A Comparison of Effectiveness Between Different Representations of Agent Cognitions Using Evolutionary Algorithms

Glenn Wilkie-Sullivan -
40208762

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Games Development

School of Computing

March 21, 2019

Authorship Declaration

I, Glenn Wilkie-Sullivan, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date:

Matriculation no:
40208762

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Over the course of over 60 years, the field of game theory has been examined comprehensively, in situations where it can be theoretically applied to economic or mathematical problems. This project will attempt to blend the field of game theory with machine learning and evolutionary algorithms in an attempt to apply theory based reasoning to applicable problems and dilemmata, in the hopes that the results will show success and possible applications to multiple other fields of computing.

Contents

List of Tables

List of Figures

Acknowledgements

Insert acknowledgements here

-

1 Introduction

Since 1713, game theory has been a prevalent and ever-rising field. Early work saw the introduction of game theory principles into solutions for economic and mathematical problems, yet didn't hit its main strides until the 1920s and 50s with the work of John von Neumann, Merrill Flood, Melvin Dresher and John Forbes Nash Jr. Books and academic journals such as, "Theory of Games and Economic Behavior" [von Neumann et al., 1944] from von Neumann and, "Two-person Cooperative Games" [Nash, 1950b] from Nash saw possibilities of game theory having applications in fields outside of economics and mathematics. Subsequently in the same decade as Nash's work, game theory saw its first appearance into fields such as philosophy, psychology and political science.
Since then, one of the main advances in game theory was made by Robert Axelrod, with his work on the iterated prisoner's dilemma - this is the most popular and instrumental occurence of game theory having application in fields such as artificial intelligence and evolutionary biology.
This project is an attempt to combine multiple facets of game theory and artificial intelligence, in hope that the end result will be educational and/or useful to the fields; there is also a goal that the end result will be applicable to video games in some way, either through AI agents or through general design. There are multiple stepping stones within the fields of game theory, mathematics, etc. which need to be examined in order to gain a further understanding of the project, which we will look at in the literature of the field(s).

1.1 Aims & Objectives

In order to carry out the implementation side of this project, a strong understanding of the principles and theory behind game theory and artificial intelligence are required. As they tie into each other in this project, there is also a necessity behind learning how to approach the blending of both fields. As seen in the following section, this project covers a considerable amount of previous work and literature which provides the knowledge and insight required to understand or recreate this project. In the literature review, the topics of game theory, artificial intelligence, neural networks and evolutionary algorithms are explored - through their history, applications and relation to this project. From this knowledge, we can form research questions to focus on, and eventually answer at the end of the project. The second part of this report will cover the application of the theory covered within the literature review, as well as a continuation into original ideas for research, as seen by

the research questions. This will entail creating a program, or programs, which can compare the validity and effectiveness of machine learning techniques using a game theory scenario - mainly, the prisoner's dilemma. This applies to the research goals as shown:

1. Research different models of agent cognition, in particular neural networks and finite state machines.
2. Implement an evolutionary algorithm that can evolve agents to play a repeated Prisoner's Dilemma, where the representation of the evolving agent strategies can be changed between neural networks and finite state machines.
3. Test the evolutionary algorithm by recreating existing results from the literature.
4. Perform experimental analysis to determine how the representation of the agent's strategy changes how easy it is for the evolutionary algorithm to find strategies that give high payoffs, i.e. that play the game well.
5. Perform additional experiments to determine how robust the results are to changes in the game, e.g. Stag Hunt vs Prisoner's Dilemma.

1.2 Scope & Limitations

Within this project, there are various delimiters on quality and timeliness - the project will follow a set 8 month time period, and with this comes a few difficulties when approaching the implementation and documentation of the project. As this project attempts to identify the strengths and weaknesses of machine learning techniques, there has to be an clarification that the implementation operates on limited computational power. This means that certain approaches cannot be taken when implementing the theory of the literature review - numerous papers which are instrumental in the game theory field assume that the application will have infinite processing power. However, this limitation is identified as a research question, soon to come. Secondary to this, the time constraint unfortunately limits the amount of possiblities of further applications to other fields such as video games. While there is a definite possibility of application in this field, this report can only cover the theory behind it, instead of an application or visualisation.

1.3 Report Structure

The reader will see that the report structure is systemically approaching the final product of this project - the literature review will examine the ideas and motivation behind the project, as well as foundations for the planning behind the implementation. This includes the approaches that will work, and any limitations the implementation will encounter. Subsequently, the implementation will identify the programming language(s), packages/libraries and environment(s) required to approach the planning of the project, and then how the final program looks in execution. The results and evaluation section will then cover the strengths and possible downfalls of this product, what went right and what went wrong. This allows any reader to recreate the approaches of the project while avoiding any problems encountered in this section. This section will also contain the comparison of effectiveness between machine learning techniques, and foundations to conclude on which approach is better or worse. With this, the project can verify its integrity and robustness, then conclude and answer the research questions posed at the beginning, which can be seen in the following section.

2 Literature Review/Background

2.1 Brief History of Game Theory

Game theory is the study of behaviours and mathematical models which result from the decisions and strategies of two or more economically rational players in either cooperative or non-cooperative strategy games. Applications of game theory have manifested in social science, psychology, mathematics and many more fields of study; however, the root interactions lie in strategic games such as the prisoner's dilemma or tit-for-tat. Game theory was introduced and popularised by mathematician John von Neumann, who first proved an optimal strategy for zero-sum games with perfect information such as chess or go called the minimax theorem in 1928. This theorem indicates that in such games, there is a pair of strategies for each player which allows them to minimise their maximum losses, while considering all responsive moves of the opponent.

After von Neumann published his initial paper on game theory, he published a book co-authored by economist Oskar Morgenstein entitled, "Theory of Games and Economic Behaviour" [von Neumann et al., 1944]. Within this book, von Neumann fixates mainly on non-cooperative games and/or zero-sum games; but most importantly, identified a method of finding consistent solutions and strategies for both players in two-person zero-sum games. This work became a milestone for game theory as it established a foundation for becoming a unique discipline.

Following this, numerous advancements in game theory occurred during the 1950s - mathematicians Merrill Flood and Melvin Dresher experimented mathematical and game versions of the prisoner's dilemma for the American think tank corporation, RAND (Research and Development) [Herdt, 2003]. In the same year, John Forbes Nash Jr published his dissertation on non-cooperative games which contained the first definitions of the Nash equilibrium - an important milestone for adaptive strategy in game theory [Nash, 1950a]. He proved that in every n-player non-zero sum game, a Nash equilibrium existed, assuming the game had a finite number of actions. This was a continuation of the work from von Neumann and Morgenstein in their 1944 book, which only covered two person zero-sum games, and was restrained by the implications of 'rational' behaviour.

In 1980, political scientist Robert Axelrod set up a multi-agent tournament for the iterated/repeated prisoner's dilemma [Axelrod, 1980]. Multiple well-known game theorists from different professions such as psychology, political science, economics, mathematics and more submitted 14 FORTRAN (Formula Translation) programs for the agents to follow as implicit strategies.

In this tournament, agents would play against each other for 200 rounds - mutual cooperation would yield 3 points, mutual defection 1 point, single defection 5 points and single cooperation 0 points. The winning strategy was a simple tit-for-tat program which cooperated on the first turn, then repeated the opponents previous move for each subsequent turn. This strategy ended the tournament with an average of 504.5 points of a maximum 1000.

2.2 Prisoner's Dilemma

The prisoner's dilemma is one of the fundamental games of game theory which shows the payoffs and consequences of two 'players' acting in their own self interests. This summary, cited from britannica.com [Britannica,], is a model version of the prisoner's dilemma:

"*Two prisoners are accused of a crime. If one confesses and the other does not, the one who confesses will be released immediately and the other will spend 20 years in prison. If neither confesses, each will be held only a few months. If both confess, they will each be jailed 15 years. They cannot communicate with one another. Given that neither prisoner knows whether the other has confessed, it is in the self-interest of each to confess himself. Paradoxically, when each prisoner pursues his self-interest, both end up worse off than they would have been had they acted otherwise.*"

The first examples of the prisoner's dilemma being used in the context of game theory date back to the 1950s, by Merrill Flood and Melvin Dresher who devised puzzles and experiments using the structure of the dilemma - mainly an attempt to verify the usefulness of a non-cooperative Nash equilibrium. In this experiment, Flood and Dresher ran 100 games between two human players - in which player 1 (economist Armen Alchian) cooperated 68 times, while player 2 (mathematician John Williams) cooperated 78 times. In game theory, if a strategic game exists with the possibility for a various number of possible outcomes, a payoff matrix can be used to visually represent the benefits and consequences of each outcome. For the prisoner's dilemma, a typical payoff matrix would look as such:

As you can see, the prisoner's would achieve the best possible equal payoff if they consistently chose to confess, but a prisoner could achieve a higher payoff if they were to follow their own self interests. However, the payoff matrix in this experiment looked like this:

		PRISONER 2	
		Confess	Lie
PRISONER 1	Confess	-8 , -8	0 , -10
	Lie	-10 , 0	-1 , -1

Figure 1: Prisoner's Dilemma Payoff Matrix

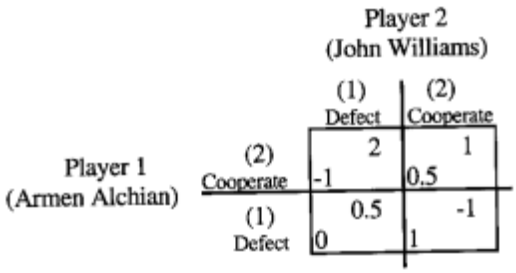


Figure 2: Flood-Dresher Experiment Payoff Matrix [Herdt, 2003, p. 184]

In the Flood-Dresher experiment, the restraints can be thought of as 'unfair' as human players have a level of empathy and other emotion which may sway their decision for reasons an A.I. program would never follow. Such an example would be the comments which player 1 made in their log of comments. Alchian, or player 1, wrote comments such as "He does not want to trick me. He is satisfied. I must teach him to share", while player 2 Williams wrote comments such as "A shiftless individual - opportunist, knave" [Herdt, 2003, p. 189] just a turn apart from each other. Many economists, game theorists and mathematicians believe that the results of this experiment may have been swayed slightly due to each player being empathetic or vindictive at numerous points in the game.

2.3 Nash Equilibrium

In 1950, John Forbes Nash Jr. published his dissertation entitled, "Non-cooperative Games" [Nash, 1950a]. Within this dissertation was proof which indicated that within a two person zero-sum game, there exists an 'equilibrium point' for both players. This equilibrium was described in the paper as

such, "Thus an equilibrium point is an n-tuple such that each player's mixed strategy maximizes his pay-off if the strategies of the others are held fixed. Thus each player's strategy is optimal against those of the others" [Nash, 1950a, p. 3]. Simply put, Nash was illustrating that within a two person game in which one player's benefit is a direct loss for the opponent, there lies a state in which neither player has any incentive to switch strategies, as it will not benefit their payoff - thus, the game sits at an equilibrium. The simplest, and most likely quickest way to prove the existence of a Nash equilibrium would be as follows:

	Left	Right
Up	0 , 0	4 , 1
Down	1 , 4	3 , 3

Figure 3: Example Matrix

Most proofs of equilibria exist if a certain number of conditions are met. Given a model payoff matrix, figure 3, our conditions for a pure/mixed strategy Nash equilibrium are as follows:

- The first player's best response is the same against any potential move of the opponent (red circle).
- The second player's best response is the same against any potential move of the opponent (blue circle).
- Nash equilibrium = (Up, Right),(Down, Left).

There are a few ways of proving the existence of a Nash equilibrium within games in a more detailed way - within Nash's dissertation, he chose to speak about the 'generalised' Kakutani fixed point theorem [Kakutani, 1941], and the Brouwer fixed point theorem. The Kakutani theorem is a more generalised proof of the Brouwer fixed point theorem, but is used to prove a Nash equilibrium in a very similar way. The conditions used in both theorems can be modified in such a way that you would prove the existence of an equilibrium state rather than a fixed point, within a set of strategies rather than tuples.

2.3.1 Applications of Game Theory

Beside its obvious magnitude in the fields of mathematics, Nash's work has had effects on fields such as computing, social science, psychology, and many more. Economists have used examples of the Nash equilibrium to calculate the prices of rival companies, predict prices of future products and calculate the best prices for supply and demand. A dissertation/report was published by the federal reserve bank of Minneapolis looking into why car insurance was so expensive in Philadelphia in the 90s, in which the writer chose to use a Nash equilibrium to demonstrate why the fluctuation of price was caused by the rivaling strategies of insurance providers. However, given the unpredictability of today's market, a Nash equilibrium may not have many uses outside of being a mathematical model in the field of economics. Another famous example would be the Cold War between the 40s and 90s - the USSR and the US were stuck in a long period of tension which could be seen as mutually assured destruction, in which each bloc knew the positions of the opponent but didn't start a war. This correlates exactly to a Nash equilibrium situation, where each side has no incentive to switch their strategy given the payoff. Aside from it's obvious applications in machine learning, game theory has appeared in subfields of computer science such as social networks, recommender systems and resource managers. Similarly, in the field of video games, game theory is becoming more prevalent - specifically in games where strategic decision making is involved, such as Firaxis Games' 'Civilization VI' or Ensemble Studios' 'Age of Empires'. In these games, it is entirely possible, and quite likely that players will end up in situations similar to the prisoner's dilemma or even a Nash equilibrium due to its resource management and turn-based system. In a 2-player game of Civilisation VI, both players have the possibility of knowing where the other's resources (soldiers/capitals/units) are, but have a mutual acceptance of not attacking each other (Cooperation). However, at any point in the game, either player could defect from this peace and attack the other in order to increase their payoff, maybe winning the game. In a field such as multi-agent systems, the prisoner's dilemma is a common occurrence when dealing with e-commerce situations such as auctions. For example, an auctioning site such as eBay has the issues of shill bidding and sniping taking over the market. The general process of an auction is a prisoner's dilemma - if the seller is viewed as the judge/prosecutor in the classic dilemma scenario and the buyer is viewed as the prisoner, there is a communication 'grey-area' in which neither side knows the true price of what is being sold, both have an incentive to overcorrect, leading to cooperative and defective choices.

2.4 Repeated Games

Repeated games, also known as iterated games or 'supergames' are either finitely or infinitely long games which repeat after finishing. These games are usually represented in extensive form, meaning each strategy and/or game is mapped out as a tree, with specific time-stamps for each game. Payoffs are included at the end of each branch. The main application and usefulness of repeated games is to examine how economically rational players may behave differently from game to game depending on previous strategies or moves. Arguably the most important instance of repeated games is Robert Axelrod's multi-agent tournament in 1980. This tournament was a simple 200 round prisoner's dilemma, in which mutual cooperation scored 3 points, mutual defection 1 point, single defection 5 points and single cooperation 0 points - with a 200 round maximum of 1000 points. Well known game theorists from multiple professions such as psychology, political science, economics, mathematics and sociology submitted FORTRAN (Formula Translation) programs which the agents would follow as strategies. The winning strategy was submitted by Professor Anatol Rapoport, which was a simple tit-for-tat program in which the agent would start with a cooperative choice, then mimic the opponent's choice on the previous turn. According to Axelrod in his primer, "This decision rule is probably the most widely known and most discussed rule for playing the Prisoner's Dilemma. It is easily understood and easily programmed" [Axelrod, 1980, p. 7]. Interestingly, each participant of the tournament was made aware of the properties of the preliminary tournament, and thus, many of them made tit-for-tat programs which they tried to improve upon; but, the original and simple tit-for-tat program ended up performing better than the modified versions. In relation to this project, FORTRAN would not be applicable/feasible to use, given its dependency on supervision (previous data or knowledge). Essentially, this means that it is incredibly difficult to evolve individual, high-level lines of code from non-functional languages such as FORTRAN, Java, C# and so on. Evolutionary algorithms essentially make random changes - thus, if this was replicated with random changes to lines of code in an imperative language, the program would undoubtedly become unstable or not work. The same can be said for any attempts to 'evolve' lines of code like an evolutionary algorithm. As a subsitute to this, the project will use neural networks and finite state machines to evolve a strategy, as both methods are reliable, efficient and have the capability to easily 'evolve'.
Within the tournament there were 3 strategies which were expected and

known by the participants in advance - always defect, always cooperate, and random. A strategy in which the agent always defects is the safest strategy of any, and could be seen as a principle of game theory. However, although such a strategy is safe, there is a low chance of it being the best strategy due to it's 'no risk, no reward' drawback. The 'always cooperate' strategy performs well when matched against itself - as you can expect a maximum payoff, but when matched against a defecting opponent, there comes a minimum payoff. The 'random' strategy is simply cooperating 50% of the time, in an attempt to reap the benefits of both strategies. This strategy is more of a utility for making sure the opponent's strategy accounts for all possibilities, but in the end the random strategy didn't perform well. Scores for these strategies can be seen in figure 4.

TABLE 2
Tournament Scores

Player \ *Other Players*	TIT FOR TAT	TIDEMAN AND CHIERUZZI	NYDEGGER	GROFMAN	SHUBIK	STEIN AND RAPOPORT	FRIEDMAN	DAVIS	GRAASKAMP	DOWNING	FELD	JOSS	TULLOCK	(Name Withheld)	RANDOM	*Average Score*
1. TIT FOR TAT (Anatol Rapoport)	600	595	600	600	600	595	600	600	597	597	280	225	279	359	441	504
2. TIDEMAN AND CHIERUZZI	600	596	600	601	600	596	600	600	310	601	271	213	291	455	573	500
3. NYDEGGER	600	595	600	600	600	595	600	600	433	158	354	374	347	368	464	486
4. GROFMAN	600	595	600	600	600	594	600	600	376	309	280	236	305	426	507	482
5. SHUBIK	600	595	600	600	600	595	600	600	348	271	274	272	265	448	543	481
6. STEIN AND RAPOPORT	600	596	600	602	600	596	600	600	319	200	252	249	280	480	592	478
7. FRIEDMAN	600	595	600	600	600	595	600	600	307	207	235	213	263	489	598	473
8. DAVIS	600	595	600	600	600	595	600	600	307	194	238	247	253	450	598	472
9. GRAASKAMP	597	305	462	375	348	314	302	302	588	625	268	238	274	466	548	401
10. DOWNING	597	591	398	289	261	215	202	239	555	202	436	540	243	487	604	391
11. FELD	285	272	426	286	297	255	235	239	274	704	246	236	272	420	467	328
12. JOSS	230	214	409	237	286	254	213	252	244	634	236	224	273	390	469	304
13. TULLOCK	284	287	415	293	318	271	243	229	278	193	271	260	273	416	478	301
14. (Name Withheld)	362	231	397	273	230	149	133	173	187	133	317	366	345	413	526	282
15. RANDOM	442	142	407	313	219	141	108	137	189	102	360	416	419	300	450	276

Figure 4: Scores from the Axelrod Tournament [Axelrod, 1980, p. 11]

2.4.1 Folk Theorem

Folk theorem is used within repeated games to show that a Nash equilibrium outcome in a game which is repeated infinitely is quantitatively and qualitatively equal and rational to that of a single game. While the origin of this theorem is unknown, it appeared in the 1950s and was quickly spread through the game theory field - thus the name, Folk Theorem. The first instance of a research paper to use the theorem was authored by James W. Friedman (1971) in his article, "Non-cooperative Equilibrium for Supergames" [Friedman, 1971], in which he details the payoffs of subgame-perfect [1] Nash equilibria in an infinitely repeated game, instead of using a single Nash equilibrium.

[1] A subgame-perfect equilibrium means that if players were playing a smaller game which was part of a bigger game, their behaviour would represent a Nash equilibrium of

This means that within a game such as the prisoner's dilemma, where mutual defection is a Nash equilibrium, folk theorem allows the possibility of a non-defection Nash equilibria in infinitely repeated games. Another important concept of folk theorem is that of duopolies and oligopolies; in an economic circumstance, a duopoly is a point in which two suppliers own all or nearly all of the market for a product or service. An oligopoly is the same, except the number of suppliers is more than 2 but remains a small number. When applied to the prisoner's dilemma, any choice other than mutual defection is unstable - however, if the games are infinitely repeated, there exists a possibility that one player may 'threaten' the other player to defect, in which case they would always play defect from that point onwards. In such a situation, if the second player is aware of this threat, they may choose to collude with their opponent and play cooperate, assuming there is a beneficial payoff guaranteed. In both economics and game theory, you can see a riskier, higher payoff as 'discounted' when colluding. Given that scenarios using the folk theorem are infinitely repeated, there are possibities of replicating the results with an indefinitely-long game, as long as there was a high chance of success that an agent would follow the same strategy. As a final thought in reference to Axelrod's tournament, it is confusing that Axelrod didn't consider or even mention folk theorem in his findings. Given the structure of the tournament, there are definite foundations for subgame Nash equilibria, which would've made for interesting comments from Axelrod on the value of specific equilbria at various points of the tournament.

2.5 Machine Learning

Machine learning is a subfield of articifical intelligence which combines pattern recognition and computational learning theory, an idea pioneered by Alan Turing in 1950, and developed by Arthur Samuel in 1959 through his paper, "Some Studies in Machine Learning Using the Game of Checkers" [Samuel, 1959].[2] The main goal of machine learning is for algorithms to become 'smarter' on each iteration of instructions, such as a move in a game of checkers, to then make predictions based on the data it has constructed. Within Samuel's introduction to his paper, he states, "The studies reported here have been concerned with the programming of a digital computer to behave in a way which, if done by human beings or animals, would be described

that smaller game.

[2]While some may argue that Marvin Minsky pioneered the first instance of a self-learning machine in 1951, many still question whether or not this project was artificial intelligence, given the amount of missing information.

as involving the process of learning" (Samuel, 1959, p. 1). Samuel's checkers algorithm used a search tree to identify each of the board positions reachable from the available pieces[3], which would feed into a scoring algorithm - incorporating von Neumann's minimax strategy to choose the best move. The result of this algorithm, through rigorous testing, was a piece of artificial intelligence which could, "greatly outperform an average person", and was envisioned to be economically viable in real-life situations/problems. This is the first instance of an algorithm which has developed itself without being given information directly. Since the 1950s, machine learning has become almost ubiquitous; in Pat Langley's paper, "Applications of Machine Learning and Rule Induction", he shows applications of machine learning in over 15 different fields - such as economics, insurance, astronomy and more. Given the two main methods of learning, supervised and unsupervised, only unsupervised learning is applicable to this project. Supervised learning requires prior data in order to predict patterns; Instead, representations of cognitions will be used, as explained in the following section(s). The algorithm will eventually learn and get better with each generation, which is unsupervised.

2.6 Evolutionary Algorithms

Evolutionary algorithms, or evolutionary computation, is a facet of artificial intelligence in which an algorithm filteres through a set of data, removing the least fit values within a specified iteration, while keeping the most fit values until better values are found. The first appearance of a theory for automated problem solving originated in the 1950s, while application for said theory was developed in the following decade by Lawrence J. Fogel. Fogel devised the idea of evolutionary programming while working for the National Science Foundation, when he saw the approach to heuristic algorithms and primitive neural networks simulations as limited. He theorised that in order for artificial intelligence to progress, the approaches to simulating behaviour should be focused on evolution and increasing intellect rather than model human behaviour. Primarily, "Fogel considered intelligence to be based on adapting behavior to meet goals in a range of environments" (De Jong, 2014, p. 2). This meant that simulated experiments using a finite state machine could be used to investigate intelligent behaviours in a range of situations, with certain rules in consideration. Upon experimenting and publishing multiple papers on the topic, Fogel described, at a very general level, behaviour as the composite ability to predict one's environment, and thus adapt/respond to it

[3]Samuel described this process as 'looking ahead a few moves' like a human player might do.

suitably. According to Kenneth De Jong of George Mason University, Fogel made a similar proposal for a finite-state machine to replicate this behaviour to that which follows:

"*A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time*[4]*. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g. all–none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g. average payoff per symbol) indicates the fitness of the machine.*"

The general process involved picking a 'best' or best-suited machine to predict new symbols in the environment, and then the process is repeated until the payoffs can't increase any further, relative to the accuracy (or worth) of their previous predictions. Further work on this research in fields such as sequence prediction, pattern recognition and gaming was conducted by academics such as Bernard Lutter and Ralph Huntsinger (1968), Akihiro Takeuchi (1980) and many others.

Since these first occurrences of evolutionary programming, many strides have been made in the field - taking it in a number of directions such as neural network training, image processing and general computing optimisation. Within the facet of neural networks, foundational and important work comes from researchers such as Peter Angeline in 1994 with his paper on evolutionary algorithms and neural networks, J.R. McDonnell and D. E. Waagen's paper on evolving recurrent perceptrons, and Vincent Porto's paper on alternative neural network training methods. In Angeline's paper, he explains why the previous methods of constructing/modifying neural networks are limited - Timur Ash authored a paper on dynamic node creation, but Angeline outlines that only feedforward networks work in application (which isn't effective as a standardised method).[5] Another researcher of neural networks, Scott Fahlman, authored a paper explaining a recurrent version of the Cascade-Correlation learning architecture in the context of neural networks, but assumes a restricted form of reccurence according to Angeline, which

[4]For the sake of generality, the environment was described as a sequence of symbols taken from a finite alphabet.

[5]A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle.

limits the types of input/output, and is generally less robust. A version of the cascade-correlation learning architecture can be seen as follows:

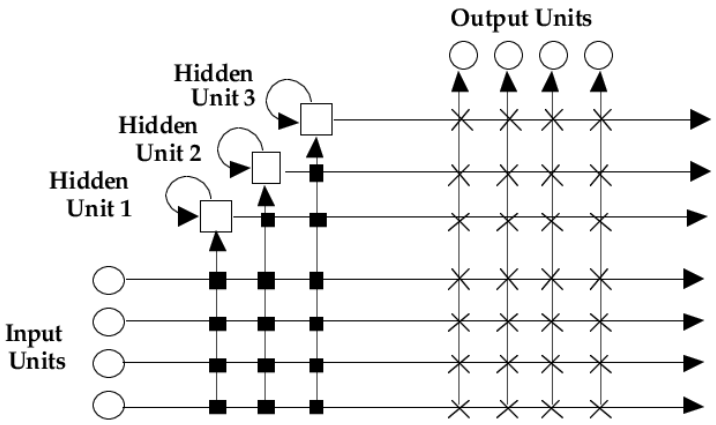


Figure 5: Cascade Correlation Learning Architecture

Finally, Dong Chen authored a paper on the constructive learning of recurrent neural networks with the help of C.L. Giles, G.Z. Suna, H.H. Chen, Y.C. Lee and M.W. Goudreaub. However, Angeline explains that the paper only explores fully connected topologies (in which all nodes are connected), without accounting for any other possibilites. Considering these limitations, Angeline presents GNARL (GeNeralized Acquisition of Recurrent Links), "a network induction algorithm that simultaneously acquires both network topology and weight values while making minimal architectural restrictions and avoiding structural hill climbing" (Angeline, 1996, p.2). This is essentially an evolutionary algorithm which is 'better suited' for evolving neural networks as opposed to genetic algorithms, as it has the boone of function optimisation and the ability to fit a variety of tasks/problems. Angeline then goes on to detail the process of evolving neural networks with both genetic and evolutionary algorithms, and eventually his own 'GNARL' algorithm, which "nonmonotonically constructs recurrent networks to solve a given task". Essentially, this algorithm decides the number of input and output nodes relative to the task it has been given, and the number of hidden nodes can range from 0 to whatever the user specifies - avoiding the limitations of the previous work mentioned before.

Following this work, one of the next milestones to occur in the field of evolutionary algorithms was the work of J. R. McDonnell and D. E. Waagen. This

paper was centered around providing an alternative to feedforward networks for 'nonlinear models of time-series data', by creating the s imple parts of evolutionary algorithms and neural networks in a more efficient and effective way, so that they can be applied to more complex structures or networks. Specifically, the authors decided that, "Once feasibility is demonstrated for simple recurrent perceptron structures, the evolutionary search method can be applied to highly recurrent perceptron networks with complex architectures" (McDonnell; Waagen, 1994, p. 1). This is an interesting way to look at this topic as it can be applied to a large number of tasks/problems, given that the method doesn't address a specific topology - such as stochastic, recurrent or hierarchical. Pseudocode for a generic evolutionary algorithm can be seen as follows:

Algorithm 1 Basic Evolutionary Algorithm

```
1: procedure GeneticAlgorithm(S - set of blocks)
2:     Initialisation()
3:     t ← 0
4:     Initialise P_t with random individuals from S*
5: Evaluate-Fitness-GA(S, P_t):
6:     while termination condition not met do
7:         Select values from P_t (fitness proportionate)
8:         Recombine individuals
9:         Mutate individuals
10:        Evaluate-Fitness-GA(S, modified individuals)
11:        P_{t+1} ← newly created values
12:        t ← t + 1
   return (Superstring derived from best individual in P_t)
13:
14: Evaluate-Fitness-GA(S - set of blocks, P - population of individuals):
15:    foreach individual i ∈ P do
16:    generate derived string s(i)
17:    m ← all blocks from S that are not covered by s(i)
18:    s'(i) ← concatenation of s(i) and m
19:    fitness(i) ← 1 / ||s'(i)||^2
```

2.6.1 Artificial Neural Networks

Artificial neural networks (ANN) are information processing systems based on biological nervous systems such as the brain. Neural networks are a

subfield of machine learning as they make decisions and perform tasks based on previous information they have been given, called training examples. Each system has three layers - input, output, and a layer for filtering the input to something which can be used by the output layer. This is possible due to an algorithm called backpropagation (backward propagation of errors), which calculates the gradient of the error function[6] with respect to the weights of the network. When designing a neural network, there are various paradigms or methods which can be used - only one of which is applicable for this project: control tasks. The other being classification, but this requires labeled training data to train the network, which this project will not have. Backpropagation and supervised learning also isn't valid/feasible within this project, as there is no prior training data to filter/modify to the output layer. Control tasks are reward-based behaviours which can train the decision making process of an evolutionary algorithm. With a neural network being the controller in question, a control task would reward correct behaviour as a method of training the network. Classification is the process of grouping similar values within the network based on their attributes/characteristics. Assuming the possbility of a 'noisy' data set at any point during the program lifecycle, this method would allow the network to classify patterns from data which they have not yet been fed/trained.

2.6.2 Finite State Machines

Finite state machines, or finite state automaton, are computational models used to simulate sequential logic or solve problems relating to software architecture. At a very basic level, computers can be seen as state machines; each instruction that a computer receives and execute will change it's state in some way, altering the behaviour and causing subsequent actions or allowing further instructions to occur. Arguably the pioneers of finite state machines, the first proposal and description of finite automata came from neurophysiologists Warren McCulloch and Walter Pitts in 1943 with their paper, "A Logical Calculus Immanent in Nervous Activity". Within this paper, McCulloch and Pitts comprehensively cover topics such as neural network theory, the theory of automata and the theory of computation and cybernetics. Around 10 years later, the first implementation of the finite state machine appeared, created by computer scientists G. H. Mealy and E. F. Moore, with their Moore and Mealy machines. The Mealy machine is focused on determining ouput through the input and current state, while the

[6]The Gauss error function is the integral of the standard normal distribution.

Moore machine bases the output on the current state alone. Another notable paper which covers finite state machines is Matthijs van Veelen and Julián García's paper, "Direct reciprocity in structured populations". Within this paper, van Veelen and García tackle the topic of finite state automata within an 'open-ended, infinite strategy space'. This topic encompasses that, "Our simulations contain a mutation procedure that guarantees that every finite state automaton can be reached from every other finite state automaton through a sequence of mutations. Thus, every strategy that can be encoded by a finite state automaton is a possible mutant" (van Veelen; García, 2012, p. 9929). This is done because the state machines that require fewer mutations are essentially the best fit. Nowadays, one of the most significant applications of finite state machines is within video games; games ranging from basic like Pacman to incredibly detailed such as Horizon: Zero Dawn (H:ZD) both use finite state machine for their AI agents. In a game such as H:ZD, creatures will start in a state such as 'idle' or 'roaming' to simulate random animal behaviour, but will switch behaviours to something like 'hunt' once a player is spotted. Other important applications of finite state machines include things such as pattern searching in text editors or IDEs, vending machines, or even system/software modelling.

2.7 Research Questions

Although a lot of content has been created through many years of research into game theory, machine learning and evolutionary algorithms, there are still a lot questions yet to be answered. In this project, I aim to tackle some of these unanswered questions, with topics such as these:

1. Given the results of research papers in the past such as Nash's, Harrald's or Axelrod's, how do those results look now when used within a model of more realistic computational power?
2. When using different representations of an agents' cognition, how does the payoff vary between evolved strategies from neural networks and finite state machines? How easy is it to evolve those representations at a high payoff?
3. Within an evolutionary algorithm, which neural network topologies are favoured in 2-player games when a neural network is being used as a representation of the agents' cognition(s)?
4. During this project, neural networks will be used with a fixed shape and payoff matrix. If it is allowed to evolve, how does the shape of the

neural network change? (Optional)

2.8 Conclusion

In summary, we have analysed multiple facets of fields such as game theory, machine learning, neural networks and evolutionary algorithms in terms of their history, use and sometimes applicability to video games and other forms of media. In all cases, each field has some form of possibility for being in video games - machine learning, neural networks and evolutionary algorithms can all be applied to AI agents to improve their realism and general behaviour. Papers from researchers such as Nash, Harrald and Axelrod have avenues of further research, and while there are continuations of said work, there is yet to be solid proof of applications similar to this project. The results of this project will touch on applications to specific facets of video games and possibly computer science which may not have been explored before. Research for this project arguably starts with Nash's paper in the 50s - the existence of the Nash equilibrium was the driving force behind this project, which eventually led me to read more into game theory. This led to papers from researchers such as Axelrod and Harrald; work from these reports was instrumental in aiding my understanding game theory from a theoretical standpoint, but not necessarily from an application standpoint. The next step was to read work from Kakutani and Brouwer, then Flood and Dresher. While the Kakutani and Brouwer papers were helpful in understanding the processes behind finding a Nash equilibrium, the Flood-Dresher paper was helpful in understanding how game theory scenarios can be applied to real life situations.

With this knowledge of game theory and the Nash equilibrium in mind, a necessary step was to find ways to apply it in application. The first works to read were that of Fogel, Waagen and Porto. These papers comprehensively covered neural networks, such as their creation, evolution and learning methods. While helpful, it was the later papers from Fogel and works from researchers such as Chen, Ash and Fahlman that solidified my knowledge on the operation of neural networks. However, what was lacking was a comparitive component. Very little, if any, papers did a comprehensive comparison of methods for game theory application, so it seemed beneficial to tackle that exact topic. With a topic in mind, I came across papers from Angeline, van Veelen and Jong which were extremely useful for learning the history and applications of both machine learning and evolutionary algorithms - another possible method of game theory application. I had to find more papers to read to make sure that a comparison of finite state machines and neural net-

works within evolutionary algorithms was a fair avenue of research, which led me to papers from Samuel, Lutter and Huntsinger, and Langley. This finalised my understanding of how to approach the project, as well as possible research questions, which you can see above. This project will be an amalgam of the knowledge gained from all of these papers, in an effort to tackle the questions which previous papers did not ask or answer.

3 Methodology

As discussed previously, a working knowledge of game theory, machine learning and evolutionary algorithms is required to implement the base dilemmata of classic game theory scenarios. In the following section, the required knowledge will be comprehensively examined - such as game theory terms, associated technologies and operations of artificial intelligence architectures.

3.1 Approach

To have a robust and intensive implementation of the proposed ideas, a selection of milestones and criteria must be met to execute the goals of the project. Before attempting said implementation, the end result of the project must be abstracted into clearer and easier-to-understand points which will be clarified as follows. First, in order to simulate a game theory scenario, a program must be created which can support two agents, represented as both neural networks or finite state machines - playing against each other in a game determined through a payoff matrix as discussed previously. Following this, an evolutionary algorithm must be used to progress the game and evolve the agents through summation of payoffs and fitness functions. Assuming these facets are working as intended, the program should now be able to play a repeated prisoner's dilemma game, all the while evolving the agents to become better/fitter as the game progresses. The parameters associated with the neural networks/finite state machines and the evolutionary algorithm should be modifiable to recreate results of research papers conducted previously, such as, "Evolving continuous behaviors in the Iterated Prisoner's Dilemma" by Harrald in 1996. With this done, the topic of the dissertation can be tackled - assuming there are multiple working versions of the program discussed previously; using both finite state machines and neural networks as representations of the agents' cognitions, both methods can be compared and evaluated based on their use of computation power[7] and difficulty. The remainder of this section will cover all of the knowledge required to understand these milestones - such as the general terminology used, program structure and explanation of milestones.

[7]Computation power is referring to the amount of generations it takes to evolve the agents.

3.2 Technologies

In order to execute these milestones, a suitable method of application must be identfied. For this project, the implementation of the proposed ideas will use the Python programming language, which will be touched on in a following subsection. Multiple integrated development environments (IDE) exist for Python such as PyCharm, created by the Czech company, Jetbrains, and Spyder (Scientific PYthon Development EnviRonment), created by Pierre Raybaut. This project will be executed using Spyder, given its overall suitability for theoretical and scientific programming.

3.2.1 Python

The forementioned programming language, Python, is a high-level, interpreted language developed by Guido van Rossum in 1991 which can utilise multiple programming architectures such as the object-oriented, imperative, functional and procedural paradigms. Given its overall versatility, general abstraction and library support, Python seems more suited than other languages such as C++ or Java for the task(s) at hand. To aid the process of implementation, the NEAT library will be used to handle network creation and management. Python is generally more suited for this project as the library support and level of abstraction makes the language easier to use and easier to read. In other languages such as C++ or Java, the prisoners in this scenario would most likely have to be represented as objects - which involves a whole other layer of difficulty in the context of this task. Ideally, the code should be managed and maintained to a point where it should be easily understand solely from the information given in these sections.

3.2.2 NEAT

According to their official website, "NEAT (NeuroEvolution of Augmenting Topologies) is an evolutionary algorithm that creates artificial neural networks". NEAT was developed by Kenneth O. Stanley in 2001 and has been upheld since 2015 by the group, CodeReclaimers. The purpose of NEAT in context of this project is to initialise a population of agents, contain and handle the attributes associated with them and control the speciation between rounds of the game. In some games such as the prisoner's dilemma, speciation is not applicable and will be turned off. Speciation is the process of splitting up agents into specfiic groups, known as species, by their related

attributes. If allowed in this context, there is a possibiltiy that agents will only play against similar strategies or agents of similar fitness repeatedly. NEAT also has the functionality of visualising various facets of the game, such as population size per generation, or fitness per generation. NEAT was chosen for this project due to the fact that its functionality in neural networks and evolutionary algorithms makes the implementation facet of the project much less prone to errors, as well as making the code much easier to read and manageable. If the project had it's own implementation of neural networks, there is a high chance that the project milestones would take a larger amount of time to accomplish, and the results are much more likely to be incorrect or inaccurate.

3.2.3 Transitions

In order to create and manage a finite state machine implementation, the 'transitions' [Neumann,] package by Alexander Neumann was used. According to their GitHub page, transitions is, "a lightweight, object-oriented state machine implementation in Python". This package was integral in the finite state machine method of representing agents, as will be explained in the following section(s). The reason this package was used specifially as opposed to others is due to it's simplicity and level of functions. While easy to use and very readable, transitions accommodates a large amount of functionality for finite state machines, which proved useful when making a prisoner's dilemma scenario with an evolutionary algorithm.

3.2.4 Python Development Environments

Integrated development environments (IDE) are software applications which aid program creation, offering tools such as a source code editor, build automation tools, and a debugger. Programs can be compiled, tested and executed by simply clicking a run button, as opposed to compiling the code manually through the command line, respective of your operating system. In relation to Python specifically, a number of open-source (free) IDEs exist. Within these are PyCharm, PyDev, Spyder and many others. Each IDE differs in what it offers, usually suiting a particular type of task. PyCharm is specifically aimed at new Python developers, offering easy Python installations and useful documentation pages on various facets of the program. Conversely, Spyder is aimed at adept Python developers, offering a number of scientific libraries in the base installation package, as well as easy library

inclusion for advanced functions. With that in mind, Spyder was justifiably the best platform for the project development.

3.3 Terminology

When discussing the program in following sections, a range of different technical terms will be used - some of which have been used but not explained in previous sections.

3.3.1 Neural Network

Initially pitched by Warren McCullough and Walter Pitts in 1944, artificial neural networks are computational models which are modeled loosely on the human brain, with interlinked 'nodes', which behave similar to that of neurons/perceptrons in a human brain. Neural networks can have differing types, such as recurrent or feed-forward, which will be explained shortly. While a biological neural network can have millions of nodes, it can also have a large amount of layers - varying in size between input, hidden, and output. Simply put, the input layer(s) will be fed information from other sources, or the user. The hidden layer(s) will attempt to filter and refine that input to output. The output layer(s) give out the information which was previously fed in.

3.3.2 Neural Network Types

There exists two types of neural networks which can be used in the context of this project - recurrent and feedforward. The difference between the two networks discerns their applicability; feedforward networks strictly allow signals to only travel one way, from input to output. Feedforward networks are useful and extensively used in speech and pattern recognition. Recurrent networks, in contrast, allow signals to travel in both directions - introducing loops to the network. A loop means that the output of any layer cannot affect that same layer. Recurrent networks are specifically useful for identifying how the input variable(s) can affect the output variable - this relates to the project as there has to be multiple moves from the agents/prisoners (inputs) and the resulting payoff will be the output variable.

3.3.3 Weights

For each incoming connection, a node/neuron will assign a number known as a weight, which represents the strength of connections between nodes. When the neural network is active, each node receives a number for each of it's connectons, which is then multiplied by the associated weight. By summing the resulting products together, this creates a single weighted number. Depending on the activation function of the network, the number may be passed to the next layer regardless if it is above or below a specific threshold value. For example, if the activation function is sigmoid, a value which is lower than the threshold may be passed to the next layer, but wouldn't be passed for another activation function. This passed value is the sum of weighted inputs for the node, and the corresponding node is activated if the value is passed to the next layer. Assuming a node A, has a strong weight to node B, this means that A has a stronger influence over B - strengthening or weakening the level of activation.

3.3.4 Activation Functions

In the context of neural networks, an activation function is the deciding factor in whether a neuron should be activated or not. It does this by calculating a weighted sum of it's input, and adding a bias value to it. By doing this, the output of said can be seen as non-linear. Non-linearity means that the output cannot be reproduced from a linear combination of the inputs. Without a non-linear activation function in the network, a neural network would behave just like a single-layer perceptron regardless of the amount of layers.

3.3.5 Bias

When using activation functions, a bias value is added for the function to better fit the data. The bias value can range from 0 to 1, and can offset the sigmoid curve between the input and output values. The bias only interacts with the output values, and thus, determines how easy/likely it is for a node to be activated. If the bias value is high, the output of the network will tend to be higher, even if the weights and inputs have smaller values. If the bias is smaller, or 0, the output value will be decided solely by the weights and inputs.

3.3.6 Aggregation

When collating the input and weight values to filter into input, there needs to be a way to gather input from the previous neurons. This process is called an aggregation function, and is used in almost every instance of a neural network. In this project, the aggregation function used is called 'sum of product', which sums the weights and the input values, in addition to the bias.

3.3.7 Elitism

When a population is moving into the next generation, elitism is a value which copies a certain amount of individuals into the next generation without any changes. Meaning, if a genetic algorithm uses an elitism value of two, then two individuals will pass into the next generation with no changes to their attributes - this is usually determined by the individuals with the highest fitness values.

3.3.8 Agents

According to Stuart Russell and Peter Norvig in their textbook, "Artificial Intelligence: A Mordern Approach", an agent is, "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators". This means that a software agent may utilise keystrokes, file contents or network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets. In the prisoner's dilemma scenario used within this project, an agent will represent each prisoner, in the effort to emulate a real world scenario in which the agents play against each other. Each agent/prisoner will have a fitness, which could be compared to a real-world aptitude or ability. As seen in section 2.4, the resulting agents from a repeated prisoner's dilemma can very much vary - Robert Axelrod detailed four different characteristics [Axelrod, 1980] in order to be successful in a repeated prisoner's dilemma:

- Be nice, cooperate and never be the first to defect. It also paid off to be forgiving.
- Be provocable - many of the competitors defected early in the game without provocation, a characteristic which was very costly in the long

run.

- Don't be envious - in the short run, neither side can benefit itself with a selfish choice enough to make up for the harm done to it from a selfish choice by the other.
- Have clarity, or don't be too clever. Many of the competitors did less well in an environment in which their defections were usually echoed back at them.

Axelrod found that very basic strategies such as tit-for-tat were successful in the tournament. The forementioned attributes played a large role in being successful, even though the programs operate under economic rationality.

3.3.9 Evolutionary Algorithm

Evolutionary algorithms are heuristic-based approaches to solving complex problems that cannot be easily solved otherwise. Just as neural networks model the brain, evolutionary algorithms model evolutionary concepts such as reproduction, mutation and selection. First, an evolutionary algorithm will initialise a population of members (in the context of this project, **agents** are initialised). The initial population can have the same or differing attributes, dependent on the particular type of problem which is being solved. After initialisation and attempts at a solution, the members are evaluated by a fitness function, which will be explained shortly. Based on the output of the fitness function, particular members of the current generation, or species, are taken into the next, while the rest become 'extinct'. Children of the parent members may be created, with attributes from each of the selected parents. These processes are called crossover and mutation. By doing this for each generation, ideally the evolutionary algorithm will refine the members of the population to solve the posed problem(s) more quickly and efficiently.

3.3.10 Fitness Function

Within an evolutionary algorithm, a fitness function will calculate the quality of a solution to a given problem. The algorithm will find the best input value for the fitness function, and select with of the solutions are the most adequate for the scenario. In this case, the fitness function will sum the payoffs of each agent for every round, and use the eventual value as the new

fitness. The results will detail what moves the agents have chosen on average. However, what defines the fittest agent, and how can it be chosen? Assuming the population of agents all have mixed strategies, the 'fittest' agent could be seen as a loose term without much proof of being correct. However, by pitting an agent against itself and the rest of the population in a prisoner's dilemma scenario, the fittest agent will be a fair representative of the whole population, as they have played against each possible strategy. The fitness function used in this project is as follows:

$$F = F + (P1 \parallel P2)$$

Where F is the fitness of an agent, and P1/P2 is the payoff for the agent and opponent's move respectively.

3.3.11 Finite State Machine

A finite state machine, or finite automata, is a computational model which is widely used to simulate sequential logic, based on ideas such as a system changing state due to its inputs.

3.3.12 States

A state is a representation of the finite state machine's current status. Each state will have a transition into another state, such that a collection of states can become a fully working, possibly cyclical system.

3.3.13 Computational Power

A myriad of computing journals, papers and academic reports mention computational power in the context of their research, including this report. Computational power can have multiple definitions in other contexts, but within this context, it refers to the amount of generations required to evolve an agent successfully. This is tied to the general definition of the term, which is, "the amount of useful work a computer system accomplishes (i.e. instructions it executes)".

3.3.14 Speciation

According to Timothy Nodine in his paper, "Speciation in NEAT", [Nodine, 2010] speciation is the set of processes by which NEAT creates, maintains and uses several disjoint groups of similar genomes for guiding reproduction. This means that given a species, A, of an initial population, NEAT will create new subspecies, B, C, D, etc. each generation based on the similar attributes of the agents/genomes. This is useful for examining the internal behaviour between agents, as well as refining the eventual output of the prisoner's dilemma simulation. Speciation is used to diversify the search space of the population, such that results are not converged to too quickly.

4 Implementation and Testing

4.1 Introduction

In order to comprehensively examine the differences between neural networks and finite state machines, two versions of the project had to be created, one with each representation of an agent, respectively. This section will accommodate both methods, and will attempt to talk about each in equal amounts. Additionally, a comparison to the plans in the methodology will be included, such as what went right, and what went wrong.

4.2 Neural Networks Method

The first attempt at creating a prisoner's dilemma was created using neural networks to represent the agents. In order to correctly execute this, a handful of requirements had to be met, such as:

- Concept of agents as prisoners.
- Attributes for the prisoners, such as a unique ID and fitness.
- Concept of dilemma, being able to select a move (Cooperate/Defect) and receive a payoff.
- Basic evolutionary algorithm capable of initialising a population and handling multiple generations.
- Interaction between agents.
- Extending the dilemma to facilitate play between two agents, as well as giving payoffs accordingly This includes each agent being able to independently select a move, receive a payoff and adapt their strategy accordingly for the next round/generation.
- Output and plot the attributes of each generation.

While the program shows other pieces of code associated to this list, this is a general overview of how the program is structured.

4.2.1 Initialisation of Agents

The first part of implementation was initialising the agents. Thankfully, NEAT aids the initialisation process a lot - alongside the main Python file containing the code for the networks and the evolutionary algorithm, there exists a configuration file containing all of the attributes relating to the networks. These attributes include things such as activation function control, bias control, compatibility control and more. The main attributes worth mentioning are as follows:

- Activation Function - Sigmoid
- Aggregation - Sum
- Elitism - 2

A lot of the contained/default attributes were already suited for this type of project, but allowed for a wide range of results. Outside of the configuration file, a few attributes were required in order to facilitate functional behaviour and fair play. The agents had to be kept in some form of list to facilitate randomised play. However, NEAT has no way of identifying networks from one another; So, a list of agent IDs were created and attached to each individual network for the evolutionary algorithm to access the agents directly.

4.2.2 Agent Strategies

In order to give the agents economically realistic behaviour, there has to be an incentive to choose particular actions, or incentives to aim for a particular outcome. While this can be achieved, or aided, by the attributes contained within the configuration file, a history of moves has to be maintained in order to extract a strategy from the agents' previous behaviours. The activation function can then use this collection of moves with an iterator to determine a strategy to follow. But what can incentivise an agent to follow any particular strategy?

4.2.3 Incentivisation and Payoff

To evolve a strategy, an agent must have incentive to change - without it, the game will see fairly random play, or the game reaches a Nash equilibrium. In

the prisoner's dilemma, the prisoners are aiming for the best possible payoff from their actions; this is easily replicable by giving the agents an integer attribute to contain their fitness. By directly correlating payoff to fitness, the agents are incentived to aim for the highest payoffs as it dictates their position in the population. The pseudocode for such a concept looks as follows:

Algorithm 2 Payoff Calculation

```
1: procedure CALCULATE_PAYOFF(A1A - AGENT ONE ACTION, A2A
   - AGENT TWO ACTION)
2:     if (A1A == cooperate and A2A == defect) then
3:         return 0
4:     if (A1A == defect and A2A == defect) then
5:         return 2
6:     if (A1A == cooperate and A2A == cooperate) then
7:         return 3
8:     if (A1A == defect and A2A == cooperate) then
9:         return 5
```

Although the code is extremely simplified, this is all that is required to incentivise an agent in a prisoner's dilemma scenario. To explain, the method itself takes in two arguments - the actions/moves of agents one and two. The method itself directly returns a payoff which can be accumulated onto an agent's fitness - for a population of 50, where 2,500 rounds of the prisoner's dilemma are played, an agent's fitness will rapidly increase for a successful strategy, or will stay average for an unsuccessful strategy. The different findings of fitness based on strategy will be shown in the following section. The fitness function itself, which can be seen almost directly as the incentive for the prisoner's dilemma, undeniably has the largest effect on the results of the program.

4.2.4 Visualisation of Agent Performance

A key factor in judging the performance of the population, as well as the program, was visualisation. The general process of this was to collate every action the agents had taken in a run of the program, and plot it onto a graph against the number of generations. Examining the average fitness per generation was integral in making sure the program was working correctly, as well as comparing the effect of changing variables within the code

(such as the fitness function). To accomplish this, NEAT has a statistics reporter [CodeReclaimers, b] - the code for which can be found as a reference. Essentially, this reporter will gather statistics related to the population such as average fitness and the standard deviation of fitness and store them as 'checkpoints' as the program progresses. Once the program has finished executing, NEAT will use these checkpoints to collate the fitness values across generations and allow the user to easily see the performace of the agents in the population. An example for this will be seen in the following section. Another utilised package for visualisation was 'matplotlib' [Hunter et al.,] - an extemely useful tool for creating and managing graphs of varying complexity. The reason for using this was to plot the ratio of cooperative and defective moves across generations. Unfortunately, NEAT has no easy way for the user to plot their own statistics, as this isn't the nature of the library. Again, an example of the matplotlib visualisations will be seen in the following section.

4.3 Finite State Machines Method

As with the neural networks method of agent representation, the finite state machine method expects very similar requirements and milestones, such as:

- Concept of agents as prisoners.
- Attributes for the prisoners, such as a unique ID and fitness.
- Some concept of states and transitions attached to the prisoners.
- Explicity coded strategies.
- Concept of dilemma, being able to select a move (Cooperate/Defect) and receive a payoff.
- Basic evolutionary algorithm capable of initialising a population and handling multiple generations.
- Interaction between agents.
- Extending the dilemma to facilitate play between two agents, as well as giving payoffs accordingly This includes each agent being able to independently select a move, receive a payoff and adapt their strategy accordingly for the next round/generation.
- Mutation, crossover and selection operators.

- Output and plot the attributes of each generation.

These requirements are somewhat more complex than that of the neural networks method, which will subsequently be elaborated on. Something to keep in mind is that the finite state machines method utilises the **transitions** package, which was explained in the previous section.

4.3.1 Initialisation of Agents

Creating and managing agents for this method was very different than that of neural networks - in order to attach various attributes to a finite state machine, the prisoners had to be made as objects. Thankfully, the transitons package allows states, attributes, state transitions and functions to all be contained within objects. The attributes range from fitness, ID, the chosen move of the agent, their history of moves and the strategy they have adopted. The states are simply: *Thinking*, *Cooperate* and *Defect*. The 'thinking' state is simply the state which each prisoner starts the simulation from. The transitions part of the package allows the user to move from one state to another by simply using a trigger. The triggers are function calls from other places in the program, and the function calls within the object scope will handle the necessary logic. This encompasses all of the facets of an agent object.

4.3.2 Agent Strategies

The agent strategies for finite state machines are explicitly stated at the beginning and evolved during runtime. Popular strategies include the tit-for-tat and random strategies, both used in this simulation. A basic tit-for-tat state diagram would look as follows:

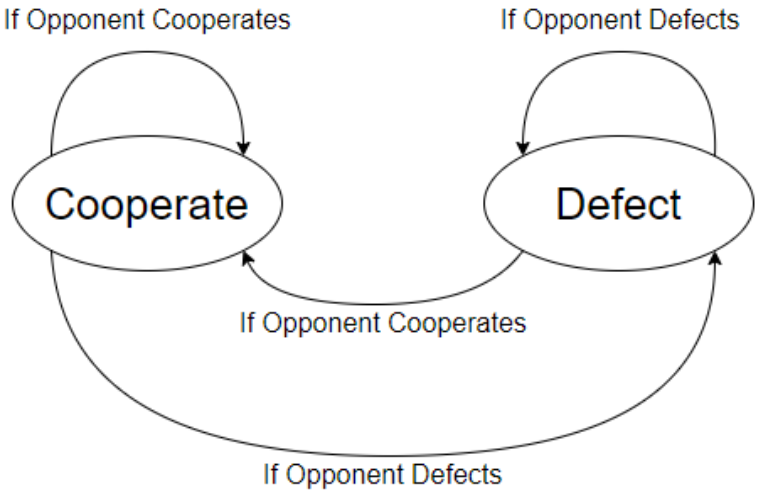


Figure 6: Tit-For-Tat State Diagram

In the program, this simply comes down to:

- Check the current state of the agent.
- Check the opponent's previous move.
- Use that move as the agent's current move - i.e if the opponent's last move was defective, defect this round.
- Update the current state of agent.

Another explicitly-coded strategy is the random strategy, which is self explanatory. But how do strategies evolve? In reference to van Veelen's paper, "In and out of equilibrium: evolution of strategies in repeated games with discounting" [Veelen and García, 2016], strategies evolve after the end of each generation using mutation operators. This means that there is a small random chance for each agent of the population at the end of the generation to be mutated - this could be either adding or removing a state. Based on the games they play and moves they pick, adding and removing states will make one choice more prominent in their list of states, and becomes the more likely state to be picked.

4.3.3 Incentivisation, Payoff and Visualisation

Exactly like the neural networks method, the incentive for agents to pick a particular move relies solely on the fitness function, which is functionally the

same. Fixed and continuous fitness calculations yield different results, which will be highlighted in the following section. For future note, fixed fitness refers to adding the payoffs from moves to the fitness directly, while continuous fitness refers to the fitness function of the Harrald [Harrald and Fogel, 1996, p. 139] paper. Visualisation also follows suit, in which the average fitness of the population is tallied for each generation and plotted using the matplotlib package.

5 Results

In this section, the results following the implementation of the program will be discussed. As with the forementioned section, this section will cover both neural networks and finite state machines separately, and then discuss them comparatively. In regards to the output of the program, there will be a handful of topical questions which must be asked and answered in order to determine if the results are correct.

5.1 Neural Networks Results

In the previous section, the structure of the program was discussed, in terms of how it was built and how it executes. The first topical question to start with is as follows: **After the strategies of the networks are evolved, do the agents ever reach a point in which they always defect?** In order to effectively examine these results, the program must be examined in the form of a single-shot prisoner's dilemma, as well as a repeated game. Interestingly, speciation plays a large role in how the strategies are evolved; the ability to control speciation within the population yielded a decent range of results, as will now be discussed - starting with single-shot games.

5.1.1 Single-Shot Prisoner's Dilemma (Fixed Fitness)

Before the results are presented, an important note: as stated before in the methodology section, the fitness for the program is calculated as follows: F = F + (P1 || P2). Where P1/P2 is the payoff from the move made by the first or second agent respectively. As a comparison, the following subsection will look at using Harrald's continuous fitness calculation [Harrald and Fogel, 1996] found on page 139 of his paper. This fitness calculation will be explained in said subsection.

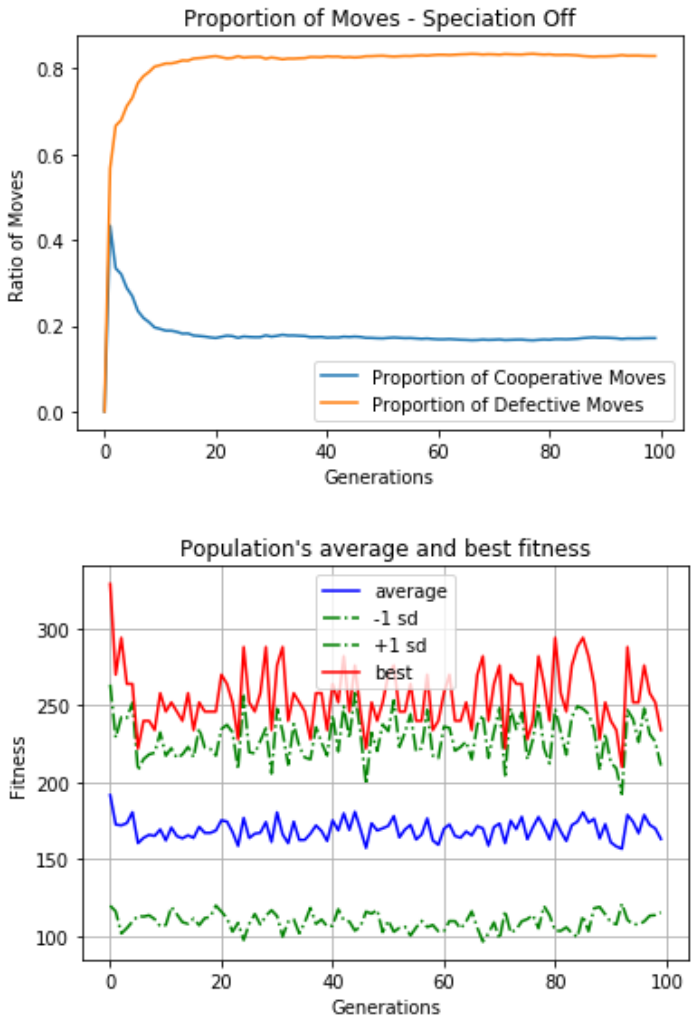


Figure 7: Neural Networks Proportion of Moves With No Speciation

As seen in figure 6, a large fraction of the population hit a preferred strategy after roughly 15 - 20 generations. In the program, the agents are playing against themselves and each other agent in the population **only once**. With this in mind, their strategies will yield better payoffs each generation, regardless of the opposing strategy. In regards to the forementioned question which was posed, there very obviously seems to be a point in which defecting becomes the preferred strategy, but does this prove the existence of a Nash equilibrium?

Figure 8: Neural Networks Proportion of Moves With Speciation

With speciation turned on, the results are vastly different - the fitness of the networks starts out very high as the mix of moves (one agent choosing cooperate and the other choosing defect) brigns the highest payoff. After testing this strategy for roughly 25 - 30 generations, the agents realise that constant mutual defection yields the best possible payoff in the long run. After generation 40, each agent in the population is choosing defect at an exponential rate, which proves indefinitely the existence of a Nash equilibrium within the population. Another interesting point worth bringing up is the comparison of results between figure 7 and the results from Harrald's [Harrald and Fogel, 1996] paper in 1996. In this paper, Harrald has very similar results to that of figure 7, using networks with 20 hidden perceptrons as opposed to 2. But

how well, or differently, does the program perform using Harrald's continuous fitness function?

5.1.2 Single-Shot Prisoner's Dilemma (Continuous Fitness)

On page 139 of Harrald's [Harrald and Fogel, 1996] paper, it is stated that the fitness for a given player, A, is calculated as follows:

$$f(\alpha, \beta) = \text{-}0.75\alpha + 1.75\beta + 2.25$$

Where α and β are the moves of players A and B respectively. This subsection will examine the results of the program using the same parameters, but with this fitness function instead.

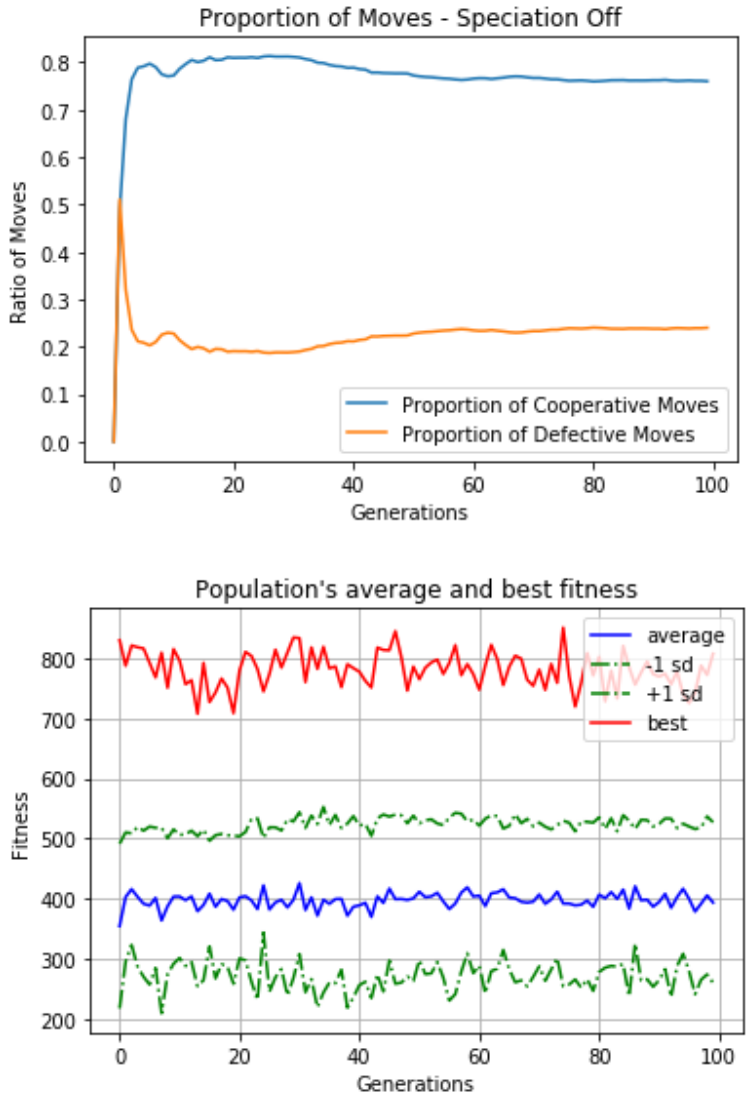


Figure 9: Neural Networks Continuous Fitness With No Speciation

Firstly, with speciation off, the continuous fitness function seems to encourage a long-term play strategy from the very start of the simulation. In comparison to the fixed fitness function, where the results seemed to be somewhat random and no preferred strategy arose. However, in figure 9, cooperation has a much higher rate of play, and may present a dominant strategy of mutual cooperation. Let's compare this to a simulation with speciation:

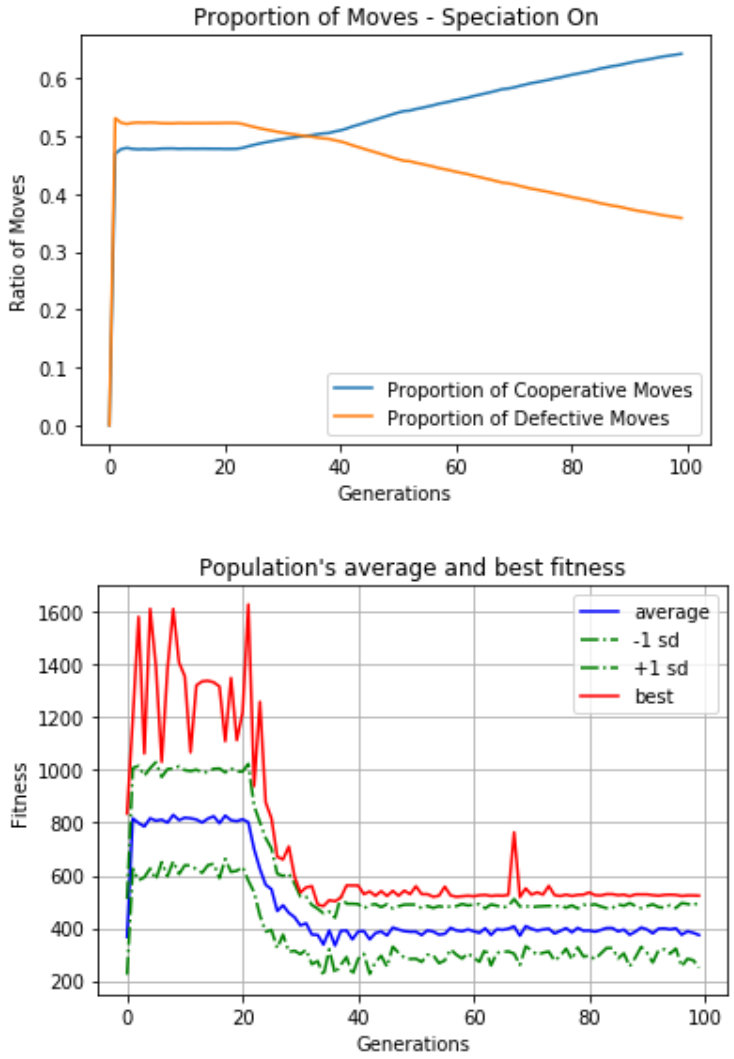


Figure 10: Neural Networks Continuous Fitness With No Speciation

This figure yields far more interesting results. While similar to that of the fixed fitness graph, figure 8, the agents progress to a complete state of co-operation. When the number of generations is increased to a larger number, the number of defective moves goes down, and will eventually reach either none, or close to none.

5.1.3 Repeated Prisoner's Dilemma (Fixed Fitness)

5.1.4 Repeated Prisoner's Dilemma (Continuous Fitness)

5.1.5 Multiple Replicants

5.2 Finite State Machine Results

On the other side of the implementation, the results for the finite state machine prisoners' dilemma came about in very different ways. As the finite state machines didn't have any speciation affecting the output, the results only came in one set, but looked similar to those of the neural networks. The first draft of the implementation produced an output as shown:

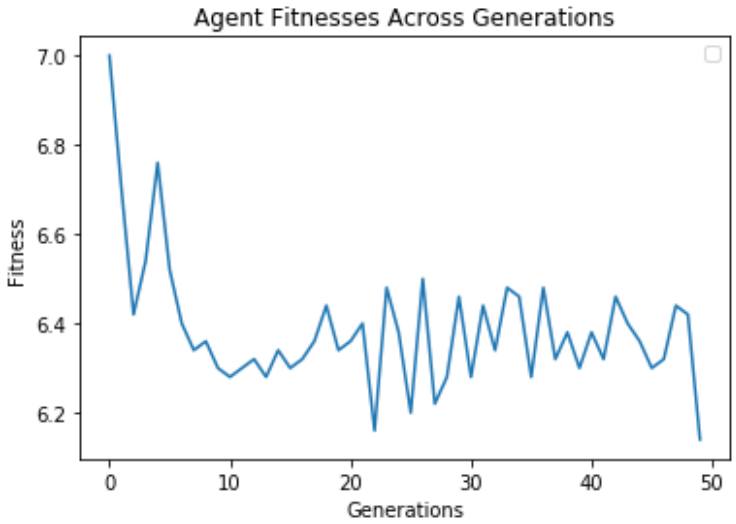


Figure 11: Finite State Machines Continuous Fitness

While this doesn't seem similar to the neural networks results, figure 11 shows a fitness trend that is not unlike the trend of figure 10. The previous neural network simulations with no speciation produced very random, erratic results with no causality; these results show the fitness starting high, then dropping down to a lower figure, dipping between roughly 6.25 and 6.45 fitness (using single shot rounds).

6 Evaluation

Given the two facets of the project, neural networks and finite state machines, this section will evaluate the effectiveness and rigour of both approaches; this will involve comparing the original project schedule to the final project schedule, how the original approaches to the software were laid out versus how the software actually turned out, how the software compares to some of the literature and previous work and finally what could be done differently with more time or if the project was re-attempted. First, the initial and final project schedules must be examined.

6.1 Project Timelines

As a reminder, the initial deliverables of the project (in order) were as follows:

- First draft of the literature review finished (**October 10th 2018**).
- Evolutionary algorithm, any number of agents, 2-player prisoners' dilemma, finite state machine, log of game, recorded fitness of each generation. (**November 9th 2018**)
- Working evolutionary algorithm which can recreate results of other experiments in the literature, mainly Harrald and van Veelen. (**December 7th 2018**)
- Comparison of effectiveness between algorithms. (**February 28th 2019**)

Appendix D shows the initial project timeline with the forementioned dates, created in October of 2018. The updated project timeline, created in March of 2019, can be seen as Appendix E. In short, a comparison of the initially posed dates and final deliverable completion dates is:

- Deliverable One (Initial Date - **October 10th 2018**)
  (Completed Date - **October 9th 2018**).
- Deliverable Two (Initial Date - **November 9th 2018**)
  (Completed Date - **November 15th 2018**))
- Deliverable Three (Initial Date - **December 7th 2018**)
  (Completed Date - **November 23rd 2018**))

- Deliverable Four (Initial Date - **February 28th 2019**)
  (Completed Date - **? March 2019**))

Now, each deliverable will be evaluated and the dates of completion justified.

6.1.1 First Deliverable

Initially posed to be completed on the 10th of October, completing a first draft of the literature review seemed realistic and attainable by that date. The review itself had been intermittenly worked on over the course of a couple of months, and as long as there was a consistent addition of content, there were no insurmountable hurdles to face. Overall, this deliverable went very smoothly and no changes would be made to it if the project was re-attempted.

6.1.2 Second Deliverable

The second deliverable was a lot harder than the first - essentially containing 6 sub-goals, there were 5 days on average to complete each sub-goal. Initially, upon creating the dates for the deliverables, this seemed fair and realistic. However, when configuring NEAT, creating a prisoners' dilemma with pseudo-realistic behaviour and a plethora of feedback, it quickly became apparent that this deliverable wouldn't be completed on time. In addition, the neural networks facet of the project started to require a few more things to be created at each interval of the project - as a contingency, the completion of the finite state machines was moved to be part of the fourth deliverable. If this part wasn't moved, the second deliverable could've taken a few more weeks, or maybe a month more to complete. In hindsight, this deliverable should've been split more evenly between the second and third deliverable - justification for this will be explained now.

6.1.3 Third Deliverable

This deliverable seemed as though it would be more difficult than it actually was. After creating the simulation with variables, methods and functions made to model a prisoners' dilemma with neural networks in the most convenient way, it seemed likely that a separate version of the code would have

to be created to accommodate results achieved in experiments from papers in the past such as Harrald's "Evolving continuous behaviors in the Iterated Prisoner's Dilemma" [Harrald and Fogel, 1996] or van Veelen's "Direct reciprocity in structured populations". [Veelen et al., 2012] However, in this part of the project, only Harrald's results had to be modelled, and this simply came down to changing a few inputs, variables, functions and expected outputs to replicate some of the graphs from those papers. In comparison to the previous deliverable, this deliverable progressed very well and showed promising results for future parts of the project. This compensated well for the shortcomings of the second deliverable.

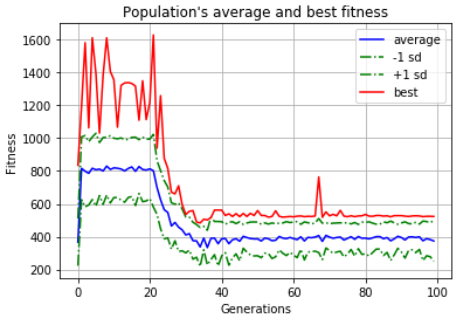
6.1.4 Fourth Deliverable

Unfortunately, as well as previous parts of the project went, this deliverable slowed down progress quite dramatically as the implementation of finite state machines had an unforeseen dimension of difficulty. As the fourth deliverable approached, the state machine facet of the project ran into a few issues mainly concerned with understanding, and how the transitions library operated. This also coupled with the neural networks facet requiring a few more additions to refine the results into something more presentable. This slowed down progress to the point where the state machine facet would likely not be completed to the best of standards, which skews the comparison to the neural networks. If the project was re-attempted, implementation of the state machines would likely start earlier to alleviate some of the problems encountered.

6.2 Comparison to Literature

As stated before, part of the project was comparing the implemented results to those of experiments in the literature review. Namely, Harrald and van Veelen conducted experiments while providing a large amount of knowledge necessary for replication. In Harrald's paper, "Evolving continuous behaviors in the Iterated Prisoner's Dilemma", a graph is included [Harrald and Fogel, 1996, p. 142] for one of the experiments which relates to this project. Using a graph of similar inputs from the previous section, a comparison looks as follows:

Project Results



Harrald Results

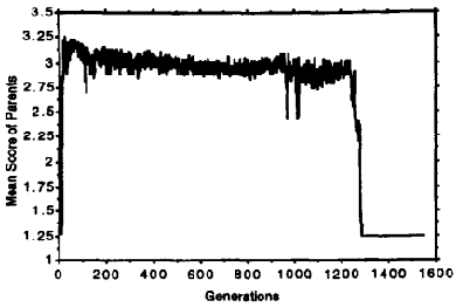


Figure 12: Project Results vs. Harrald Results

While the results appear similar, this project used different inputs to the Harrald paper - specifically, this project used a network topology of three input layers, six hidden layers and two output layers. The Harrald results assume six input layers, twenty hidden layers and one output layer. By changing the amount of layers to match, the results would most likely look close to identical, if the simulation was ran for 1600 generations and the fitnesses were clamped.

6.3 Comparison of Effectiveness Between Representations

In the literature review, a handful of research questions were posed to eventually evaluate the implementation of the program. These questions were as follows:

1. Given the results of research papers in the past such as Nash's, Harrald's or Axelrod's, how do those results look now when used within a model of more realistic computational power?
2. When using different representations of an agents' cognition, how does the payoff vary between evolved strategies from neural networks and finite state machines? How easy is it to evolve those representations at a high payoff?
3. Within an evolutionary algorithm, which neural network topologies are favoured in 2-player games when a neural network is being used as a representation of the agents' cognition(s)?

4. During this project, neural networks will be used with a fixed shape and payoff matrix. If it is allowed to evolve, how does the shape of the neural network change? (Optional)

The first question has been researched and evaluated in parts of each section; in the previous section, the results were shown to prove the existence of a Nash equilibrium through complete defection, as well as being similar to the results of the Harrald paper. However, the results look slightly different, as the results in this project (specifically the neural networks side) seems to evolve a strategy much faster than that of the literature. It takes roughly 1250 generations for the strategy to be evolved in the literature, whereas the strategies in this project seem to evolve and trend towards complete defection/cooperation after 40 generations.

The second question poses a slightly vague answer. Varying the payoff for the neural networks or finite state machines, by either making the payoff smaller or larger, doesn't have a particularly adverse effect - if the neural networks implementation used one output neuron which continuously output between -1 and 1, the payoff matrix would become redundant. Due to these reasons, it seems likely that changing the payoff in either payoff matrix won't evolve the strategies any faster or slower.

In reference to the third question, feedforward networks are favoured. In the neural networks prisoners' dilemma, there is a very explicit flow of data from the input layers to the hidden layers, then from hidden to the output layers. Backpropagation is not applicable as the information is not required to travel back the way. Feedback is collected at the end of the simulation, and sending it backwards at any point during runtime is superfluous. A recurrent network isn't favoured as the nodes do not necessarily have to form a sequence.

Unfortunately, the last research question could not be attempted in the time-frame of the project.

The main focus of this project was to examine the differences in effectiveness between using neural networks or finite state machines to represent agents in the prisoners' dilemma, using an evolutionary algorithm to advance and evolve the game. As stated before, it was undeniably easier to implement neural networks to achieve this, as proved by the updated project timeline. However, labelling it more 'effective' due to its ease of implementation is unjust.

6.4 Experimentation

Upon implementing the program (both the neural networks and finite state machine versions), some form of experimentation must be conducted in order to validate the correctness/robustness of the program. In order to do this, the following tests can be conducted on the program, such as ...

7 Conclusion

After covering the previous sections, what can be taken away from this project? The research goals and questions must be revisited in order to answer whether or not they were achieved. There may also be possible discussions on further research.

References

[Angeline et al., 1994] Angeline, P., Saunders, G., and Pollack, J. (1994). An evolutionary algorithm that constructs recurrent neural networks. Report, Ohio State University.

[Ash, 1989] Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, 1:365–375.

[Axelrod, 1980] Axelrod, R. (1980). Effective choice in the prisoner's dilemma. Report, Sage Publications, Inc.

[Axelrod, 1984] Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books Inc, New York.

[Aziz et al.,] Aziz, D. A., Cackler, J., and Yung, R. Basics of automata theory. Stanford University report detailing automata theory/finite state machines.

[Binmore, 2005] Binmore, K. (2005). *Natural Justice*. Oxford University Press, Oxford.

[Britannica,] Britannica, E. Prisoner's dilemma. https://www.britannica.com/topic/prisoners-dilemma.

[Chen et al., 1995] Chen, D., Giles, C., Sun, G., Chen, H., Lee, Y., and Goudreau, M. (1995). Constructive learning of recurrent neural networks. *IEEE Transactions on Neural Networks*, 6:1196–1197.

[Chen et al., 1998] Chen, J., Lu, S., and Vekhter, D. (1998). Game theory. Report, Stanford University.

[CodeReclaimers, a] CodeReclaimers. Neat overview. $https://neat-python.readthedocs.io/en/latest/neat_overview.html$.

[CodeReclaimers, b] CodeReclaimers. Source code for statistics. $https://neat-python.readthedocs.io/en/latest/_modules/statistics.html$.

[Collis,] Collis, J. Glossary of deep learning: Bias. https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2.

[Duignan,] Duignan, B. Prisoner's dilemma. Website article detailing game theory.

[Fahlman, 1991] Fahlman, S. (1991). The recurrent cascade-correlation architecture. Report, Carnegie Mellon University.

[Fogel et al., 1995] Fogel, III, D., C., E., and Boughton, E. (1995). Evolving neural networks for detecting breast cancer. *Cancer Letters*, 96:49–53.

[Friedman, 1971] Friedman, J. (1971). A non-cooperative equilibrium for supergames. Report, Oxford University.

[Gifford,] Gifford, A. Payoff matrix in economics: Theory & examples. Useful website detailing payoff matrices.

[Harrald and Fogel, 1996] Harrald, P. G. and Fogel, D. B. (1996). Evolving continous behaviours in the iterated prisoner's dilemma. Report, Manchester School of Management.

[Herdt, 2003] Herdt, T. (2003). Cooperation and fairness: the flood–dresher experiment. *Review of Social Economy*, 61:184–191.

[Hunter et al.,] Hunter, J., D., D., Firing, E., and Droettboom, M. matplotlib. https://matplotlib.org/.

[Jong et al., 1997] Jong, K., Fogel, D., and Schwefel, H.-P. (1997). A history of evolutionary computation. Report.

[Kakutani, 1941] Kakutani, S. (1941). A generalization of brouwer's fixed point theorem. *Duke Mathematical Journal*, 8:457–459.

[Kaznatcheev,] Kaznatcheev, A. Short history of iterated prisoner's dilemma tournaments. Website detailing iterated prisoner's dilemma tournaments.

[Knight,] Knight, V. Background to axelrod's tournament. Website detailing Robert Axelrod's tournament.

[Langley, 1995] Langley, P. (1995). Applications of machine learning and rule induction. Report.

[Levine,] Levine, D. What is game theory? Website detailing general game theory principles.

[Lutter and Huntsinger, 1968] Lutter, B. and Huntsinger, R. (1968). Engineering appllications of finite automata. *Sage Journals*, 47:264–265.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

[McDonnell and Waagen, 1994] McDonnell, J. and Waagen, D. (1994). Evolving recurrent perceptrons for time-series modeling. *IEEE Transactions on Neural Networks*, 5:24–38.

[McIntyre et al.,] McIntyre, A., Kallada, M., Miguel, C. G., and da Silva, C. F. neat-python. https://github.com/CodeReclaimers/neat-python.

[Nash, 1950a] Nash, J. (1950a). Non-cooperative games. Report, Princeton University.

[Nash, 1950b] Nash, J. (1950b). Two-person cooperative games. Report, Air Force Project RAND.

[Neumann,] Neumann, A. transitions. https://github.com/pytransitions/transitions.

[Nodine, 2010] Nodine, T. (2010). Speciation in neat. Undergraduate Honors Thesis HR-10-06, Department of Computer Science, The University of Texas at Austin.

[Paci,] Paci, G. What is an agent. http://wiki.c2.com/?WhatIsAnAgent.

[Policonomics,] Policonomics. Game theory iii: Folk theorem. Website detailing folk theorem.

[Porto et al., 1995] Porto, V., Fogel, D., and Fogel, L. (1995). Alternative neural network training methods. *IEEE Expert: Intelligent Systems and Their Applications*, 10:16–22.

[Ross,] Ross, D. Game theory. Website detailing game theory principles.

[Russell and Norvig, 1994] Russell, S. and Norvig, P. (1994). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

[Samuel, 1959] Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal*, 3:535–554.

[Smith and Wright, 1991] Smith, E. and Wright, R. (1991). Why is automobile insurance in philadelphia so damn expensive? Report, University of Essex.

[Smith, 1982] Smith, J. (1982). *Evolution and the theory of games*. Cambridge University Press, Cambridge.

[Stergiou, nd] Stergiou, Christos. Siganos, D. (n.d.). Neural networks. Report, Imperial College London.

[Teknomo,] Teknomo, K. Aggregation function. https://people.revoledu.com/kardi/tutorial/NeuralNetwork/AggregationFunctions.html.

[Tiwari,] Tiwari, S. Activation functions in neural networks. https://www.geeksforgeeks.org/activation-functions-neural-networks/.

[Veelen and García, 2016] Veelen, M. and García, J. (2016). In and out of equilibrium: evolution of strategies in repeated games with discounting. 161:161 – 189.

[Veelen et al., 2012] Veelen, M., García, J., Rand, D., and Nowak, M. (2012). Direct reciprocity in structured populations. 109:9929–9934.

[von Neumann et al., 1944] von Neumann, J., Morgenstern, O., and Rubinstein, A. (1944). *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press.

Appendices

A Initial Project Overview (IPO)

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Learning to Play: Nash Equilibria in Repeated Games Using Machine Learning and Neural Networks

Overview of Project Content and Milestones

The aim of the project is to investigate how changing the representation of an agent's cognition changes how easy or difficult it is to evolve effective strategies for playing repeated games.

The key milestones to achieving this are:

1. Research different models of agent cognition, in particular neural networks and finite state machines.
2. Implement an evolutionary algorithm that can evolve agents to play a repeated Prisoner's Dilemma, where the representation of the evolving agent strategies can be changed between neural networks and finite state machines.
3. Test the evolutionary algorithm by recreating existing results from the literature.
4. Perform experimental analysis to determine how the representation of the agent's strategy changes how easy it is for the evolutionary algorithm to find strategies that give high payoffs, i.e. that play the game well.
5. Perform additional experiments to determine how robust the results are to changes in the game, e.g. Stag Hunt vs Prisoner's Dilemma.

The Main Deliverable(s):

1. The main deliverable of the project is to have a robust program which houses an evolutionary algorithm capable of handling both finite state

machines and neural networks, as forms of representations of an agents' cognition within a repeated game theory scenario. These agents should be able to play against each other in multiple games, with evolving strategies. These representations should be able to evolve, whether they be controllers such as neural networks or finite state machines.

2. The secondary deliverable will be an experimental analysis into the controllers of the program: neural networks and finite state machines. When these techniques are implemented within the program, they must be compared to identify the best method of evolving an agents' strategy/cognition by ways of algorithm fitness or strategy payoff.

The Target Audience for the Deliverable(s):

The finished product of the project will most likely require some working knowledge of game theory and/or machine learning to understand. Ideally, the finished product will have enough of an interesting result to yield possibilities of further research, which may interest those with experience in the applicable fields.

The Work to be Undertaken:

The project will first require a great amount of research and review. Most of the foundational work applicable to this project was published in the 1950s/60s, with important work on evolutionary algorithms and neural networks between then and the present. Thus, most of the information needed for the project is a collection of professional papers and/or publications written by well-established mathematicians and game theorists.
A system will need to be designed and implemented to sustain 2 agents, a repeated game scenario, a neural network and a relay of information which the agents can train from. This system will all be built in the Python programming language - and may use code packages as reference material.
After a suitable system is designed and implemented, the user should have control over the power of computation for the agents, and the weights/payoffs of the neural network should be serialised to a CSV file – which can used to compare results between iterations of the games.
The end result of the project should have the ability of experimentation, although the best strategies should more or less be the same/similar on each iteration depending on the situation. Ideally, the project should be able to replicate the results of the iterated prisoner's dilemma experiment by Har-

rald and Fogel in 1996, if given the same initial values.

Additional Information / Knowledge Required:

I will be honing my skills in Python and project management. I will ideally have a working knowledge of artificial intelligence, machine learning and evolutionary algorithms. Additionally, I will learn about the various packages and features of Python which are applicable to the project.

Information Sources that Provide a Context for the Project:

Work leading up to the eventual discovery of the Nash equilibrium was authored by John von Neumann and John Forbes Nash. Work on repeated games was introduced and popularised by Robert Axelrod. Work on neural networks was arguably pioneered by Marvin Lee Minsky with his creation of the SNARC (Stochastic Neural Analog Reinforcement Calculator).

The Importance of the Project:

From a personal standpoint, the project will be important for gaining useful information on topics I've never studied, and the end result will hopefully be a proud accomplishment which I can show to employers. If said result is good enough, the project itself may be a useful research topic/experiment for others in the future. The project as a whole should also be a good investigation tool into how much computation power is needed for a project of this type, as this project will have limited power. These results should be helpful in answering the question of whether or not these scenarios can be adapted to evolve non-player characters in video games.

The Key Challenge(s) to be Overcome:

There is an inevitable difficulty when dealing with these topics – game theory, artificial intelligence and evolutionary algorithms are difficult fields to put into application, and I suspect there will be an inherited difficulty which comes with the task. Blending such fields may prove difficult given the free open-source software, in which work like this project may/may not have been done before. Finally, given the time and effort constraints of university, the timelines and workloads other modules may make it difficult to complete each deliverable on time.

B Interim Report Overview

Insert a copy of the project review form you were given at the end of the review by the second marker

C Diary Sheets & Project Management Documents

September

Week 1

Objectives:
The initial goal of the meeting was to examine and modify the first draft of the initial project overview document, with the hope of improving it to a standard for submission. Another goal was to discuss the project timeline, and how it should be laid out in a chart.

Progress:
Upon reading the draft of the document, Simon discussed and agreed suitable goals/milestones of the project, which would eventually be placed into a project timeline chart (Gantt). Simon and I then made changes to a couple of sections in the IPO document, then discussed plans for how the program should be designed. This led to Simon proposing an unofficial, rough change of the project title, given the new scope of design and implementation: "Learning to play: A comparison of effectiveness between different representation of agent cognitions using evolutionary algorithms and machine learning". Simon also suggested reading up on evolutionary algorithms, as well as emailing the new second marker as per my prompt.

Goal for Next Week:
Finish IPO document, refine for submission.
Finish project timeline (Gantt Chart).

October

Week 3

Objectives:

The goal of this meeting was to scan over the literature review of the project, so that Simon could see the structure and overall content quality and eventually offer feedback for each section. I also had to iron out a few queries/thoughts regarding the referencing and citation facet of the project.

Progress:
Simon started by looking over the list of references for the literature review to make sure they were all in the correct format and reliable sources. He then suggested a few papers to read and then add to the list, as a good general reference for the review such as books/papers by John Maynard Smith and Ken Binmore. After this, Simon started to read over the content of the literature review, and offered a number of suggestions/changes for the benefit of quality/readability.

Goal for Next Week:
Finalise first draft of the literature review, to eventually look over the content.

November

Week 7

Objectives:
The goal of this meeting was plan the implementation of the project.

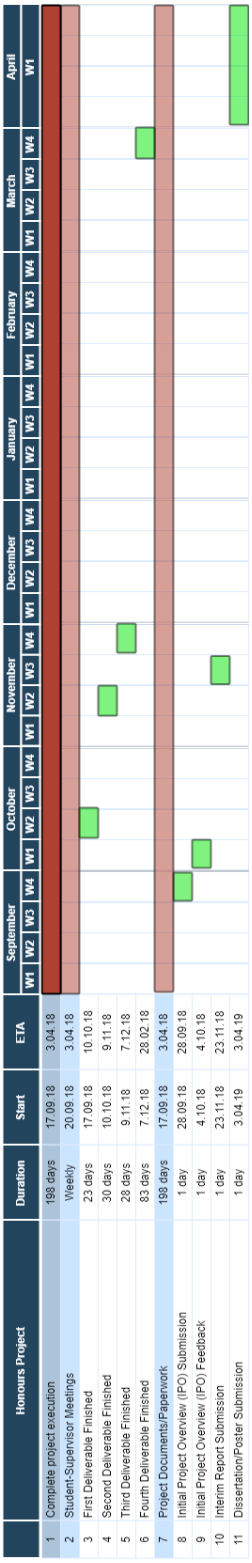
Progress:
Simon suggested a good amount of tips for how to approach the program, and what the eventual output should be.

Goal for Next Week:
Work on implementation – neural networks + evolutionary algorithms.

D Original Proposed Project Timeline

	Honours Project	Duration	Start	ETA
1	Complete project execution	198 days	17.09.18	3.04.18
2	Student-Supervisor Meetings	Weekly	20.09.18	3.04.18
3	First Deliverable Finished	23 days	17.09.18	10.10.18
4	Second Deliverable Finished	30 days	10.10.18	9.11.18
5	Third Deliverable Finished	28 days	9.11.18	7.12.18
6	Fourth Deliverable Finished	83 days	7.12.18	28.02.18
7	Project Documents/Paperwork	198 days	17.09.18	3.04.18
8	Initial Project Overview (IPO) Submission	1 day	28.09.18	28.09.18
9	Initial Project Overview (IPO) Feedback	1 day	4.10.18	4.10.18
10	Interim Report Submission	1 day	23.11.18	23.11.18
11	Dissertation/Poster Submission	1 day	3.04.19	3.04.19

E Updated Project Timeline



	Honours Project	Duration	Start	ETA
1	Complete project execution	198 days	17.09.18	3.04.18
2	Student-Supervisor Meetings	Weekly	20.09.18	3.04.18
3	First Deliverable Finished	23 days	17.09.18	10.10.18
4	Second Deliverable Finished	30 days	10.10.18	9.11.18
5	Third Deliverable Finished	28 days	9.11.18	7.12.18
6	Fourth Deliverable Finished	83 days	7.12.18	28.02.18
7	Project Documents/Paperwork	198 days	17.09.18	3.04.18
8	Initial Project Overview (IPO) Submission	1 day	28.09.18	28.09.18
9	Initial Project Overview (IPO) Feedback	1 day	4.10.18	4.10.18
10	Interim Report Submission	1 day	23.11.18	23.11.18
11	Dissertation/Poster Submission	1 day	3.04.19	3.04.19