2011

An Exploration of Multi-agent Learning Within the Game of Sheephead

Brady Brau
Minnesota State University, Mankato

An Exploration of Multi-agent Learning Within the Game of Sheephead

by

Brady Charles Brau

An Alternate Plan Paper submitted in partial fulfillment of the requirements for

Master of Science

in

Computer Science

Minnesota State University, Mankato

Mankato, Minnesota

November 2011

An Exploration of Multi-agent Learning Within the Game of Sheephead

Brady Charles Brau

This alternate plan paper has been examined and approved by the following members of the alternate plan paper committee.

Dr. Dean Kelley, Advisor

Dr. Rebecca Bates

Dr. Daniel Swart

Abstract

In this paper, we examine a machine learning technique presented by Ishii et al. [7] used to allow for learning in a multi-agent environment and apply an adaptation of this learning technique to the card game Sheephead. We then evaluate the effectiveness of our adaptation by running simulations against rule-based opponents. Multi-agent learning presents several layers of complexity on top of a single-agent learning in a stationary environment. This added complexity and increased state space is just beginning to be addressed by researchers. We utilize techniques used by Ishii et al. to facilitate this multi-agent learning. We model the environment of Sheephead as a partially observable Markov decision process (POMDP). This model will allow us to estimate the hidden state information inherent within the game of Sheephead. By first estimating this information, we restore the Markov property needed to model the problem as a Markov decison problem. We then solve the problem as Ishii et al. did by using a reinforcement learning technique based on the actor-critic algorithm [13]. Though our results were positive, they were skewed by a rules-based implementation of part of the algorithm. Future research will be needed complete this implementation via a learning-based action predictor. Future research should also include testing against human subjects thus removing the rules-based bias inherent in the current algorithm. Given increased processing power, disk space, and improved AI techniques such as the techniques described above, complex multi-agent learning problems which once proved difficult may find solutions

from the AI world.

Glossary

Belief State	a set of probabilities making up the learning agent's understanding of the current state space
Blind	within the card game Sheephead, two cards that are placed face down which can be picked up during the bidding stage
Fail	the set of cards that are not trump
Hand	describes both a complete process of dealing the cards, the bidding stage, the play of all cards by all players, and scoring as well as the set of cards a player has at any time
Nola	within the card game sheephead, a seperate game that results from no player picking up the blind during the bidding state that ends by either one player taking all of the tricks in which the other four players pay him, or by the last player who took a trick paying the other four players
RL	reinforcement learning

Schmere	playing points on a trick to be taken by someone else, typically used in terms of playing points on your partner's trick in order to increase the team score for that hand
Spitz	another name for the joker in sheephead
State Space	consists of all possible variables (both hidden and visible) making up the existing environment within the learning scenario
Trick	the play of a single card by all players, won by the player who plays the highest cards based on the rules of Sheephead and the current type of game played
Trump	a set of cards within Sheephead during regular (not Nola) play that are always higher than non-trump card regardless of the order of play
Unknown Card	a card placed face down by the player who picks up the blind when they do not have a fail suit in their hand without also having the ace of that fail suit, which is played when the called ace trick is played

Contents

List of Tables

List of Figures

1 Introduction

Many machine learning scenarios involve very dynamic environments, making traditional static learning techniques difficult to employ. We define static learning as machine learning within the context of a stationary or static environment. This type of learning includes problems such as that of solving a Rubik's cube or a crossword puzzle.

The dynamic environment described above is especially true for multi-agent games where there exists a vast amount of state space, much of it hidden from the learning agent. Multi-agent learning describes a scenario in which two or more agents exist that are employing some form of machine learning within the environment and within the confines of the scenario or game setup. State space describes the variables that define the environment or context of the given game. For example, in the card game Hearts, state space would define the cards that have been played, the cards in each player's hand, and the state of the current hand and trick. Games such as those that involve multiple players, dynamic team attributes including individual and team rewards, and complex rules provide a challenge for AI techniques such as decision trees, neural networks, and other supervised training algorithms.

Games involving hidden state space or variables belong to a group of games classified as *imperfect information* games. The games Hearts and Sheephead, specifically, which are both discussed in this paper, belong to a subset of imperfect information games referred to as *perfect recall* games.

They are defined as such due to previous play being completely visible while current state (or in the case of Hearts and Sheephead cards in the player's hands) remains hidden. Perfect recall games have been shown to be NP-hard for computer simulation [4].

Challenge such as that of a dynamic learning environment begin to approach challenges of everyday life and thus are becoming more important to solve. As our machine learning techniques, as well as the technology they operate on, improve we are able to start tackling these difficult problems. We are very good at solving very specific, constrained problems, but seem to struggle when approaching much more realistic, "real-world" scenarios such as operating within a group of other dynamic agents. These challenges continue to push us to understand how the human brain is able to accomplish what machine learning currently cannot.

In this paper, we propose a technique for learning within this scenario of multi-agent learning, as well as dealing with the difficulty of imperfect information games. We will delve into the details of the learning algorithm presented by Ishii et al. [7], and then present an implementation of our own research based on their research.

The game we will be using for this project and future work is our family card game Sheephead. Sheephead is a multi-player card game in which every player keeps his or her own score, but plays each individual hand with a partner or partners. These rules, as well as others, contribute to making it a good example of a dynamic learning environment. We will describe these

and other complexities associated with this game in Chapter 2.

Ishii et al. base their algorithm or learning technique on the card game Hearts [7]. This is similar to Sheephead in that each individual must learn while playing against other individual agents who will adjust and adapt their playing techniques. Hearts, as well as Sheephead, consists of a very large hidden state space making brute force types of approaches next to impossible.

In this paper, we will first examine the background details involving the algorithm presented by Ishii et al. that will be used in this work. This includes examining the various components of this algorithm, as well as those that parts of the algorithm presented here are based on. We will then give a brief description of the game of Sheephead, including a subset of the rules. (We only describe a subset of rules for brevity's sake, as well as to focus on those that are used in this simulation.) Methods will then be presented in the initial setup of the project and simulation. We will then conclude with a presentation of initial results of the simulation and summary of work currently done on the project, as well as suggested future work.

2 Background

2.1 Games and Machine Learning

Machine learning techniques for games such as checkers [12], Hearts [7], and chess [11] have seen significant gains with techniques such as Bayesian networks, neural networks, and hidden Markov models. Research such as

Tesauro [17] involving learning within the game of Backgammon and Ginsberg involving learning within the game of bridge [6] continue to advance this area of learning.

Tesauro's work involved utilizing neural networks and reinforcement learning to play the game of backgammon. It was very effective in interrogating a large number of scenarios and learning throughout play resulting in a program capable of performing well against even expert players. An important characteristic of Tesauro's work is the fact that the entire game is observable, i.e., all states within the game are known by the pieces on the board [17].

Ginsberg, prior to Tesauro, focused on a learning agent capable of playing the game Bridge. This research worked on solving problems with imperfect, unobservable games such as that of our research. It used techniques such as partitioned search and Monte Carlo methods. GIB, the learning agent created in their work, performed very well at competitions with various other learning agents. Over two very large competitions, it lost only one match to another learning agent. No resultsm, though, were given regarding play against a human agent. So, while performance was good, it was still just relative to other learning agents and algorithms [6].

We will investigate one specific approach within this project which builds on several learning techniques to achieve learning and performance objectives. This approach, presented by Ishii et al. [7], shows a reinforcement learning (RL) technique which works with the game of hearts, modeled as a partially observable Markov decision process (POMDP). After we describe

the problem of multi-agent learning within the context of perfect and imperfect information games further, we will then delve into the details of several of the techniques used within the algorithm.

2.1.1 The Multi-Agent Learning Problem

While much of the past machine learning research was focused on single-agent learning scenarios, this has recently shifted to include multi-agent scenarios [6]. The complexity of the multi-agent learning scenario arises because of to several aspects including accounting for the opponent's strategy, accounting for that strategy changing, and dealing with the increased state space caused by hidden variables due to the multiple agents. While reinforcement learning techniques such as Q-learning, a reinforcement learning technique introduced by Watkins in 1989 [18], have done well in single agent repeatable scenarios, they have struggled with multi-agent problems or games.

The major difficulty with many of the single-agent reinforcement learning techniques is that the mathematics relies on the environment being stationary, or consisting of a finite set of unchanging variables. More specifically, if there are two or more agents, the scenario will not actually have a Markov property utilized in formulating the problem as a Markov decision problem [7]. This presents a problem when other agents act in a manner or strategy unknown to the learning agent. Further complicating things, other agents will be learning or adjusting their strategy just like the main learning agent will, thus resulting in a constantly evolving game state [14]. These difficulties

lead to an ever increasing state space in regards to the problem description due to the addition of an exponentially large set of variables. This ever increasing state space makes solutions more difficult and complex. Brute force techniques are no longer an option and reinforcement learning becomes a necessity [11].

Multi-agent learning problems also introduce the additional complexity of needing to adjust quickly to an ever changing environment. This includes both the change in strategy of the learning agent and actions by the learning agent regarding the other learning agents as well as the change to the environment caused by the other agents. Overall, the problem of dealing with a multi-agent environment is much more complex than that of a single-agent environment.

2.1.2 Perfect Recall Games

Games dealing with unobservable state information are referred to as *imperfect information games*. Imperfect information games can be further classified based on the agent or agents' ability to observe all previous plays made within the game. *Imperfect recall games* describe those games in which a play can be made by a player that is not observable to every other player. We look at *perfect recall games* in this research in which all plays are observable to all players [5].

Perfect recall games means that all previous play is known and discrete. This is advantageous as it allows us to utilize reinforcement learning tech-

niques without having to estimate past performance. If past performance was not discretized, we would need to constantly adjust our learning to the constantly adjusting perceived performance of past actions. Also, due to the context of our research, i.e., card games, we have a set block of play that can be "assessed" at very specific times, namely at the end of each trick and each hand.

The difficulty in perfect recall games, or more generally in imperfect information games, arises from the lack of state knowledge during play. In the context of Sheephead, this results from not knowing the cards in the other player's hands. This can be addressed by estimation techniques to gain a belief state of what the current environment is. We use information such as historical play, game knowledge and known state information to estimate probabilities of the missing state information.

2.2 Ishii et al. Algorithm Background

In this section we describe the algorithm techniques used by Ishii et al. and state the specific issues addressed in their work. The issues are all related to learning within the confines of a multi-agent, dynamic environment or scenario. The three aspects represent symptoms of such an environment and are listed below:

- Learning within a scenario involving other learning agents.
- Learning within a scenario involving unobservable state space.

- Learning within the bounds of available processing time and space.

The techniques used as part of the Ishii et al. algorithm are described below and are made up of generic techniques in dealing with areas such as function approximation, modeling unobserveable state space and feature extraction. The techniques are all used in order to deal with scenarios described above in an effective and efficient manner.

2.2.1 Partially Observable Markov Decision Process (POMDP)

The algorithm used in this research follows the model of a *partially observable Markov decision process* (POMDP). A partially observable Markov decision process is similar to a Markov decision process, but also models a hidden state scenario where the learning agent needs to maintain a belief state regarding the current environment [7].

A POMDP is defined by the following aspects:

- S - set of states
- A - set of actions
- X - set of observations
- $P(s_{t+1}|s_t, a_t)$ - state transition probability
- $P(x_t|s_t)$ - observation probability
- $R(s_t, a_t)$ - reward function

- An agent is in state s_t at time t.
- By taking action a_t the agent reaches state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$.
- An unobservable state must be defined by a set of probabilites around the state of certain variables or features.

Given the above aspects, the POMDP gives us the mechanisms to represent a game such as Sheephead in which we need to maintain the known or observable state, transition probabilities, reward functions to support the learning of the environment and the transition probabilities realized by the playing of cards within a hand.

2.2.2 Feature Extraction

Feature extraction is a technique used to reduce the complexity of the state space and thus reduce the processing time and data size. A scenario such as that in a card game can consist of a very large number of potential states due to the hidden nature of cards in each player's hand. To deal with this large space, we use feature extraction to reduce the dimensions to something more manageable that can be processed in a reasonable amount of time.

Feature extraction involves taking certain important aspects of the state and describing those rather than the entire state. For example, in the case of a card game, the features might consist of where certain cards lie, how many of a type of card still have not been played, and how many total cards have been played. This technique allows us to use a small number of

feature dimensions to represent the entire state space which could consist of an exponential number of potential possibilities. The increase in importance for those features found to be valuable from our learning algorithm helps us overcome the need to represent the entire state space. In the example of cards, this would be the set of all possible scenarios of cards played. In Sheephead, because we use a 32 card deck, this would be 32!.

2.2.3 Function Approximation - NGNet / EM Algorithm

We use the technique of function approximation to improve performance and allow for working with a large number of dimensions. This allows us to map a large set of input parameters to a large set of output parameters without exponential processing. More specifically, within this research, we use feature extraction to remove layers of summation by turning such calculations into a linear function evaluation.This is especially important in our research as the state space consists of a large number of known parameters as input and a large number of learned information as output.

To handle function approximation within this research we will use Normalized Gaussian Networks (NGNet) [10] to allow for this function approximation. NGNets use linear techniques to approximate multi-dimension Gaussian distributions. This is just one way to map mutliple features or dimensions to a function that allows us to evaluate potential values very quickly and with limited memory or space. This is represented by equations 1 and 2.

$$y = \sum_{y=1}^{M} \left(\frac{G_i(x)}{\sum_{j=1}^{M} G_j(x)} \right) (W_i x + b_i) \tag{1}$$

and

$$G_i(x) \equiv (2\pi)^{-1/2} exp \left[-\frac{1}{2}(x - \mu_i)' \sum_i^{-1} (x - \mu_i) \right] \tag{2}$$

where

- M - number of units or events,
- $'$ - denotes a transpose,
- $G_i(x)$ - N-dimensional Gaussian function which has an N-dimensional center μ_i and an $(N \times N)$-dimensional covariance matrix $\sum_i$,
- W_i - $(D \times N)$-dimensional linear regression matrix, and
- b_i - D-dimensional bias vector.

We then use a multi-dimension version of the Expectation-Maximization (EM) algorithm to match the set of input dimensions to the appropriate NGNet maximizing on log-likelihood. The EM algorithm is a technique used to approximate a set of gaussian distributions based on the data tested. We first define θ as the set of model parameters, i.e. mean, μ_i, and variance $\sum_i$, associated with the above function. We then apply the EM algorithm to the above functions in two steps to find the expected value of the parameters

given the initial assumptions and then to maximize the likelihood given the model and the data.

Expectation Step:

$$P_{ij}(c_i|X_j) = \alpha P(X_j|c_i)P(c_i) = P_{ij} \tag{3}$$

$$N_i = \sum_j P_{ij} \tag{4}$$

where α =the learning rate between 0 and 1.

Maximization Step:

$$\hat{\mu_i} = \frac{\sum_j P_{ij} x_j}{N_i} \tag{5}$$

where $\hat{\mu_i}$ =the new mean.

$$\hat{\sigma_i} = \sqrt{\frac{\sum_j P_{ij} x_j^2}{N_i} - \left(\frac{\sum_j P_{ij} xj}{N_i}\right)^2} \tag{6}$$

where $\hat{\sigma_i}$ =the new standard deviation.

$$\hat{P}(c_i) = \frac{N_i}{\sum_j N_j} \tag{7}$$

where $\hat{P}(c_i)$ =the new class prior used in the expectation step.

We iterate over the EM steps above until we converge on a result. Ishii et al. use an adaptive, online algorithm which builds upon results from already

gathered data [13]. Their approach reduces the processing time because we are not running the EM algorithm over the entire data set, but rather introducing and adapting to the new data points since the last run. Ishii et al. showed the online version to be much faster than the normal EM algorithm. However, for simplicity, we run the EM algorithm over the entire set each step. This does increase computation time, but limits the scope of this research effort as the non-online version of the algorithm is a much simpler implemenation.

2.2.4 Actor-Critic Algorithm

The actor-critic algorithm provides the foundation for the reinforcement learning portion of the Ishii et al. algorithm [13]. In the actor-critic algorithm, the critic is responsible for maintaining the state of the system, and the actor selects an action based on a merit function with regards to state. The critic then updates itself based on an error evaluation for the given state transition. This temporal difference (TD) error is evaluated using the following function:

$$\delta = R(x_{t+1}) + \gamma V(x_{t+1}) - V(x_t) \tag{8}$$

where R is the reward function, V is the value function, and γ is a constant which affects the rate at which we adjust and is between 0 and 1.

The critic then updates the value function based on the following formula:

$$V(x_t) \leftarrow V(x_t) + \eta_c \delta t \tag{9}$$

where η_c is the critic learning rate and is between 0 and 1.

The actor then updates its merit function based on the following formula:

$$U(x_t, a_t) \leftarrow U(x_t, a_t) + \eta_a \delta c \tag{10}$$

where η_a is the actor learning rate and is between 0 and 1.

2.3 Algorithm Implementation

The algorithm or learning process presented by Ishii et al. [7] can be described from a high level perspective as consisting of a few modules that work together to provide the full learning scheme. The primary modules are the state evaluation model and the action control module. The action control module is then made up of an action selector module and action predictor modules for all of the other agents participating in the game. Figure 1 shows these modules as well as the hierarchical relationship amongst them. This section describes the state evaluation module, the actor-critic algorithm used in the modules and the action control module.

2.3.1 State Evaluation Module

The state evaluation module is responsible for estimating the current state of both the observable and unobservable environment around the agent. In the

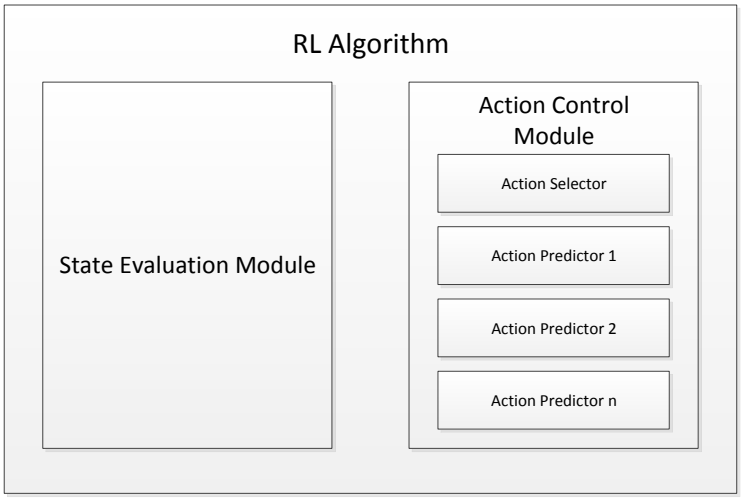

Figure 1: Basic architecture of the reinforcement learning implementation scheme reviewed in this paper.

example of Hearts, this consists of the cards in the learning agent's hand, the cards that have already been played and the cards in other agents' hands. The responsibility of the state evaluation module is to calculate the belief state over all of the possible scenarios.

Our biggest concern with this module is the ability to enumerate over all of the state possibilities. Even in the small scenario presented by Ishii et al. [7] representing the game of Hearts, the state space is so large it required them to minimize the state space for both computational and storage reasons. They accomplish this by using the technique of feature extraction. For example, instead of trying to estimate every card that is in every player's hand, you would set certain variables such as how many hearts each player has and who

has the queen of spades. The other benefit from feature extraction is certain limitations based on these features. For example, three features may be tied together with a total probability of one (i.e., the probability that each player has a particular card). This allows you to limit the possibilities of the entire feature set based on interdependencies amongst features.

On top of using feature extraction as a technique to reduce complexity and size, Ishii et al. also use the technique of function approximation or function approximators to remove several layers of summation when evaluating the state after observable actions are made. This reduces summations across the many features or possibilities. The features are then passed into a multi-dimensional function which maps to an output set of features, in this case representing the new state space estimation.

2.3.2 Action Control Module

The action control module consists of the action selector module and a set of action predictor modules for each of the opponent agents within the scenario. These modules all work to build a representation of a given agent's strategy. The action selector module is responsible for building the learning agent's strategy and setting up the output functions that control the agent's actions. The action predictor modules are responsible for predicting opponent agents' actions by modeling their strategy.

The action predictor module works much like the actor-critic algorithm presented by Barto, Sutton and Anderson [14]. It calculates a merit func-

tion for all of the possible actions and selects the action based on a specific probability function. It then updates its merit function by replaying the past game, which updates a portion of the merit function, and observing the actual action taken of the agent. The action predictor, like the state evaluation module, uses both the techniques of feature extraction and function approximation to reduce the state space of evaluating performance and updating its merit function.

The action selector module determines actions based on its merit function similar to the action predictor. However, unlike the action predictor, it updates its expected error, and likewise its merit function after every play by iterating over possibilities. It does not iterate over all possibilities though, but instead uses a pruning technique to reduce the size of the summation associated with this step. It then simply selects its actions based on a probability function. This pruning is based on a merit function being evaluated for only a couple of plays, values under a certain threshold being pruned from the evaluated possibilities.

2.4 The Game of Sheephead

Sheephead is a zero-sum, multi-player card game involving five or more players. Each player maintains his or her own score (usually in terms of money). Within each hand, partners are determined. This can consist of a"one versus four" scenario, a "two versus three" scenario, or a special scenario called "Nola" in which each player is essentially on their own.

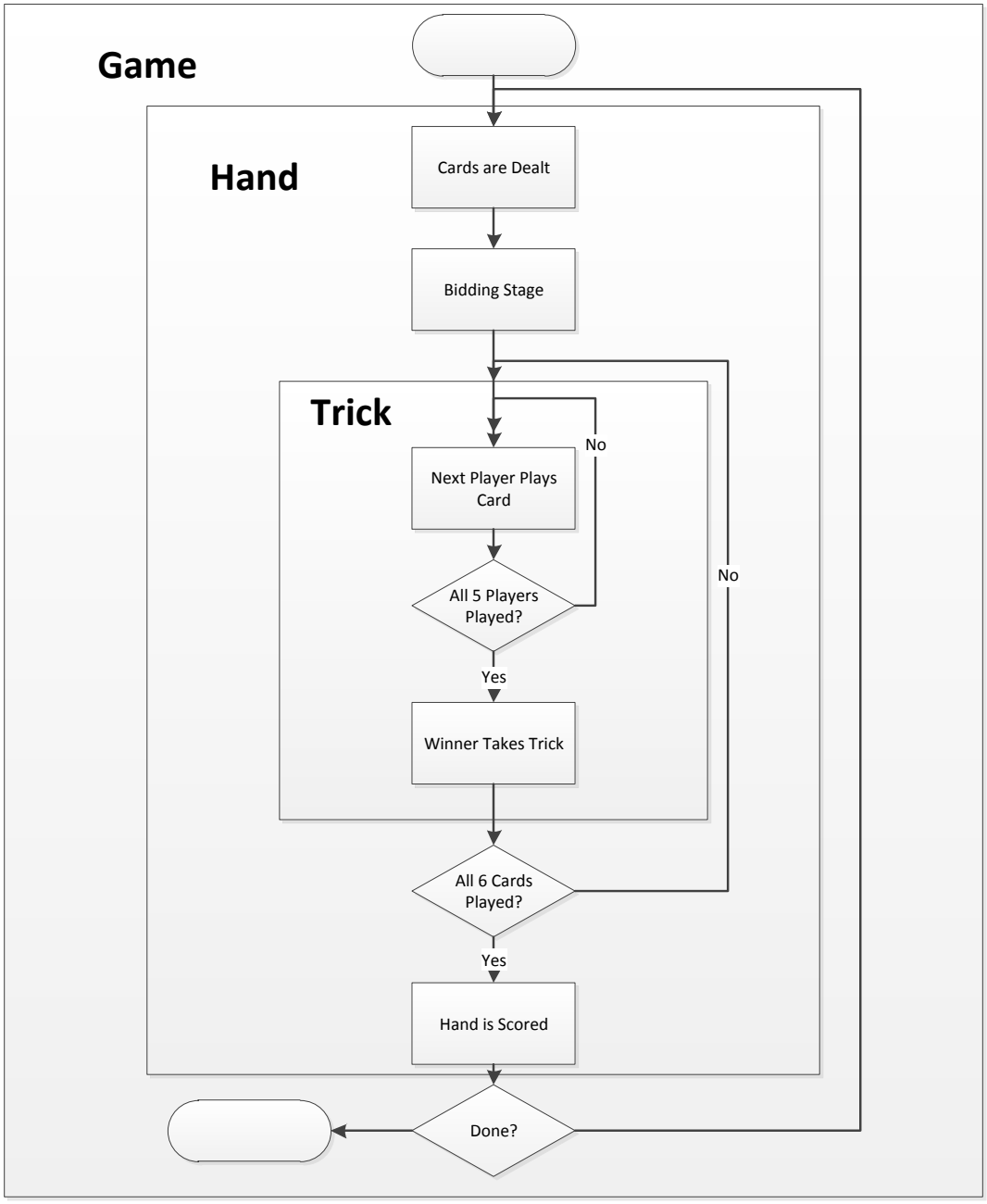

Figure 2: This flowchart shows the basic flow of a game of Sheephead. It is seperated into three levels of play: game, hand, and trick.

Figure 2 helps to clarify the difference between a game, a hand, and a trick and illustrates the transition flow of a single trick. As shown in the flowchart in Figure 2, each hand within Sheephead consists of three states, bidding, optional partner choice and playing. In the bidding state each player takes a turn, starting to the right of the dealer, choosing to pick up or not two cards that were set aside face down during the deal (the blind). Once a player picks up the cards or everyone passes (decides not to pick up the two card blind) we move on to the partner choice state. In the partner choice state, only applicable if the cards are picked up, the player picking up the blind can call a card which will require the holder to become his or her partner. Finally, in the playing state, six tricks are played where each player plays a card.

Results are scored based on a 120 point scale for each hand resulting in certain players winning and collecting money (usually in quantities such as nickels and dimes) and certain players paying money. This is determined differently based on many scenarios, but in most games it involves determining the winning team by the team that scores more than 60 points.

While the rules of Sheephead are too extensive to include here we will include a subset of the rules to show some of the complexities in this game and simulation. The full set of rules was coded in the simulation. The overall algorithm is also presented in psuedocode in Apeendix B.

- Players must follow the suit of the lead card when possible.

- Trump cards, consisting of queens, the joker, jacks and all diamonds, and are higher than all non-trump cards.
- A player must declare a partner by selecting (or calling) an ace of a fail suit (ace of clubs, ace of spades or ace of hearts) of a card they keep.
- In the scenario when the player declaring the partner only has aces of the fail suit they have, they must identify an "unknown" card and call an ace of a fail they do not have. The unknown card will then be played on the ace trick.
- If the player declaring a partner has all three aces, they then must declare a partner by selecting (or calling) a ten of a fail suit of a card they can keep. Partners are unknown then until the fail trick described above is played.
- In normal play, the order of trump during play is: queen of clubs, joker, queen of spades, queen of hearts, queen of diamonds, jack of clubs, jack of spades, jack of hearts, jack of diamonds, ace of diamonds, ten of diamonds, king of diamonds, 9 of diamond, 8 of diamonds.
- In normal play the order of all non-trump suits (clubs, spades and hearts) is ace, ten, king, 9, 8, 7.

Nola is a different type of game which is played in a hand when no players pick up the two card blind during the bidding stage. In a Nola hand:

- Kings are higher than tens.
- The joker is the seven of diamonds.
- The order of cards during a Nola hand is ace, king, queen, jack, ten, 9, 8, 7.

Two scenarios result from Nola: The winner takes all tricks and gets paid by other players or the loser takes the last trick and pays all other players.

Scoring:

- Cards are worth points during normal play as follows: ace - 11, ten - 10, king - 4, queen - 3, jack -2
- If the score of a hand ends up 60 to 60, players will pay and receive double normal payment for the next five hands.

There are a total of 120 points played with each hand resulting in scores for each team between 0 and 120. In most games (non-Nola games), a score of 61 or more wins the hand, 91 or more wins the hand with the opposing team paying twice the normal payment, 120 (or all of the tricks) wins the hand with the opposing team paying three times the normal payment.

Sheephead was chosen for this research project due to both a high level of familiarity for the author as well as several similarities to Hearts on which the Ishii et al. research was based. Those similarities include:

- Zero-sum game (every gain by a player is offset by a loss of another player or players),

- Score is maintained per individual,
- Belongs to the class of perfect recall games,
- Set of distinct rules affecting current play state,
- Distinct reward system per trick.

The research of Ishii et al. is adapted specifically for Sheephead to explore the algorithm. Sheephead provides a good base for this project as it involves fewer cards than hearts, so processing time is reduced. Any number of rules cane be included or excluded as needed by simply eliminating hands that result in those non-normal scenarios from the learning and thus the results without affecting overall play or the results of the entire history of games.

3 Implementation and Design

This section covers the methods used to implement this research. It covers both the details of the algorithm itself, including all formulas and mathematical specifics, as well as the logistics regarding the simulation and gathering of results.

3.1 Algorithm Details

We first describe the overall process flow regarding the learning agent algorithm. We then show the details of algorithm components organized based

on the state evaluation module, the action predictor module and the action selector module.

3.1.1 Transition Flow

Figure 2 shows the overall structure of a game of Sheepead. The learning algorithm can be described from a high-level perspective by the transition diagram shown in Figure 3. This diagram shows the transition flow from one play to the next of the learning agent, including the play of the opponent agents.

As Figure 3 shows, actions by both the learning agent and opponent agents (by means of the action predictor) are determined by both observations and strategy. We represent the strategy of the learning agent by the action selection module and RL technique described in Section 2.3. We represent the strategy of each opponent agent, shown in the figure by a superscript offset from the learning agent, as the action prediction module. The state s is understood by means of observations x and the creation of a belief state represented by the state evaluation module.

As seen in Figure 3, the process through one play is that the player plays based on a combination of strategy, observations and estimation of opponents' strategies. The learning agent calculates probabilities for the given state as well as what the state will be based on calculated probabilities regarding the opponenents strategy via the action prediction modules. The action selection process of the learning agent then iterates over all of the

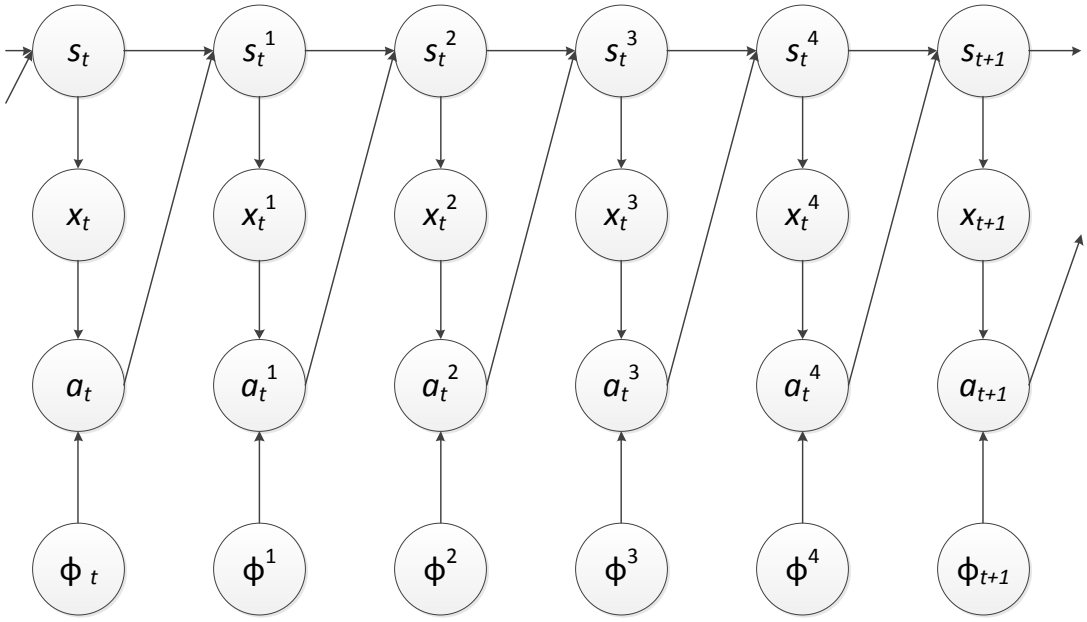

Figure 3: This diagram shows the transition process flow for the learning algorithm for one trick t as well as the learning agent at the start of the next trick $t+1$. A hand is made up of six tricks as each player starts a hand with six cards. s represents the entire state space, x represents the observation state, a represents the action played, t represents a trick consisting of each player playing a single card, the superscript numbers represent the opponent offset from the learning agent, and ϕ represents the strategy of the player.

possibilities and plays the card with a probability resulting from the function approximator learned via the RL technique.

3.1.2 State Evaluation Module

The state evaluation module is responsible for representing the known state using feature extraction and function approximation techniques. The state represenation includes features extracted from the full state space. These features are selected variables that are represented by either binary or integer values. These features are then turned into an N-dimensional function by using the EM algorithm to estimate the function based on the output vector of the NGNet. The features utilized for this research are shown in Table 1.

Table 1: State Evaluation Features

Feature	Data Type	Values
Number of trump cards played or held by agent	int	0 - 6
Number of clubs played or held by agent	int	0 - 6
Number of spades played or held by agent	int	0 - 6
Number of hearts played or held by agent	int	0 - 6
Opponent Player 1 has trump	boolean	0 or 1
Opponent Player 2 has trump	boolean	0 or 1
Opponent Player 3 has trump	boolean	0 or 1

Feature	Data Type	Values
Opponent Player 4 has trump	boolean	0 or 1
Status of queen of clubs	int - not played, agent has it, played	<-1,0,1>
Status of joker	int - not played, agent has it, played	<-1,0,1>
Status of queen of spades	int - not played, agent has it, played	<-1,0,1>
Status of queen of hearts	int - not played, agent has it, played	<-1,0,1>
Status of queen of diamonds	int - not played, agent has it, played	<-1,0,1>
Status of jack of clubs	int - not played, agent has it, played	<-1,0,1>
Status of jack of spades	int - not played, agent has it, played	<-1,0,1>
Status of jack of hearts	int - not played, agent has it, played	<-1,0,1>
Status of jack of diamonds	int - not played, agent has it, played	<-1,0,1>
Learning Agent is leading player of trick	boolean	0 or 1
Opponent Player 1 is leading player of trick	boolean	0 or 1
Opponent Player 2 is leading player of trick	boolean	0 or 1
Opponent Player 3 is leading player of trick	boolean	0 or 1

Feature	Data Type	Values
Opponent Player 4 is leading player of trick	boolean	0 or 1
Status of ace and ten of diamonds	int - ace played, agent has ace, ten played, agent has ace	0 - 8 (bitwise)
Status of ace and ten of clubs	int - ace played, agent has ace, ten played, agent has ace	0 - 8 (bitwise)
Status of ace and ten of spades	int - ace played, agent has ace, ten played, agent has ace	0 - 8 (bitwise)
Status of ace and ten of hearts	int - ace played, agent has ace, ten played, agent has ace	0 - 8 (bitwise)

3.1.3 Action Predictor Module

The action predictor is used to predict the card played by each opponent during a trick. It uses a similar method of prediction as the actor uses (i.e., the merit function) in the actor-critic algorithm described in Section 2.2.4. This is also the method that we use in the action selection module. The action prediction module predicts an action for agent M_i based on the following formula:

$$P(a_t^i|y_t^i(a_t, H_t, K), \theta^i) = \frac{exp(U^i(y_t^i(a_t, H_t, K), a_t^i)/T^i)}{\sum_{a_t^i \in A^i} exp(U^i(y_t^i(a_t, H_t, K), a_t^i)/T^i)} \tag{11}$$

where

- A^i denotes the set of possible actions (a^i for the agent at a particular time t),

- H_t represents the historical state of the current game at time t,
- K represents the overall game knowledge which is static,
- T_i is a constant that represents the randomness of the agent's actions, and
- $U^i(y_t^i(a_t, H_t, K), a_t^i)$ approximates the merit function $U(x_t, a_t)$ per [7] as described in the actor-critic algorithm (Section 2.2.4).

Like the state evaluation module, we use a feature extraction technique to reduce the input to the action predictor module. The features used as input are based on probabilities and incorporate game state into the logic. As an example, we consider the feature of a player having a card higher than those played within a particular suit. We know if the player has not followed suit in a previous play when that suit was led, that this probablity is 0, thus adjusting all the other players' probabilities as well. The learning output of the function approximator learns and approximates the probabilities of ways that a player will respond to the various playing scenarios. In the given example when a player has a higher card, the learned feature will be the probability that the player will play that particular card. The features used as input to the action predictor and as output from the action predictor are in Tables 2 and 3.

Table 2: Action Predictor Inputs

Number	Input
1	If the leading card is a trump, the expected number of trump cards held by the opponent which are weaker than the strongest card already played in the current trick, otherwise zero.
2	If the leading card is a trump, the expected number of trump cards held by the opponent, which are stronger than the strongest card already played in the current trick, otherwise zero.
3	If the leading card is a club, the expected number of clubs held by the opponent, which are weaker than the strongest card already played in the current trick, otherwise zero.
4	If the leading card is a club, the expected number of clubs held by the opponent, which are stronger than the strongest card already played in the current trick, otherwise zero.
5	If the leading card is a spade, the expected number of spades held by the opponent, which are weaker than the strongest card already played in the current trick, otherwise zero.
6	If the leading card is a spade, the expected number of spades held by the opponent, which are stronger than the strongest card already played in the current trick, otherwise zero.
7	If the leading card is a heart, the expected number of hearts held by the opponent, which are weaker than the strongest card already played in the current trick, otherwise zero.

Number	Input
8	If the leading card is a heart, the expected number of hearts held by the opponent, which are stronger than the strongest card already played in the current trick, otherwise zero.
9	The probability that the opponent has the queen of clubs.
10	The probability that the opponent has the joker.
11	The probability that the opponent has the queen of spades.
12	The probability that the opponent has the queen of hearts.
13	The probability that the opponent has the queen of diamonds.
14	The probability that the opponent has the jack of clubs.
15	The probability that the opponent has the jack of spades.
16	The probability that the opponent has the jack of hearts.
17	The probability that the opponent has the jack of diamonds.
18	The probability that the opponent has the ace of diamonds.
19	The probability that the opponent has the ace of clubs.
20	The probability that the opponent has the ace of spades.
21	The probability that the opponent has the ace of hearts.
22	The probability that the opponent has the ten of diamonds.

Number	Input
23	The probability that the opponent has the ten of clubs.
24	The probability that the opponent has the ten of spades.
25	The probability that the opponent has the ten of hearts.
26	The lead player of the current trick.

Table 3: Action Predictor Outputs

Number	Output
1	If the leading card is a trump, the merit value for the opponent playing a card which is weaker than the strongest card already played in the current trick.
2	If the leading card is a trump, the merit value for the opponent playing a card which is stronger than the strongest card already played in the current trick.
3	If the leading card is a club, the merit value for the opponent playing a card which is weaker than the strongest card already played in the current trick.
4	If the leading card is a club, the merit value for the opponent playing a card which is stronger than the strongest card already played in the current trick.
5	If the leading card is a spade, the merit value for the opponent playing a card which is weaker than the strongest card already played in the current trick.
6	If the leading card is a spade, the merit value for the opponent playing a card which is stronger than the strongest card already played in the current trick.

Number	Output
7	If the leading card is a heart, the merit value for the opponent playing a card which is weaker than the strongest card already played in the current trick.
8	If the leading card is a heart, the merit value for the opponent playing a card which is stronger than the strongest card already played in the current trick.
9	The merit value that the opponent will play the queen of clubs.
10	The merit value that the opponent will play the joker.
11	The merit value that the opponent will play the queen of spades.
12	The merit value that the opponent will play the queen of hearts.
13	The merit value that the opponent will play the queen of diamonds.
14	The merit value that the opponent will play the jack of clubs.
15	The merit value that the opponent will play the jack of spades.
16	The merit value that the opponent will play the jack of hearts.
17	The merit value that the opponent will play the jack of diamonds.
18	The merit value that the opponent will play the ace of diamonds.
19	The merit value that the opponent will play the ace of clubs.

Number	Output
20	The merit value that the opponent will play the ace of spades.
21	The merit value that the opponent will play the ace of hearts.
22	The merit value that the opponent will play the ten of diamonds.
23	The merit value that the opponent will play the ten of clubs.
24	The merit value that the opponent will play the ten of spades.
25	The merit value that the opponent will play the ten of hearts.
26	The merit value that the opponent will play the nine of diamonds.

3.1.4 Action Selection Module

The action selection module works in a similar fashion to the action predictor module. It bases its action on the merit function of the actor-critic algorithm described in Section 2.2.4. It then plays the rest of the hand based on the action predictor module, choosing the action which results in the highest merit value for all actions. In Ishii et al., they use a pruning technique which runs the merit function only on the first couple of actions and eliminates all low values up front, enumerating only those remaining to find the maximum merit value. We, however, enumerate over all possibilities in order to reduce

the development time needed for our approach. We are also able to do this because of the limited number of plays per trick [9], and no constraint put on overall test time regarding real-time play. The action selection module then simply plays the card with the highest merit value over the remainder of plays in that hand.

3.2 Simulation Architecture

In this section we describe the code implementation of the algorithm adaptation. This description includes a review of the system and code architecture showing both class and process flow diagrams. We use this section to describe the architecture and patterns of the source code and application used for implementation.

3.2.1 Application Architecture

The simulation was written as a single command-line Java application. The application architecture was kept simple so as to minimize the amount of impact the overhead could have on either performance or quality. Eclipse was used for all development and testing.

Results are written to a flat file by the Java application. A line is written for every one-hundredth hand, recording each player's score (money) in comma delimited columns. The results are graphed using Microsoft Excel to show the performance of each run.

Special attention was paid to the random number generator used to select

card plays and predictions based on probabilities. Due to the heavy usage of random number generation as well as the close proximity with regards to time of each call, we take special care to use a single seeded random number generator. After testing with a few non-platform random number generators and no significant differences, the platform built-in random number generator was used for final results.

Figure 4 shows the high-level view of all components of application architecture and the relationships between them. As shown in the figure we have a small number of components which should hopefully minimize the potential for individual components to skew the results or performance.

3.2.2 Class Architecture

In this section we describe the class architecture used within the Java application described above. We review the main classes and relationships used to support the simulation functionality. The architecture was motivated by the need for simplicity as well as the ability to quickly make modifications to the simulation during initial runs. The need for simplicity, as previously noted, will help us minimize the potential for performance or quality bugs within the code.

We show a subset of the class architecture in Figure 5. The architecture incorporates player interfaces to allow us to easily swap out a combination of players during each simulation run. A card game provides for a fairly straight-forward object mapping as shown in the figure. We map domain

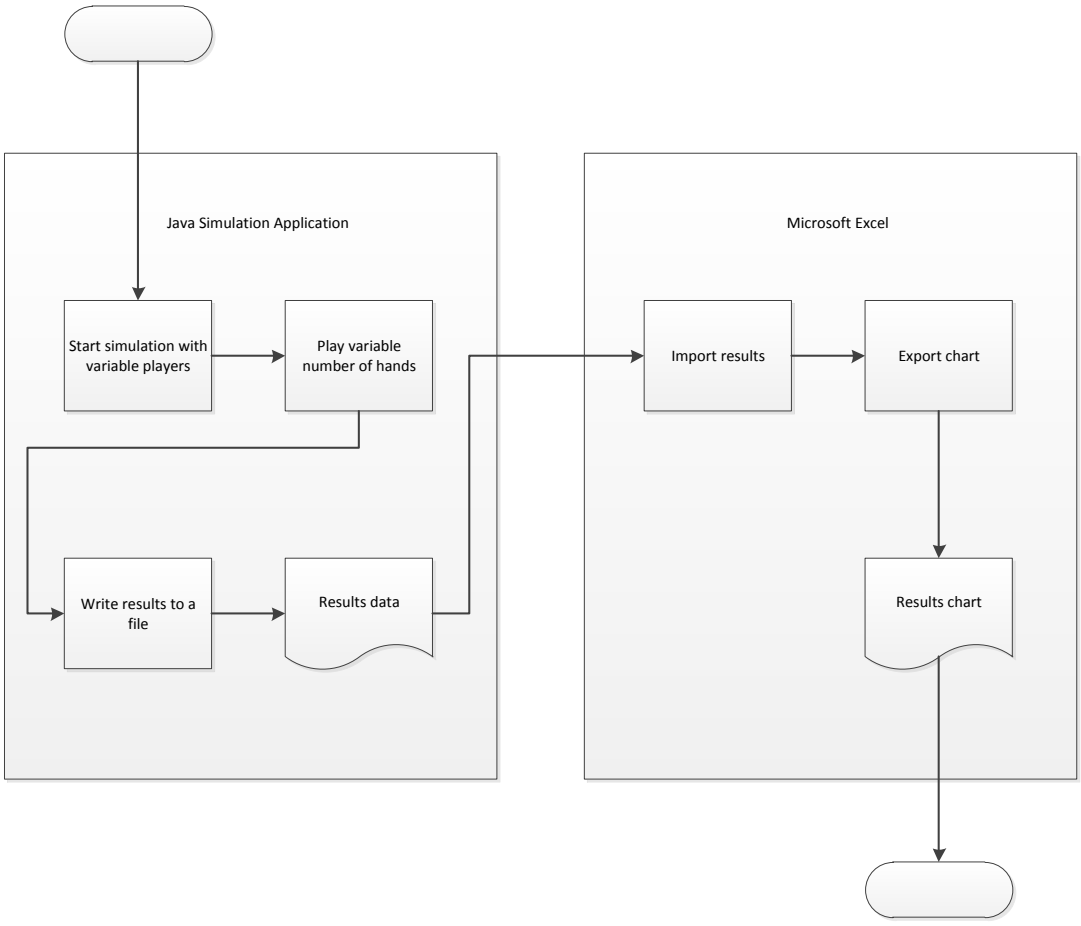

Figure 4: The basic architecture of the simulation application. The majority of processing and logic is run within the Java simulation application shown on the left. Microsoft Excel, shown on the right, is only used in gathering and presenting the results.

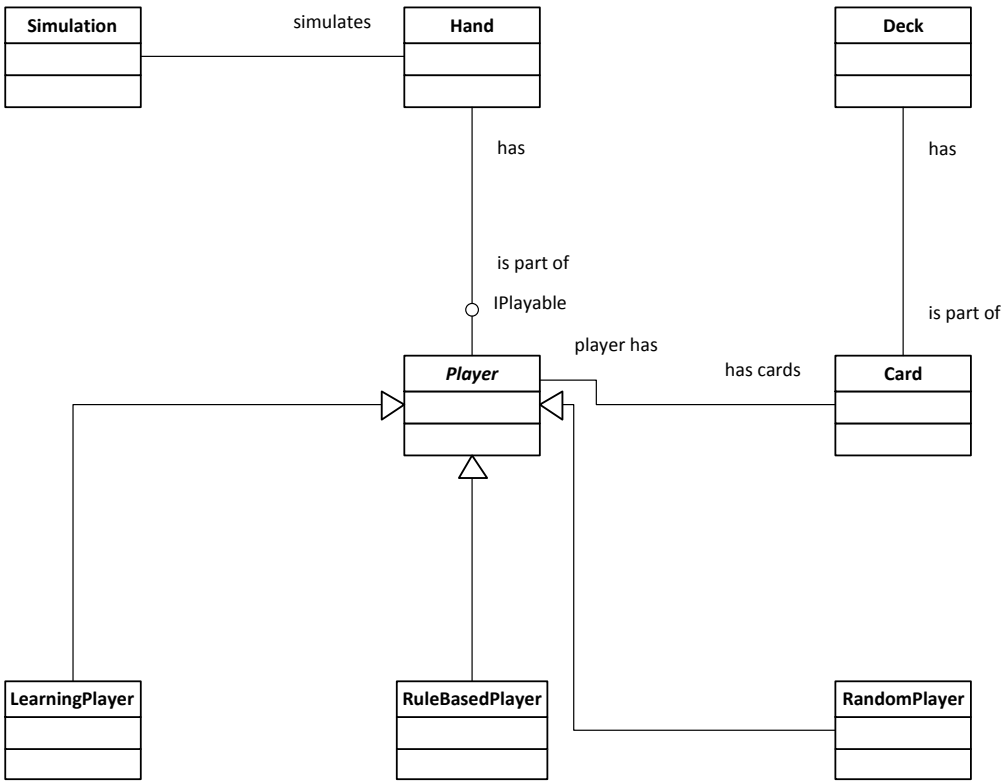

Figure 5: A subset of the code architecture used within the Java application. The simulation comprises the entire test run. A simulation is made up of many hands. Each hand has exactly five players participating. Player is a base object implementing the IPlayable interface. Each type of player is then a subset of Player.

objects such as a simulation, card, deck, player and hand directly to class objects. The relationships between each follows as such:

- A simulation is made up of zero to many hands.
- A hand is played by players.
- A player has cards during a game.
- A deck is made up of 32 cards.

4 Experimental Methodology

The simulation was run per the description of the algorithm details in the previous section. An initial simulation was run to evaluate performance improvements of a rule-based agent vs. an agent that plays cards at random. This initial evaluation was used to prove the ablity of the rule-based performance in which we will base our learning performance on. Performance was then measured by comparing the learning agents with rule-based agents.

Certain limitations were imposed on our implementation of the learning algorithm. These limitations were done for two reasons. The first, and most important reason, was to limit the scope of this project and make it manageable within the given time frame and resources. The second was to adjust the Ishii et al. algorithm to the more rule-based nature of the Sheephead game.

The largest adjustment from the original algorithm was to replace a learning-based action predictor with a rule-based one. This was possible due to the very strict rule-based play involved in what cards can and cannot be played, and what cards should be played in certain scenarios. This could be considered a "cheating" experiment because we were going against a rule-based opponent. A rule-based agent would obviously be more susceptible to losing to a rule-based action prediction.

The simulation was modeled and written using the Java programming language. The simulation itself was built to be configurable to allow for various scenarios of players and numbers of hands. Support was added for the dynamic creation and modification of the agents themselves. As part of this portion of the research, specific to this paper, three agents were coded: a random play agent, a rule-based agent and a learning agent implementing the algorithm described in this paper.

Included in the simulation are all of the game rules and complexities representing the full game of Sheephead except for the sub-game of Nola. This was not included because of the increased complexity and the difficulty expressing this as a rule-based action predictor. There are a few areas of the game where all players, including the rule-based player, act the same such as the bidding stage. This will focus the learning efforts initially to a smaller number of game situations. These additional game stages can easily be reintroduced to the learning stage at a later date.

Results from the simulation are written to a file based on scoring of each

hand. Scoring is described in detail in Appendix A.

5 Results and Discussion

Results overall show that the rule-based agents perform significantly better than the random players. The results also show that as long as there are more than 2 rule-based agents, the learning agent performed better than the rule-based agents. Each test consisted of tracking money won for all players every 100th hand of a 100,000 hand simulation.

Figure 6 shows one rule-based agent against four random players. In this test the rule-based agent performed significantly better than all random agents. Figure 7 shows thow rule-based agents against three random players. In this scenario the rule-based agent performed significantly better than the random agents like the previous two scenarios. Figures 8 and 9 show the same results, the rule-based agents peforming better than the random agents in scenarios of three rule-based agents vs. two random agents and four rule-based agents vs. one random agent.

Figure 10 shows one learning agent vs. four rule-based players. In this scenario the learning agent performed significantly better than all four rule-based players. Likewise, Figure 11 shows two learning agents perform significantly better than three rule-based players. Figure 12 however shows us a change in which only one of three learning agents performs significantly better than the rule-based players. The other two learning agents performed

about the same as one of the rule-based players. The other rule-based players performed significantly worse than the other four agents. Figure 13 shows the results with only one rule-based player. The rule-based player performs worse than all of the learning agents, but only slightly worse than the lesser of the learning agents.

When the learning agent had to compete with less than three rule-based agents, it lost the advantage of the rule-based action predictor which actually would backfire when it had three other agents competing who would also work off of action predictions based on rules. This resulted in very poor action predictions and thus very poor results.

One interesting note to the data was that position seemed to affect the players. Initial tests showed that while a rule-based player performed significantly better on each run and at least one random player performed the worst, the other players typically varied based on where they were sitting. There did not appear to be anything consistent about how the data varied amongst the other players outside of one rule-based player always performing better and one random player always performing worse. Future research will hopefully give us an explanation as to why this is happening as a limited amount of time was spent on this as part of this specific study. This led us to believe it would be important to execute runs with various seating arrangements eliminate that as something that might skew the data. This should only be done in large batches though, so as not to affect the typical play of the game which would be several hands played with the same seating

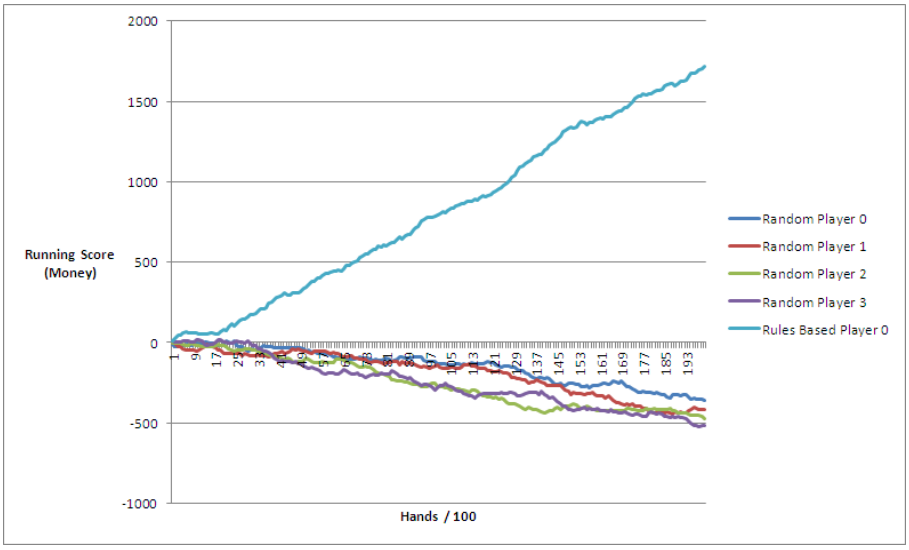

Figure 6: Results are shown for 100,000 simulations with 1 rule-based player and 4 random players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

arrangement.

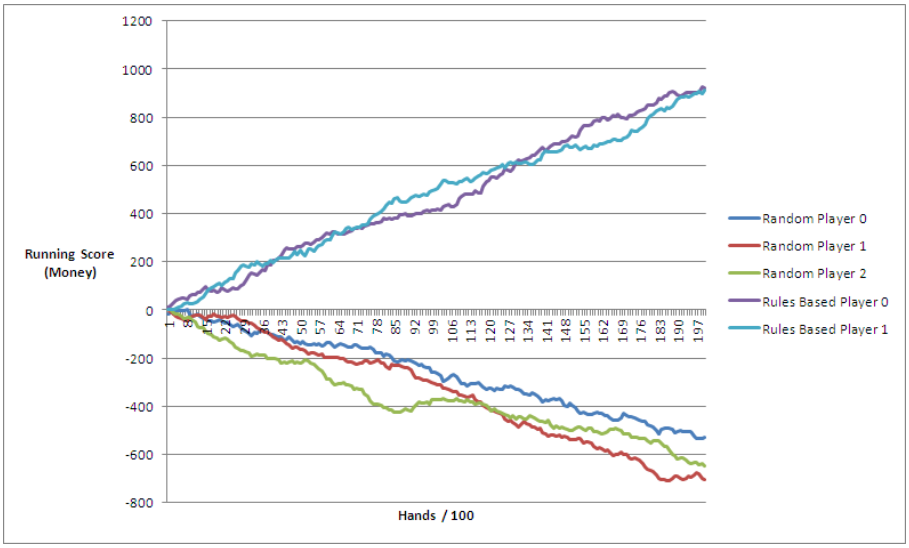

Figure 7: Results are shown for 100,000 simulations with 2 rule-based players and 3 random players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

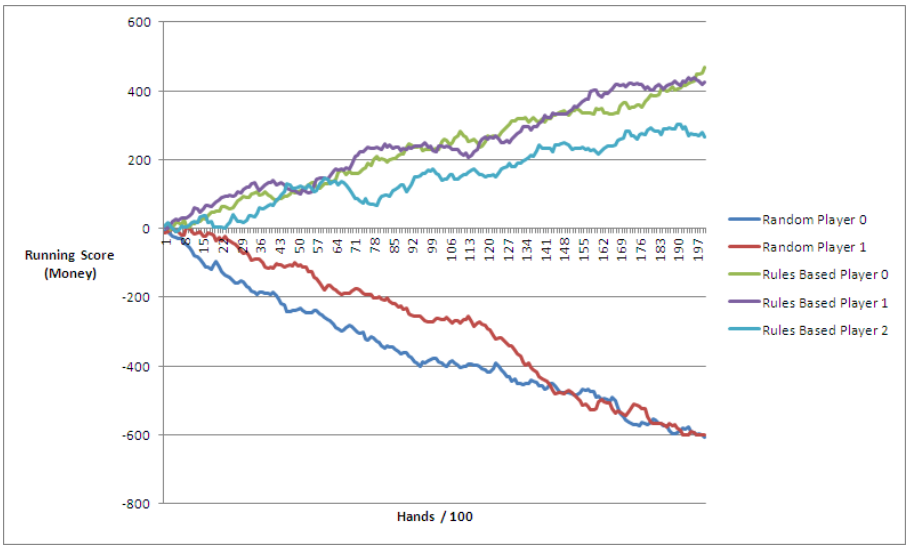

Figure 8: Results are shown for 100,000 simulations with 3 rule-based players and 2 random players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

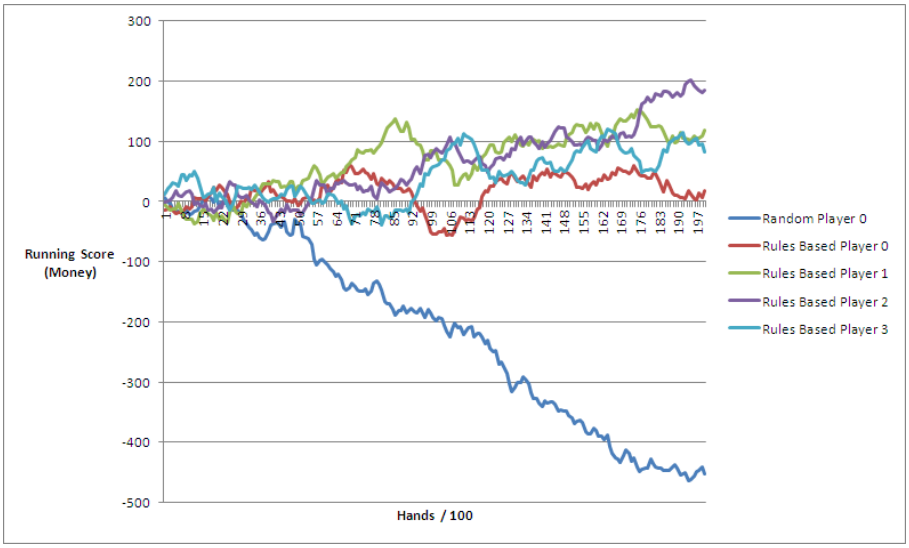

Figure 9: Results are shown for 100,000 simulations with 4 rule-based players and 1 random player. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

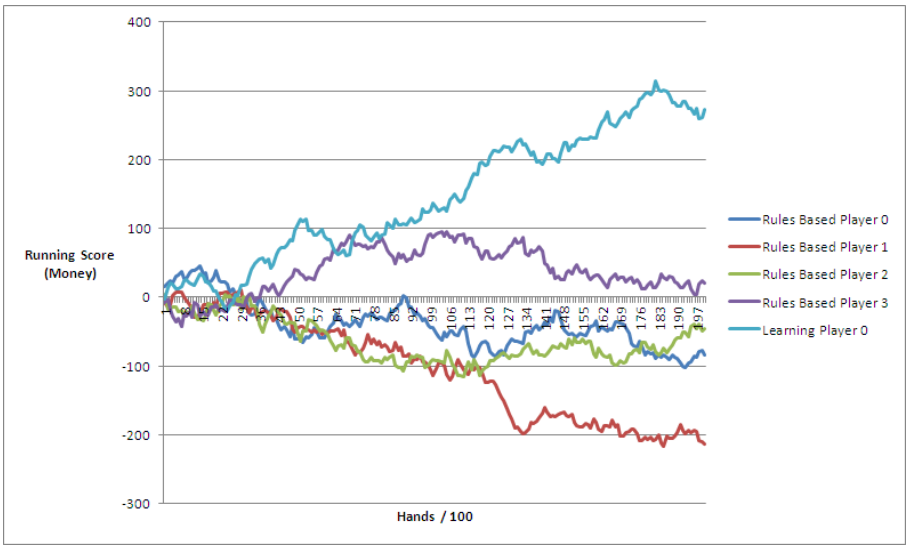

Figure 10: Results are shown for 100,000 simulations with 1 learning agent and 4 rule-based players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

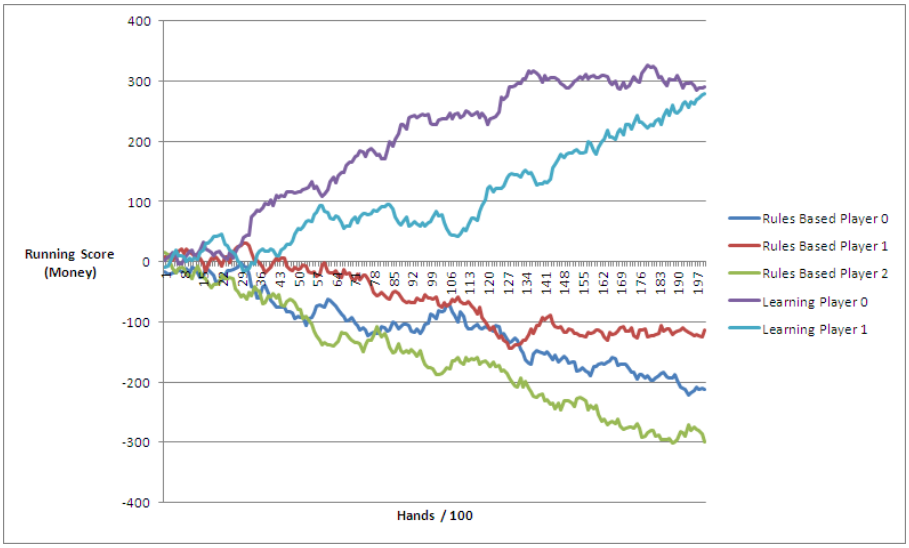

Figure 11: Results are shown for 100,000 simulations with 2 learning agents and 3 rule-based players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

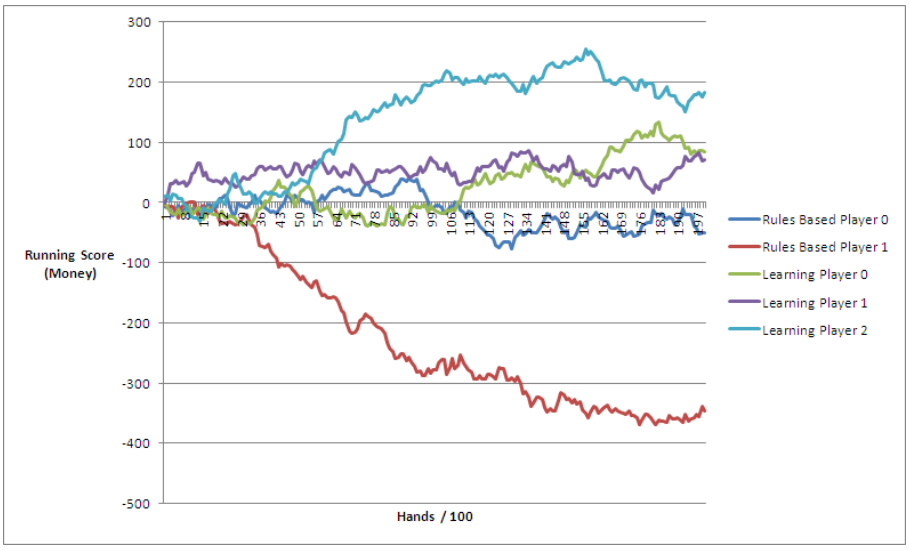

Figure 12: Results are shown for 100,000 simulations with 3 learning agents and 2 rule-based players. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

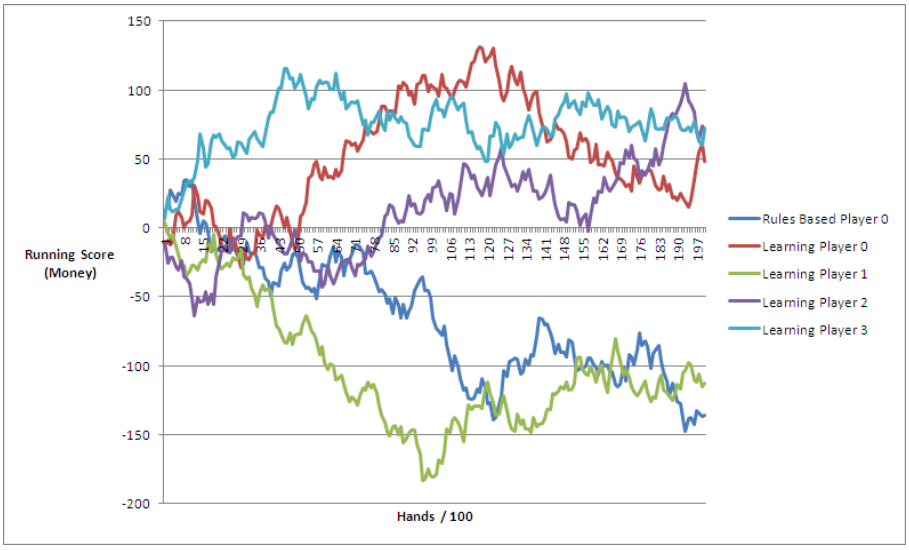

Figure 13: Results are shown for 100,000 simulations with 4 learning agents and 1 rule-based player. The y axis represents hand number divided by 100. The x axis represents the running score (money) of the player.

6 Conclusion and Future Work

Given the complex scenario of multi-agent games, machine learning within one is extremely difficult. Ishii et al. [7] presented a successful technique for tackling this problem, specifically within the card game Hearts. While the research presented in this paper is only the start of implementing the full technique for the more complex game Sheephead, it does begin to demonstrate the uniqueness of the scenario in the complexity of the game itself and the difficulty of applying machine learning techniques to imperfect information situations. By setting the benchmark of performance with an initial version of the algorithm, as well as setting up the simulation software itself, we prepare for future work implementing all modules of the learning

algorithm described in Section 2.

While multi-agent gaming adds a tremendous amount of complexity to machine learning approaches it also offers highly valuable solutions to potential real-world problems. In this work, we found ourselves getting closer and closer to approximating real world thinking and interactions amongst agents. Through the research shown in this paper, we present techniques for dealing with the problem of learning within a world that is changing and collaborating or competing with other agents who are learning alongside of us. Future work will take this further by implementation of the technique in full for the game of Sheephead and examining the impact of seating arrangements in the game.

References

[1] Mazda Ahmadi, Matthew E. Taylor, and Peter Stone. IFSA: Incremental feature-set augmentation for reinforcement learning tasks. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.

[2] Asaf Amit and Shaul Markovitch. Learning to bid in bridge. *Machine Learning*, 63:287–327, June 2006.

[3] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. *Neuronlike adaptive elements that can solve difficult learning control problems*, pages 81–93. IEEE Press, Piscataway, NJ, USA, 1990.

[4] Jean R. S. Blair, David Mutchler, and Michael van Lent. Perfect recall and pruning in games with imperfect information1. *Computational Intelligence*, 12(1):131–154, 1996.

[5] Joan Feigenbaum. Games, complexity classes, and approximation algorithms. In *Proceedings of the International Congress of Mathematicians*, volume 3, 1998.

[6] Matthew L. Ginsberg. GIB: imperfect information in a computationally challenging game. *J. Artificial Intelligence Research*, 14:303–358, June 2001.

[7] Shin Ishii, Hajime Fujita, Masaoki Mitsutake, Tatsuya Yamazaki, Jun Matsuda, and Yoichiro Matsuno. A reinforcement learning scheme for a partially-observable multi-agent game. *Machine Learning*, 59:31–54, May 2005.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, May 1998.

[9] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[10] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[11] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[12] A. L. Samuel. *Some studies in machine learning using the game of checkers*, chapter 3, pages 71–105. MIT Press, Cambridge, MA, USA, 1995.

[13] Masa-aki Sato and Shin Ishii. Reinforcement learning based on on-line em algorithm. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 1052–1058, Cambridge, MA, USA, 1999. MIT Press.

[14] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365 – 377, 2007.

[15] Nathan R. Sturtevant and Adam M. White. Feature construction for reinforcement learning in hearts. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 122–134, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[17] Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134:181–199, January 2002.

[18] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. 10.1007/BF00992698.

A Sheephead Rules

In this section we describe the full set of rules for the card game Sheephead. The game of sheephead is made up of a variable number of hands, each resulting in some of the players getting paid, and the other players paying money. Each hand consists of five players. If more than five players are sitting at the table, the dealer and however many number of players behind the dealer must sit out until five are left. Players can play as many hands as they choose as each hand is distinct and does not impact any other hands. Players' money increases or decreases with each hand played until players decide to stop.

The deck in sheephead is made up of 32 cards. This includes the following cards:

- 7, 8, 9, 10, jack, queen, king, ace of clubs
- 7, 8, 9, 10, jack, queen, king, ace of spades
- 7, 8, 9, 10, jack, queen, king, ace of hearts
- 8, 9, 10, jack, queen, king, ace of diamonds
- one joker

Play begins by the current dealer dealing the cards to each player. This includes dealing 6 cards to each player and placing two cards face down on the table. These two cards are referred to as the blind. The players then each

have an opportunity to pick up the two card blind. This decision is usually made based on how good each player's cards are. A player with good cards is more inclined to pick up the two card blind. The player to the left of the dealer is the first to be given an opportunity to pick up the cards or not. If he passed or does not pick up the cards, the player to his left is then given the opportunity to pick up the cards. This continues counter-clockwise until one player picks up the blind or all five players, including the dealer, pass. If a player picks up the two card blind, play continues using the normal rules described below. If all five players pass, play continues with *Nola* rules as described below the normal rules. After the hand is played the player to the left of the dealer becomes the next dealer and the process repeats.

A.1 Normal Play

During normal play we have already established a player has picked up the two card blind. The player first buries, or places face down, two cards in their hand. They automatically win these cards and the points associated with them. The player must then either declare a partner, or state that they will play the hand alone. If they declare a partner they will playing the hand two (the player who picked up the blind and a partner) against three (the other three players). If they play it alone it will be the player picking up the blind versus the other four players.

The player picking up the blind declares a partner (if they do not play the hand alone) by announcing a card which his or her eventual partner will

have. This is done by abiding by the following rules:

- If a player has a card of a fail suit in their hand in which they do not have the ace of that same suit in their hand, they must declare an ace of a suit in which this condition holds.
- If a player does not meet the above criteria, but does not have all three fail suit aces, they must pick a fail suit card in their hand, place it face down and declare an ace of a fail suit they do not have in their hand. This is known as calling an unknown ace.
- If a player has all three fail aces in their hand they must declare a ten of one of the fail suits which they do not hold in their hand.
- If a player does declare a card which they have in their hand, they then must play the hand alone.

Play then begins with the player to the left of the dealer leading. Each trick consists of every player playing one card. The winning player based on card order, described below, takes the trick and keeps those cards. The winning player then leads the next trick. The first card played in each trick determines the suit of the trick. Each player must follow, or play a card of that suit, if they are able to. Suits consist of trump, clubs, spades and hearts. Trump is a special suit that if played when a fail suit is played, is always higher than the fail suit played first. The suits and order of cards within each suit is listed below:

- trump - queen of clubs, joker, queen of spades, queen of hearts, queen of diamonds, jack of clubs, jack of spades, jack of hearts, jack of diamonds, ace of diamonds, ten of diamonds, king of diamonds, nine of diamonds, eight of diamonds
- clubs - ace of clubs, ten of clubs, king of clubs, nine of clubs, eight of clubs, seven of clubs
- spades - ace of spades, ten of spades, king of spades, nine of spades, eight of spades, seven of spades
- hearts - ace of hearts, ten of hearts, king of hearts, nine of hearts, eight of hearts, seven of hearts

After all six tricks have been played the hand is scored. Each team counts the points of all of the cards they have won or buried (the player who picked up the blind) based on the point values listed below:

- aces = 11
- tens = 10
- kings = 4
- queens = 3
- jacks = 2

This adds up to a total of 120 points. Money is then exchanged based on the rules below. The team with the player who picked up the cards will be referred to as team A and the other team will be referred to as team B.

If the player picking up the blind played alone:

- If the player picking up the blind took all six tricks, all other players pay that player 25 cents.
- If the player picking up the blind scored between 91 and 119 points, all other players pay that player 20 cents.
- If the player picking up the blind scored between 61 and 90 points, all other players pay that player 15 cents.
- If the player picking up the blind scored between 31 and 60 points, that player pays all other players 20 cents.
- If the player picking up the blind scored between 1 and 30 points, that player pays all other players 25 cents.
- If the player picking up the blind took zero tricks, that player pays all other players 30 cents.

If the player picking up the blind called a partner:

- If team A took all six tricks, team B pays the player who picked up the blind 30 cents and the partner 15 cents (each player of team B paying 15 cents).

- If team A scored between 91 and 119 points, team B pays the player who picked up the blind 20 cents and the partner 10 cents (each player of team B paying 10 cents)
- If team A scored between 61 and 90 points, team B pays the player who picked up the blind 10 cents and the partner 5 cents (each player of team B paying 5 cents)
- If team A scored between 31 and 60 points, the player who picked up the blind pays 20 cents and the partner pays 10 cents (each player of team B recieves 10 cents)
- If team A scored between 1 and 30 points, the player who picked up the blind pays 30 cents and the partner pays 15 cents (each player of team B recieves 15 cents)
- If team A takes no tricks, the player who picked up the blind pays 40 cents and the partner pays 20 cents (each player of team B recieves 20 cents)

If a game ends with each team scoring 60 points, payments on the next five games will be double.

A.2 Nola

Nola is played when each player passes when given the opportunity to pick up the two card blind. The rules of Nola are very different from normal play

described above. There is no longer trump and the order of cards within suits is shown below:

- clubs - ace of clubs, king of clubs, queen of clubs, jack of clubs, ten of clubs, nine of clubs, eight of clubs, seven of clubs
- spades - ace of spades, king of spades, queen of spades, jack of spades, ten of spades, nine of spades, eight of spades, seven of spades
- hearts - ace of hearts, king of hearts, queen of hearts, jack of hearts, ten of hearts, nine of hearts, eight of hearts, seven of hearts
- diamonds - ace of diamonds, king of diamonds, queen of diamonds, jack of diamonds, ten of diamonds, nine of diamonds, eight of diamonds, joker

Given these differences, Nola hands are played in the same way as normal hands.

B Simulation Algorithm

In this appendix we will show the algorithm used in the simulation. This will be shown at a fairly high level, but should help with understanding how the simulation was written and run. Figure 2 shows a high-level overview of the alogrithm shown.

SIMULATIONRUN:

```
CREATEPLAYERS;
for i = 1 to NUMBEROFHANDS do
  DEALHAND;
  PLAYHAND;
  if i mod 500 = 0 then
    for j = 1 to 5 do
      WRITEOPUTPLAYERMONEY j;
    end for
  end if
  if i mod 100 = 0 then
    REARRANGEPLAYERS;
  end if
  SETNEXTDEALER;
end for

DEALHAND:
for i = 1 to 5 do
  GIVE3CARDSTOPLAYER i;
end for
GIVE2CARDSTOBLIND;
for i = 1 to 5 do
  GIVE3CARDSTOPLAYER i;
end for
```

PLAYHAND:

BID;

for $i = 1$ to 6 **do**

  for $j = 1$ to 5 **do**

    PLAYCARD CURRENTPLAYER;

    SETNEXTPLAYER;

  end for

  SCORETRICK;

end for

SCOREHAND;

BID:

for $i = 1$ to 5 **do**

  PICKEDUP = false;

  if PICKUP i **then**

    PICKEDUP = true;

    PICKUPBLIND i;

    DECLAREPARTNER i;

    BREAK;

  end if

end for

if ! PICKEDUP **then**

ISNOLA = true;

else

ISNOLA = false;

end if

SCOREHAND;

SCORETRICK:

MAXPLAYER = null;

MAXCARD = null;

for $i = 1$ to 5 **do**

if MAXCARD == null OR CARD i ¿ MAXCARD **then**

MAXCARD = CARD i;

MAXPLAYER = PLAYER i;

end if

end for

for $i = 1$ to 5 **do**

ADDPOINTS CARDPOINTS, i, MAXPLAYER;

end for

SCOREHAND:

DOUBLEMULTIPLIER = 1;

if ISDOUBLES **then**

DOUBLESMULTIPLIER = 2;

```
end if
if ISDOUBLES then
  DOUBLESCOUNTER–;
  if DOUBLESCOUNTER == 0 then
    ISDOUBLES = false;
  end if
end if
if ISNOLA then
  for i = 1 to 5 do
    if TOOKALLTRICKS i then
      ADDMONEY i, .20 * DOUBLESMULTIPLIER
      for j = 1 to j do
        if i <> j then
          SUBTRACTMONEY i, .05 * DOUBLESMULTIPLIER
        end if
      end for
      BREAK;
    else if TOOKLASTTRICK i then
      SUBTRACTMONEY i, .20 * DOUBLESMULTIPLIER
      for j = 1 to j do
        if i <> j then
          ADDMONEY i, .05 * DOUBLESMULTIPLIER
        end if
```

```
        end for
        BREAK;
      end if
    end for
  else
    for i = 1 to 5 do
      if ISMAINPLAYER i then
        if PLAYEDALONE then
          if PLAYERPOINTS i == 120 then
            ADDMONEY i, 1 * DOUBLESMULTIPLIER
            for j = 1 to j do
              if i <> j then
                SUBTRACTMONEY i, .25 * DOUBLESMULTIPLIER
              end if
            end for
            BREAK;
          else if PLAYERPOINTS i ¿ 90 then
            ADDMONEY i, .8 * DOUBLESMULTIPLIER
            for j = 1 to j do
              if i <> j then
                SUBTRACTMONEY i, .20 * DOUBLESMULTIPLIER
              end if
            end for
```

```
    BREAK;
else if PLAYERPOINTS i ¿ 60 then
    ADDMONEY i, .6 * DOUBLESMULTIPLIER
    for j = 1 to j do
        if i <> j then
            SUBTRACTMONEY i, .15 * DOUBLESMULTIPLIER
        end if
    end for
    BREAK;
else if PLAYERPOINTS i == 30 then
    SUBTRACTMONEY i, .8 * DOUBLESMULTIPLIER
    for j = 1 to j do
        if i <> j then
            SUBTRACTMONEY i, .20 * DOUBLESMULTIPLIER
        end if
    end for
    ISDOUBLES = true;
    DOUBLESCOUNTER = 5;
    BREAK;
else if PLAYERPOINTS i ¿ 30 then
    SUBTRACTMONEY i, .8 * DOUBLESMULTIPLIER
    for j = 1 to j do
        if i <> j then
```

```
            SUBTRACTMONEY i, .20 * DOUBLESMULTIPLIER
          end if
        end for
        BREAK;
      else if PLAYERPOINTS i ¿ 0 then
        SUBTRACTMONEY i, 1 * DOUBLESMULTIPLIER
        for j = 1 to j do
          if i <> j then
            SUBTRACTMONEY i, .25 * DOUBLESMULTIPLIER
          end if
        end for
        BREAK;
      else if PLAYERPOINTS i == 60 then
        SUBTRACTMONEY i, 1.25 * DOUBLESMULTIPLIER
        for j = 1 to j do
          if i <> j then
            SUBTRACTMONEY i, .30 * DOUBLESMULTIPLIER
          end if
        end for
        BREAK;
      end if
    end if
  else
```

```
if PLAYERPOINTS i == 120 then
   ADDMONEY i, .3 * DOUBLESMULTIPLIER
   for j = 1 to j do
      if i <> j then
         if ISPARTNER j then
            ADDMONEY i, .15 * DOUBLESMULTIPLIER
         else
            SUBTRACTMONEY i, .15 * DOUBLESMULTIPLIER
         end if
      end if
   end for
   BREAK;
else if PLAYERPOINTS i ¿ 90 then
   ADDMONEY i, .2 * DOUBLESMULTIPLIER
   for j = 1 to j do
      if i <> j then
         if ISPARTNER j then
            ADDMONEY i, .10 * DOUBLESMULTIPLIER
         else
            SUBTRACTMONEY i, .10 * DOUBLESMULTIPLIER
         end if
      end if
   end for
```

```
    BREAK;
else if PLAYERPOINTS i ¿ 60 then
    ADDMONEY i, .1 * DOUBLESMULTIPLIER
    for j = 1 to j do
        if i <> j then
            if ISPARTNER j then
                ADDMONEY i, .05 * DOUBLESMULTIPLIER
            else
                SUBTRACTMONEY i, .05 * DOUBLESMULTIPLIER
            end if
        end if
    end for
    BREAK;
else if PLAYERPOINTS i == 60 then
    SUBTRACTMONEY i, .2 * DOUBLESMULTIPLIER
    for j = 1 to j do
        if i <> j then
            if ISPARTNER j then
                SUBTRACTMONEY i, .1 * DOUBLESMULTIPLIER
            else
                ADDMONEY i, .1 * DOUBLESMULTIPLIER
            end if
        end if
```

```
    end for
    ISDOUBLES = true;
    DOUBLESCOUNTER = 5;
    BREAK;
  else if PLAYERPOINTS i ¿ 30 then
    SUBTRACTMONEY i, .2 * DOUBLESMULTIPLIER
    for j = 1 to j do
      if i <> j then
        if ISPARTNER j then
          SUBTRACTMONEY i, .1 * DOUBLESMULTIPLIER
        else
          ADDMONEY i, .1 * DOUBLESMULTIPLIER
        end if
      end if
    end for
    BREAK;
  else if PLAYERPOINTS i ¿ 0 then
    SUBTRACTMONEY i, .3 * DOUBLESMULTIPLIER
    for j = 1 to j do
      if i <> j then
        if ISPARTNER j then
          SUBTRACTMONEY i, .15 * DOUBLESMULTIPLIER
        else
```

```
                ADDMONEY i, .15 * DOUBLESMULTIPLIER
              end if
            end if
          end for
          BREAK;
        else if PLAYERPOINTS i == 0 then
          SUBTRACTMONEY i, .4 * DOUBLESMULTIPLIER
          for j = 1 to j do
            if i <> j then
              if ISPARTNER j then
                SUBTRACTMONEY i, .2 * DOUBLESMULTIPLIER
              else
                ADDMONEY i, .2 * DOUBLESMULTIPLIER
              end if
            end if
          end for
          BREAK;
        end if
      end if
    end for
  end if
```