of Knowledge-based Modeling and Simulation of Natural Systems", Simulation, September 1991, Vol. 57, No. 3, pages 152-165.

36. Sargent, Robert G.; "Validation and Verification of Simulation Models", Proceedings of the 1992 Winter Simulation Conference, edited by J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, pages 104-114.

37. Penfold, H. B.; Diessel, O. F.; Bentink, M. W.; "A Genetic

Breeding Algorithm Which Exhibits Self-Organizing in Neural Networks", Artificial Intelligence Application and Neural Networks, AINN '90, Zurich, Switzerland, June 25-27, 1990, pages 293-296.

38. Karp, Richard M.; Zhang, Yanjun; "On Parallel Evaluation of Game Trees", Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures, June 18-21, 1989, Santa Fe, New Mexico, pages 409-420.

39. Arunkumar, S.; Chockalingam, T.; "Randomized Heuristics for the Mapping Problem", International Journal of High Speed Computing, Vol. 4, No. 4, 1992, pages 289-299.

40. Crawford, Walt; "Digital Diversions: Taking a Break From Productivity", Library Hi Tech, Vol. 9, Iss. 34, No. 2, 1991, pages 81-98.

41. Gottinger, H. W.; "Complexity of Games and Bounded Rationality", Optimization, Volume 21, Number 6, 1990, pages 991-1003.

42. Berghel, Hal; Roach, David; Talburt, John; "Approximate String Matching and the Automation of Word Games", Proceedings of the 1990 Symposium on Applied Computing, April 5-6, 1990, pages 209-212.

43. Driessen, T.; Muto, S.; Nakayama, M.; "A Cooperative Game of Information Trading: The Core, the Nucleolus and the Kernel", Methods and Models of Operations Research, 1992, Vol. 36, pages 55-72.

44. Miller, Douglas A.; Zucker, Steven W.; "Efficient Simplex-Like Methods for Equilibira of Nonsymmetric Analog Networks", Neural Computation, 1992, Vol. 4, pages 167-190.

45. van den Nouweland, Anne; Borm, Peter; Tijs, Stef; "On Hypergraph Communication Situations", Methods of Operations Research, Vol. 64, 1991, pages 319-322.

46. Bitar, Javier; Goles, Eric; "Parallel Chip Firing Games on Graphs", Theoretical Computer Science, Vol. 92, 1992, pages 291-300

47. Böhm, Max; Speckenmeyer, Ewald; "A Dynamic Processor Tree for Solving Game Trees in Parallel", Methods of Operations Research, Vol. 63, 1990, pages 479-489.

48. Wysocki, M.; Kwolek, B.; Irzenski, W.; "Solving of Dynamic Nash and Stackelberg Games in Parallel on a Network of Microcomputers", Parallel and Distributed Computing in Engineering Systems, edited by S. Tzafestas, P. Borne, and L.

Grandinetti, 1992.

49. Economides, Anastasios A.; Silvester, John A.; "A Game Theory Approach to Cooperative and Non-Cooperative Routing Problems", SBT/IEEE International Telecommunications Symposium ITS '90 Symposium Record, September 3-6, 1990, pages 597-601.

50. Mastaglio, Thomas W.; "Networked Simulators and Computer-Supported Wargame Simulations", 1991 IEEE International Conference On Systems, Man and Cybernetics, Vol. 1, pages 303-307.

51. Sebald, A. V.; Schlenzig, J.; Fogel, D. B.; "Minimax Design of CMAC Encoded Neural Network Controllers Using Evolutionary Programming", Conference Record of the 25th Asilomar Conference on Signals, Systems, and Computers, November 4-6, 1991, pages 551-555.

52. Stiller, Lewis; "Group Graphs and Computational Symmetry on Massively Parallel Architecture", The Journal of Supercomputing, Vol. 5, 1991, pages 99-117.

53. Crooks, Kyle Evan; Kandel, Abraham; "Combat Modeling using Fuzzy Expected Values", Fuzzy Sets and Systems, Vol. 47, 1992, pages 293-301.

54. Breugnot, Daniel; Gourgand, Michel; Hill, David; Kellert, Patrick; "GAME: An Object-Oriented Approach to Computer Animation in Flexible Manufacturing System Modelling", IEEE, 1991.

55. Heitkoetter, Joerg, ed. (1993) "The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ)", Usenet: comp.ai.genetic. Available via anonymous ftp from rtfm.mit.edu in pub/usenet/news.answers/ai-faq/genetic/part?. ~80 pages. Or simply call it "the Guide", or "HHGTEC" for acronymaniacs.

56. Gurari, Eitan M.; An Introduction to the Theory of Computation, Rockville, MD, Computer Science Press, 1989.

57. Aho; Hopcraft; Ullman; The Design and Analysis of Computer Algorithms, Reading, Mass., Addison-Wesley Pub., 1974.

58. Qian, Yu; Tessier, Patrick; Dumont, Guy A.; "Process Modelling and Optimization of Systems with Imprecise and Conflicting Equations" Engineering Applications of Artificial Intelligence Volume 6, Number 1 (1993): 39-47

59. Glance, Natalie S.; Huberman, Bernardo A.; "The Dynamics of Social Dilemmas", Scientific American, March 1994, pages 76-81.

60. Bui, Tung X.; A Group Decision Support System for Cooperative Multiple Criteria Group Decision Making, Edited by G. Goos and J. Hartmanis, Springer-Verlag, 1987.

61. Barkakati, Nabajyoti; Unix Desktop Guide to X/Motif, Carmel, IN, Hayden Books, 1991.

62. Johnson, Eric F.; Reichard, Kevin; X Window Applications Programming, Second Edition, New York, NY, MIS Press, 1992.

63. Beck, Kent; Cunningham, Ward; "A Laboratory For Teaching Object-Oriented Thinking", Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA) 1989 Proceedings October 1-6, 1989, pages 1-6.

64. Jones, A. J.; Game Theory Mathematical Models of Conflict, Chichester, West Sussex, England, Ellis Horwood Limited, 1980.
65. Ditto, William L.; Pecora, Louis M.; "Mastering Chaos", Scientific American, August 1993, Pages 78-84.

66. Fudenberg, Drew; Tirole, Jean; Game Theory, Second Printing, Cambridge, Mass., The MIT Press, 1992.

## APPENDIX : SAMPLE PROGRAM OUTPUT

**Output from "Payoffs and Rankings", the 10 test runs for the 5 player case.**

```
   Only cases where 2*CC > CD + DC are printed.
   All payoff values conform to DC > CC > DD > CD.
   Total occurrences = 406

   d1 = the average difference between DC and CC values.
   d2 = the average difference between CC and DD values.
   d3 = the average difference between DD and CD values.
```

**Run #1**

```
pattern    # occurrences      d1  d2  d3     dc  cc  dd  cd
1 4 2 5 3     66 times   16%  34  58  13  |  124  90  31  18
1 2 3 5 2     53 times   13%  45  47  10  |  118  73  25  15
1 5 2 4 3     18 times    4%   6  27  43  |   96  90  62  18
1 5 3 4 2    138 times   33%  29  21  43  |  115  86  65  21
2 3 1 5 4      9 times    2%   6 119   4  |  161 155  35  30
2 4 1 5 3    121 times   29%  14  77  14  |  124 110  32  18
2 5 1 4 3      1 times    0%   6  56  63  |  136 130  74  11
```

**Run #2**

```
pattern    # occurrences      d1  d2  d3     dc  cc  dd  cd
1 2 3 4 5    10 times    2%  47   7  55 | 120  72  65   9
1 2 3 5 4    144 times   35%  42  42  17  |  119  77  34  17
1 2 4 5 3    121 times   29%  21  35  36  |  114  92  56  20
1 3 4 5 2    131 times   32%  10  75  20  |  127 116  41  21
```

**Run #3**

```
pattern    # occurrences      d1  d2  d3     dc  cc  dd  cd
1 2 3 4 5    169 times   41%  21  76  10  |  126 104  28  17
1 3 2 4 5     36 times    8%  27  52  20  |  122  95  42  22
1 4 2 3 5     87 times   21%  22  33  35  |  110  88  54  19
1 4 3 2 5     60 times   14%  45  34  28  |  126  80  46  17
2 4 1 3 5      4 times    0%   5  19  66  |  112 107  87  21
2 4 3 1 5      9 times    2%  31  12  25  |   81  50  38  12
3 4 1 2 5     11 times    2%   5   9  62  |  120 115 105  43
3 4 2 1 5     30 times    7%  29   6  53  |  110  80  73  19
```

**Run #4**

| pattern | # occurrences | | d1 | d2 | d3 | dc | cc | dd | cd |
|---|---|---|---|---|---|---|---|---|---|
| 2 1 3 4 5 | 37 times | 9% | 13 | 81 | 14 | 129 | 116 | 34 | 20 |
| 2 1 3 5 4 | 16 times | 3% | 17 | 89 | 5 | 123 | 105 | 16 | 10 |
| 2 1 4 5 3 | 28 times | 6% | 14 | 120 | 5 | 158 | 144 | 24 | 19 |
| 3 1 2 4 5 | 26 times | 6% | 16 | 58 | 12 | 106 | 89 | 30 | 18 |
| 3 2 1 4 5 | 2 times | 0% | 4 | 73 | 39 | 139 | 135 | 62 | 23 |
| 3 4 1 2 5 | 87 times | 21% | 14 | 21 | 54 | 115 | 101 | 80 | 25 |
| 4 1 2 3 5 | 90 times | 22% | 36 | 52 | 13 | 120 | 83 | 31 | 17 |
| 4 2 1 3 5 | 17 times | 4% | 39 | 55 | 33 | 162 | 122 | 67 | 33 |
| 4 3 1 2 5 | 69 times | 16% | 32 | 23 | 31 | 103 | 71 | 47 | 15 |
| 5 1 2 3 4 | 34 times | 8% | 45 | 51 | 4 | 113 | 67 | 15 | 10 |

**Run #5**

| pattern | # occurrences | | d1 | d2 | d3 | dc | cc | dd | cd |
|---|---|---|---|---|---|---|---|---|---|
| 3 2 4 5 1 | 103 times | 25% | 21 | 82 | 5 | 124 | 102 | 20 | 14 |
| 3 2 5 4 1 | 85 times | 20% | 45 | 55 | 12 | 128 | 83 | 28 | 15 |
| 4 2 3 5 1 | 11 times | 2% | 4 | 100 | 16 | 163 | 159 | 58 | 41 |
| 4 2 5 3 1 | 110 times | 27% | 31 | 25 | 31 | 105 | 73 | 48 | 17 |
| 5 2 3 4 1 | 3 times | 0% | 6 | 92 | 27 | 129 | 123 | 30 | 3 |
| 5 2 4 3 1 | 94 times | 23% | 10 | 33 | 49 | 120 | 110 | 77 | 28 |

**Run #6**

| pattern | # occurrences | | d1 | d2 | d3 | dc | cc | dd | cd |
|---|---|---|---|---|---|---|---|---|---|
| 2 3 5 4 1 | 10 times | 2% | 22 | 12 | 55 | 109 | 87 | 74 | 18 |
| 2 4 5 3 1 | 24 times | 5% | 17 | 6 | 67 | 125 | 108 | 101 | 33 |
| 3 2 4 5 1 | 55 times | 13% | 32 | 42 | 13 | 107 | 74 | 32 | 18 |
| 3 2 5 4 1 | 137 times | 33% | 29 | 28 | 36 | 113 | 83 | 55 | 19 |
| 4 1 2 5 3 | 11 times | 2% | 6 | 117 | 5 | 155 | 149 | 31 | 26 |
| 4 1 3 5 2 | 167 times | 41% | 23 | 75 | 12 | 128 | 104 | 29 | 17 |
| 4 2 3 5 1 | 2 times | 0% | 47 | 60 | 10 | 131 | 83 | 23 | 13 |

**Run #7**

| pattern | # occurrences | | d1 | d2 | d3 | dc | cc | dd | cd |
|---|---|---|---|---|---|---|---|---|---|
| 2 1 3 5 4 | 21 times | 5% | 30 | 57 | 4 | 102 | 71 | 14 | 10 |
| 2 1 4 5 3 | 2 times | 0% | 10 | 30 | 3 | 48 | 38 | 7 | 4 |
| 2 1 5 4 3 | 10 times | 2% | 33 | 88 | 3 | 134 | 101 | 12 | 8 |
| 3 1 2 5 4 | 111 times | 27% | 33 | 52 | 13 | 118 | 84 | 32 | 18 |
| 3 1 4 5 2 | 1 times | 0% | 26 | 103 | 15 | 186 | 160 | 57 | 42 |
| 3 1 5 4 2 | 10 times | 2% | 20 | 82 | 8 | 128 | 107 | 25 | 16 |
| 3 2 1 5 4 | 174 times | 42% | 27 | 24 | 41 | 114 | 87 | 62 | 21 |
| 4 1 2 5 3 | 17 times | 4% | 7 | 69 | 29 | 132 | 124 | 54 | 25 |
| 4 1 3 5 2 | 6 times | 1% | 6 | 65 | 19 | 105 | 99 | 34 | 15 |
| 4 1 5 3 2 | 54 times | 13% | 12 | 105 | 8 | 146 | 133 | 27 | 19 |

**Run #8**

```
pattern    # occurrences       d1   d2   d3     dc   cc   dd   cd
5 3 2 1 4    406 times  100%   26   50   24 |  120   94   44   19
```

**Run #9**

```
pattern    # occurrences       d1   d2   d3     dc   cc   dd   cd
2 4 1 3 5     23 times    5%    7  119    7 |  153  146   26   18
3 4 1 2 5    100 times   24%   18   75   10 |  123  104   29   18
3 5 1 2 4     85 times   20%   36   57   13 |  124   88   30   17
4 5 1 2 3     53 times   13%   34   39   18 |  113   79   39   20
5 4 1 2 3     21 times    5%   32   30   23 |   97   65   34   10
5 4 2 1 3    107 times   26%   25   21   45 |  113   88   66   20
5 4 3 1 2     17 times    4%   16    5   73 |  131  114  109   36
```

**Run #10**

```
pattern    # occurrences       d1   d2   d3     dc   cc   dd   cd
4 2 3 1 5     83 times   20%   15   96    8 |  137  121   25   17
5 2 3 1 4    323 times   79%   28   38   28 |  116   87   48   19
```

Figure 29 - Test Runs from "Payoffs and Rankings"

**Output from the first run of "Payoffs and Rankings"**

```
                                                 Number of
      FSM code                                 outcomes by type
Name  i 0   1   2   3   4   5   6   7         dc   cc   dd   cd
  A   d7b7e5f0a64565a367d2ea58b8a24771ac     256  046  156  042
  B   392fbe049cb2947b07dc2ebccfe7fc1aa8     050  200  038  212
  C   beb1a00441a191c70a5bf70c27fa4e2181     096  188  096  120
  D   2beb886353352defd7437fd00207b31370     132  118  122  128
  E   80e26f853136e0e854eade70f5441e8082     184  104  144  068
  F   55224cf4a57204168ae3491b6abafd0091     148  122  132  098
  G   fbd6ecddd6b010f869192c50ca0a4f8fa3     106  178  082  134
  H   46e9e93d62c0094cd4ed9461257278459 8    220  068  166  046
  I   7eb3325c01ad20b595291e10707bf47d5e     114  150  124  112
  J   2d019364d08e1f2b348ea9beb70742bde8     094  162  110  134
  K   108b9738b37cebed8cd19a147aa7b29b45     096  182  064  158
  L   36bf67841d9e82a044682651c794e164d7     192  088  162  058
  M   db7d291c01f15f831a0ef2f25e176bde24     114  150  106  130
  N   11613922efcda989558c900a3ef9cf6c1b     140  140  112  108
  O   e0bdc1c1b954d7d47202f3dbaa197ec8b1     130  150  092  128
  P   57b0232281d28d7b7f39d675fc816791fe     228  068  134  070
  Q   2f23946b4f8c66f51a5b1269523e56d255     140  130  142  088
  R   01876f8dcc36ec57b854dfea7049f86513     164  114  138  084
  S   c2e79611946193418c4cbbdc8b7e38a0b9     278  026  111  024
  T   0e9ccb2eaab12be9ef10fc713995c6eede     068  162  084  186
  U   5bf618533770d7366dc5e5a698534ff078     116  166  106  112
  V   fa2c01acae619ff189c8c90ba9995de389     022  228  028  222
  W   6951f7048d94b3d34e0ecba0b79f348152     092  152  084  172
  X   b4c3f7c054254b642828621e6d0ccc5357     208  082  164  046
  Y   dc4e1cca719609f1414b369e783f19fc8d     098  156  108  138
  Z   f6e272dc33b54c68ccb00b4c7bfbdc05db     084  194  088  134
 AA   ce35245bb202804beb6887cd5389b4d17b     084  160  082  174
 AB   4fdee482a55efe572db00bdfcdac6dc228     066  208  070  156
 AC   dd213b6e27d1cd6e3a7a2786ffe8183c8b     040  216  052  192
 AD   beea1bb344f6c2dfa8d48e21fff1f14aa4     074  170  074  182
 AE   f0d6ec47a91c034f189449d11d81d9d665     206  094  122  078
 AF   0d2c2249b1153507ffcec32bd4ae557458     090  154  088  168
 AG   9e4e26d3eb0cfecd885cc66059fbc153a1     080  166  078  176
 AH   57e0d7284ffd743326170de55ec38524bd     230  064  150  056
 AI   c1ef95270bc1c88f261033b1273b07d625     216  066  166  052
 AJ   0c874994bae8c937d061a12f6a7ca7d2c1     060  176  078  186
 AK   3ad2c87b2ea92cc22637148e68cd5c1e36     110  158  104  128
 AL   2f00179e88e34b30b8638bbe548694abf5     074  170  068  188
 AM   167bf721999e8984618be6a09fd3d4c5ce     158  130  092  120
 AN   491e754aa1f5f564e21745e3aa532e2fbb     106  138  102  154
 AO   77a70cc64e443ce9bceddee99688e58448     132  146  124  098
 AP   dc74efea628be3cf34c83e33d7a3ad1a20     058  200  068  174
 AQ   f750fade2cd83d89e6184aaa8177164043     236  060  154  050
 AR   4ae5c18d794410eb83e4fe5c3a67aca74e     082  188  104  126
 AS   ae7011ffd380570d55193747985b6ebd51     126  162  098  114
 AT   8c7408ec03d9a2996d9bc4f8a341c822d0     088  160  084  168
 AU   237d0217ac6fb678f591f720854e30f4a3     168  110  132  090
 AV   6c586efabe1ba2f9238492c0bfa58befeb     088  156  098  158
 AW   63cc38dc0122ff6a72bbf1f820e4d6f72c     136  134  140  090
 AX   ea8bf2cbd767aad641f82a1fe85e56ab53     000  252  000  248
```

Table 7 – The randomized player FSMs.

## VITA

Michael Clark Weeks was born in Louisville, Kentucky, on May 1, 1968. He is son of R. Clark Weeks and Mildred Carricato Weeks. He grew up in Louisville, and attended Saint Xavier High School, graduating in 1986. He went on to study at the University of Louisville Speed Scientific School, in the Engineering Math and Computer Science Department. He entered the graduate program in May, 1992.

In 1991, he entered Engineers' Days with a game program written in the language C. This project won second place. In 1992, he won first place in Engineers' Days with a beer brewing program that used a Genetic Algorithm. The following January, 1993, he was granted a Graduate Service Assistanceship with the Disability Resource Center at the University of Louisville. He was awarded the Outstanding Senior Award by the Speed School Alumni Foundation in May of 1993.