Investigation into the effect of Social Learning in Reinforcement Learning Board Game Playing Agents

Vukosi Ntsakisi Marivate

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, South Africa, in fulfilment of the requirements of the degree of Master of Science in Engineering

Declaration

I declare that this is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering, at the University of the Witwatersrand. It has not been submitted before for any other degree or examination at any other university.

Signed this _____of_____ 2009

_____

Vukosi Ntsakisi Marivate

Abstract

This thesis presents the use of social learning to improve the performance of game playing reinforcement learning agents. Agents are placed in a social learning environment as opposed to the Self-Play learning environment. Their performance is monitored and analysed in order to observe how the performance changes compared to Self-Play agents. Two case studies were conducted, one with the game Tic-Tac-Toe and the other with the African board game of Morabaraba. The Tic-Tac-Toe agents used a table based TD (λ) algorithm to learn the Q values. The results from the tests for the Tic-Tac-Toe agents indicate that the social learning agents perform better than the Self-Play agents in both board tests and competitive tests. By increasing the population sizes of the agents the number of superior social agents also increases as well as improvements in their skill level. In the second case study the agents use function approximation and the TD (λ) algorithm because of a larger number of states. The social agents performed better than the Self-Play agents in the board tests and are not superior in the test where they compete against each other. Larger populations were not possible with the Morabaraba agents but the results are still positive as the agents perform well in the board tests.

Acknowledgements

I would like to thank the following people for their assistance and advice throughout this research:

My Family and Thembekile Masanabo for their unwavering support and encouragement throughout the year.

Professor Tshilidzi Marwala for his support, patience and guidance throughout the research period.

Doron Horwitz and David Vannuci of the University of the Witwatersrand Convergence Lab for their assistance with computational resources and advice with Morabaraba.

Rofhiwa Musehane, Linda Mthembu, Dr. Anthony Gidudu and Fulufhelo Netshiongolwe for their assistance with proof reading.

Nomenclature

AI – Artificial Intelligence

DP – Dynamic Programming

RBF – Radial Basis Function

SP – Self Play

Table of Contents

Table of Figures

Chapter One: Introduction

Artificial Intelligence (AI) (Russell et al. 2003) is a field in which computer programs that exhibit intelligence are developed and built. Agents (Russell et al. 2003) are an abstraction that is used to describe computer programs that reside in some environment and can interact within that environment. Through AI research and development, agents that can play board games have been built and tested against humans. Some board games are completely solved while others still trouble researchers; in that the computer agents cannot beat humans in those games. One way in which agents are taught to play games is Reinforcement learning (Sutton, Barto 1998). A problem that arises when training such agents is that they cannot learn a policy for a board game that can compete successfully against most of the opponents they encounter. The benchmark in this domain is measured by how well an agent can play against a human master player for that specific board game (Schaeffer 2001). Most Reinforcement learning agents are trained using Self-Play (Schaeffer 2001) (An agent learns by playing against itself) and this has led to successes in some games (Tesauro 1994) and failures in others (Schaeffer 2001). The agents sometimes exhibit behaviour that is not expected when they face an opponent that they have never played against, meaning they do not have any knowledge of that particular strategy (Wong, Lim & Gao 2004). This thesis proposes and carries through experiments using Social learning theory (Bandura 1977) in tandem with Reinforcement learning to try and improve the performance of game playing agents. Social Learning is concerned with how humans learn from observations of others' actions. By fusing this theory with reinforcement learning, the author aims to observe how the performance of game playing

agents changes in social settings and hopefully result in positive improvements in the agents' performance. The agents are trained with other agents as opposed to the manner of Self-Play that has been used in some of the board game agents. The social learning configuration is tested against the Self-Play configuration on two case studies; the case studies are the game of Tic-Tac-Toe and Morabaraba. Two social configurations are used in training the social agents and are compared at the end of the thesis.

Chapter Two introduces the use of artificial intelligence in games and describes the problems and challenges that arise with current solutions. Chapter Three presents a background on Reinforcement learning and how it is used in this thesis for the two case studies that are presented. Chapter Four presents a summary of social learning theory, how it relates to reinforcement learning and how it differs from other population based algorithms. Chapter Five presents the first case study with the game of Tic-Tac-Toe and findings using the proposed frameworks. Chapter Six presents the second study with the game of Morabaraba and the findings. The last Chapter draws conclusions from the work and also recommends some future work.

The following papers from this dissertation have been published/submitted for publication:

- V.N. Marivate and T. Marwala. *Introduction of Social Methods in Board Game Agents, Proceedings of the IEEE Computational Intelligence and Games Conference*, December 2008, To Appear.

- V.N. Marivate and T. Marwala. Social Learning in Board Games, *Proceedings of the International Symposium on Intelligent Informatics*, December 2008, To Appear.

- V.N. Marivate and T. Marwala. Study of the effects Social Learning with Reinforcement Learning Board Game Agents, IEEE *Transactions on Computational Intelligence and AI in Games*, Submitted.

Chapter Two: Artificial Intelligence in Board Games

2.1 Introduction

Games are an area where humans can pit their intellect against each other in a competitive manner. Making machines/computer programs that have the ability to play games (Schaeffer 2001) against human opponents has been a challenge since the beginning of research in Artificial Intelligence (Schaeffer 2001). Games are a domain in which the performance and boundaries of machine learning and Artificial Intelligence (AI) have been tested. Games offer an environment that has rigid rules and a specified manner in which interactions can take place. The board game environment is deterministic and easy to track. This though does not mean that the game domain is not a fruitful area of research. Breakthroughs that are made in AI in games can then be extended to solve real world problems where the environments are more uncertain and are less constrained. Application domains such as economics use game theory and game modelling to model economic phenomena (Roth 2002). Through the years there have been machines that have been taught to learn and play a multitude of games. This chapter introduces the areas where artificial intelligence has been applied in games. It also discusses some of the problems that have arisen within the game domain.

2.2 Early Stages

Making intelligent machines that can play games against humans has been a fascination for many years. For many years there has been substantial research in designing and building machines that could beat humans at any board game (Schaeffer 2001). Early on

the mechanical contraption of the chess playing Turk (Standage 2002, Wolfgang von Kempelen 1783) enthralled audiences around Europe. The machine could beat most human players who played against it. Even after the hoax, about a chess master who was hidden in the contraption (Figure 1), was revealed the idea of a machine that can play games against humans intelligently stayed in the minds of scientists.

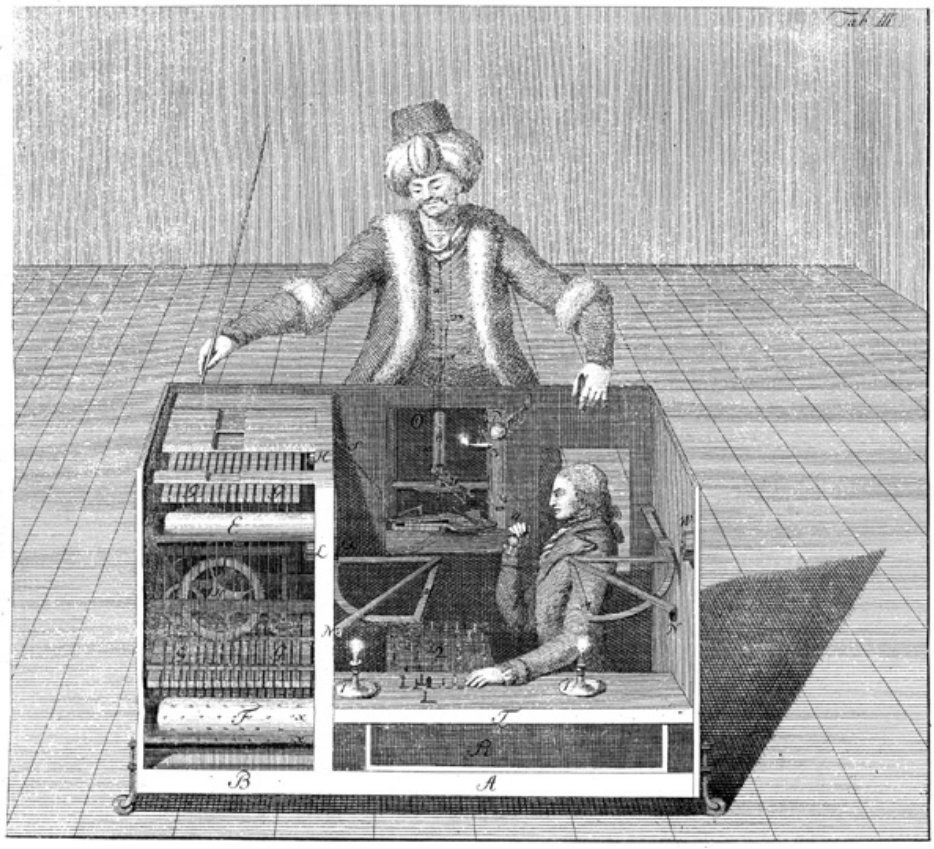

Figure 1: Chess Playing Turk, taken from (Wolfgang von Kempelen 1783)

Shannon (1950) would later on explore the possibilities of building high performance game playing programs. At his time he was among a number of computer scientists, such as Alan Turing, who were trying to solve problems in the game domain. The game of choice for computer scientists during the 1950s was chess. Breakthroughs in the

algorithms developed to play chess against humans would later be used in other domains (Schaeffer 2001). This would form inspiration for the game playing machines of today. Whether it is machines that play Board Games (Ghory 2004, Kalles 2008, Neto, Julia 2008, Runarsson, Lucas 2005), Card Games (Hurwitz, Marwala 2007, Hurwitz 2007) or Video Games (Gold 2005, Laird, Lent 2000), researchers have been able to model them or are still actively pursuing solutions to some harder games with larger state spaces and larger branching factors.

2.3 Current Stages

Artificial Intelligence techniques have been used in games such as Chess, Othello, Backgammon and Scrabble. As more computing power became available to researchers, better machines have been developed and built. Solutions for game playing machines have used methods such as reinforcement learning (Sutton, Barto 1998), neural networks (Bishop 1995), Search methods (Schaeffer 2001) etc. Most of the methods have relied on the background knowledge of the researchers (Schaeffer 2001). This implies that the performance of the solution is mostly dependant on the domain knowledge that is introduced by the researcher. Thus in order to build good game playing machines, one has to know what are the best strategies, what are the potential pitfalls and what are general patterns in that particular game. This can become tedious and in some games unattainable as it is hard to represent some strategies in a machine (Runarsson, Lucas 2005, Bouzy, Chaslot 2007). Data for training of an AI method or for the search databases is normally collected from analysing a number of different games. Some game

playing machines have used self-play mechanisms. One famous example is TD-Gammon. It was a backgammon playing reinforcement learning agent (computer program that performs actions to reach a goal (Wooldridge 2002)) built by Tesauro (1994). It learned how to play backgammon by playing against itself and having its structure tuned for best results. Thus self-play allows the AI machine to play against itself for a large number of iterations as a way of training and learning. This then forgoes the use of databases of previous games (and thus search algorithms) as a learning method, because the AI method used captures the dynamics of the game. No human player would like to play 1 million rounds of a game against a machine so that it can learn to play that game. Thus by introducing self-play researchers now have an abundance of training data. Another manner to train with databases is to use reinforcement learning algorithms that use the database games to learn (Mannen 2003). The agent can use the databases to train its reinforcement learning method. An example is the use of database games to train a Chess playing reinforcement learning agent (Mannen 2003) .

2.4 Problems Arising

There are board games which intelligent agents still have not mastered. These games include game such as GO, an ancient Chinese war game. The reinforcement learning agents that have been used in this can only play at a very elementary level. Even with some domain knowledge included in the agents the agents still cannot play at a level that can beat even novice human players. The methods that have been used successfully in other board games perform poorly (Schaeffer 2001). Another problem that arises in

agents is mal-adaptive behaviour (Wong, Lim & Gao 2004, Wong, Lim & Gao 2005) of agents in cases where they have never played against a strategy that is dynamic, meaning the opponent is also learning and changing its strategy as time progresses. Even TD-Gammon's success has been argued to be due to the dynamics of the game and not completely due to Self-Play (Pollack, Blair 1997). Another problem that may arise is that an agent playing against itself might then get stuck using a strategy that is not best but it is good to adopt when playing against that strategy (Ghory 2004). This then leads to large losses when this agent plays against an opponent who might not even be good but has a strategy that can just beat the one the agent has. There has been research into how reinforcement learning agents play board games against other agents that have set strategies and it was then noted how reinforcement learning is or could be more superior (Hurwitz 2007). Multiagent reinforcement learning has been proposed as a solution or better approach to modelling games with multiple players, rather than traditional game theory (Hurwitz 2007).

The main question that is addressed by this thesis is how do reinforcement learning agents then cope with agents that have changing strategies and are also learning? Is there also a possibility of building solutions that are more general and can take on different games without a great deal of tuning by the researcher?

Chapter Three: Reinforcement Learning

3.1 Introduction

This chapter gives a brief overview on Reinforcement learning. The relevant theory used for this thesis is discussed for insight and expanding on some problems associated with some reinforcement learning methods. This chapter also discusses the learning methods used by the agents that are developed in this thesis.

3.2 Reinforcement Learning

3.2.1 Agents

An intelligent agent is defined (Sutton, Barto 1998) as a computer system/program that resides in some environment and is allowed to perform actions in that environment (Figure 2).

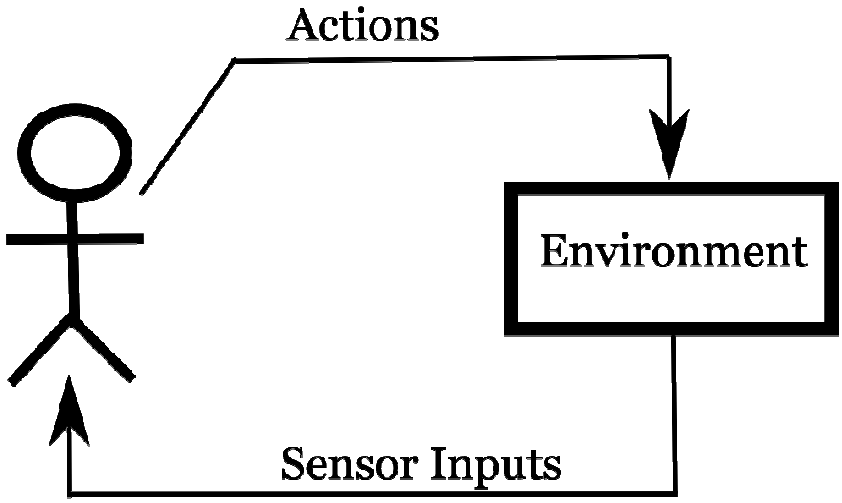

Figure 2: Agent and Environment

Intelligent agents (Wooldridge 2002) are defined as agents that can *react* to changes in their environment, have *social ability* (communication) and the ability to use AI to reach their goals by being *proactive*. Agents are active, task-oriented – modelled to perform specific tasks – and are capable of autonomous action and decision making.

3.2.2 Reinforcement Learning

Humans learn by interacting with each other and then experiencing or being told what is right or wrong about what they are doing. This is even truer for early childhood development of humans (Sutton, Barto 1998). Extending this to agents that reside in an environment, the agents can learn from being reinforced for their actions. Lessons are learned from being rewarded or punished after performing an action in a certain situation; this is termed Reinforcement Learning (Sutton, Barto 1998). This is different from most supervised learning methods in machine learning (Kotsiantis, Zaharakis & Pintelas 2006). In supervised learning, a learning algorithm is given test cases that have inputs and the corresponding correct outputs. The learning method tries to learn/approximate a functional relationship between the input and output data. This for example, can be in the form of approximating data that might have been generated using an equation as shown in (3.1).

$$\bar{y} = f(\bar{x}) \tag{3.1}$$

Where $\bar{y}$ is the output from a function *f* that accepts input $\bar{x}$. This can be viewed as a case of learning from a teacher, different from reinforcement learning where it is assumed

that an action has been performed and then a reward or punishment is given. In reinforcement learning the agents must create their own experiences. They perform actions and only after then are they given some form of indication of how well the action they took is. The agent is not told which action to take but tries to find the actions that will lead to the most rewards (Singh et al. 1996). Thus the agent has to explore different actions. The rewards might be delayed (as is the case in board games, one wins only at the end of the game) and thus the agent must be able to track which actions lead to the best returns in the long term. Reinforcement learning (Sutton, Barto 1998) is learning that entails an agent trying to maximise its reward given the actions it has taken in a certain environment as shown in Figure 3.

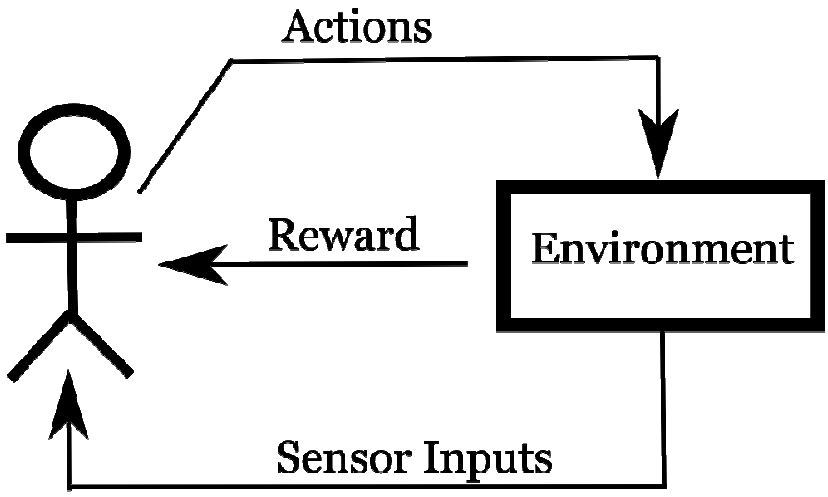

Figure 3: Reinforcement Learning Framework

In reinforcement learning the agent is not told which actions to take and when to do so. It just perceives the environment through its sensors and thus has a state that represents the environment, then takes an action to maximise the reward gained from the action or

sequence of actions. The reward can come from within the agent itself (Intrinsic) or from the environment (Extrinsic) (Stout, Konidaris & Barto 2005). A good example of reinforcement learning in games is the game of checkers (Dubel et al. 2006). In checkers the agent will be given the state of the board (Coordinates of all the pieces) and then only given a reward if it wins. If there is a loss, the agent will then be given a punishment signal. Through this the agent will learn not to repeat the action that caused the loss (Ghory 2004). Before going into further details into the theory of how reinforcement learning is carried through, a few definitions and notations are in order. These are discussed in the next subsection.

3.2.3 Markov Decision Processes

When observing an agent and the actions it takes in its environment there is a succession of states. When the state of an environment contains all information about all previous states and actions performed, then it is said that the environment has the Markov property (Sutton, Barto 1998). This for example is the position of a board game pieces. If a reinforcement learning task satisfies the Markov property then it is termed a Markov Decision Problem (Sutton, Barto 1998) (MDP). An MDP is made up of a number of entities.

- *S* - set of states of the environment (*Possible positions on the board*)
- *s* - current state of the environment (*Current position during a game*)
- *s'* - The next state (*The next game position*)
- *A* - set of actions that can be taken by the agent (*Set of possible moves*)

- a - current action chosen by the agent (*Current move*)
- R – Reward given *(R(s), (R(s,a), R(s,a,s'))*
- $P(s'|s,a)$ -Transitional Probability

The transitional probability is the probability of moving into another state (s') given an action (a) and a state (s). Given the above information, an agent can make a decision on which actions are best to take in a specific state. This is termed the policy (π) of the agent. It is a mapping of a state to a specific action ($a=\pi(s)$). The transitional probabilities of an environment are not normally provided or known. Thus a challenge in reinforcement learning is modelling an environmental dynamics within the agent. To do this the concept of the value of a state is introduced. This is done through the introduction of Value Function and Action Value functions.

3.2.4 Value Functions and Action Value Functions

In solving the reinforcement learning problem, one needs a measure of evaluating how good it is to be in a certain state or how good it is to be in a certain state and choosing a certain action. The value of a state can be interpreted as the likelihood of winning a board game given the state of a board game at a given time. Through these functions one can evaluate the policy that the agent is taking. The value function is defined in (3.2) as:

$$V_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_0 = s] \tag{3.2}$$

This equation entails the expected value (E) of the summation of the discounted (γ) reward (r) of all possible future states given that the agent is executing a policy π and starting at the current state s. The policy (π) is the mappings of state to actions. The second function is to evaluate how good it is to be in a certain state and choosing a certain action, thus the Action-Value function is introduced as equation (3.3).

$$Q_{\pi}(s,a) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \mid s_0 = s, a_0 = a] \tag{3.3}$$

Where $Q(s,a)$ takes into account not only starting at the current state but also evaluates the current action. The maximization of (3.2) and (3.3) by carrying out an optimal policy π^* will result in higher rewards in the end. To find the policy that maximises the value function or action-value function we use the Bellman Optimality equations (Sutton, Barto 1998). To learn in reinforcement learning from a system without the complete model (Model free) of the system, the agent needs to learn through experience. The agent thus has to go through interactions and find an optimal policy that optimizes (3.2) or (3.3), thus finding V^* and Q^* which are termed the optimal Value function and Action-Value function respectively. In the case of board game playing agents, the agents play multiple games and learn through their numerous games. Thus they create their own training data.

3.3 Reinforcement Learning Algorithms

There are a number of solutions that have been used to solve the reinforcement learning problem.

3.3.1 Classical Solutions (Dynamic Programming and Monte-Carlo)

Dynamic programming (DP) (Sutton, Barto 1998) solutions for the reinforcement learning problem are iterative based methods. Mostly they exploit the Bellman Equation, shown in (3.4), to find their solutions.

$$V_\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_\pi(s') \right] \tag{3.4}$$

These methods are iterative and can be used to evaluate a policy, do policy iteration and value iteration (Sutton, Barto 1998). Dynamic programming methods converge towards the optimal values being sought. The second batch of solutions for finding the Value Function or Action Value Functions is the Monte-Carlo methods (Sutton, Barto 1998). In Monte-Carlo methods, agents follow a certain policy and keep a running average of the awards they are receiving either for the Value function or the Action-Value function. Through episodes, these estimates converge to the correct *V* and *Q* values. These methods suffer the curse of dimensionality when you have large state spaces. Is it possible to visit all possible states within a certain reasonable number of episodes?

3.3.2 Q-Learning

Q-Learning is an online reinforcement learning algorithm. It does not need the model of the environment. The algorithm functions by trying to approximate the values of the state action pairs for a given problem. This equation was shown earlier in (3.3). After learning the *Q*-value of each state and action the best action to take in a state would be the one with the highest *Q*-value. The agent learns by populating a *Q*-value table. This table can

be visualised as a matrix whose values are evaluations of each state action pair in the game that is possible. The agent first starts off with an empty table with all *Q*-values equal to 0. At each state the agent can choose an action. This action can either be sampled randomly or chosen from the *Q*-Table (Sutton, Barto 1998). The update of each *Q*-Value after an action is chosen is shown in (3.5)

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha(r(s) + \gamma \cdot \max_{a'} Q(s',a')) \tag{3.5}$$

Where α is the learning rate and γ is the discount factor. The rest of the symbols were defined in 3.2.3. The γ discount factor is adjusted so as to make immediate rewards or long term returns more important.

3.3.3 Temporal Difference Learning

Temporal difference learning is taken as the algorithm that was created to solve the reinforcement learning problem (Sutton, Barto 1998). The update to a value function is made using equation (3.6):

$$V'(s_t) = V(s_t) + \alpha \cdot (r_t + \gamma \cdot V(s_{t+1}) - V(s_t)) \tag{3.6}$$

Where r_t is the reward just received, α is the learning rate, γ is the discount factor, $V(s_{t+1})$ is the value of the state that the agent has transitioned into, $V(s_t)$ is the old estimate of the value of the current state and $V'(s_t)$ is the new estimate of the value of the current state. Temporal difference learning uses estimates to update other estimates and this is termed bootstrapping (Sutton, Barto 1998). Temporal difference learning can

also be used to estimate Action-Value functions (Sutton, Barto 1998). In this research it is used for finding Action-Value functions for the board game states and actions to be chosen for the agents.

3.3.4 TD (λ) - Learning

TD (λ) uses Eligibility Traces (Sutton, Barto 1998) as part of its updating. The eligibility traces can be viewed as a manner of keeping information about the previous states that have been visited so that if they do occur again within a certain time, they have more importance in the learning phase. The TD (λ) algorithm is shown below in Figure 4.

Initialise $V(s)$ to an arbitrary value and $e(s) = 0$
Repeat (for each game)
    Initialise s
    Repeat (for each game step)
        $a \leftarrow$ action given for π for s
        Take action a, observe reward, r, and next state, s'
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For all s:
            $V(s) \leftarrow V(s) + \alpha \delta e(s)$
            $e(s) \leftarrow \gamma \lambda e(s)$
        $s \leftarrow s'$
    Until end of game

Figure 4: TD-Lambda Algorithm (Sutton, Barto 1998)

e(s) above is the eligibility trace (Sutton, Barto 1998) of a certain state. Thus if a certain state repeats itself, its update is taken into account with a higher importance depending on how recent the previous occurrence was.

3.3.5 Action Selection (Exploration vs. Exploitation)

In order to explore actions that are available the agent needs to choose whether to explore or exploit its gathered knowledge. This is termed the exploitation-exploration dilemma. In tandem with the TD (λ) algorithms the agents have an action selection method. The selection of actions can be sampled from a probability distribution that may change given the number of episodes (games) that have been completed (Sutton, Barto 1998).

In this research only one action selection method was used. For choosing the actions and allowing exploration and exploitation, actions were sampled from an epsilon greedy distribution which can be written as (3.7):

$$P(s,a_i) = \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A|}, if\ \ a_i = \arg\max_{a' \in A} Q(s,a') \\ \dfrac{\varepsilon}{|A|}, else \end{cases} \tag{3.7}$$

The agent chooses a random action with probability ε and takes a greedy action with probability $1 - \varepsilon$. This makes sure that the agents initially are more likely to explore but as more and more games are played ε decreases and thus the agents start to then exploit more, using the knowledge that they have gained through playing the games.

3.3.6 Function Approximation

For large state spaces (large number of states in one environment) the use of tables as a means of storing value function or action-value function data becomes impractical. This is due to the large amount of data that would need to be stored in the data structure used. Secondly with large state spaces it may not be possible to visit each and every possible state or state action pair. Thus there will be empty boxes in the Value table or Action-

Value table. To solve this problem function approximation schemes can be utilized. The different function approximation schemes that have been used include Feed Forward Neural Networks (Bishop 1995) and Radial Basis Function Networks (Park, Sandberg 1993). For the Morabaraba board game, a Radial Basis function network is utilized. RBF networks are linear function approximators. The network replaces the action-value function table. Radial basis function networks are similar to multi-layer perceptron neural networks. They have 3 layers: the input layer, the radial basis function layer (replaces the hidden layer) and the output layer. The configuration is shown in Figure 5.

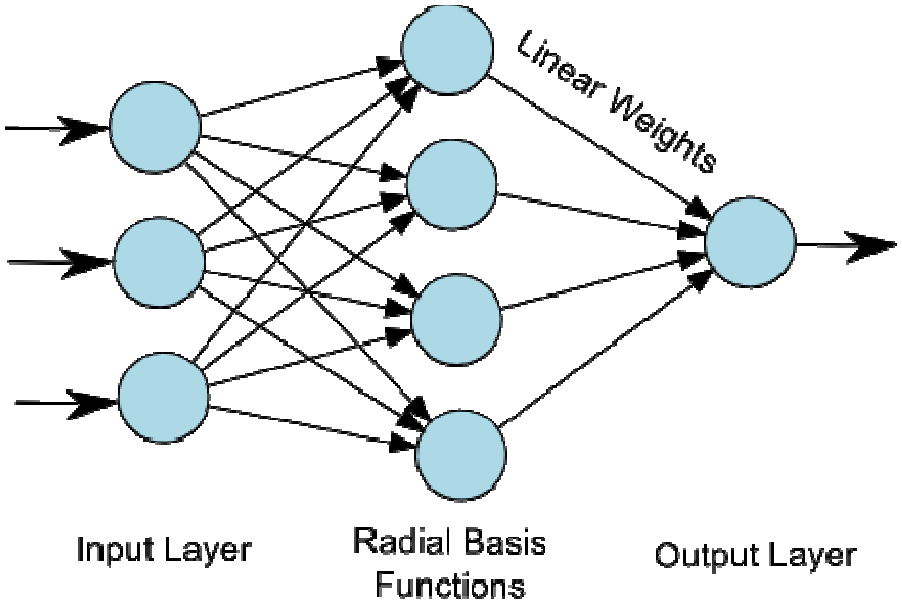

Figure 5: Radial Basis Function Network

The equation that defines the network is given by (Park, Sandberg 1993)

$$y(x) = \sum_{i=0}^{N} w_i \phi(\|x - c_i\|) \tag{3.8}$$

where y is the output, x is the input, N is the total number of centers (c), w_i is the ith linear output weight and ϕ is the radial basis function. The radial basis function used in this paper is a Gaussian function (Orr 1996) which is defined in (3.9) as

$$\phi_i(x) = \left(\frac{-\|x - c_i\|^2}{2\sigma} \right) \tag{3.9}$$

σ here represents the variance. Now as we are using a function approximation scheme, the learning algorithm is adjusted accordingly. The value function *V(s)* changes to a *V(s,w)* where w is the weight vector (in this case the second layer linear weights). Thus the value function or action-value function is now not only a function of the state but also adjustable weights. The RBF network used has a uniform grid of centers along all possible values of the input. The widths are then fixed before learning can commence (this adjustment is done in this research to tune the agents). The linear weights are adjusted during learning using a gradient descent algorithm (Baird, Moore 1999). The radial basis function network allows for generalisation in estimating Value functions or action value functions. This comes in handy for environments that have large state spaces. It may not be possible to visit every possible state but the RBF can generalise and approximate the values. The downside of using Radial basis functions is the computational complexity (Sutton, Barto 1998, Neumann 2003) of its structure and the need for adjusting and tuning the network and other parameters of the learning algorithms. For this research, RBF networks were chosen because of their generalisation characteristics. The board positions would be the inputs into the RBF network. The other

inputs would be current player and in some cases be the current game stage. The output would be the estimated V or Q value.

Chapter Four: Social Learning

4.1 Introduction

Since Reinforcement Learning is inspired by modelling how humans are taught to develop in their early stages of life, another complimentary theory can be used in conjunction with reinforcement learning. Humans seldom learn only by themselves. They live in a society and thus observe what others do. From observing the action of others, children learn what is right or wrong, what is best and what can lead to maximum rewards (Miller 1944).This is termed social learning. In early stages, children observe how their parents behave. They reciprocate this behaviour in time. For effective learning a number of factors must be present, and according to (Miller 1944). These fundamentals of learning are:

- Drive
- Cue
- Response
- Reward

The learner has to have motivation to do an action, thus *Drive*. The learner also has to know when to act, thus *Cue*. After completing the action, the learner then has to get a *Response*. From the response the learner can then get a *Reward* if his actions led to the correct state. There are a number of theories that describe social/observational learning. The fundamental points by Miller are used to highlight the general learning theory. In learning, there can be an explicit teacher (mirroring Supervised Learning) as seen in Figure 6, or one can use observation and imitation of others actions as a way of learning

(Less supervised). The next section describes social/observational learning and its connections to reinforcement learning.

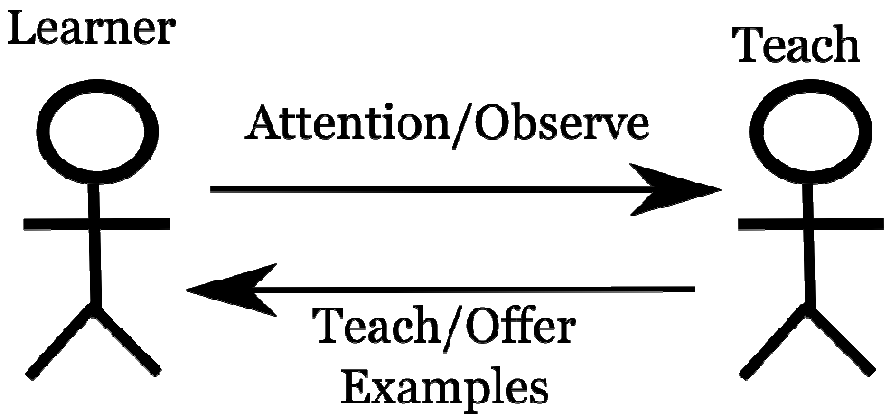

Figure 6: Supervised Learning Environment.

4.2 Social Learning, Games and Reinforcement Learning

Social learning in game communities is more common. Social learning itself is the most common way of learning (Miller 1944). Players of such board games such as chess, Scrabble and checkers mentor each other in their clubs (Okulicz, Vialle & Verenikina 2007). Through the player's interactions with others in their competitive games, they observe how others play. From these they can be rewarded if they show superior skills or can then learn from watching other player's moves. In chess clubs, the clubs do not merely rely on teaching the players directly but also encourage playing as many games as possible with other players of the club as illustrated in Figure 7, thus the observational learning is heavily encouraged (Okulicz, Vialle & Verenikina 2007).

Figure 7: Players in Chess Club, taken from (Detroit City Chess Club 2008)

In social learning there are a number of important factors that a being must have in order to be able to learn. These factors are additions or variations of the before mentioned fundamentals of learning. In order to benefit from observational learning the learner must be able to (Bandura 1977):

- Pay attention to what is being observed
- Remember the observations
- Be able to replicate the behaviour
- Be motivated to demonstrate what they have learnt

These properties can be carried through to reinforcement learning; and the agent can:

- *Observe* the State and its Transitions
- *Remember* the actions taken in specific states
- Choose different actions to change states (*Replicate* observed behaviour)

- Is rewarded if the actions lead to the goal (*Motivation*)

Furthermore Vygotsky (1978) discusses the concept of the "more knowledgeable other". This concept takes into account that in a social setting a learner would learn more from observing another being, which has more experience or is at the same skill level, thus making it the "more knowledgeable other". This can also be observed in chess clubs where members are paired to train with stronger players or peers.

4.3 Proposed benefits to competitive game agents

For learning in competitive games, social learning can be used to improve the performance of reinforcement learning agents. By introducing other agents as opponents in the learning stage, one introduces a dynamic playing environment (Sandholm, Crites 1996). If for example the opponent is a logic based intelligent computer program, a reinforcement learning agent would learn a strategy or policy that would optimally beat the logic opponent (Hurwitz 2007). By introducing two different opponents, with different set of strategies that the agent would play against, the agent would learn a strategy that would beat the two effectively. In board games this would mean creating opponents that have varying strategies, thus stimulating a social setting is needed. This would introduce reinforcement learning agents that have different strategies and learning paths. The non-stationary environment would be more dynamic than having Self-Play as all of the agents have their own identities.

4.4 Difference from other socially inspired Population based algorithms

The proposed use of social learning is different from socially inspired algorithms. Socially inspired algorithms in evolutionary methods include Particle Swarm Optimisation (Kennedy, Eberhart 1995, van den Bergh, Engelbrecht 2004), Ant Colony Optmisation (Dorigo, Stützle 2004), Memetic Algorithms (Moscato 1999), Fish School Optmisation (Filho et al. 2008). These algorithms are used mostly for search/optimisation. In the algorithms that have multiple agents, each agent has some knowledge of how well the rest of the agents are performing. In the social frameworks used in this study, the agents have no knowledge of how well their opponent plays/performs. Another factor is that in the evolutionary methods the agents are acting to reach a common goal. Again in the case of the social learning framework the agents are competitive. There is no common goal that the agents are trying to reach in their playing. Some work has been done in the field of co-evolution of agents. Most studies have focused on how co-evolution in multi-agent systems can improve the way in which the whole multi-agent system can improve its performance. This means that the focus is on getting the agents to work together in solving problems. This is more attributed to the standard definition of a multi-agent system.

Chapter Five: Study 1 – Tic-Tac-Toe

5.1 Introduction

To simulate the social setting an environment was setup in which agents could reside and play games. From this social setting the effects of social learning can be monitored by testing agents that are taught using social learning and comparing them to Self-Play agents. The first game modelled and tested with social learning in play is Tic-Tac-Toe. This game is used as a popular example in describing Reinforcement learning in games. This chapter describes the game, its setup and how the agents are put in a social environment. Tic-Tac-Toe has a smaller state-space and branching factor than other games such as Chess or Backgammon. It allows for a comparative test of agent performance as well as a population based test. The methodology discussed in this case study is also used in the second case study in the game of Morabaraba. The last part of the chapter deals with the testing of the agents and the results of these tests. The results of the social learning agents are compared against self-play agents and the behaviour of the agents is analysed.

5.1.1 Implementation for the Agent Environment

The agents and reinforcement learning algorithms were implemented using the C++ programming language due to its speed and memory (RAM) considerations. All of the implementation of this research was done using the C++ Reinforcement Learning toolbox (Neumann 2003). All of the models of the agents and environments were designed as objects in the toolbox. The games and agents are managed by a game

controller. The controller allocates who has to play next and also keeps track of game statistics such as wins, test results and how many times each agent has played games. It also matches winners and losers and thus implements the social frameworks described in section 5.4.

5.2 Modelling the Game and Learning

Tic-Tac-Toe (Todd, Reser. Paul, Laska. 2006) is a 3 x 3 board game. Two players place pieces on the board trying to connect three of their own pieces in a row. Figure 8 illustrates the player with the noughts defeating the player with the crosses.

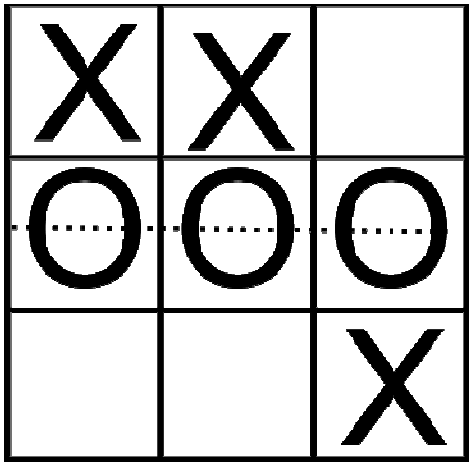

Figure 8: Tic-Tac-Toe Board

If two perfect players play a game of Tic-Tac-Toe, the game should always end with a draw (Schaeffer 2001). The game has been modelled with reinforcement learning in the past, and is frequently used to test and benchmark new algorithms in Reinforcement Learning (Ghory 2004). It has been recorded that agents take up to 50000 learning

episodes (Mannen 2003) to be able to play at a beginner level. In this experiment this is the amount of iterations used for the training of the agents.

5.3 The Game Model

5.3.1 States

To model the game for reinforcement learning the game was represented by 10 state variables. Nine of the variables can each have 3 different values which represent the positions on the board. Each position on the board can be empty (2) or have a nought (0) or cross (1). The tenth state represents the current player who is supposed to play. The total number of states is $3^9 \times 2 = 39366/2 \approx 19833$. The first total is divided by 2 due to the fact that one player cannot place all of his pieces on every spot on the board.

5.3.2 Actions

Actions are represented by number 1 to 9, each corresponding to a position on the board. The model also keeps track of which actions are available to an agent in a certain state. Thus an illegal move such as placing a piece on a board area that already has a piece is not possible. Actions are chosen using the method described in 3.3.5.

5.3.3 Reward Schedule

When an agent wins a game, it is rewarded with a reward of 1.0. When the agent loses, it then gets a reward of -1.0. When there is a draw, the agents get a reward of 0.0. For all other game states that are not terminal (not the end of the game) the reward is 0.0.

5.4 Agent Learning

5.4.1 The Environment and Agent Diversity

To simulate a social learning environment, multiple agents need to be created. In this research each agent is given its own identity. By identity it means that each agent has different initialization structural parameters. This induces what is termed as Structural Diversity (Masisi, Nelwamondo & Marwala 2008). The agents have the same learning algorithm but have different initialization options. These were chosen to be the learning rate, the discount factor and the lambda for the TD (λ) algorithm. The range of the agents structural parameters are shown below in Table 1.

TABLE I
AGENT IDENTITIES

Parameter	Range
Learning Rate	0.2 – 0.3
Discount Factor	0.95 -0.99
Lambda(λ)	0.9-1.0

The ranges in Table 1 were chosen by using the tests sets described in section 5.5 on a self-play population of agents. The ranges above resulted in the best set of agents that where created. As an agent learns, its opponents' policies are also changing and thus a learner will have to adjust its policy, to be a policy that can play against more than one known opponent.

5.4.2 Social Learning Training Configurations

Two training configurations are used in training the agents in the social setting. The two methods are derived from tournament styles; a modified Swiss (Just, Burg 2003) and a Round Robin system are used and compared.

5.4.2.1 Modified Swiss

In chess tournaments similar skilled players are paired to play rounds against each other. As a player wins games they are further paired with others who have won their games in their same skill level, this is termed the Swiss Tournament configuration (Just, Burg 2003). To simulate an agent always pairing with a player who was more knowledgeable the Swiss configuration was modified. In the modified Swiss configuration, agents are paired up to play one round of a game which is a full episode. When the game is finished there is either a winner, a loser or there is a draw. A tournament like structure was utilised for the agents to play in. The structure is shown in Figure 9.

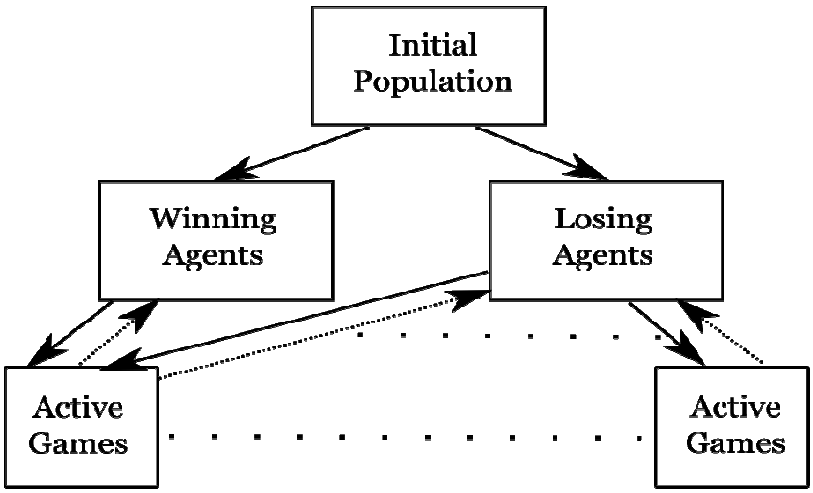

Figure 9: Tournament Learning Framework

The agents are first initialised and placed in an initial population. In the first iteration they are arbitrarily put in two sub-classes (Winning Agents and Losing Agents). In the second iteration and for the rest of the game the agents play games against each other. A winning agent is pitted against a losing agent. After a game/episode the winner is placed in the winner agent list and the losing agent in the losing agent list, thus a direct simulation of a mentor and a learner. At the end of a playing round the agents will be in two groups. A number of rounds are played and the process of pairing losers and winners repeats until the maximum number of rounds is reached. In this configuration there is a large focus on getting agents to be paired with players that have better experience. This is emulating the concept of a player learning from a "more knowledgeable" other/player. Another variation of this is when winners are paired with other winners and losers with other losers. This is closer to the normal Swiss tournament configuration.

5.4.2.2 Round Robin

In a Round Robin setting each agent plays against every other agent. There is no splitting of the group to winners and losers. After a round of playing, the agents are then pitted against the next agent. This is done until the maximum number of games is played. Round Robin configuration has less of a focus on having a more knowledgeable other or peer as an opponent. This configuration will be compared to the modified Swiss configuration.

5.4.2.3 Benchmarking Agent

Benchmark agents are also needed, thus another set of agents were created which are the self-play agents. These agents learn by only playing against themselves. Each agent plays a move as one player and then plays another move as the other player. These agents were created so as to be able to benchmark how well the social agents fair against conventional Self-Play learning.

5.5 Testing and Results

5.5.1 Board Test

Two tests were setup for the agents. The first test for the agents was an assessment on how well the agents perform at trying to pick correct actions in given test states. The Tic-Tac-Toe board is setup with pieces already on it. There is only one correct move that can be made. There were a total of 10 test boards with different levels of difficulty. The level of difficulty was achieved by testing different board configurations that required

different responses. The needed responses were then gauged as ranging from easy to hard. The agents are given one try at each board. Some boards have to reach a terminal state (end of game) while in others the agent has to choose an action that will result in forcing a draw in the game. There are 5 easy boards, 2 intermediate boards and 3 hard boards. The easy boards test if the agent can notice states that will make them win as shown in Figure 10.

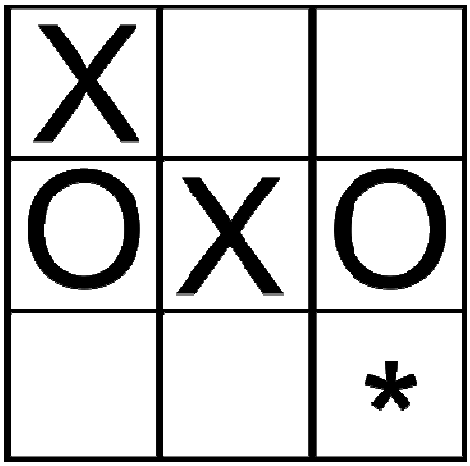

Figure 10: Cross to Play (Easy)

These are 1 move to win boards. They are relatively easy and test how the agents try to choose actions that will maximise reward in their next action choice. The intermediate boards are defensive boards where they test how well an agent can block a win by the other opponent, which means a loss for the agent, or force a draw. These tests show that the agent is trying to avoid losing or getting a lower return. An example is illustrated in Figure 11.

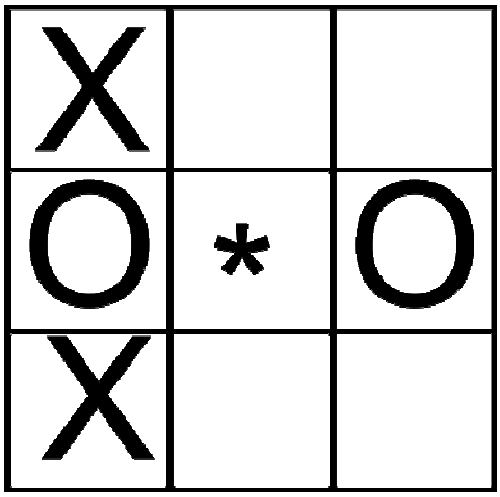

Figure 11: Cross to Play (Intermediate)

The difficult boards test how an agent can force a win his future move and not the next move. These are trickier but test how the agent is trying to maximize its future returns. The board is shown in Figure 12.

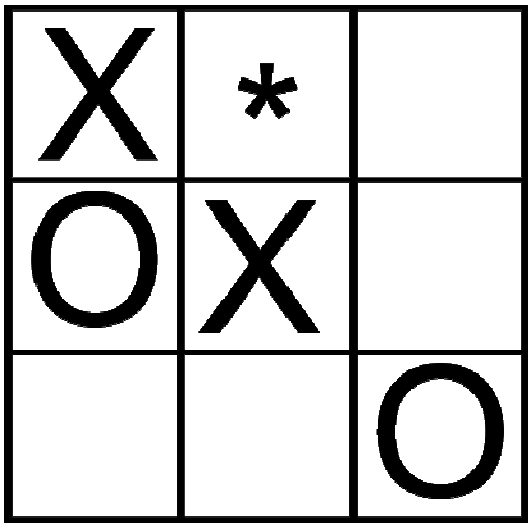

Figure 12: Cross to move (Hard)

5.5.2 Results for Board Test

The tests were carried through with different agent populations. First, 100 agents were created to learn in self-play mode. Their board test results are shown in Figure 13.

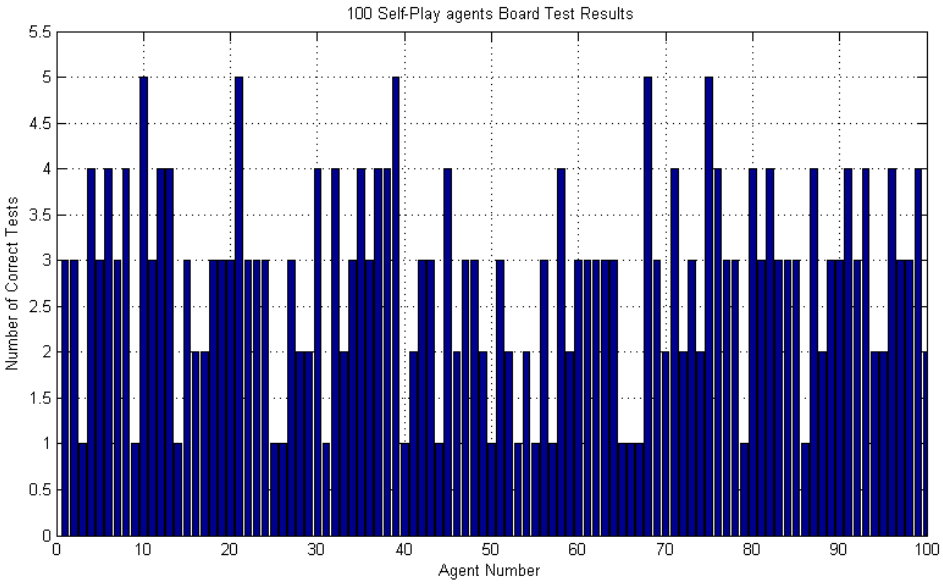

Figure 13: Self-Play Board Test Results

The best Self-Play agent succeeded in 5 of the tests. This was rare as it was only 5 agents in the whole population. On average, most of the Self-Play agents got 3 tests correct. For the social agents the agent populations were small. This was chosen to see how the dynamics change initially. The results of the tests for the modified Swiss configuration are shown below in Figure 14.

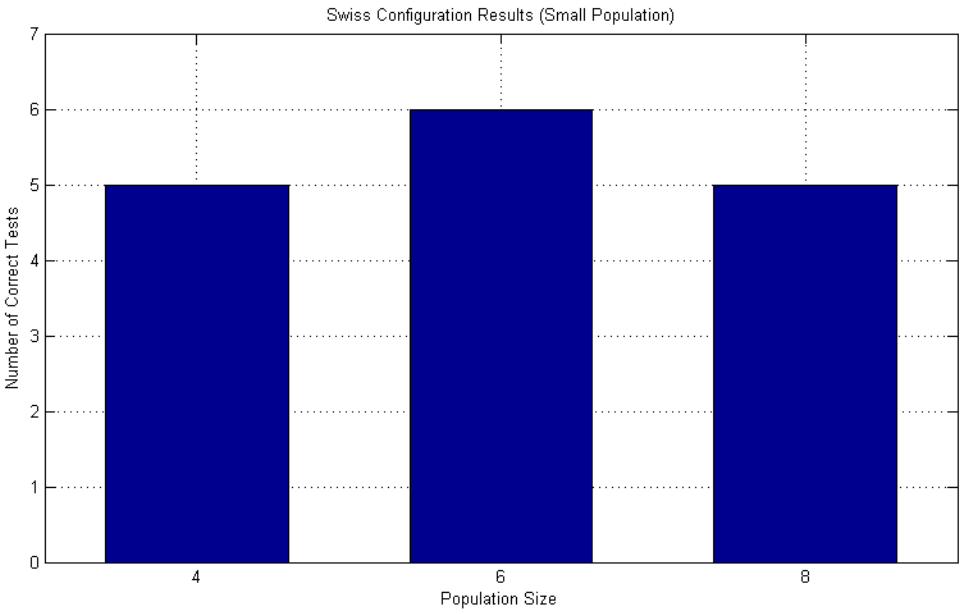

Figure 14: Board test results for Swiss Self Play

The best Self-Play agents get 5 moves correct while the best Swiss social agent in the 4 agent population gets 5 tests correct, while the one in the 6 agent population gets 6 tests correct. This implies that the Self-Play agents can play at a beginner level while the best agent in the 6 agent population is playing at an intermediate level compared to the other agents. None of the agents are advanced (correctly play the hard boards).

The other test was with the Round Robin Configuration. The results are in shown in Figure 15.

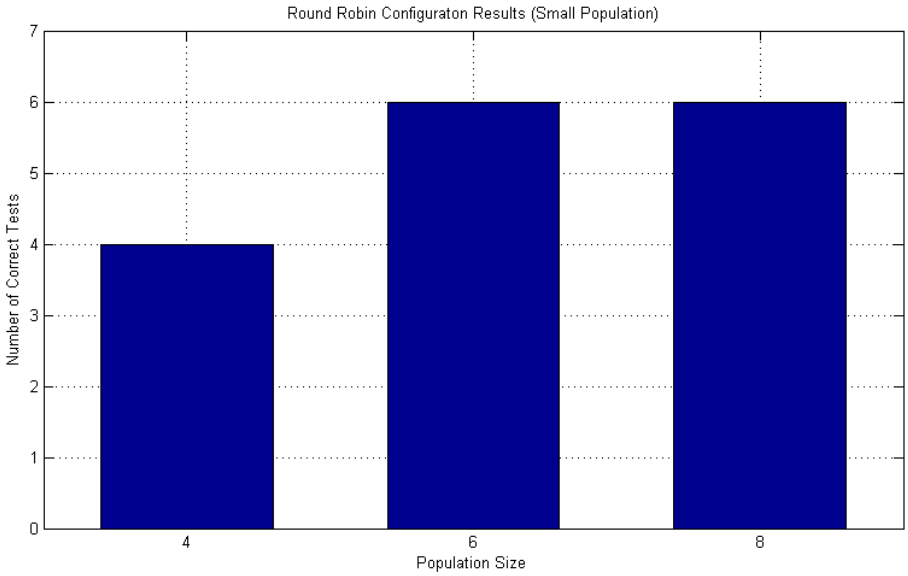

Figure 15: Round Robin Agent Performance

Another observation from the social agents is that as more agents (>8) are used in the population there is an increase in the number of intermediate agents in one generation. This is more evident in the modified Swiss tournament setting as opposed to the Round Robin configuration. Both configurations were tested with 16 and 32 agent sized populations. When the populations are increased with the modified Swiss configuration more than one intermediate agent emerges. In some stages up to 6 intermediate agents emerge. With the Round Robin configuration, 2 intermediate playing agents have emerged. By introducing multiple different agents as opponents in the training phases, one has been able to create agents that are superior to the S-P agent. This is further expanded in section 5.5.3.

5.5.3 Large Population Effects on Board Tests

Another study is concerned with dynamics that are created with large populations. This is different from small populations and is analysed. Tic-Tac-Toe is suitable for this as up to 300 agents can be created within 3 GB of RAM. Large state space games are not feasible for this type of test on a single computer. A distributed solution (Doran 1996) would be needed to create a social environment for a game such as Chess or Checkers, Backgammon or Morabaraba. With larger population models, one can study the effects of the social learning on a higher number of agents. The distribution and performance of 300 Modified Swiss tournament agents are shown in Figure 16. Most of the agents get 4 tests correct although others perform very well with 7 correct tests. Another test was then conducted with 300 agents in the Round Robin Tournament configuration. The results for this large population test are shown in Figure 17. Here the best agents only get 6 tests correct.

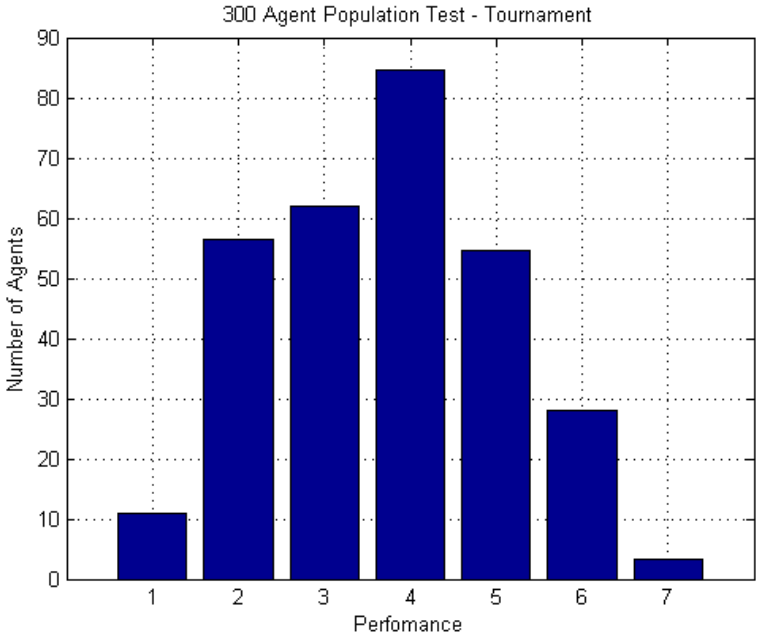

Figure 16: Results for 300 Swiss Agents

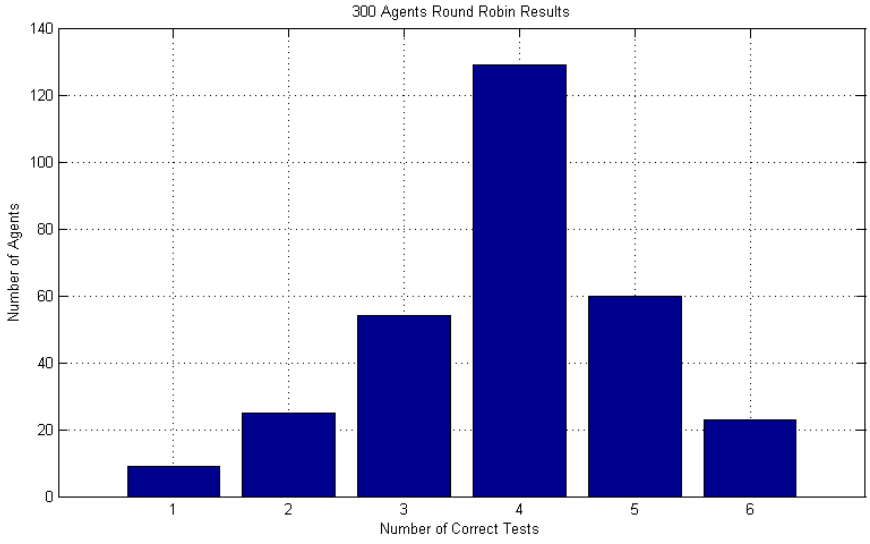

Figure 17: Round Robin 300 Agent Performance

From the above two results from the large population social agents, one can observe that there are now agents that get 7 tests correct. The Swiss configuration results with 4 agents that get 7 correct tests. The amounts of agents that get 6 correct tests are just under thirty. For the Round Robin configuration results in just a bit over 24 agents that get 6 tests correct. The number of agents that get 4 tests correct is larger in the Round Robin configuration than the Swiss configuration. The Swiss configuration has more agents that get 3 or less tests correct. The agents that do get 7 tests correct either get all easy and intermediate boards correct or get 5 easy 1 intermediate and 1 hard board. This then can be used to interpret those agents with 1 hard board correct as being advanced agents. The reason for not being able to make a correct move in the intermediate boards may be due to the limited number of games played. Thus a branch (a certain game progression from the different number of progressions) might have not been explored adequately. A second test was done on the Swiss configuration. This time instead of winning agents playing against losing agents in the next round, winners play against winners and losers play against losers. The results are in Figure 18.

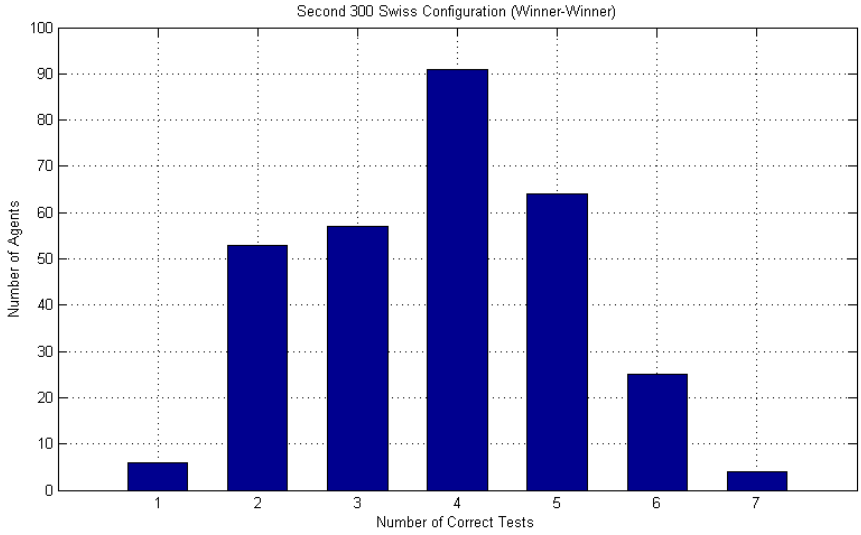

Figure 18: Second Swiss Configuration Results for 300 Agents

By changing the Swiss configuration, the numbers of agents that get 5 or more tests correct increases, but there is no increase in the agents that get 7 tests correct. The rest of the tests results stay similar to the initial Swiss results in Figure 16.

5.5.4 Play Test

The second test entails the agents playing games in a league setting. All of the agents are allowed to play with all the other agents for a specified number of games. This time they are not learning but are exploiting the knowledge they would have gained during the training stage. The wins, losses and draws are recorded and used for analysis. These are used to find which of the agents are the strongest. 5000 games are played by the agents against each other. This was applied to the best modified Swiss agents and Self-Play agents. This test was administered to gauge how the agents play against each other. It also checks how well the Social agents play against the Self-play Agents. The best Self-

Play agents from the board test play against the social agents. The social agents that performed well in the board test perform well in the play test. The best social agents win 2900 games on average out of 5000. The rest of the games resulted in draws or losses. Some social agents that had results of 5 or 7 correct tests on the board test still perform very well against the self-play agents. This hints on the possibility of the self-play agents not being able to adjust to different policies from the other agents. Furthermore for each of the agents, if they start the game first they have a higher chance of winning the game.

5.5.5 A Closer Look at the Agents

By monitoring the reward schedule, the agents can also be compared by how they learn and also perform during that learning. Figure 19 presents the reward schedules of 3 agents at the beginning of learning with the first 50 games.

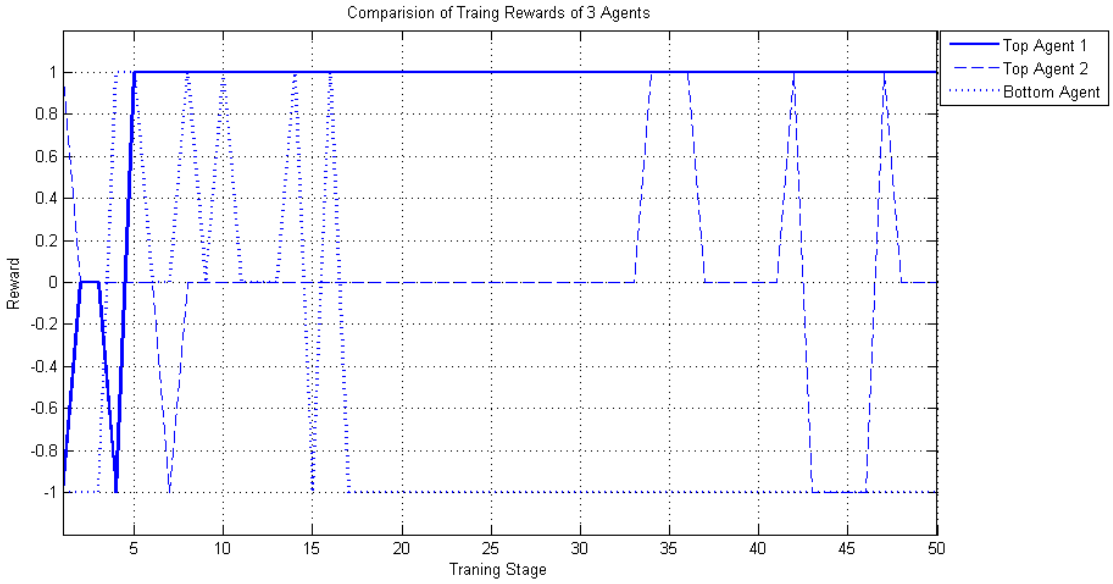

Figure 19: Comparison of 3 Social Learning Agents

From the figure, the agent that performs the worst starts with losses, then wins, draws and then starts losing as the episodes progress. Top Agent 1 (6 correct tests) starts with a loss but then starts winning all games after 5 games. Top Agent 2 (7 correct tests) starts with some wins and losses, but then starts drawing for a number of games. Top Agent 2 still loses in the training but at the end has one of the best performances in the board and play tests. From this one can see that learning does take place and is not only due to the initialisation parameters.

5.6 Conclusion

The agents all play the game at beginner level. This is indicated by how they perform at the board test. All of the agents fare very well on the easy boards but struggle on the intermediate ones and the difficult ones. There are a number of intermediate agents that are created in the social settings. Thus without increasing the number of training cycles, but by introducing non-stationary opponents in social settings the agent's performance have been improved.

The larger the population sizes the more likely the number of superior agents. In the larger populations even better playing agents are trained with the modified Swiss configuration. In the play test the beginner level of the agents is further shown as they all have higher chances of winning if they start the game first. The social agents fair better when playing against the Self-Play agents in the play test. The social agents have made it

possible to create agents that are superior to the best self-play agents. This is a positive result and merits the potential for the use of social methods in agent reinforcement learning.

Chapter Six: Study 2 – Morabaraba

6.1 Introduction

The second game studied is Morabaraba (Mind Sports South Africa 2008). It has a large state space and branching factor. Due to this, only small populations of Morabaraba agents could be implemented with the computing resources available. The game allows for an observation of how the social framework has impact on a higher state space game. The rest of this chapter describes this game and the results from the social learning experiments. The social learning training configuration and agent structural parameters are the same as those used in Tic-Tac-Toe in chapter Five.

6.2 Morabaraba

Morabaraba (Mind Sports South Africa 2008) is a two player African board game. It is similar to Nine Man Morris (Burns 2005). The board is made up of 24 positions. These positions are arranged in and around 3 squares. The game board is shown in Figure 20

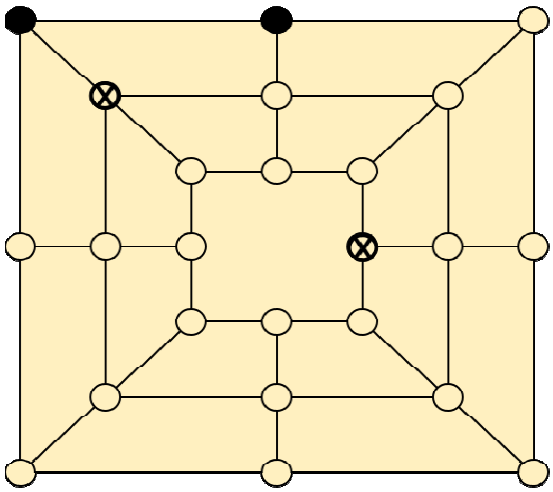

Figure 20: Morabaraba Board

Two players each have 12 pieces or cows at the beginning of the game. The players take turns placing and moving the cows on the board. The game has three stages:

- Placing the Cows: Players take turns first placing their cows. If a player manages to get three cows in a row they can remove one of the opponent's cows from the board.
- Moving the Cows: After both players have used up all of their 12 cows, they then take turns moving their cows around interconnected nodes. If a player aligns three in a row with their cows they can remove an opponent's cow.
- Flying the cows: When a player has 3 cows left they are then allowed to "fly" or move anywhere on the board where there is an empty node. They no longer have to move to an adjacent connected node.

Figure 20 illustrates the board with 2 cows from each player. The Morabaraba game is categorized as a complex war-game (International War Games Federation 2008). The experiment is also carried through with only the first stage of the game, which is placing the cows. A player gets a reward if they have captured the highest number of cows from the opponent at the end of the first round. The game has not gotten a large following in providing agents that can play using AI to learn. The programmed Morabaraba solution that exists exploits the developer's knowledge of Morabaraba and databases (Oellermann 2007). Part of using this game as a case study is to delve into the problems that might be faced while trying to use an AI learning agent to learn to play the game.

6.3 Modelling the Game

6.3.1 States

To represent the game to a reinforcement learning algorithm one has to define the states to be used to represent the environment. The game was modelled with 26 states (24 for the board, 1 for the game stage and the last for the player). 24 states are used to represent the positions on the board. The positions on the board can have 3 values, (2) for empty position, (0) for player 1 cow or (1) for player 2 cow. The numbers are just for representation to the learning method. Another state was used for representing which player is playing next (0 or 1). The last state was a representation of what stage the game is in. The game stages would be: placing a cow, removing an opponent's cow or end of game. All of the states are presented as being continuous in the toolbox as to be able to use the function approximation scheme. The upper bound of the number of possible states is $3^{24} \times 2 \times 3 \approx 1.7 \times 10^{12} / 2 = 0.85 \times 10^{12}$. This is larger than the number of possible states for Tic-Tac-Toe by a factor of approximately 43×10^{6}. Given that the computational running time for each game is longer than Tic-Tac-Toe with the implementation only 100000 games are used and again reinforce the need for a function approximation scheme. Games like backgammon used over 1 million games of Self-Play (Tesauro 1994).

6.3.2 Actions

There are 24 actions available to the agents at the beginning of the game. As the game progresses the availability of each action changes depending on what positions are empty

on the board. When an agent has to remove a cow the cows available to be removed are represented as actions available from the initial set of 24 actions.

6.3.3 Rewards

At the end of the game, the number of captured cows by both players is tallied. The player with the most captured cows is given a reward of 5.0. The player who has less captured cows is then given a reward of -5.0. When a player captures an opponent's cow they receive a reward of 1.0. This is done to encourage the agents to capture as many cows as possible. At all other times and when there is a draw the reward is 0.0. A complete episode is taken as a complete game. A step in the game is when a player makes a move. Thus multiple steps take place before a reward of 5.0 or -5.0 is given. Action selection is as described in 3.3.5. This reward schedule encourages the agents to not only be rewarded for winning the game but also to try and capture as many of the opponents cows as possible during the round. It also punishes the agent that loses cows and thus should result in agents trying to protect their cows while trying to capture as many of the opponents cows.

6.3.4 Function Approximation

To learn the *Q*-Function a radial basis function network was used as described in 3.3.6. For best results adjustment of the variance parameter has to be done, in line with the number of partitions for each of the board positions. Each board position can have 3 values and is portioned into 3. The variance is tweaked by checking how each value has

an effect on the playing performance of the agents. Unfortunately this is a variable process and has to be done when building function approximation schemes that learn from TD (λ). Thus a researcher always has to carry out the adjustments (Thrun, Schwartz 1993). There is active research in trying to find more generalised ways of solving this problem (Hurwitz 2007). In Radial Basis functions the centre's and their associated variances are normally initialised using an unsupervised learning method on part of the training data. In the case of reinforcement learning there is no training data and it is up to the developer to adjust the settings so as to have the best performance. Thus a stumbling block in building a good Morabaraba playing agent is finding the best function approximation scheme to use as well as an algorithm that will maximise its performance. For the purpose of this research, making the best Morabaraba agent was not the goal. The goal was to observe the performance of reinforcement learning agents in social settings. A reinforcement learning agent using tabular methods was introduced with Tic-Tac-Toe and now an agent that used a function approximator is used to play a more difficult game.

6.4 Testing

6.4.1 Board Test

The test setup for the agents was an assessment on how well the agents perform at trying to pick correct actions in given test states. The Morabaraba board is setup with a predetermined state. From this state, one or two moves are judged as being the best to make. This could be a transition into a state where the player has three cows in a row or the player blocks a sequence of the opponent's cows. There were a total of 10 test boards

with different levels of difficulty. In the case of Morabaraba, the test boards are not difficult but are designed to test if the agents have learnt a strategy that will give them higher rewards. The agents are given one try at each board. Some boards have to reach a terminal state (end of game) while in others the agent has to choose an action that will block an opponent from completing a three in a row. A typical test configuration of the board is shown in Figure 21.

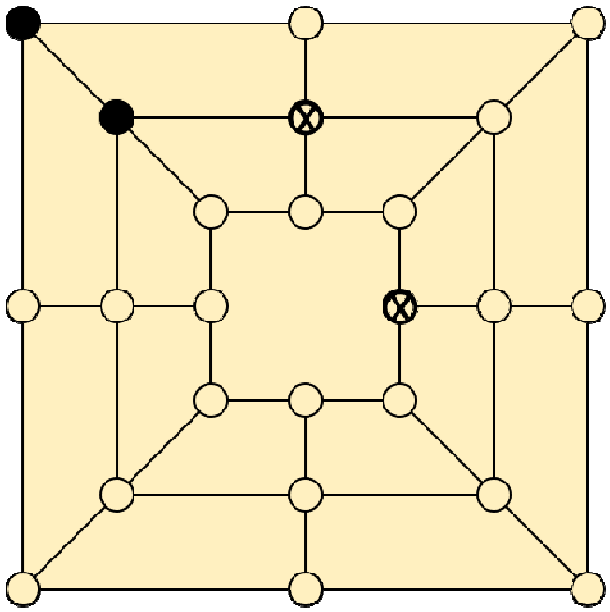

Figure 21: Test Board - Black to move.

The above board is an example of a “one move to capture” boards. They are relatively easy and test how the agents try to choose actions that will maximize reward in their next action choice. In this case the player using the black cows will be able to capture the opponent’s cow if they complete their three cows in a row. The other boards test if the agent can block another agent from completing their three in a row. In the case of Figure

21 if the player using the crosses cows were to move, then the best move would be to move to the space that would block the opponents three in a row.

6.4.2 Board Test Results

Firstly a test set was created with Self-Play agents. 50 agents were created and used Self-Play to learn. The results are shown in Figure 22.

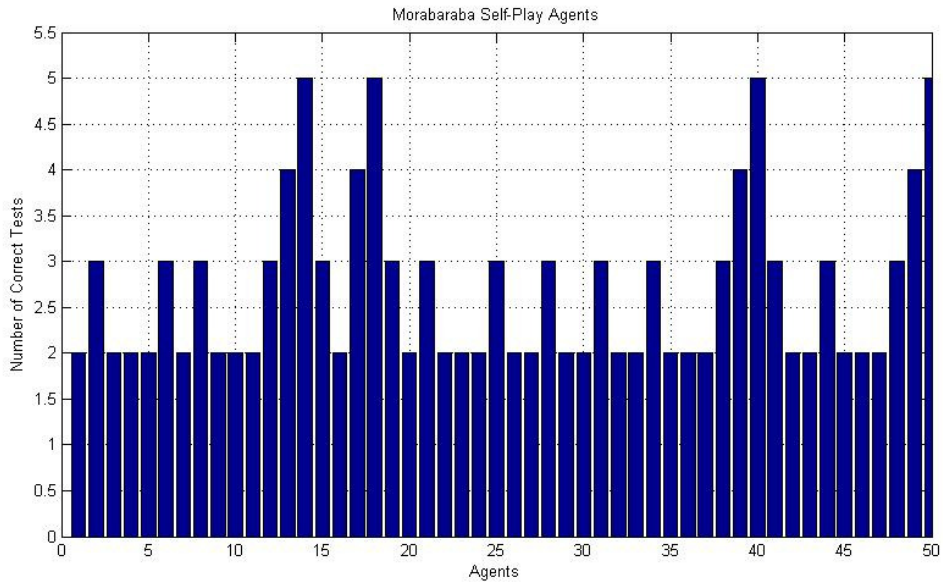

Figure 22: Morabaraba Self-Play Results

The majority of the agents get two tests correct. The best Self-Play agents manage to get 5 tests correct. The social agents were tested with different social population sizes. There are 4 population sizes that were used in the investigation: 4, 6, 8 and 10. Each of these was tested 5 different times with the board test (meaning they have been trained differently 5 times). The tests were carried through with different agent populations and the average of the results is presented below. From the social configurations the best

agent's tally is used. The results of the board tests for the modified Swiss configuration are shown below in Figure 23.

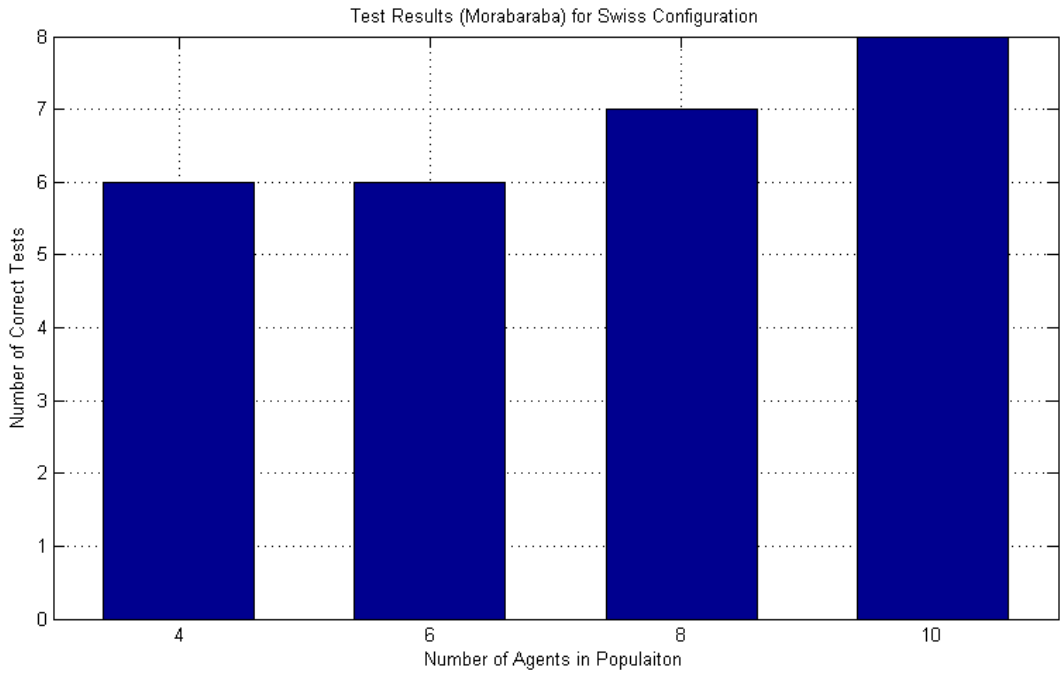

Figure 23: Swiss Configuration Results

The best modified Swiss configuration agent manages to get 8 tests correct. This is with a population size of 8. The smaller population sizes result in an emergence of agents that still perform better than Self-Play. The results for the Round Robin configuration are shown in Figure 24.

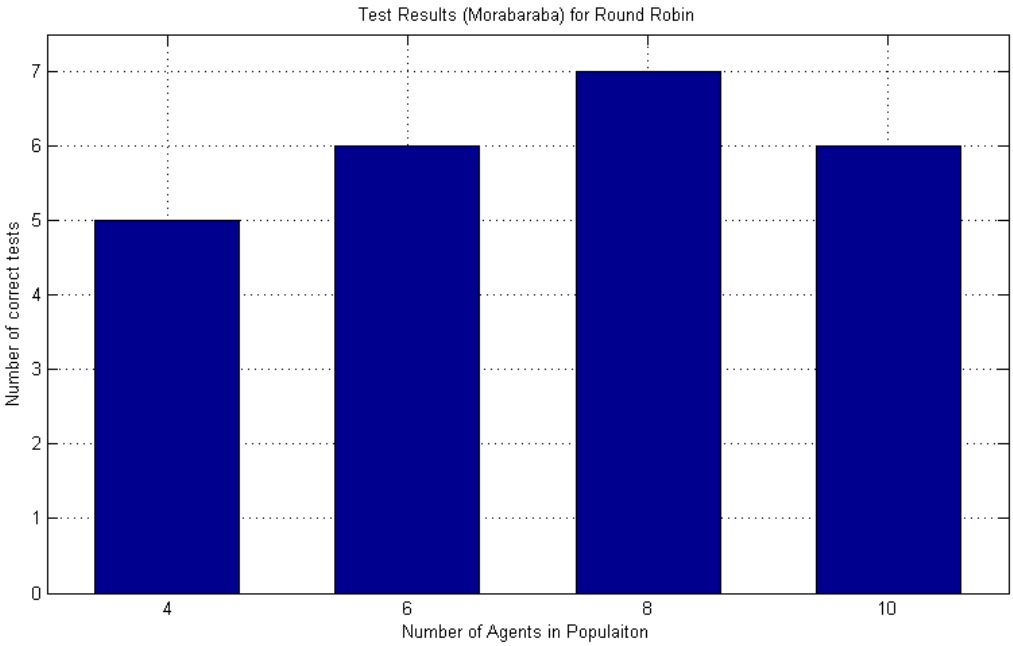

Figure 24: Round Robin Results

In the Round Robin configuration the best Morabaraba social agent gets 7 tests correct within a population of 8. From both these results it can be deduced that as the number of agents in a population is increased the more the overall best agent performs better than the self-play agents. The social setting of using the modified Swiss would be expected to perform better. This configuration allows agents who are weaker to observe strong players. Thus the overall strength of the population should increase at every game. By using the agents in the social setting the experiment was able to spawn agents that are superior to self-play agents. The increase in population size also increases the computational complexity of the overall agent experiment. Thus having higher population sizes increases the amount of computing power needed thus increasing computational cost. This problem was partially solved by using function approximation,

but this also suffers the curse of dimensionality (Sutton, Barto 1998) and thus is limiting. By increasing the size of each state the training also becomes more complex thus resulting in more memory needed and more time needed to train the methods.

6.5 Play Test

In the play test the best agents (from the board test) from the self-play games played 5000 games against the social agents. The best social agents (from the board test) performed relatively well compared to the weaker agents. The best social agents won 2300 games against the Self-Play agents. The rest of the games were lost (2000) or drawn (700). The best Self-Play agent won 2100 games, lost 2045 and the rest were drawn. Similar results were found when running the tests multiple times. From these it was inconclusive if the social agents learned better strategies than the self play agents.

6.6 Conclusion

By using a human inspired observational/social learning framework in training reinforcement learning, reinforcement learning agents that play basic Morabaraba have been developed. Morabaraba has a large state space and thus a learning algorithm that uses radial basis function networks for function approximation was used. This function approximation scheme was used in training the agents at a basic beginner level. The use of social learning or evolutionary algorithms within AI is needed in order to be able to build machines that are more robust and less prone to sub-optimal behaviour. The results

from the board tests indicate that the modified Swiss social configuration have slightly better strategies.

The play test though reveals that the agents (Self-Play and Social) play against each other at a relatively same level. This then would make the results inconclusive. One problem is that the population sizes of the agents cannot be increased as there is a limitation on computational resources. If this were made possible, from analysing results from Tic-Tac-Toe, it could be possible to build better Morabaraba agents. The results from the board tests are promising and comparing them to the Tic-Tac-Toe tests it is expected that with an increase in population there should also be an increase in performance in the play tests; resulting in robust Morabaraba playing agents.

Chapter Seven: Conclusions and Suggestions for Further Work

7.1 Conclusions

The goal of this research was to observe and analyse what changes (if any) that would occur if agents were put into a social setting during their training phases. The stimulation of social learning with reinforcement learning agents had positive results in the first case study of Tic-Tac-Toe against traditional Self-Play agents. All of the agents are initialised with different structural parameters and thus are diverse set of opponents with dynamic strategies. The modified Swiss social configuration yielded agents that played better and also performed best in the board tests. The Round Robin social setting also had positive results but not as good as the modified Swiss. The larger the population for the modified Swiss agents the more advanced agents emerged. From this, one can conclude that the more the agents the better the skills transfer in the social learning. The agents have more dynamic opponents to play against. As the agents play against each other and learn, their policies change. In order for the best agents to stay on top they change their own strategies so as to be better equipped to play well against most of the opponents during the learning phase. The play test for Tic-Tac-Toe revealed that the best social agents were superior to the self-play agents in their play. The best social agents are able to win more games against the different Self-Play agents and other social agents. Thus the best social agents developed policies that were less likely to be mal-adaptive. Their policies played well against a varying number of different opponents.

The second case study used the board game of Morabaraba. During learning, tables were replaced with a function approximation scheme for generalisation. The results for Morabaraba are inconclusive but positive. There is a slight improvement in the social agent's performance in the board tests but this though does not lead to domination in the play test as in the Tic-Tac-Toe. The best social agents win a similar number of games as the Self-Play agents when they play against each other after learning. The modified Swiss configuration was again better than the Round Robin social configuration in stimulating the emergence of agents that perform well in the board tests. There were limitations with the computational complexity due to the Radial Basis functions but agents that play Morabaraba intelligently at a basic level were created.

7.2 Further Work

As the purpose of this research was to observe how agents react to social settings where they learn, where the other agents are also learning while playing there is room for improvement. The following recommendations for future work are suggested:

- The use of evolutionary methods can be further mimicked. The agents could be tested after a set number of training games. Then weak agents could be removed from the training group and leave only stronger agents to interact. Another option is to then find ways to allow "reproduction" of the good agents in order to introduce new agents to the training phase. This would be mimicking such algorithms as Genetic algorithms.

- Given the results from Tic-Tac-Toe, a study can be carried through in trying to get some of the world's best reinforcement learning game playing agents to learn while playing against each other and not only against themselves.
- The focus of this research was not to develop best Morabaraba reinforcement learning agent but a functional one. The RBF networks used were adequate for the tests performed but had limitations. Developing an agent that uses a less demanding function approximation scheme can be done. From this a social learning study of a large state space game can be performed with larger population sizes.

References

Baird, L. & Moore, A.W. 1999, "Gradient Descent for General Reinforcement Learning", *Advances in Neural Informaiton Processing Systems,* vol. 11, pp. 968-974.

Bandura, A. 1977, *Social Learning Theory,* Prentice Hall, Englewood Cliffs, NJ, USA.

Bishop, C.M. 1995, *Neural Networks for Pattern Recognition,* Oxford University Press, USA.

Bouzy, B. & Chaslot, G. 2007, "Monte-Carlo Go reinforcement learning experiments", *2006 IEEE Symposium on Computational Intelligence and Games*, pp. 187.

Burns, B. 2005, *The Encyclopedia of Games,* Silverdale books, United Kingdom.

Detroit City Chess Club 2008, 31 October 2008-last update*, Detroit City Chess Club Photos* [Homepage of Detroit City Chess Club], [Online]. Available: http://www.detroitcitychessclub.com/chessphotos.html [2008, November/01] .

Doran, J. 1996, "Simulating Societies using Distributed Artificial Intelligence", *Social Science Microsimulation*, pp. 381.

Dorigo, M. & Stützle, T. 2004, *Ant Colony Optimization,* 1st edn, MIT Press, USA.

Dubel, I.C.L., Bsc, L.L., Brandsema, I.J. & Bsc, S.S. 2006, *Checkers Reinforcement learning project: AI Checkers*.

Filho, C.J.A.B., de Lima Neto, F.B., Lins, A.J.C.C., Nascimento, A.I.S. & Lima, M.P. 2008, "A Novel Search Algorithm based on Fish School Behavior", *IEEE International Conference on Systems, Man and Cybernetics (SMC 2008)*, pp. 2646.

Ghory, I. 2004, *Reinforcement Learning in Board Games*, Department of Computer Science, University of Bristol.

Gold, A. 2005, "Academic AI and video games: A case study of incorporating innovative academic research into a video game prototype", *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 141.

Hurwitz, E. & Marwala, T. 2007, "Learning to bluff", *Proceedings of th IEEE International Conference on Systems, Man and Cybernetics*, pp. 1188.

Hurwitz, E. 2007, *Multi-agent modelling using intelligent agents in competative games,* M.Sc. Thesis, University of the Witwatersrand, Johannesburg, South Africa.

International War Games Federation 2008, *, The International Wargames Federation* [Homepage of International War Games Federation], [Online]. Available: http://www.theiwf.org/ [2008, August/30] .

Just, T. & Burg, D. 2003, *U.S. Chess Federation's Official Rules of Chess,* 5th edn, McKay.

Kalles, D. 2008, "Player co-modelling in a strategy board game: Discovering how to play fast", *Cybernetics and Systems,* vol. 39, no. 1, pp. 1-17.

Kennedy, J. & Eberhart, R. 1995, "Particle swarm optimization", *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942.

Kotsiantis, S., Zaharakis, I. & Pintelas, P. 2006, "Machine learning: a review of classification and combining techniques", *Artificial Intelligence Review,* vol. 26, no. 3, pp. 159-190.

Laird, J.E. & Lent, M.v. 2000, "Human-Level AI's Killer Application: Interactive Computer Games", *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 1171.

Mannen, H. 2003, *Learning to play chess using reinforcement learning with database game, Masters Thesis*, Utrecht University.

Masisi, L., Nelwamondo, F.V. & Marwala, T. 2008, "The effect of structural diversity of an ensemble of classifiers on classification accuracy", *Proceedings of the IASTED International Conference on Modelling and Simulation (Africa-MS)*, pp. 135.

Miller, N.E. 1944, "Social Learning and Imitation.", *The Journal of nervous and mental disease,* vol. 99, no. 2, pp. 221.

Mind Sports South Africa 2008, , *Morabaraba*. Available: http://www.mindsportssa.freeservers.com/rules-morabaraba.htm [2008, August/30] .

Moscato, P. 1999, "Memetic algorithms: a short introduction", *Mcgraw-Hill'S Advanced Topics In Computer Science Series*, pp. 219-234.

Neto, H.C. & Julia, R.M.S. 2008, "LS-Draughts - A draughts learning system based on Genetic Algorithms, neural network and temporal differences", *IEEE Congress on Evolutionary Computation, CEC*, pp. 2523.

Neumann, G. 2003, , *Reinforcement Learning Toolbox* [Homepage of Neumann, Gerhard.], [Online]. Available: http://www.igi.tugraz.at/ril-toolbox/general/overview.html [2008, June/30] .

Oellermann, A. 2007, 30 September 2007-last update, *Adam's Happy Electronic Morabaraba* [Homepage of Oellermann Adam], [Online]. Available: http://www.morabaraba.org/ahemcomputer [2008, October/21] .

Okulicz, E., Vialle, W. & Verenikina, I. 2007, "The Development of Expertise Within a Community of Practice of Scrabble Players", *Learning and Socio-cultural Theory: Exploring Modern Vygotskian Perspectives International Workshop,* vol. 1, no. 1, pp. 65-76.

Orr, M.J.L. 1996, *Introduction to radial basis function networks*, Tutorial edn, Centre for Cognitive Science, University of Edinburgh.

Park, J. & Sandberg, I.W. 1993, "Approximation and Radial-Basis-Function Networks", *Neural computation,* vol. 5, no. 2, pp. 305-316.

Pollack, J.B. & Blair, A.D. 1997, "Why did TD-Gammon Work", *Advances in Neural Information Processing Systems*, pp. 10.

Roth, E. 2002, "The Economist as Engineer: Game Theory, Experimental Economics and Computation as Tools of Design Economics", *Econometrica,* vol. 70, pp. 1341-1378.

Runarsson, T.P. & Lucas, S.M. 2005, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go", *IEEE Transactions on Evolutionary Computation,* vol. 9, no. 6, pp. 628-640.

Russell, S.J., Norvig, P., Canny, J.F., Malik, J. & Edwards, D.D. 2003, *Artificial intelligence: A modern approach,* 2nd edn, Prentice Hall, Englewood Cliffs, NJ.

Sandholm, T.W. & Crites, R.H. 1996, "On Multiagent Q-learning in a Semi-Competitive Domain", *Lecture Notes in Computer Science: Proceedings of the Workshop on Adaptation and Learning in Multi-Agent Systems,* vol. 1042, pp. 191-205.

Schaeffer, J. 2001, "A Gamut of Games", *AI Magazine,* vol. 22, no. 3, pp. 29-46.

Shannon, C.E. 1950, "Programming a computer for playing chess", *Philosophical Magazine,* vol. 41, no. 4, pp. 256-275.

Singh, S., Norvig, P., Cohn, D. & Inc, H. 1996, *How to make software agents do the right thing: An introduction to RL. http://envy.cs.umass.edu/People/singh/RLMasses/RL.html*, Adaptive Systems Group, Harlequin Inc.

Standage, T. 2002, *The Turk: The Life and Times of the Famous Eighteenth-century Chess-playing Machine,* Walker & Co.

Stout, A., Konidaris, G. & Barto, A. 2005, "Intrinsically Motivated Reinforcement Learning: A Promising Framework for Developmental Robot Learning", *Proceedings of The AAAI Spring Symposium on Developmental Robotics*.

Sutton, R.S. & Barto, A.G. 1998, *Reinforcement Learning : An Introduction,* 1st edn, MIT Press, Cambridge, Mass, USA.

Tesauro, G. 1994, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play", *Neural Computation,* vol. 6, no. 2, pp. 215-219.

Thrun, S. & Schwartz, A. 1993, "Issues in Using Function Approximation for Reinforcement Learning", *Proceedings of the Fourth Connectionist Models Summer School*.

Todd, Reser. Paul, Laska. 2006, *, Tic Tac Toe* [Homepage of Cyber Oculus], [Online]. Available: http://www.cyberoculus.com/tic-tac-toe.asp?Action=Rules [2008, June/30] .

van den Bergh, F. & Engelbrecht, A.P. 2004, "A Cooperative approach to particle swarm optimization", *IEEE Transactions on Evolutionary Computation,* vol. 8, no. 3, pp. 225-239.

Vygotsky, L.S. (ed) 1978, *The development of higher psychological process*, Harvard University Press, Cambridge, Mass, USA.

Wolfgang von Kempelen (ed) 1783, *Briefe über den Schachspieler des Hrn. von Kempelen, nebst drei Kupferstichen die diese berühmte Maschine vorstellen*, 1st edn.

Wong, K.Y.M., Lim, S.W. & Gao, Z. 2005, "Multi-agent Cooperation in Diverse Population Games", *Advances in Neural Information Processing Systems,* vol. 17, pp. 1521.

Wong, K.Y.M., Lim, S.W. & Gao, Z. 2004, "Diversity and Adaptation in large population games", *International Journal of Modern Physics B,* vol. 18, no. 17/19, pp. 2422.

Wooldridge, M.J. 2002, *Introduction to Multiagent Systems,* John Wiley & Sons, Inc., New York, NY, USA.