School of Computing
FACULTY OF ENGINEERING

Developing a Strategy for the Game 'Why First?': A Reinforcement Learning Approach

Reece Banbury

Submitted in accordance with the requirements for the degree of MSc Advanced Computer Science

2014/1015

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Project Report	Report	SSO (30/09/15)
All software agent implementation code	Software codes	Dr. Brandon Bennett, Dr. Raymond Kwan (30/09/15)
README files for final software version use	User manuals	Dr. Brandon Bennett, Dr. Raymond Kwan (30/09/15)
Strategy files to configure final software version	Binary files	Dr. Brandon Bennett, Dr. Raymond Kwan (30/09/15)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

This projects looks at evaluating reinforcement learning techniques when applied to the game '*Why First?*'.

Many games can be viewed as abstract versions of real world problems, by demonstrating methods of solving various types of games we can generalise these techniques to solve problems various problems in fields such as medicine(Shipp et al. 2002). '*Why First?*' is a rather unusual game, so studying it has the potential to yield new insights. It is counter-intuitive to humans since the aim is to place second.

In this project I have demonstrated that reinforcement learning can be applied to games such as '*Why First?*' that do not hold properties found in more well understood games.

Early implementations focus on developing an understanding of the game and what information is important when making a decision. Using this information we develop a board state representation that provides enough information while minimising the board state search space.

The final implementation uses a defined board state representation and random weighted selection to make decisions during the course of a play. By increasing the weights of actions for board states that lead to wins and decreasing those that lead to losses, the implementation effectively learns to play the game without supervision.

Acknowledgements

First and foremost I would like to thank Dr. Brandon Bennett for going above and beyond what I would expect from a supervisor. For taking the time to support and guide me whenever I needed it and for teaching me how to write.

Secondly, I would like to thank everyone in the School of Computing who took an interest in this project and played the game with me. In particular Oliver Skidmore for his insight and his time taken to help me track down bugs. And Dr. Raymond Kwan for taking the time to further help me understand Hyper-Heuristics.

I would also like to thank Dr. Lydia Lau and everyone in the School of Computing's Student Support Office for their help and understanding with the difficulties I faced during this project.

And finally I would like to thank everyone who gave me somewhere to sleep and work during the last few weeks of this project, I couldn't have done this without you.

Contents

1 Introduction **3**

1.1 Aim and Objectives . . . 3
1.2 Minimum Requirements . . . 3
1.3 Deliverables . . . 3
1.4 Methodology . . . 3

2 Background and Related Research **5**

2.1 Introduction . . . 5
2.2 Game Theory . . . 5
2.2.1 Definition of a Game . . . 5
2.2.2 Types of Games . . . 6
2.2.3 Minimax . . . 8
2.2.4 Conclusions . . . 9
2.3 'Why First?' . . . 11
2.3.1 Overview . . . 11
2.3.2 The Rules . . . 11
2.3.3 The Scope of this Project: The 3-Player Single Stage Game . . . 13
2.4 Artificial Intelligence . . . 14
2.4.1 Machine Learning . . . 14
2.4.2 Reinforcement Learning . . . 15
2.4.3 Hyper-Heuristics . . . 15

3 Understanding the Problem **16**

3.1 Observations and Speculation . . . 16
3.2 Formulation and Representation . . . 16
3.3 Measurement of Effectiveness . . . 16

4 Naive Approaches **17**

4.1 Fixed Strategies . . . 17
4.1.1 Results . . . 20
4.1.2 Conclusions . . . 20

5 Learning Approaches **22**

5.1 Predefined Mixed Strategies . . . 22
5.2 Random Probabilistic Selection (RP) . . . 22
5.2.1 Implementation . . . 22
5.2.2 Random Weighted Selection Algorithm . . . 23
5.2.3 Experiments and Results . . . 26
5.2.4 Conclusions . . . 31
5.3 Generated Mixed Strategies . . . 31
5.3.1 Random Weighted Selection Algorithm Revisited . . . 32
5.3.2 Conclusions . . . 32
5.4 Developing Strategies from Experience . . . 33

5.4.1 Board State Representation 33
5.4.2 Reinforcement Learning from Board States (HL) 34
5.4.3 Implementation 35
5.4.4 Experimentation and Results 37
5.4.5 Training Against Different Opponents 38
5.4.6 Strategy Analysis 43

6 Conclusion **48**

7 Future Work **48**
7.1 Genetic Algorithms 48
7.2 Minimax 48
7.3 Different Board State Representations 49
7.4 A General Learner 49

References **50**

Appendices **51**

1 Introduction

Developing algorithms that can play games has been a significant sub-field of Artificial Intelligence since its inception. Much research has been devoted to achieving human-level or better playing ability in a variety of board games, such as *Chess*, *Checkers*, *Backgammon*, *Go* etc. Considerable success has been achieved and many techniques that are used widely in AI were originally developed from game playing, for example Minimax and Alpha-beta pruning.

1.1 Aim and Objectives

The aim of this project is to test the effectiveness of machine learning techniques when applied to a game that has not yet been investigated by previous research and deviates characteristically from the more popular and well understood games.

Since the game in question holds properties not common in traditional games such as *Chess* or *Checkers*, this project aims to demonstrate the applications of machine learning techniques for a less familiar type of game.

1.2 Minimum Requirements

- Develop a software prototype that demonstrates that either Reinforcement Learning techniques can be used as an effective approach to play the game '*Why First?*', or that they are not.

1.3 Deliverables

- This report that shall include:
  - A literature review on Game Theory and existing Machine Learning techniques.
  - The design process and evaluation of all software created.
  - Details of experiments and results run.
- Prototype software implementing an AI agent that can play '*Why First?*'. Several versions will be implemented and tested.

1.4 Methodology

For this project I shall be creating *exploratory softare*. This software will not be of a commercial standard. It is a tool for demonstrating the techniques discussed and is only intended for use as a research tool. Therefore error checking and user interface are minimal. The program may not be completely efficient, but it will adequetely perform the task it was designed for. Efficiency is only important in so far as it affects the number of learning iterations that can be performed.

I will be using an *iterative and incremental development* methodology(Basil and Turner 1975). I will be iterating upon software versions to develop them to demonstrate what is required. Subsequent versions that are intended to demonstrate something new and different, will

be developed on the framework of the previous software verison using the results of the previous experiments to direct the development of the new software version. Since each version will be developed on the framework of the previous version, code may contain redundant remnants of previous software versions. Again this is unimportant and will not effect the software's performance.

For each software version that demonstrates something new, a number of experiments will be run to determine how to proceed with developing the next software version. Experiments will also be run and devised in an iterative method. Subsequent experiments will be devised based on the results of previous experiments from the current and previous software versions.

I will be conducting an initial literature review on *Basic Game Theory* and *Machine Learning*. Using this and my own observations and speculations from playing the game, I will develop a simple initial software version to confirm some basic hypotheses and further build upon.

For each software version and its experiments I will be documenting its purpose, design and the results of the experiments, as well as any changes made during development and the reasons for them. I will then summarise these results and conclude whether it has succeeded or failed in it's purpose. Using this summary I will outline what was learned and how I intend to proceed to the next version.

2 Background and Related Research

2.1 Introduction

Games have a variety of uses, from modelling real world behaviour such as biological or economical behaviour(Galindo and Tamayo 2000) to purely recreational purposes, both have been studied extensively. Here we shall look at what games have been studied, what approaches have been used and how effective they have been.

2.2 Game Theory

Game Theory is the mathematical study of strategic decision making, though originally used to address '*zero-sum games*', whereby the total benefit across all players of any action taken sums to zero, it has since been extended to games where the total benefit does not necessarily sum to zero or so-called '*general games*' (Neumann and Morgenstern 1944). These *games* have been used to study, model and predict patterns of behaviour in a variety of fields particularly biology and economics.

2.2.1 Definition of a Game

Before we can begin discussing games, we must first define the concept of a game and its components. This is important since in everyday language a '*game*' may refer to the rules and equipment required to play or a specific instance of play, if left undefined this ambiguity will lead to confusion later on. For this (and for most of this section on game theory) I shall be using the definitions used by Neuman and Morgenstern in *Theory of Games and Economic Behvaior* (1944).

Definition 1. Game
A *game* is simply the rules which describe it.

Definition 2. Play
A *play* is every particular instance at which the game is played, following the rules to completion.

Definition 3. Move
A move is the occasion of choice that leads to various different outcomes in play, this decision is to by made by either a player or some random element.

Definition 4. Choice
The specific outcome chosen in play is the *Choice*.

Definition 5. Strategy
The ideas that influence how the player makes their choices.

Remark. If we imagine then that a game is some tree T with the root being the beginning state and each leaf being an end state, then a play is some path P from the root to some leaf node. Each move in the game would correspond an inner node in T, and each choice in a play would correspond to an edge in P.

2.2.2 Types of Games

Zero-Sum and General Non-Zero-Sum Games

In Game Theory a *zero-sum game* is a game where the sum of all players' quantified advantages and disadvantages sum to zero. That is whenever a player gains an advantage the other opponent(s) will have a equivalent disadvantage. For greater than 2 player games the disadvantage is split between the other players. For example in a 3 player game, if player 1 gains some advantage with value 10, then players 2 and 3 will take a loss with value 10 split between them (e.g. player 2 will lose 3 and player 3 will lose 7).

Non-zero-sum games are then the set of games where the above *zero-sum* rule does not apply. Sometimes misleadingly this term refers to the set of both *zero-sum* and *non-zero-sum* games. To avoid this confusion the term *general games* is more commonly used to refer to these type of games. These *general games* can actually be reduced to *n+1 player zero-sum-games* making techniques for playing *zero-sum games* applicable for *general games* (Neumann and Morgenstern 1944, Chapter 11).

n-Player Games

Games vary considerably just by the number of participants playing them and must be considered accordingly.

One-player games cannot be *zero-sum*. The strategy involved is usually just making choices that maximise the players own outcome since there are no other players to consider. Though this may seem simple, *one-player games* (such as 'solitaire') often involve aspects of chance which make the outcome of choices more difficult to determine (Neumann and Morgenstern 1944, Chapter 3).

Two-player games, such as *Chess* or *Checkers*, are the more commonly studied and most well understood games in game theory. Effective strategies such as '*minimax*' (and it's variants) exist, though there is no known strategy for all *two-player games*. Since these games can vary so greatly, strategies for more complex games must be developed on a game-by-game basis depending on it's characteristics.

For **n-player games** where $n \geq 3$, strategies generally involve reducing or abstracting the game into a simpler version and using techniques appropriate to the simpler version to solve it. For example, for *zero-sum games* if we consider that players will form into two coalitions, then we can reduce the game to some *two-player zero-sum game* between the two coalitions (Neumann and Morgenstern 1944, Chapter 5).

Perfect and Imperfect Information

Games with *perfect information* are games whereby all players know all information from previously occuring states of a play (Osborne and Rubinstein 1994, Chapter 6). This should not be confused with what *can* happen, for example, games with an element of chance such as *Backgammon* can still be games of perfect information. Though players do not know what moves will be possible, they do know information about all events that have occurred previously. *Two-player zero-sum games of perfect information* are *strictly determined* (Neumann and Morgenstern 1944, Chapter 3).

Games of *imperfect information* are then games whereby not all information from previous states is known to all players. Games such as 'poker' or other games where each player has information only available to themselves are examples of this.

Sequential and Simultaneous Games

Sequential games are games where each player has knowledge of the other players' previous choices before making their own. This can be an advantage because a player can make a more fully informed decision before making their choice, however the player has the disadvantage that opponents can do the same effectively nullifying this advantage. Examples of this form of game include *Chess* and *Checkers*.

Simultaneous games are games where players make choices are the same time without knowledge of each others' decisions. Games of this type are of *imperfect information* and are thus subject to the same problems since they lack knowledge of the opponents choice when making their own. Examples of this form of game include *Prisoners Dilemma* and *Matching pennies*.

Deterministic and Stochastic Games

Deterministic games are games where a player can at all times predict with complete accuracy the outcome of their actions. A player does not necessairly need to know exactly the outcome of a full play from the beginning, but must be able to know exactly what actions the opponent(s) can take for any action or sequence of actions. Games such as *Chess* are deterministic since a player knows exactly what actions they can take and what responses an opponent can make and so on from the beginning. However the search space for these games is usually too large for a person to feasibly consider all possible sequences of actions.

Stochastic games or *games that involve an element of chance* are games where all outcomes cannot be predicted due to some form of randomness, such as a dice roll or random cards being drawn form a deck. This includes games such as *Backgammon* where available actions are dictated by dice rolls, or *Poker* where choice of actions is influenced by card draws. These games differ from *deterministic games* in that strategies have to take into account the randomness. This is usually done in the form of using probablities of random random events occurring to best predict the outcomes. A varition of *minimax* called *expectiminimax* is used for games involving chance (Shapley 1953, Russell and Norvig 2003).

Summary
Games require various different approaches based on the different combinations of characteristics that define the game. The characteristics of most **traditional and well understood games** studied such as 'chess' and 'checkers' are:

- *Zero-sum*
- *Two-player*
- *Sequential*
- *Deterministic*
- *With perfect information*

However, the characteristics of '***Why First?!***' are:

- *Non-zero-sum or general game* (Since more than one player may win)
- *(3–6)-player*
- *Simultaneous*
- *Stochastic*
- *With imperfect information*

It is this deviation from the traditional characteristics of more well known games that makes '*Why First?!*' interesting to study.

2.2.3 Minimax

Minimax Theorem states that for all *finite*, *zero-sum*, *two-player games* there exists an optimal mixed strategy (Willem 1996). Not all games ares *finite*, *zero-sum*, *two-player games*, but the general principles can be extended and applied to other types of games.

Maximin and Minimax
Using minimax, a player can find the maximum garunteed value of their next choice regardless of what actions the opponent(s) choose. Or find the minimum value they are garunteed to receive without opponent(s) know their intended actions. These values are calculated assuming that opponent(s) play optimally (Osborne and Rubinstein 1994).

A *ply* is a player's single *move* and the opponent(s) reply. To calculate the maximin value for a single ply, a player considers all possible actions available. For each action the player considers the opponent(s) reply. The player will then choose the action which results in the highest personal gain assuming the opponent is trying to minimize that value. To calculate the minimax value, the player considers the same but chooses the action which results in the opponents' lowest gain or highest loss assuming they are trying to maximize that value.

For most games multiple plys are considered before a decision is made. For smaller games such as '*tic-tac-toe*' the entire game tree can be considered before making even the first choice.

Minimax Algorithm

A typical Minimax algorithm (Russell and Norvig 2003, Chapter 6) for a *two-player* game would look as follows (code is based on *Python* syntax):

```
minimax(player, state):
        if state.endstate():
                # returns the value of an endstate for a specific player
                return value(player, state)

        if player == you:
                maxvalue = -∞
                for i in state.actions(player):
                        newstate = state.applyAction(i)
                        newvalue = minimax(opponent, newstate)
                        maxvalue = max(maxvalue, newvalue)

                return maxvalue
        else:
                minvalue = ∞
                for i in state.actions(player):
                        newstate = state.applyAction(i)
                        newvalue = minimax(you, newstate)
                        minvalue = min(minvalue, newvalue)

                return minvalue
```

Endstate values for a *zero-sum* game would typically be 1 for a win, 0 for a draw and -1 for a loss. Though for other games this value may indicate a player's final score for the play.

Variations

Though Minimax was originally designed for *two-player, zero-sum, deterministic games*, the principles have been applied to other types of games with variations on the Minimax algorithm.

For example, *Expectiminimax* treats events of chance as another player. Instead of calculating the minimum or maximum outcome of their *move*, Expectiminimax calculates the value of the expected outcome.

2.2.4 Conclusions

Though techniques in Game Theory such as a Minimax based approach may be effective in developing a strategy for '*Why First?!*', this project is not focussed on such techniques. Most techniques such as Minimax were developed with more traditional games in mind, and '*Why First?!*' exibits lots of characteristics not present in those games. Though I believe it is possible an effective strategy could be developed using this approach, I believe it would also be time

comsuming to reach a point where such an approach becomes effective. Since that is not the main focus of this project I will be disregarding it in order to dedicate more time to Maching Learning techniques which are the intended topic of study.

Given more time I would be interested to see how a Game Theory approach compares to the Machine Learning. I believe this would be a good direction to go in as an extension of this project for future work, though probably worth another project in it's own right.

Remark. Though in this project we shall be taking a more Maching Learning based approach than a Game Theory based approach, it is important to note the role Game Theory has played in research into solving and playing games and why it has be disregarded in favour of Maching Learning.

Figure 1: The 'Why First?' board with 3 player pieces and 1 of each of the different value cards.

2.3 'Why First?'

2.3.1 Overview

The game '*Why First?*' was chosen to study for several reasons. Firstly it is a previously unresearched and little known game that has no well known or documented strategy for winning. This means that if any good strategy does exist, we are not aware of it and so our approach in this project will not be influenced by it. The rules are simple enough to learn quickly, but the strategy involved is complex enough to be worth studying. The simplicity to understand is important due to the time constraints of this project, however a game too simple would be trivial to study for a masters level thesis. Here it can be argued to game of '*Why First?!*' strikes a good balance.

2.3.2 The Rules

The game requires 2-6 players (though with 2 players a 3rd 'random' player is added whereby each choice of this player is made at random) and each player gets a coloured token. The game board consists of 29 spaces in a path labelled -12 to 16 and there is a deck of 32 cards with a number from -4 to $+5$ on it, the distribution is as follows:

Card value:	-4	-3	-2	-1	$+1$	$+2$	$+3$	$+4$	$+5$
Card amount:	1	3	4	5	6	5	4	3	1

The struction of a *play* of the game to completion is as follows:

- A *play* consists of 5 *stages*
- Each *stage* consists of 5 *rounds*
- For each *play*:
  - each player selects a coloured token
  - each player begins with a score of 0 *points*
  - each *stage* of the *play* is then played in sequence
  - once the final *stage* ends, the player(s) with the second highest score win the *play*
  - if all players end the *play* with equal scores, no player wins the *play*
- For each *stage*:
  - each player places their coloured token on the 0 space of the game board
  - all the cards are placed in a deck and shuffled
  - each player randomly draws 5 cards from the deck, these cards are the player's *hand*
  - each *round* of the *stage* is then played in sequence
  - once the final round ends, the player(s) in the second highest position win the stage
  - all winners of the stage recieve points equal to the label of the space they are on, points are added to the player's current score for the *play*
  - if all players end the stage on the same space, no player wins the stage and no points are allocated
- For each *round* (except the final *round* of each *stage*):
  - players simultaneously select a card from their *hand* and a player to '*use*' that card on (players may select themselves)
  - all players' *choices* are then revealed at the same time
  - for each player, the sum of the value of the cards '*used*' on that player indicates the direction and distance that player must move along the board
  - once every player's new position on the board has been calculated, the current *round* ends and the next *round* begins
- For the final *round* of each *stage*:
  - players **must** '*use*' their final card on themselves
  - new positions are then calculated the same as previous rounds
  - once every player's new position on the board has been calculated the final round ends

2.3.3 The Scope of this Project: The 3-Player Single Stage Game

Due to the time constraints of this project we will not be considering the full extent of the game in its entirety. We focus on a significnatl sub-problem of the game that is '*winning a single stage*'. Though the full game offers interesting an strategic problem, I hypothesise that the key to winning a full play revolves around the deliberate winning and losing of stages or attempting to force certain players to win stages. Though there is some difficulty in planning which stages to win and lose or who should win each stage, the main problem lies in being able to intentionally win rounds. Since cards may be '*used*' freely on all players (with the exception to the final round), a strategy for winning a play can be generalised to forcing other players to win a particular stage. By this logic if we can force another player to win a stage we have developed a strategy to intentionally lose a stage. Intuitively it should be easier to develop a strategy to intentionally lose a stage since there are more positions which lose the stage than win.

Thus the main strategic difficulty lies in being able to *win a single stage*. By developing a strategy to do this we develop a large chunk of the strategy for playing the full game and form a good basis for any future work into this topic.

Furthermore, we shall only be considering the case where there are '*3-players*'. Increasing the number of players increases the amount of events that occur outside of a particular players control. That is randomness due to random card draws, and other players' choices in each round. Though this increases the challenge of the game, it is less interesting from a strategic development viewpoint since the strategy of a single player begins to have less influence on the outcome of the play. Though some effective strategies for *3-player* games may also be effective in *n-player* games, we shall demonstrate later that this is not necessairly the case.

2.4 Artificial Intelligence

Games have been the subject of much interest and research since the inception of the field in 1956 (Russell and Norvig 2003). Chess playing programs were one of the first problems to be studied in AI with continued interest in the subject still ongoing (Lai 2015).

Games are generally abstract versions of real world problems. Since all actions and interactions of each player are limited and defined by a strict rule set, game states are easy to represent and study, making them particularly appealing for AI research.

2.4.1 Machine Learning

Learning, as described by Mitchell (1997):

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Russell and Norvig (2003) classify 3 distinct types of machine learning: *supervised*, *unsupervised* and *reinforcement*.

Supervised learning requires feedback from a teacher (this can be a separate agent or the environment). The teacher provides examples of inputs with the correct output for the program to learn. For example, in a board game the teacher will provide the correct move for given board states. Using these learnt inputs the program will then attempt to provide the correct output for given inputs. This type of learning does require prior knowledge of what output is 'correct' for a given input. Examples of supervised learning include *Artificial Neural Networks* and *Decision Tree Learning*.

Unsupervised learning problems occur when no direct outputs are given and information about the environment are learned from patterns in the input. This is usually done through methods such as *Bayesian learning* or *MAP learning*.

Reinforcement learning is a more general type of learning whereby the program learns based on whether its actions result in success or failure, that is actions that result in success give the program some form of reward and actions that result in failure provide some form of punishment.

'*Why First?*' falls most naturally under the catagory of a **reinforcement learning problem**. It cannot be supervised learning since we are unaware of strategy to play the game well at this point, and hence cannot provide 'correct' moves or lines of play. Unsupervised learning would also be inappropriate since we're not looking for patterns in the input (or given board states). Reinforcement learning is appropriate however since we don't know how to play optimally, but we do get positive or negative feedback in form of a win or a loss.

2.4.2 Reinforcement Learning

Reinforcement learning differs from supervised learning in that specific training examples of input/output pairs are never explicitly given to the learner. Feedback may only be given after a series of actions, for example an entire play of a game to completion. Even then the feedback is only in the form of some positive *reinforcement* or negative *punishment* rather than a teacher stating specifically which actions should have been taken. The learner will then prioritise actions that have previously resulted in *reinforcement* and deprioritise actions that have previously resulted in *punishment*.

Multi-armed Bandit Problem
Also known as *exploration vs exploitation* problem. The problem is named after slot machines which are known as one-armed bandits, and a thus slot machine with multiple arms is a multi-armed bandit. The problem is usually posed in the analogy of a slot machine with multiple arms and the question of whether you choose to *exploit* an arm that has previously paid off a lot or *explore* other arms that are uncertain but might pay out even more.

Markov Decision Processes and Q-Learning
Markov decision proccesses (Puterman 1994) are the foundation for Q-Learning (Watkins 1989), a commonly used technique for reinforcement learning. In Q-Learning each action is given a reward and the agent will attempt to maximise their cumulative reward. For this game the reward is in the form of a win and so is not recieved until the end of play, therefore it is difficult to assign actions immediate rewards. For this reason I have chosen to not use Q-Learning here.

Passive and Active Learning
Passive learning is where an agent performs the same action for given situations. This form of learning is usually used to determine how well a particular action performs. *Active learning* is where an agent chooses different actions for given situations. This form of learning is more common and is used to teach an agent how to perform a certain task in a given enviroment.

For the task of teaching an agent how to play '*Why First?*' we will be using active learning.

2.4.3 Hyper-Heuristics

Hyper-heuristics themselves are not part of Machine Learning, though often incorperate Machine Learning into parts of their design. Hyper-heuristics are not designed to solve specific problems, but are rather a general tool for optimally selecting ways to solve problems. That is the will be several low-level-heuristics that actually perform some set of actions upon the enviroment, and the hyper-heuristic just concerns itself with how to select them. This is where hyper-heuristics generally incorperate some form of Machine Learning to learn how to best select which low-level-heuristics to favour, based on some factors such as provided training examples or observing and recording how well they perform (Russell and Norvig 2003, Chapter 4).

3 Understanding the Problem

3.1 Observations and Speculation

Even after playing many games with various people, no obvious effective strategy was apparent. This may be due to things such as making ineffective choices from lack of understanding. Since most people have not played this game before, they may be unsure as to what would be best to do each round o stage. For example, a player may forget that the player with the second most points wins the play and just attempt to win each stage, when in fact it is more beneficial for them to lose certain stages. Even if a player understands what they want to do at each stage, there may be difficulty in achieving their desired outcome. For example, player A may aim to force another player B to win that stage, but others may wish for player B to lose the stage, so player A may make ineffective choices due to lack of consideration of other players' strategies.

Due to the multiplayer nature of the game, especially when playing with greater numbers, goals within the play that benefit more players are more likely to get completed; and hence if a player is in a situation where their own goal opposes that of the majority, they're very likely to lose. Therefore, an effective strategy would appear to be to create situations where your opponents' best strategies are also ones that benefit you, whilst also minimising the number of opponents' strategies that hinder you.

This may be possible with some form if Minimax, possibly a probabilistic version due to having imperfect information. However, this assumes that opposing players will be playing effective strategies. When playing against humans, this is not always the case.

Another potentially effective approach may be some form of machine learning, where each board state maps to a choice and the program learns which choices are effective by playing and adjusts accordingly. An alternative to this could be where each choice for a board state has a probability of being chosen and the probabilities are adjusted after playing. This may be more appropriate due to the random nature of the game.

For this project I have chosen to look futher into Machine Learning since Minimax is typically designed for games with different characteristics.

3.2 Formulation and Representation

One of the difficulties in playing games is knowing what information is relevant. It is infeasible with modern technology to search through an entire game tree for games with lots of possible board states. An effective technique is the represent the board state in a more compact format that represents more important information. And indeed this is what will be implmented later in this project.

3.3 Measurement of Effectiveness

I have mentioned several times this notion of an '*effective strategy*'. For this particular game, a strategy will be deemed effective if it achieves a high win rate against other strategies. The

win rate of a player is their percentage of wins in a set of plays. Clearly this percentage gives a more accurate representation of the player the higher the number of plays.

If N players are using random strategies (picking each choice in a play at random), then each player can be expected to win $\frac{1}{N}$ of the time. Therefore, a strategy can be considered effective if it has a win rate that is significantly greater than $\frac{1}{N}$.

Another measure of an effective strategy may be *how much* a strategy wins or loses by. For a win this may be how many spaces the player is off from not being second. Or for a loss it may be how many spaces the player is from being second.

However with this approach you may have a strategy that wins more comfortably but less often, a strategy like this would be less reliable and hence less useful. One could argue that a strategy that wins more comfortably would be more likely to mitigate mistakes or the effects of chance and hence win more. But then the strategy we desire is still one with a high win rate and so it makes more sense for that to be our measurement of effectiveness.

4 Naive Approaches

So far I can only hypothesise what may work based on what I have personally observed. Though this has little merit by itself it gives me a basis in which to start experimenting and collecting data to either support or contradict these hypotheses.

4.1 Fixed Strategies

In this section I will use basic rule based strategies in a single stage of play to confirm the following hypotheses:

1. There exists a more effective strategy than completely random play.
2. There exists a less effective strategy than completely random play.
3. If a strategy is effective in an A-player game, it is not necessarily as effective in a B-player game. Where $A, B \in \{3, 4, 5, 6\}, A \neq B$.
4. There exists a strategy X and a strategy Y such that X performs better when Y is also in play.

To test the strategies I will be playing them against each other over 1000 rounds and recording their respective win rates. It should be noted that the win rates may not sum to 100% since more than 1 player may win a stage or no player may win.

The strategies I will be using for testing are as follows.

Random Play (R)
For each round

1. Select a card randomly from your hand.

2. Select a player randomly to use it on.

This strategy has no aim and is used as a baseline to test other strategies against.

Lowest First (LF)
For each round

1. Select the LOWEST card in your hand.
2. If that card is negative select the player in the LOWEST position.
3. Else if that card is positive select the player in the HIGHEST position.
4. If you have selected yourself select the next LOWEST/HIGHEST player.

This strategy aims to create a '*gap*' by pushing one player back and another forwards, leaving the highest card last to be played on yourself.

Lowest First Alternate (LFA)
For each round

1. Select the LOWEST card in your hand.
2. If that card is negative select the player in the HIGHEST position.
3. Else if that card is positive select the player in the LOWEST position.
4. If you have selected yourself select the next HIGHEST/LOWEST player.

This strategy is a slight variation on the 'Lowest First' strategy and is intended to be a demonstration of a '*bad*' strategy.

Highest First (HF)
For each round

1. Select the HIGHEST card in your hand.
2. If that card is negative select the player in the LOWEST position.
3. Else if that card is positive select the player in the HIGHEST position.
4. If you have selected yourself select the next LOWEST/HIGHEST player.

This strategy is intended to have a similar aim to 'Lowest First', but playing high cards first and leaving the lowest card last to be played on yourself.

Highest First Alternate (HFA)
For each round

1. Select the HIGHEST card in your hand.

2. If that card is negative select the player in the HIGHEST position.

3. Else if that card is positive select the player in the LOWEST position.

4. If you have selected yourself select the next HIGHEST/LOWEST player.

This strategy is a slight variation on the 'Highest First' strategy and is intended to be a demonstration of a '*bad*' strategy.

Highest Magnitude (HM)
For each round

1. Select the HIGHEST MAGNITUDE card in your hand.

2. If that card is negative select the player in the LOWEST position.

3. Else if that card is positive select the player in the HIGHEST position.

4. If you have selected yourself select the next LOWEST/HIGHEST player.

This strategy is intended to have a similar aim to 'Lowest First' and 'Highest First' By selecting the highest magnitude first it aims to have more influence on the board over the first few rounds leaving the lowest magnitude and least influential card to be played on yourself.

Lowest Magnitude (HM)
For each round

1. Select the LOWEST MAGNITUDE card in your hand.

2. If that card is negative select the player in the LOWEST position.

3. Else if that card is positive select the player in the HIGHEST position.

4. If you have selected yourself select the next LOWEST/HIGHEST player.

This strategy is intended to have a similar aim to 'Lowest First' and 'Highest First' By selecting the lowest magnitude first it aims to have more influence on the board in the later rounds of the stage however it leaves the highest magnitude and most influential card to be played on yourself.

4.1.1 Results

Winrate against 2 Random over 1000 stages (expected random winrate: 33.33%):

Lowest First	47.1%
Lowest First Alternate	26.7%
Highest First	42.6%
Highest First Alternate	24.0%
Highest Magnitude	58.9%
Lowest Magnitude	46.4%

Winrate against 5 Random over 1000 stages (expected random winrate: 16.67%):

Lowest First	32.3%
Lowest First Alternate	24.3%
Highest First	10.2%
Highest First Alternate	10.7%
Highest Magnitude	21.9%
Lowest Magnitude	29.2%

Winrate of each strategy (left) against each other strategy with a 3rd random player over 10000 stages:

	LF	LFA	HF	HFA	HM	LM
Lowest First	45.02%	65.31%	49.67%	45.48	24.01%	39.92%
Lowest First Alternate	14.20%	37.92%	6.06%	23.14%	6.08%	14.23%
Highest First	25.0%	40.68%	53.01%	76.38%	20.16%	27.38%
Highest First Alternate	12.88%	28.34%	14.01%	43.85%	10.51%	13.62%
Highest Magnitude	63.53%	64.40%	81.28%	66.82%	49.48%	54.74%
Lowest Magnitude	47.07%	61.22%	53.73%	50.68%	35.66%	42.56%

4.1.2 Conclusions

From the above results we can confirm the first 3 of all the stated hypotheses. LFA and HFA are demonstrations of (2) and all other strategies are demonstrations of (1). HF successfully demonstrates (3) since it performs better than random in the 3-player game but performs worse than random in the 6-player game. Though it would be unecessary to simulate all possibly strategy combinations in a 3-player game (216 sperate games excluding random). Running select combinations support hypothesis (4), for example LF seems to perform better in games where LFA is also present. And indeed where LF is normally beaten by HM when the third player is random with winrate of 24.01% to 63.53%, LF beats HM with a winrate of 50.4% to 34.6% when the third player is LFA.

From the confirmation of these hypotheses we have verified some important properties of the game. Though some of these properties may seem trivial, they confirm certain assumptions and give direction in how to proceed in developing a strategy for the game.

Firstly we have verified there exists a strategy better than random play. This confirms the assumption that there is some level of strategy involved to playing the game and the outcome is not purely random. This is important to verify so as to assure we are not wasting our time persuing a fruitless endeavor. This may not be immediately obvious due to the high amount of random elements involved.

Next, though less important but still worth considering, we confirm that there is a worse strategy than random play. For the full game it may be advantageous to lose certain stages. Though not considered here this aspect may be important for future work on the full 5 stage game. We have also demonstrated strategies that work well for the 3-player game do not necessarily work well for the 6-player game. Again this is less important in this project, but shows that this needs to be taken into consideration when studying the full game with more players.

Finally we demonstrate that certain fixed strategies work better (or worse) in conjunction with other strategies. The implications of this is that fixed strategies are likely less effective than dynamic strategies. For example, when in a game where an opponent is playing a strategy such as LFA, a strategy such as HM (though generally good) does not take advantage of the opponent's strategy and loses to a strategy that does i.e LF. An effective dynamic strategy should be able to take advantage of such situations or at least mitagate the advantage gained by strategies such as LF.

5 Learning Approaches

From here we have more insight into the game and a sense of direction in which to proceed. We will be using a machine learning based approach to automatically *learn* to play against and adapt to different strategies. In particular we will be using a reinforcement learning based approach since this most appropriately fits the problem at hand.

5.1 Predefined Mixed Strategies

In this section I will be designing an agent that selects one of the fixed strategies from the previous section and uses reinforcement learning to select the most effective strategy(s) for given fixed opponents. By doing this I shall verify my hypothesis that *a dynamic mixed strategy is more effective than any given single strategy.* I do this to demonstrate and test the effectiveness of reinforcement learning at its most basic level.

During this phase of development I looked briefly into *hyper-heuristics* (Edmund Burke 2003) as inspiration for learning techniques. Though the emphasis in this project is more on Machine Learning, this version of the software clearly follows a similar structure as it has been influenced and inspired by *hyper-heuristics.*

5.2 Random Probabilistic Selection (RP)

At the beginning of each playing session:[1]

- Create a weighted list of all predefined strategies where each strategy has EQUAL weight

then, do the following for each round:

1. Before the round begins, randomly select a strategy from the list with probability respective to the strategies' weights
2. Play the round according to the rules defined by the strategy selected
3. If you win the round, increase the weight of the strategy selected in the list
4. Else if you lose the round, decrease the weight of the strategy selected in the list unless the weight is already at the lowest threshold

5.2.1 Implementation

In the previous version, each player had a fixed strategy defined at the start of each play and the implmentation worked in the following manner:

1. At the beginning of each stage, the '*game*' generates the deck and gives each player their 5 cards
2. Then for each round

[1] By playing session I mean a session of subsequent plays of the game to completion.

(a) The '*game*' asks each player for a '*choice*' in the form of a 2-tuple of (player, card)

(b) The '*game*' then does the necessary calculations and provides the necessary information to the players for the next round

In this implementation the player has no opourtunity to change strategies or make any strategic decisions before the beginning of a new stage or between stages. This gives rise to the new implmentation whereby the '*game*' asks for a '*high-level choice*' (referred to as '*hyper_play*' in the code) before each stage, and a '*low-level choice*' for each round.

The new implmention for this version each player has a predefined '*high-level strategy*' (referred to as *hyper_strategy* in the code). Players still using a fixed strategy also have a predefined '*low-level strategy*'. The implementation works as follows:

1. Before each stage the '*game*' asks each player to perform a '*high-level choice*'
2. At the beginning of each stage, the '*game*' generates the deck and gives each player their 5 cards
3. Then for each round
   (a) The '*game*' asks each player for a '*low-level choice*' in the form of a 2-tuple of (player, card)
   (b) The '*game*' then does the necessary calculations and provides the necessary information to the players for the next round

For the **Random Probabilistic Selection**, the '*high-level choice*' is the weighted selection of a predefined strategy from the weighted list to be used for the rest of the stage. I also created a high-level strategy called **Single Player (SP)** that simply selects the same low-level strategy each stage for using fixed strategies in this implmentation.

5.2.2 Random Weighted Selection Algorithm

This particular problem posed the biggest challenge in this version of the software. Though there exists support for random selection in Python's standard library, there exists no in-built functions for random weighted selection. Several approaches were tried during this development phase. Though I managed to get a solution that works while developing this version, the most optimal solution I created was not developed until a later development phase.

My **first approach** was to first assign each low-level strategy an initially uniform percentage. To do this I created a Python *Dictionary* with the low-level strategies as *keys* and the percentage of being picked as *values* represented as floats.

```
self.heuristic_options = {
        'R': self.random_play,
```

```
        'LF': self.low_first,
        'LFA': self.low_first_alt,
        'HF': self.high_first,
        'HFA': self.high_first_alt,
        'HM': self.high_mag,
        'LM': self.low_mag
        }

prob = 1/len(self.heuristic_options) #initial probability

self.heuristics_by_name = {}
for i in self.heuristic_options:
        self.heuristics_by_name[i] = prob
```

I then used these values to generate a second *Dictionary* where the *keys* were the *cumulative values* of the percentages, and the *values* were the low-level strategies themselves.

Standard Python *Dictionaries* are orderless so it was necessary to implement the second *Dictionary* as an *Ordered Dictionary* which is available in the Python standard library.

```
total_prob = 0
temp = {}
for i in self.heuristic_options:
        total_prob += (self.heuristics_by_name[i])
        temp[self.total_prob] = i

self.cumulative = OrderedDict(sorted(temp.items()))
```

To select a strategy I then generated a random number n, $n \in [0, 1]$. Then for each *cumulative value* in order, if n is less than that value, select the strategy it corresponds to.

```
options = {
        'R': self.random_play,
        'LF': self.low_first,
        'LFA': self.low_first_alt,
        'HF': self.high_first,
        'HFA': self.high_first_alt,
        'HM': self.high_mag
        }

num = uniform(0,1)
print(num)
for i in self.cumulative:
        if num < i:
                self.low_name = self.cumulative[i]
```

```
self.low_heuristic = options[self.low_name]
```

After a win, the percentage of the strategy used was increased while decreasing all other strategies, while the inverse was done for a loss. Particular care was taken to ensure the totals always summed to 1. This approach was particularly expensive since the second *Dictionary* had to be recalculated after each stage. Furthermore, the approach proved to be ineffective with winrates equivlent to that of random play. Despite trying many varations of combinations of *reward* and *punishment* values, the algorithm never seemed to converge on a good low-level strategy.

After this failure I reevaluated the algorithm and had another look at how *hyper-heuristics* work.

For my **second approach** I changed to a *score* based approach rather than using percentages. After studying the previous algorithm I deduced that several of the problems were due to the way I had implmented the weights of each low-level strategy.

Firstly, the way I had used the percentages limited how well a certain strategy could be regarded. What I mean by this is that at a certain point, either the percentage of the winning strategy became so high or the percentages of the other strategies became so low they could not be increased or decreased any more respectively. The result of this is that after many plays, if a strategy had done well, further wins would not increase its likelyhood of being picked while losses still did. Also, for strategies that had done badly, further losses were not punished, but wins were taken into account. For the *score* based approach, a strategy can have its score increased infinitely (or rather up to the limitations of the machine). Increasing one strategy's score implicitely decreases another's even when that strategy has a score of 1. The result of this is that even after many plays, the program is still learning.

Furthermore, the way rewards and punishments were implmented meant that a win in the first play had the same impact as a win in the 1000th. Intuitively we can see that, for example, after 900 wins, 10 subsequent losses should not reduce the likelyhood of being picked so heavily. Similarly, if one strategy has 900 wins and another 0 wins, should the latter win 10 sebsequent plays in a row, it should not be made so much more desirable. With the *score* based approach, as the total score across all strategies increases, arbitary increases and decreases begin to have less impact on the probability of a strategy being picked. The effect of this is that it naturally decreases the learning rate as more plays are completed and a good strategy is determined.

```
score = 10 #initial score

self.heuristics_by_name = {}
self.max_score = score
for i in self.heuristic_options:
        self.heuristics_by_name[i] = score
```

Initially I used to same method of using a second *Ordered Dictionary* with *cumulative scores*. Though still more effective, this still suffered the same problems of inefficiency.

My **final approach** in this development phase I managed to improve the effciency drastically and remove the need for the second *Ordered Dictionary*.

Keeping the first *Dictonary* the same, I altered the way the algorithm selected a low-level strategy from the weighted list. The new selection method is as follows:

1. Let T be the *total score* across all strategies
2. Generate a random integer r between 0 and T
3. Iterates through the *Dictionary*
   (a) If r is less than the score select that low-level strategy
   (b) Else $r = r - score$

As implemented:

```
num = uniform(0,self.total_score)
for i in self.heuristics_by_name:
        if num < self.heuristics_by_name[i]:
                self.low_name = self.heuristics_by_name[i]
                self.low_heuristic = options[self.low_name]
        else:
                num = num - self.heuristics_by_name[i]
```

While this is an improvement over the previous algorithm, it often has to iteration through a large portion of the *Dictionary* before finding the strategy to select. This is reasonable for this version where there are only 6 strategies to choose from, however we shall later see this is not reasonably fast enough when running a large number of plays.

5.2.3 Experiments and Results

Now we have established the implementation we look at measuring the effectiveness of the approach and fine tuning the variables to better improve performance. I performed a series of experiments to test the effectiveness when adjusting the following variables:

- **Plays**: the number of plays completed and *recorded* to test the effectiveness of the agent with the particular settings.
- **Warmup**: the number of plays completed and *unrecorded*. These plays give the agent time to learn.
- **Initial Score**: the weight all fixed strategies start with. Increasing this means less-used strategies have a higher chance of being picked later on.
- **Reward/Punishment**: the amount each strategy's weight is adjusted by upon a win/loss.
- **Opposition**: the strategy the agent is playing against.

For each variable I performed a set of experiements, adjust only one or two of the variables while keeping the rest the same. Settings that performed the best were carried over to the next set of experiments as the baseline.

Plays and Warmup

The aim of this set of experiments is to find a suitable number of *plays* that accurately measures the effectiveness of the agent and a minimal number of warmup plays that give the agent enough experience to perform optimally. Clearly we want these numbers to be as small as possible to save on computing time, but large enough to perform the task required.

Fixed Variables:

Initial Score:	10
Reward/Punishment:	2/1
Opposition:	R/R

Experiment Results:

Plays	Warmup	Winrate	Favoured Strategy	Score
100	0	42.0%	HM	37
1000	0	53.7%	HM	433
10000	0	54.96%	HM	5713
10000	10000	57.69%	HM	12964
10000	100000	58.12%	HM	74907
10000	1000000	58.83%	HM	723293

From the results we conclude that 10,000 plays is enough to accurately measure the effectiveness of the agent. We can also see that 100,000 play warmup is enough to get the most optimal performance out of the agent in a reasonable time frame. The 1,000,000 play warmup did increase the winrate, though not marginally and took significantly longer to run.

Initial Score

The aim of this set of experiments is to find how much to choose *exploration* over *exploitation*. This is an example of a *multi-armed bandit* problem, since higher values of the inital score lead to favoured exploration whereas lower values lead to exploitation. This is a problem of balance, too high values will lead to the agent more frequently exploring poor options, whilst too low values may cause the agent to begin to attempt to exploit poor options without considering good ones.

Fixed Variables:

Plays:	10000
Warmup:	100000
Reward/Punishment:	2/1
Opposition:	R/R

Experiment Results:

		Strategy Scores						
Initial Score	Winrate	R	LF	LFA	HF	HFA	HM	LM
1	56.25%	7	2879	1	531	2	56852	12245
5	58.54%	13	2293	1	101	3	76145	1906
10	57.64%	24	3582	1	411	1	74847	800
25	57.58%	106	3361	16	434	1	71491	1625
50	57.61%	121	3053	15	369	4	73711	2044
100	57.27%	257	5541	20	686	5	64966	4264

From the results we can see that 5 appears to be the most optimal initial score. It appears from the table that an inital score of 1 led to the agent attempting to exploit LM, though a good strategy this strategy is not optimal. As we increase the initial score past 5, the agent explores sub-optimal strategies more and more. This can be seen in the table where as the initial score is increased the score of HM decreases, while the score of all other strategies increases.

Reward/Punishment

The aim of this set of experiments is to find how much to *reward* a win by and how much to *punish* a loss by. Here we want to sufficiently reward wins so the agent learns a correct strategy. However, since random play has a 33% winrate against itself, we don't want to reward wins so much that bad strategies that get the occasional win become more prioritised. Likewise we wish to punish bad strategies, but not punish good strategies too much for the occasional loss.

Fixed Variables:

Plays:	10000
Warmup:	100000
Initial Score:	5
Opposition:	R/R

Experiment Results:

			Strategy Scores						
Reward	Punishment	Winrate	R	LF	LFA	HF	HFA	HM	LM
1	1	57.88%	3	2	1	1	1	18049	15
2	1	58.48%	14	1419	1	9	1	80967	15
1	2	43.66%	2	3	3	3	1	5	2
5	1	49.27%	3388	30187	45	18	281	12688	162628
1	5	42.90%	3	6	2	1	3	2	1
3	1	57.99%	63	725	16	5796	14	134420	243
3	2	58.11%	2	63	1	146	4	98431	1297

From these results we can conclude several things. Firstly, having a higher punishment than reward is clearly a poor choice of variables. We can see from the table that the agent plays significantly worse and does not converge on any strategy. Next we can see that having higher difference between the reward and punishment values means that the agent is more forgiving of

losses. The effect of this is that the agent explores other strategies more depite losses. Later we shall see that against more difficult combinations of opponents this is actually beneficial, since even the most optimal strategy still suffers losses the agent punishes these losses less and still identifies the optimal strategy. Both the reward/punishment combinations of 2/1 and 3/2 seem to perfiorm well, converging on the optimal strategy while still exlporing some of the better strategies. We shall arbitrarily pick 2/1 to proceed with, though it does have a slightly higher winrate this is not actually significantly higher than either 3/2 or 3/1.

Opposition

The aim of this set of experiments is to see how the agent fairs against various opponents with the settings based off the previous experiments. These experiments are more to test how effective the agent is rather than being used to tune the settings.

Fixed Variables:

Plays:	10000
Warmup:	100000
Initial Score:	5
Reward/Punishment:	2/1

Experiment Results:

			Strategies			
Opponents	Opponents' Winrate	Agent's Winrate	Favoured	Score	2nd Choice	Score
R/R	22.92%/24.88%	57.79%	HM	71545	LM	5090
HM/R	48.69%/6.99%	49.28%	HM	47431	LM	2734
LM/R	35.88%/13.89%	53.42%	HM	68451	LF	42
LF/R	27.29%/14.20%	60.62%	HM	93088	LM	995
HF/R	19.59%/6.43%	78.41%	HM	111891	LM	23012
HM/HM	42.87%/40.98%	18.75%	HF	3	ALL	1
LM/LM	27.68%/27.34%	46.53%	HM	29558	LF	11692
LF/LF	30.84%/29.15%	40.56%	LF	17434	HM	1661
HF/HF	28.20%/25.83%	46.11%	HF	7275	HM	59
LF/LFA	57.59%/1.33%	56.16%	LF	64103	LM	9812
HF/HFA	42.15%/15.56%	42.29%	HF	28532	ALL	1
RP/R	48.16%/8.04%	48.43%	HM	33857	LM	5691
RP/RP	33.16%/34.70%	32.67%	HM	4501	HF	278

The combinations chosen are ones known to be effective. The agent performs well against all tested combinations except HM/HM. For those strategies the agent's choice converges on the most appropriate strategy for the opponents, with its second choice generally with a significantly lower score. However for the HM/HM combination we can see from the strategies favoured that it does not converge on any strategy. This can be expected since if the agent picks HM every time, it can be expected to lose two plays for every one win. With the reward/punishment values as they are the expected change in score for this would be 0.

Beating HM/HM

Though the agent performs well against most other strategy combinations, against HM/HM it seems to struggle. This experiment aims to set the variables in such a way that it can beat HM/HM. Since this combination of opponents highlights a weakness of the agent, we intend to determine why this is in order to correct and hopefully improve the performance of the agent.

Fixed Variables:

Plays:	10000
Opposition:	HM/HM

Experiment Results:

Warmup	Initial Score	Reward	Punishment	Winrate	Opponents' Winrate
100000	5	2	1	17.28%	43.29%/42.25%
1000000	5	2	1	18.92%	41.98%/41.82%
100000	10	2	1	16.14%	42.64%/43.95%
100000	5	5	1	33.72%	33.03%/33.81%
100000	5	5	3	15.71%	43.03%/44.28%
100000	5	10	1	32.21%	36.12%/32.32%
100000	7	2	1	33.97%	33.79%/32.90%

I tried several variations in order to improve the strategy here. Firstly, increasing the warmup time seems to slightly improve the winrate, though not significantly. Next, I tried increasing the intial score, this decreased the winrate so was returned to its original value. As expected, the solution lay in increasing the reward. The reasoning behind this being that the agent was expected to lose every 2 out of 3 games. With the previous reward/punishment values the expected score change every 3 games was 0. With the new values of 5/1 and 7/1 this increased to 3 and 5 respectively. There was no significant difference in the winrate between the two so 7/1 was chosen arbitrarily.

Reapplying the New Settings

Now we've found settings that make the agent perform more optimally against the combination HM/HM, we wish to rematch the agent against other combinations of opponents with the new settings to see if it still plays optimally against those.

Fixed Variables:

Plays:	10000
Warmup	100000
Initial Score:	5
Reward/Punishment:	7/1

Experiment Results:

Opponents	Opponents' Winrate	Agent's Winrate	Agent's Previous Winrate
R/R	23.81%/24.24%	57.24%	57.79%
HM/R	48.26%/9.18%	46.81%	49.28%
LM/R	35.71%/15.42%	51.86%	53.42%
LF/R	37.05%/22.21%	43.35%	60.62%
HF/R	20.28%/6.18%	78.20%	78.41%
HM/HM	32.94%/33.22%	34.35%	18.75%
LM/LM	27.13%/28.43%	46.00%	46.53%
LF/LF	40.85%/41.04%	18.52%	40.56%
HF/HF	25.54%/29.62%	45.05%	46.11%
LF/LFA	56.19%/1.88%	56.56%	56.16%
HF/HFA	42.53%/16.25%	41.22%	42.29%

The new settings allow the agent to beat most other strategy combinations, though some have slightly lower winrates there is no significant difference. Though the agent now performs well against HM/HM it appears to perform worse against combinations containing LF. It is difficult to say whether or not this agent is an improvement over the previous one, since it swaps one weakness for another.

5.2.4 Conclusions

The approach in section performs well against a range of fixed strategies and stratey combinations. It performs better than the fixed strategy HM since it adapts to situations where HM is weak. There are certain combinations of opponents that the agent performs poorly against, further tuning of the variables may further improve performance in these situations. However due to the time constraints of this project the decision was made to persue different, more promising approaches rather than perfect this prototype.

5.3 Generated Mixed Strategies

From the previous section I attempted to generalise the approach and generate fixed strategies that the agent then selected using the same random probabilistic selection. Strategies that performed well would then be prioritised in the same way. I defined these strategies in the form of *what cards would be chosen each round* and *what player would be chosen each round*. Cards were defined by their position in the order of the hand, and players were chosen by what rank they were on the board. However both approaches attempted in this manner failed. Since there were few significant results from this phase of development I shall only breifly outline approach.

Generating Strategies by Parts

For my first attempt the agent would select three lists before each round. These were:

- What cards to choose for each round
- What player to choose if that card was positive for each round

- What player to choose if that card was negative for each round

Despite lots of experiments run to adjust variables, the agent never performed better than random play. I hypothesised that certain combinations of these *parts* did not work well together and was the cause of the failure.

Generating All Combinations of Strategies by Parts
To attempt to fix the aforementioned problem I generated all possible combinations of *parts*. The agent then selected one of these combinations before each stage and played accordingly.

Due to the large search space of possible strategies, plays were being completely relatively slowly. I hypothesised that performance would increase with a greater number of Warmup plays since only a small sub-section of the possible strategies were being selected each playing session. My hypothesis was that the agent would need significantly more training iterations than there were strategies. However as the approach was implemented this was infeasible.

5.3.1 Random Weighted Selection Algorithm Revisited

By timing various parts of the program, I discovered that the majority of the running time was spent in the random weighted selection.

With the implementation as it was, when all strategies have equal weights, it could be expected that the algorithm will select a strategy in the latter half of the list half of the time. This means that half of the time the algorithm has to iterate through at least half of the list. With the search space so large this was very inefficient way to select a strategy.

To remedy this I reimplemented the algorithm as follows:

1. Generate a number between 0 and the maximum score
2. Randomly select a strategy from all possible strategies
3. If the score of that strategy is greater than or equal to the number select that strategy
4. Else go to 1

For this each strategy has an equal probability of being picked, then a probablility of $score/total\_score$ of being selected to be used. By artificially assigning different scores to a list of items and using the new random weighted selection algorithm to select a large number of items, I verified that indeed items were selected with frequency proportional to their scores.

This was a significant improvement over the previous implementation and allowed the agent to play significantly more training plays. However the agent still did not perform better than random play.

5.3.2 Conclusions

Using the representation of the strategies defined, I tested strategies similar to fixed strategies that have proved to be effective. I found that the agent still did not perform significantly better

than random play. The only difference between the strategies defined in this section and the fixed strategies defined in section 4.1 is that these strategies did nothing to consider their own position on the board. I conluded that the agent must take this into consideration to play the game effectively, and that naively deciding what choices to make before the stage begins is ineffective.

Though these approaches failed, I gained valuable insight into how the game is played effectively and applied this knowledge to my later approaches. The random weighted selection algorithm developed during this phase was also used in my final implementation.

5.4 Developing Strategies from Experience

In the previous section the agent did not consider its own position on the board. This taught us that it is too naive for an agent to select all its moves before the stage begins. In this section I aim to create an agent that considers its board state and makes decisions appropriately. The difficulty here lies in creating a consise enough representation of the board state such that there aren't too many states to record. Having too many states means more plays will have to be completed for the agent to reliably visit the same state frequently and in turn learn the correct action. Having too few states however means the agent does not gain enough information from the board state to make an appropriate decision. That is it will not distinguish between certain board states that require separate actions to play effectively. From these board states the agent will assign scores to all possible actions available. It will then use the random weighted selection algorithm from previous sections to select an action. Choices in each play will be recorded and the scores of those plays will be increased upon a win and decreased upon a loss.

By developing this agent I shall more fully demonstrate the effectiveness of reinforcement learning for these classes of games.

5.4.1 Board State Representation

For this approach I have chosen that the board state will consist of the following:

- **The player's hand:** this will be in the form of a list of the card's values in order from lowest to highest.
- **A *catagory* of position combination:** this will be a number representing a group of possible position combinations. For example, 0 will represent that all players are in the same position and 4 will represent that the opponents are in tied position and the player is ahead of them.
- **The player's relative position to the frontmost opponent:** this will be a value represnting the distance from the frontmost opponent. Positive numbers represent that the opponent is ahead of the player, while negative numbers representive the opponent is behind the player.
- **The player's relative position to the rearmost opponent:** this will be a value represnting the distance from the rearmost opponent. Positive numbers represent that

the opponent is ahead of the player, while negative numbers representive the opponent is behind the player.

From these features, the agent can infer the following unstated things about the state of the board:

- **Round number:** this is inferred from the number of cards in hand.
- **The player's rank:** this is inferred from the relative positions.
- **The opponents' rank:** this is also inferred from the relative positions.
- **Distance from a particular rank:** for example second. Inferred from the relative positions.
- **Limited information on what cards have been played:** for example, if plays are relatively far apart in an early round, this infers that large magnitude cards have been played.

Effects on Learning
Information unavailable to the agent is exactly what cards have been played and on who. Though knowing exactly what cards have been played may give the agent an edge, it would significantly increase the number of board states to consider and thus slow down learning. Knowledge of what cards have been played on who would perhaps give the agent an idea of the opponents' strategies, however guessing and playing around strategies is beyond the scope of what this approach intends to do.

Though the *position catagory* infers no further information, and indeed can be calculated based on relative positions, it makes calculations easier and removes some symmetry when considering actions in states where opponents are on the same space. This means less actions are considered for those states and speeds up learning. For example, if player one is considering a board state where players two and three are on the space, then choices to play cards on the frontmost or rearmost player are equivalent. By storing these actions as the same action that makes a random choice of opponent, we remove half of these actions from consideration.

Also, by considering the opponents as the *frontmost* and *rearmost* opponents we cut the number of boardstates stored significantly. This representation removes the symmetry of where there are two states with the opponents in opposite positions. For example, player one is considering the state where player two is 2 spaces ahead and player three is 2 spaces behind. If player one has not encountered this state before, but has encountered the state where player three is 2 spaces ahead and player two is two spaces behind, by considering both as the same player one has more knowledge of what action to take.

5.4.2 Reinforcement Learning from Board States (HL)

At the beginning of each playing session:

- Create an empty dictionary of board states. This dictionary will contain a mapping from board states to a weighted list of possible actions.

then, for each round:

1. Convert the available information to a board state representation as defined.
2. If this board state is not in the dictionary:
   (a) Add the board state to the dictionary.
   (b) Create a weighted list of all possible actions, where each actions has EQUAL weight, that the board state maps to.
3. Find the weighted list of possible actions associated with that board state in the dictionary.
4. Randomly select an action from the list with probabilities respective to the action's weight.
5. Record what action was taken for that board state.
6. Interpret the action and make the choice associated with it.

after the stage:

1. If you win the round: increase the weights of the actions recorded for their respective board states.
2. Else if you lose the round: decrease the weights of the actions recorded for their respective board states, unless the weight is already at its lowest threshold.

5.4.3 Implementation

Upon the players creation the implementation does some initialisation in form of creating a Dictionary to hold the encountered board states and setting the variables for learning.

The indices of this Dictionary are the various board states, these are added when encountered. The values for each board state are another Dictionary. For these Dictionaries, the indices are the actions available to the player for the board state the references them. While the values are the weights in the form of scores of each action. These second Dictionaries each also have two 'special' entries. The first is a list of the actions available, this is to allow the Random Probabilistic Selection Algorithm to iterate through the actions to select one. The second is the *maxium score* of all actions for that board state, this is updated upon wins and is also used in the Random Probabilistic Selection Algorithm.

Using the same framework as previous versions, for the *high-level choice* the player simply creates an empty list to record what actions they take for each board state.

Board state catagories are defined as follows:

0 if ranks equal
1 if all ranks unequal, player in rank 1
2 if all ranks unequal, player in rank 2
3 if ranks unequal, player in rank 3
4 if opponents tied, player in rank 1
5 if opponents tied, player in rank 2
6 if opponent and player tied, player in rank 1
7 if oppenent and player tied, player in rank 2

For the *low-level choice* the implementation works in the following manner:

1. Get the required values of the board state:
   (a) Find the ranking of the player and the opponents.
   (b) Sum the ranks of all players.
   (c) Based on the sum of the ranks and the player's own position, determine the catagory of board state.
   (d) Find the relative positions of the opponents.[2]
   (e) Create an ordered list of the values of the cards in the player's hand.
   (f) Combine these elements to get the representation of the board state.
2. If the board state is not in the Dictionary of board states:
   (a) Get possible actions available for that board state:
      i. Create an empty Dictionary of possible actions.
      ii. Add to this Dictionary the *key* 'max_score' with the *value* the defined intial score.
      iii. Add to this Dictionary the *key* 'keys' with the value an empty list.
      iv. If catagory is 0, 4 or 5: player choices are Yourself or Opponent[3].
      v. Else: player choices are Yourself, Frontmost Opponent or Rearmost Opponent
      vi. For all possible combinations of player choice and card choice: add to the dictionary the *key* (card choice, player choice) with *value* as the defined inital score. Also append the *key* to the list of 'keys'.
   (b) Add these possible actions to the Dictionary of board states.
3. Using the Random Probabilistic Selection Algorithm, select an action from the list of actions for that board state.
4. Convert action selected to a 2-tuple of (player, card) .
5. Return that 2-tuple.

[2]This is calculated differently based on the catagory.
[3]Random choice of opponent

5.4.4 Experimentation and Results

With the algorithm established we now work on fine tuning the variables and improving performance. As well as considering the variables from previous versions: Games/Warmup, Initial Score, Reward/Punishment and what Opponents the agent is being *tested* against[4]. We also consider what Opponents the agent is *trained* against.

Plays and Warmup

We have established in previous sections that 10,000 plays is enough to accurately verify the effectiveness of an agent at that state in its training. Since the agents are learning by board states and there are so many board states to consider, effectiveness will rely heavily on the number of Warmup plays. Due to this we will leave these experiments until after all other variables have been calibrated.

Initial Score

Again these experiements aim to adjust how much the agent chooses *exploring* new options over *exploiting* previously rewarding options. Since the search space for each individual board state is smaller[5] it is likely that the agent will favour *exploitation*.

Fixed Variables:

Plays:	10000
Warmup	100000
Reward/Punishment:	2/1
Opposition:	R/R

Experiment Results:

Initial Score	Winrate
1	40.64%
5	38.22%
10	36.44%
25	35.98%
50	34.62%
100	34.87%

The initial score that provided the best performance was that of value 1. However, having an initial score of 1 means that there is no punishment for losing a stage if that is the first time a board state has been encountered. Due to this I have chosen to use an initial score of 5.

Reward/Punishment

The aim of these experiements is to balance *reward* and *punishment*. As with previous sections, I have tested a varity of combinations though we desire a combination that sums to an overall increase for 1 win for every 2 losses. This is since when playing against an optimal strategy, the agent can be expected to win at most 1 in 3 plays if also playing optimally.

[4] See section 5.2.3 for explanation of variables

[5] maximum of 12 possible actions.

Fixed Variables:

Plays:	10000
Initial Score:	5
Opposition:	R/R

Warmup:		100000	200000
Reward	Punishment	Winrate	
1	1	36.96%	38.30%
2	1	38.94%	40.39%
1	2	37.66%	39.45%
5	1	39.83%	42.70%
1	5	34.45%	36.28%
5	2	40.17%	44.01%
5	3	40.27%	43.34%
5	4	40.40%	43.09%
7	1	39.34%	41.59%
7	3	41.32%	42.85%
7	5	39.00%	41.82%

For 100,000 Warmup plays the best combination of values is 7/3. However for 200,000 Warmup plays 5/2 performs better. Both combinations perform well since they sum to an increase in the case of 1 win in 3 plays. However since 5/2 performs better with a larger number of Warmup plays I have chosen this to proceed with.

5.4.5 Training Against Different Opponents

For these experiments I have generated different strategies based on amount of Warmup plays and Opposition trained against. The aim of this is to demonstrate how much increasing the number of Warmup plays can improve performance aswell as attempting to determine the most effective opposition to train the agent against. To generate these strategies, I have run the program with the desired settings and saved the Dictionary of board states to a binary file. To run the experiments, the program loads Dictionary into memory and then performs the set plays with the desired settings without further Warmup.

Fixed Variables:

Plays:	10000
Initial Score:	5
Reward/Punishment:	5/2

Training Against Two Random Players (RR)

Training against two random players provides the agent with a wide variety of board states to learn, since random play may lead to board states not usually seen when playing against more sophisticated strategies. However the agent may not learn optimal actions since sub-optimal actions may be enough to win the stage against these opponents.

	Warmup: 100,000		Warmup: 500,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	33.34%/33.86%	40.37%	28.48%/29.97%	49.25%
LF/R	45.04%/27.04%	32.81%	39.49%/25.55%	40.51%
HF/R	41.91%/28.50%	35.83%	37.55%/25.84%	42.86%
HM/R	55.99%/22.07%	27.48%	51.05%/20.75%	34.63%
LM/R	44.88%/27.70%	32.65%	39.65%/24.67%	41.72%

	Warmup: 1,000,000	
Opponents	Opponents' Winrate	Agent's Winrate
R/R	28.56%/27.55%	51.72%
LF/R	38.24%/24.18%	43.19%
HF/R	34.33%/24.68%	46.95%
HM/R	47.45%/20.19%	39.21%
LM/R	36.39%/24.47%	45.30%

	Warmup: 5,000,000		Warmup: 10,000,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	25.21%/24.14%	57.21%	23.28%/23.90%	59.97%
LF/R	32.95%/21.51%	51.60%	30.60%/21.63%	54.51%
HF/R	29.13%/20.67%	55.42%	27.76%/20.00%	57.30%
HM/R	40.99%/18.01%	47.72%	38.18%/17.82%	50.55%
LM/R	32.57%/21.51%	52.02%	31.35%/20.46%	54.45%
LF/LF	34.57%/34.23%	34.37%	33.04%/33.39%	36.61%
HF/HF	36.05%/36.81%	33.08%	35.22%/35.43%	35.70%
HM/HM	36.36%/36.37%	31.18%	35.84%/35.38%	32.43%
LM/LM	33.01%/32.24%	38.68%	30.82%/30.83%	42.18%
LF/LFA	43.31%/12.74%	54.05%	42.87%/12.43%	55.41%
HF/HFA	40.24%/17.59%	45.62%	37.93%/17.83%	47.49%

Training Against One Other Learner and One Random Player (LR)

Training against one other learner and one random player may provide enough challenge to the agent that it must learn optimal actions, while still being provided with a wider variety of board states. However for more common states it may not learn as optimal actions as when playing against two more optimal playing players.

	Warmup: 100,000		Warmup: 500,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	33.24%/33.43%	40.50%	30.25%/29.66%	47.32%
LF/R	45.41%/26.78%	32.77%	39.34%/26.05%	40.57%
HF/R	40.76%/29.48%	35.90%	37.99%/26.20%	42.47%
HM/R	55.84%/22.16%	27.72%	50.36%/20.58%	35.17%
LM/R	44.69%/27.89%	32.07%	40.43%/24.76%	40.65%

	Warmup: 1,000,000	
Opponents	Opponents' Winrate	Agent's Winrate
R/R	36.34%/24.57%	44.71%
LF/R	37.58%/24.88%	43.62%
HF/R	34.47%/23.86%	47.16%
HM/R	47.47%/20.21%	38.55%
LM/R	37.24%/24.02%	45.18%

	Warmup: 5,000,000		Warmup: 10,000,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	24.73%/24.34%	58.68%	23.26%/24.28%	59.54%
LF/R	32.08%/21.97%	52.45%	30.43%/21.25%	54.94%
HF/R	28.74%/20.51%	56.35%	26.87%/20.33%	58.13%
HM/R	39.21%/18.25%	49.22%	36.49%/17.54%	52.67%
LM/R	31.60%/20.75%	54.08%	30.17%/19.54%	56.59%
LF/LF	34.23%/33.97%	34.85%	33.94%/33.35%	35.61%
HF/HF	35.37%/35.56%	34.88%	34.43%/34.65%	36.84%
HM/HM	36.34%/35.83%	31.58%	35.04%/35.38%	33.42%
LM/LM	32.04%/31.67%	39.97%	30.70%/31.29%	41.71%
LF/LFA	42.45%/12.23%	56.39%	40.35%/12.00%	58.91%
HF/HFA	36.92%/18.01%	48.72%	33.35%/17.87%	52.07%

	Warmup: 50,000,000	
Opponents	Opponents' Winrate	Agent's Winrate
R/R	21.91%/22.34%	63.20%
LF/R	27.07%/19.11%	60.62%
HF/R	23.45%/18.05%	63.17%
HM/R	33.01%/15.96%	56.98%
LM/R	27.88%/18.99%	59.42%
LF/LF	31.92%/31.77%	38.59%
HF/HF	31.51%/33.32%	40.89%
HM/HM	33.33%/34.14%	36.18%
LM/LM	29.48%/29.53%	44.43%
LF/LFA	36.07%/10.53%	64.65%
HF/HFA	28.71%/16.81%	56.92%

Training Against Two Other Learners (LL)

Training against two other learners forces the agent to learn more optimal actions in common board states since sub-optimal actions are less likely to win against other learners. However certain, less common board states may not be encountered by the agent since optimal play by the other learners may not lead to these states.

	Warmup: 100,000		Warmup: 500,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	33.59%/33.54%	39.86%	29.79%/29.13%	48.30%
LF/R	45.07%/27.21%	32.34%	39.69%/26.12%	39.62%
HF/R	41.94%/29.08%	35.39%	37.27%/24.80%	43.77%
HM/R	56.27%/22.33%	27.09%	50.73%/21.39%	34.28%
LM/R	45.46%/26.94%	32.56%	40.17%/24.98%	40.78%

	Warmup: 1,000,000	
Opponents	Opponents' Winrate	Agent's Winrate
R/R	28.23%/28.14%	51.33%
LF/R	38.10%/24.44%	43.33%
HF/R	35.13%/24.02%	47.16%
HM/R	47.89%/19.49%	39.30%
LM/R	37.32%/23.63%	45.02%

	Warmup: 5,000,000		Warmup: 10,000,000	
Opponents	Opponents' Winrate	Agent's Winrate	Opponents' Winrate	Agent's Winrate
R/R	26.24%/25.80%	55.77%	25.65%/25.41%	57.04%
LF/R	32.43%/23.40%	50.52%	33.59%/20.93%	55.89%
HF/R	28.87%/20.58%	56.29%	28.60%/20.93%	55.89%
HM/R	41.16%/17.98%	47.97%	41.31%/17.46%	48.24%
LM/R	33.65%/21.76%	50.99%	32.73%/22.64%	51.70%
LF/LF	34.55%/34.55%	33.81%	35.01%/34.78%	33.68%
HF/HF	35.20%/34.51%	36.25%	36.04%/34.97%	35.50%
HM/HM	35.97%/35.45%	32.50%	34.98%/35.97%	32.59%
LM/LM	32.39%/32.82%	38.58%	31.63%/33.26%	38.85%
LF/LFA	44.81%/13.99%	51.39%	44.84%/13.99%	51.23%
HF/HFA	36.43%/16.18%	51.82%	36.72%/16.40%	51.59%

Comparisons

The following are the results of how the differently trained agents perform when playing against each other.

	Warmup: 100,000			Warmup: 500,000		
	Winrates			Winrates		
Players	Player One	Player Two	Player Three	Player One	Player Two	Player Three
$RR/LR/R$	38.66%	38.99%	29.99%	40.57%	42.21%	25.73%
$RR/LL/R$	37.90%	38.37%	31.11%	41.54%	41.56%	25.33%
$LR/LL/R$	38.31%	38.69%	30.42%	42.48%	41.68%	24.62%
$RR/LR/LL$	37.09%	35.96%	34.56%	35.25%	36.39%	36.85%

	Warmup: 1,000,000			Warmup: 5,000,000		
	Winrates			Winrates		
Players	Player One	Player Two	Player Three	Player One	Player Two	Player Three
$RR/LR/R$	41.52%	43.84%	23.01%	43.78%	47.01%	18.75%
$RR/LL/R$	42.58%	42.33%	23.46%	44.35%	45.67%	19.16%
$LR/LL/R$	43.16%	43.18%	22.47%	45.74%	44.29%	19.37%
$RR/LR/LL$	36.08%	36.72%	36.43%	34.45%	37.35%	38.85%

	Warmup: 10,000,000		
	Winrates		
Players	Player One	Player Two	Player Three
$RR/LR/R$	43.11%	47.40%	18.21%
$RR/LL/R$	44.29%	47.05%	18.40%
$LR/LL/R$	47.88%	45.04%	17.41%
$RR/LR/LL$	33.51%	37.65%	39.39%

	Warmup: 50,000,000		
	Winrates		
Players	Player One	Player Two	Player Three
$LR/RR(10M)^6/R$	52.37%	40.36%	16.94%
$LR/LR(10M)/R$	50.07%	42.47%	16.58%
$LR/LL(10M)/R$	49.81%	42.67%	17.45%

From the results of the differently trained agents playing against fixed strategies. It appears that they all seem to learn and improve at a similar rate and perform similarly against the same opponents. The main factor influencing performance is the number of Warmup plays, with all agents performing reasonably against all the tested combinations of fixed strategies at 10,000,000 Warmup plays. At 50,000,000 Warmup plays, the agent's performance is still improving, beating all other fixed strategy combinations tested. Due to time limitatons, only one agent was able to be trained for 50,000,000 Warmup plays. Unfortunately an even longer training period was unfeasible with the time allowed for this project, though the data suggests that further training would still increase the agent's performance.

Against each other they perform similarly well, though the agent trained against two random players performs slightly better at the lower number of Warmup plays. As the number of Warmup plays is increased, the agents trained against other learners perform better, with the agent trained against two other learners performing the best.

The implication of this is that because the agent trained against two random players experiences more different board states intially, it has the edge for few Warmup plays. However, as the number of Warmup plays is increased the other agents experience more board states but also learn more optimal moves since playing against more difficult opponents. The result of this is that eventually the agents trained against other learners begin to more significantly surpass those that aren't. Number of Warmup plays is still the deciding factor in the agents performance. All agents trained for 10,000,000 Warmup plays perform significantly worse when played against the agent trained for 50,000,000 Warmup plays. With the jump from 10,000,000 to 50,000,000 Warmup plays still showing significant improvement, this suggests that further increasing the number of Warmup plays will further increase performance.

5.4.6 Strategy Analysis

We will now analyse the strategy developed by the agent to gain insight into how the game is played effectively. For this we will use the strategy generated by the agent trained against one other learner and one random player for 50,000,000 Warmup plays. We use this because it was by far the most effective strategy developed.

To analyse the strategy I have extracted the Dictionary of board states from the agent. Using this I have created a new Dictionary which maps the board state to the highest scoring action of that state. I have then grouped board states by round[7] and catagory and then analysed the

[6] 10M denotes that the agent was only trained for 10,000,000 plays.
[7] Calculated by number of cards in hand.

best moves for each.

For each round, I have recorded which move was selected as the best the most often for each catagory, and to what percentage it was selected as the best. I have also recorded for each catagory which individual card choice was most popular and which individual player choice was most popular, with the relevant percentages for each. I have also included the expected value of the percentages for each action if choices were purely random. Expected percentage values for card choice will be presented each round. Expected percentage values for player choice for each catagory are as follows:

Catagory	Player Choice
0	50.00%
1	33.33%
2	33.33%
3	33.33%
4	50.00%
5	50.00%
6	33.33%
7	33.33%

The results are as follows:

First Round

Total board states recorded: 946

Expected random card choice: 20.00%

				Most Popular				
Catagory	Action	%[8]	Expected %[9]	Card[10]	%	Player	%	Total States
0	(2, 'O')	19.13%	10.00%	2	25.79%	'O'	75.79%	946

[8]% Indicates to what percentage of board this was chosen as the most popular move.

[9]Expected percentage value if actions were chosen at random.

[10]By hand index.

Second Round

Total board states recorded: 70871

Expected random card choice: 25.00%

	Most Popular							
Catagory	Action	%	Expected %	Card	%	Player	%	Total States
0	(1, 'Y')	16.58%	12.50%	1	32.67%	'O'	56.93%	404
1	(0, 'Y')	14.41%	8.33%	2	29.47%	'F'	39.60%	17881
2	(0, 'R')	12.81%	8.33%	1	28.82%	'F'	39.15%	18100
3	(0, 'R')	14.25%	8.33%	0	32.52%	'Y'	34.47%	18028
4	(0, 'Y')	20.33%	12.50%	1	30.38%	'O'	52.77%	4510
5	(0, 'O')	17.29%	12.50%	1	30.95%	'O'	50.68%	3528
6	(3, 'F')	14.49%	8.33%	2	28.99%	'F'	43.18%	3629
7	(0, 'R')	15.65%	8.33%	1	30.68%	'F'	33.96%	4791

Third Round

Total board states recorded: 69558

Expected random card choice: 33.33%

	Most Popular							
Catagory	Action	%	Expected %	Card	%	Player	%	Total States
0	(2, 'O')	29.93%	16.67%	1	42.18%	'O'	72.11%	147
1	(0, 'Y')	27.79%	11.11%	0	39.64%	'Y'	41.60%	19840
2	(0, 'R')	17.16%	11.11%	0	38.04%	'F'	39.53%	20107
3	(0, 'Y')	20.66%	11.11%	0	52.82%	'Y'	43.20%	20491
4	(0, 'Y')	31.06%	16.67%	0	41.52%	'Y'	51.11%	2389
5	(0, 'O')	22.92%	16.67%	0	48.88%	'Y'	51.12%	1972
6	(2, 'F')	30.02%	11.11%	2	38.22%	'F'	54.07%	2025
7	(0, 'R')	24.82%	11.11%	0	44.45%	'R'	40.63%	2587

Fourth Round

Total board states recorded: 34644

Expected random card choice: 50.00%

	Most Popular							
Catagory	Action	%	Expected %	Card	%	Player	%	Total States
0	(1, 'O')	44.19%	25.00%	1	62.79%	'O'	67.44%	43
1	(0, 'Y')	43.98%	16.67%	0	57.83%	'Y'	51.77%	10203
2	(0, 'R')	25.46%	16.67%	0	53.66%	'F'	38.50%	10393
3	(0, 'Y')	33.96%	16.67%	0	79.05%	'Y'	47.34%	10580
4	(0, 'Y')	49.45%	25.00%	0	58.83%	'Y'	57.40%	906
5	(0, 'O')	35.71%	25.00%	0	70.50%	'Y'	52.66%	773
6	(1, 'F')	56.34%	16.67%	1	74.01%	'F'	63.38%	781
7	(0, 'R')	44.04%	16.67%	0	77.10%	'R'	48.39%	965

The agent has played 200,000,000 rounds (exluding the final round where no decision is made) and seen 176,019 unique board states by the used representation.

To interpret the results I have compared percentage value of each action chosen as the best move and compared it to the expected value. Percentage values closer to the expected values reveal a less significant preference toward that action, while larger differences imply a more significant reference towards that action. Significantly larger percentages than expected show that action was selected as the best move for that catagory more often and hence is generally appropriate for the round/catagory situation defined. These results reveal high level patterns or rules that exist in the strategy rather than each board state being considered individually.

The results from round one show a significant tendancy to choose an opponent as the player choice. Although the most popular card choice is the middle most card of the hand, it is not that significantly favoured. It may be that the agent is choosing the *lowest magnitude* card to choose first so it has more influence on the board later in the game.

The agent generally prefers to take actions appropriate to the situation. When the agent is ahead in catagories 1 and 4, the most popular action for all rounds it to use the lowest card on itself. However, in catagory 6 when the player and an opponent are in tied first place, the agent appears to prefer to use the highest card on the frontmost player that it is tied with. This may be preferable if the frontmost player also uses their highest card on the agent, then the agent saves their lowest card to be used on themselves in the final round.

When the agent is winning in catagory 2 the most popular action is pushing the rearmost player back with the lowest card. However, we can see from the most popular player choice that the frontmost player is most frequently chosen in actions. This may be due to the agent having similar probabilites for choosing to push the rearmost player back and push the frontmost player

forward.

For catagory 3 the agent chooses to push the rearmost player back as one would expect in round two. However, in later rounds the option is to use the lowest on itself. This may be that often the agent has exhausted negative cards early and so pushes itself forward slightly with lower cards so it appears less threatening and holds its largest card to be played on itself in the final round. There is a significant percentage of board states that have selected this action as their best action, especially in the fourth round. This implies that this is often the correct line of play since the agent performs reasonably well. This is an example of a decision that may not be immediately obvious to human players but has been incorperated in the strategy by the implmentation of reinforcement learning.

6 Conclusion

This project aimed to investigate the effectiveness of reinforcement learning in 'Why First?'; a that exhibits characteristics not found in more well understood games such as *chess*.

Due to its lack of popularity it is difficult to determine how good the implmentation here is. I am unaware of any expert players of the game or any other software implmentations that play the game well, so I have nothing to compare to. However, comparing to previous versions of the software it appears to perform well, beating the concieved fixed strategies which themselves proved effective against purely random play, and combinations thereof.

With the final implementation, the last increase of the training period still improved the effectiveness of the agent over its predessor. Given more time I would have liked to see how well the approach performs given a longer training period. However due to the time it takes to train the agent and the time contraints of this project this was unfortunately unfeasible.

After analysing the strategy of the final implementation, patterns were found in what had been learned. This gave a rough outline of what an effective strategy should be and suggest that perhaps a rule based representation of a strategy could be effective.

The effectiveness shown in this project demonstrates the flexibility of reinforcement learning. It shows that an approach can be implmented by someone who does not understand the strategy behind the problem, and also shows that reinforcement learning is an effective approach for learning a game such as this that does not have the common properties of more well understood games.

7 Future Work

Given enough time I would have liked to see how far this approach could improve given enough training plays. In addition to this I would have liked to implement different approaches that may be promising.

7.1 Genetic Algorithms

The strategy analysis demonstrated a possible rule based representation may be appropriate. By generating lots of these rules, evaluating them with a fitness function and breeding them, many possible strategies could be developed and improved. Due to the large search space of possible strategies, a Genetic Algorithm based approach could definitely be appropriate.

7.2 Minimax

As mentioned in earlier sections, Minimax may be an effective approach, though developing such an approach would be a project in its own right. By evaluation what cards have been played, it may be possible to predict the expected result of a given action. Such approaches have been shown to be effective in various other games and so may be applicable here.

7.3 Different Board State Representations

Using a more simplistic board state representation may speed up learning but at the expense of less optimal moves due to lack of information. Conversely a more complex board state representations may lead to improved play from the agent, though more complexity will increase the number of board states and thus slow down learning. With improved algorithmic efficiency or increased computational power, a more complex board state representation may be a vast improvement over the approach implemented in this project.

7.4 A General Learner

The aim of this project was to demonstrate how effectively we could teach an agent to play a less understood game. A natural progression of this is teach an agent an unknown game. My final approach was to define a representation of a board state, effectively telling the agent the features defined have something to with the outcome of the play. By teaching an agent to select their own features from the game to represent states with reinforcement learning, the agent would effectively *teach itself how to learn.*

References

Basil, V. R. and Turner, A. J.: 1975, Iterative enhancement: A practical technique for software development, *Software Engineering, IEEE Transactions on* (4), 390–396.

Edmund Burke, Graham Kendall, J. N. E. H. P. R. S. S.: 2003, *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*, Springer US.

Galindo, J. and Tamayo, P.: 2000, Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications, *Computational Economics* **15**(1-2), 107–143.

Lai, M.: 2015, *Giraffe: Using deep reinforcement learning to play chess*, Master's thesis, Department of Computing, Imperial College London.

Mitchell, T. M.: 1997, *Machine Learning*, MIT Press and The McGraw-Hill Companies, Inc.

Neumann, J. V. and Morgenstern, O.: 1944, *Theory of Games and Economic Behavior*, Princeton University Press.

Osborne, M. J. and Rubinstein, A.: 1994, *A Course in Game Theory*, MIT Press.

Puterman, M. L.: 1994, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.*, Wiley-Interscience.

Russell, S. J. and Norvig, P.: 2003, *Artificial Intelligence: A Modern Approach(2nd ed.)*, Prentice Hall.

Shapley, L. S.: 1953, *Stochastic Games*, Princetonton University.

Shipp, M. A. et al.: 2002, Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning, *Nature medicine* **8**(1), 68–74.

Watkins, C. J. C. H.: 1989, *Learning from Delayed Rewards. PhD thesis*, PhD thesis, Cambridge University, Cambridge, England.

Willem, M.: 1996, *Minimax Theorems*, Birkhuser.

Appendices

Appendix A: Provided Materials

- The game 'Why First?'.
- A very basic version of software that simulated the game 'Why First?' written in Python by Brandon Bennett. This software was heavily altered throughout development.

Appendix B: Ethical Issues

There were no ethical issues to address during this project.