AN ARTIFICIAL INTELLIGENCE AGENT FOR TEXAS HOLD'EM POKER

PATRICK MCCURLEY – 062491790

I declare that this document represents my own work except where otherwise stated.

Signed …… 08/05/2009

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

1.1 PROBLEM DESCRIPTION

Poker is currently the world's most played card game. Hundreds of thousands of people play poker every day, and can play in a real life environment or over the internet using a distributed application running a simulation of the game.

One of the biggest reasons for poker's recent success is its fundamental dynamics. The 'hidden' elements of the game means players must observe their opponent's characteristics to be able to arrive at good decisions, given their options. A very good poker player will consistently dominate a sub-optimal opponent, although stochastic elements apply heavy statistical variation to the game, allowing weak players to win occasionally.

The game of poker offers a well-defined domain in which to investigate some fundamental issues in computing science, such as how to handle deliberate misinformation, and how to make intelligent guesses based on partial knowledge.

This project will aim to investigate what Artificial Intelligence techniques can be applied to the domain in order to play up to a human standard of decision making. Online poker clients will provide a reliable test-bed where an agent to be tested continuously and the client interface will also allow a digitalized game state recording to be read by the agent to aid decisions with no need for human intervention.

The findings of the research have application beyond the realm of poker, and can be applied to financial, weather and military domains, or more generally, any domain with a non-deterministic outcome that incorporates stochastic elements.

The investigation will be ongoing, with the aims and objectives as follows:

1.2 AIMS/OBJECTIVES

1.2.1 AIMS

1. To create an Artificial Intelligence agent capable of good decision making and playing strong poker.
2. To investigate the characteristics strong poker players possess, and compare these results with the agent solution.
3. To measure the performance of the agent against human opposition over many hands, and to document the results.
4. To design the agent to play the no-limit poker variant.
5. To implement learning capabilities so the agent can improve over time.

1.2.2 OBJECTIVES

1. To investigate the effectiveness of neural networks for opponent modeling predictions applied to this domain.
2. To evaluate what factors predominantly affect opponent modeling predictions.
3. To evaluate the effectiveness of a simulated game tree in modeling the domain to provide quantified decision making values.
4. To contrast the difference in performance between a rule-based and AI-based approach.
5. To produce a finalized agent, that produces positive results over at least 10,000 hands.

1.3 DISSERTATION OUTLINE

This dissertation has been structured in the style of a software development document. As with any software implementation, there is a clearly defined problem in which the process of finding a solution is stated and explained.

The design document contains a list of requirements, technologies used, proprietary tools incorporated and most importantly, the proposed agents architecture and lower level design components. This document is followed by the implementation document in which lower level implementation and processes are explained. The results and evaluation then demonstrate and evaluate the implemented agent's success, whilst the conclusion reflects the interesting observations of the project and lays out further work in the area.

Throughout the text, when the solution is contextually applied to the poker domain it is defined as the 'agent'. Any player that is participating in the game with the agent is referred to as 'opposition'. In later sections, any person who interacts with the solution is defined as a 'developer' and any poker player described externally from the Artificial Intelligence or Computer Science context is defined simply as 'player'.

Supplementary material is provided at the end of the document which contains poker rules, a poker glossary and the entire results of simulated pre-flop roll outs. Due to the complex nature of some of the described poker scenarios, it is highly recommended that the poker rules and glossary are read before attempting to study the main document.

1.4 ETHICS

There is a large ethical component to this project, given the nature of poker, and the way in which the artificial intelligence agent is to be tested against human opposition.

All ethical issues arise from the methodology of testing the created artificial intelligence, which is split into two headings:

PLAYING HUMAN OPPOSITION

Poker mainly consists of gambling for real money, which will have significant ethical implications on the project. Given that the agent will interface with an online poker client for performance benchmarking, the agent will be actively playing humans. Given the nature of the poker clients, it would be unrealistic to communicate with all human players, informing them that they are taking part in a research experiment to give them a choice of whether to play or not. Therefore, the agent's opposition will be uninformed of their research involvement, which opens up several ethical issues where real money is involved.

Fortunately, most online poker clients offer their customers a 'free money' option, where every feature of online poker is still accessible, but instead of money being added and deducted from a real money balance, free chips are used instead. This mitigates substantial impact on the human test subjects, and therefore reduces the ethical implications.

INTERFACING WITH EXISTING APPLICATIONS

In order to extract game state information and for the agent to make its decision based on real time environment variables, the agent's inputs and outputs must be interfaced with a poker client. There are currently no poker clients that allow a public API to be used, therefore in order to interface the agent to the client; a developer must allow an external API to be generated.

Firstly, the agent will be essentially hijacking the poker client to use for its own needs. Fortunately, there are no implications in the terms and conditions of poker clients in this area, because they employ partnerships with some 3rd party tools that use these methods such as '*PokerOffice*' or '*PokerTracker*' who in turn generate more traffic for the poker client to profit from.

Additionally, the terms and conditions of poker clients state that the use of a poker agent that automatically mimics human like mouse and keyboard input to make poker decisions is strictly banned. This imposes limitation to the poker agent, where the only solution would be to implement it as an 'advisor' rather than actually making the poker actions itself. The agent could verbally inform the developer of the action it wishes to take, in which case the developer would make the action on its behalf.

2 BACKGROUND

2.1 ARTIFICIAL INTELLIGENCE AND POKER

In the domain of Artificial Intelligence for popular games there have been many games solved to date. Examples of these agents would be IBM's 'Deep Blue' for chess, the University of Alberta's 'Chinook' for checkers and Michael Buro's 'Logistello' for Othello. (Papp, Billings, Schaeffer, & Szafron, 1998)

These agents have effectively solved the game and have beaten the best human players in the world demonstrating the power of computational processing. However, all these games have one trait in common – they are games of **perfect** information. That is, all players of the game can determine the exact state of the game at any one time. (Papp, Billings, Schaeffer, & Szafron, 1998)

In these games, the well-known technique of alpha-beta search can be used to explore deep into the game tree in order to choose actions that a worst-case opponent cannot do well against. For instance, IBM's Deep Blue evaluated over 200 million chess moves a second to decide on its action. (Wikipedia - Deep Blue)

2.1.1 PROBLEM DOMAIN REALIZATION

The following table demonstrates why Poker is an effective platform for Artificial Intelligence research.

General Application Problem	*Problem Realization in Poker*
imperfect knowledge	opponents' hands are hidden
multiple competing agents	many competing players
risk management	betting strategies and their consequences
agent modeling	identifying patterns in opponent's play and exploiting them
deception	bluffing and varying style of play
unreliable information	taking into account your opponents' deceptive plays

FIGURE 1 - PROBLEM REALIZATION TABLE
(PAPP, BILLINGS, SCHAEFFER, & SZAFRON, 1998)

Poker is a *non-deterministic* game. A player's actions within the poker domain can never guarantee the same outcome.

Poker has *stochastic* outcomes. The element of chance through the random shuffling of the cards creates uncertainty, and adds a great deal of *variance* to the results, making performance benchmarking a difficult task.

Hidden states in poker are *partially observable*. A player can win a pot uncontested when all opponents fold, meaning no private information for this opponent (for example, his betting strategy) is revealed. This makes it much more difficult to model an opponent effectively.

A game tree represents abstracted possibilities that remain in a game, in a tree like hierarchical structure, and is commonly used for AI solutions such as chess or checkers. *Imperfect Knowledge* (through concealed opponents' cards) is the main characteristic that makes common 'exhaustive-tree-searching' algorithms fail against poker's domain. For example, the alpha-beta searching algorithm implemented for chess cannot judge what the best action is when applied to poker, because it cannot possibly know *where* it is situated in the *game tree*. (Davidson, Billings, Schaeffer, & Szafron, 2002) Therefore, the closer an agent can approximate itself to lie in the game tree, the more likely it can find the correct action to take. An effective method to address this property is to implement an

algorithm to calculate *Hand Evaluation* (based upon players *hole-cards* and *community cards*) as a fundamental platform for decision making. (Papp, Billings, Schaeffer, & Szafron, 1998)

2.1.2 HAND EVALUATION ALGORITHMS

Hand Evaluation is used to quantify the value of hole-cards when board cards have been dealt.

Hand Strength

Hand Strength is one of the algorithms used to quantify an agent's hand strength, regardless of more board cards being dealt. The algorithm considers all the hands that could be better, the same, and all that can be worse at the point of calculation. The algorithm iterates through all holdings and returns a percentage as a result. (Billings, 2006)

Consider the following example where an agent's starting hand is **A♣ Q♥** and the flop is **3♦ 4♠ J♥**. Simple maths calculates that there are 47 cards remaining in the deck. Out of these 47 cards there can be 1,081 different two card combinations. Presently the agents hand is Ace high, so **AK**, any pair, two of a kind or three of a kind beats the agent's current hand (444 possible combinations). Any other **AQ** is equal to the agent's strength (9 remaining combinations) and 628 other hands our currently worse. Counting the ties as half, this returns a hand strength of 0.585. In other words, the hand evaluation algorithm calculates that in this situation there is a **58.5%** chance that the agent's hand is better than a random hand.

A drawback of this method of a hand evaluation is that it is only calculated against one opponent in a pot.

To calculate against multiple opponents, the result is raised to the power of the number of opponents. Hence, against 5 opponents with random hands, **A♣ Q♥** will only fair the best hand **6.9%** of the time ($.585^5 = .069$).

Currently the calculations of hand strength and potential assume that all two combinations are equally likely (Billings, 2006). To enhance the algorithm it should take into account *hand ranges*. That is, effective algorithms should only be counting the hands that are reasonable for players to hold, or more computationally accurate method would weight them in terms of probability. In the example before, with the agent's hand as **A♣ Q♥** where the *pre-flop pot* has been *raised*, there is a very unlikely probability that our opponents hold hand with very weak potential, such as **J♥ 4♠**. If the calculation is then modified to only calculate equity against reasonable hands, such as (**66+,AJs+,KJs+,QJs,JTs,98s,AJo+,KQo**) it can be observed that in actual fact, the **A♣ Q♥** only holds a **32%** share in the pots equity. This is a rather substantial difference to the hand evaluation of a random hand, but is more accurate and representative of the situations that can occur when playing poker.

Hand Potential

Hand Potential **(HP)** computes the probability that a hand will win when *all* board cards have been dealt.

To demonstrate the importance of this calculation, consider the following example.

The agent's starting hand is **6♦ 7♦** and the flop is dealt **5♦ A♠ 8♦**. The previously discussed the *Hand Strength* algorithm would indicate that **6♦ 7♦** is a poor hand (as it is currently only **7** high). However, when the flop is analysed it can be observed that the agent's hand has potential to improve to a straight flush (described as a straight flush draw). This means that any **♦** will improve the agent's hand to a flush, any **4** or **9** will improve to a straight, and a **4♦** or **9♦** will improve to a straight flush. With 47 cards to come (7 x **♦**, 1 x **4♦**, 1 x **9♦**, 3 x **4**, 3 x **9**) 15 cards will improve the players hand dramatically.

When comparing hand strength algorithms with hand potential, some interesting calculations can be made. For instance, when the agent holds **6♦ 7♦** and its opponent holds **A♠ K♠** on the flop of **5♦ A♥ 8♦** - even though the **A♠ K♠** has a far superior hand (Top pair, top kicker) and is ahead in the hand at the current time of the flop, the **6♦ 7♦** will win the hand **56.2%** of the time on *showdown*.

2.1.3 USING HAND EVALUATION ALGORITHMS AND OPPONENT PREDICTIONS TO DETERMINE CORE ESTIMATED VALUE

A popular method of incorporating these algorithms for use in decision making is to implement them into a simulation routine. (Davidson, Billings, Schaeffer, & Szafron, 2002)
A good example of this implementation would be an earlier version of the University of Alberta's AI agent *Poki* that simulates all possible outcomes of the hand to determine how to act. The simulation enumerates all abstracted agent and opponent decisions, weighting and predicting opponent actions at each opportunity to return a final *Estimated Value* (**$EV**) calculation. Estimated Value reflects the quantified value each potential action holds for a scenario that the agent encounters.

Using the same sample, consider the situation that *Poki* was dealt 6♦ 7♦ to the flopped board of 5♦ A♥ 8♦. Consider that the poker variant was *no-limit*, the current pot was $14, the opponent has $90 of his stack remaining, Poki holds the same stack of $90 and the opponent modelling component has made a prediction that the opponent will call 50% of the time to an all-in bet and fold the rest. The simulation could then use these properties combined with hand evaluation results to calculate resulting Estimated Value for making an all-in action as shown in the table below.

$EV when opponent calls = (total pot * hand potential)
$EV when opponent calls = ((opponent amount to call + current pot) * hand potential)
$EV when opponent calls = (($90 + $14) * 0.562)
$EV when opponent calls = **$58.45**

$EV when opponent folds = (current pot)
$EV when opponent folds = **$14**

$EV of all-in = (($EV when opponent calls * likelihood of action) + ($EV when opponent folds * likelihood of action)
$EV of all-in = ($58 * 0.50) + ($14 * 0.50)
$EV of all-in = **$36.22**

It can be noted here that the above calculation is only accurate if the opponents hand distribution is equal to that of a uniform distribution. At this stage, the hand evaluation algorithm is treating the 56% equity as the agents hand against a random hand, but in reality the agent's opponents range will be much narrower if calling an all-in.
This sort of simulation is evidently a powerful tool in which the estimated value results become more accurate as the opponent modelling accuracy increases.

2.1.4 THE NASH EQUILIBRIUM

Poker can be abstracted into two components to evaluate its dynamics – Exploitive play (adapting strategies to flaws in opponents) and Optimal Play (strong decisions regardless of opponent strategies). A perfectly optimal strategy is referred to as a Nash Equilibrium.

A Nash Equilibrium strategy states that when employed by multiple opponents "no single player can do better by changing to a different strategy" [4]. An important fact of finding and using Nash equilibrium is that "If one player implements the equilibrium strategy, since their opponent cannot do better by playing a strategy other than the equilibrium; they can expect to do no worse than tie the game" (Billings, 2006). Therefore, Nash equilibrium can be used to play defensively until enough opponent modelling data has been observed to identify errors in an opponent's strategy, and to exploit him effectively using exploitive play.

When trying to find Nash equilibria in a complex game, it is rarely achievable to arrive at the precise Equilibrium (given the complex environment of poker). Instead, Nash equilibrium is approximated with a **ε-Nash Equilibrium** strategy. It is also worth noting that the more simple variants of poker (such as limit heads-up Hold'em) find it much easier to approximate an NE strategy, whilst the more complex (such as NL, full-ring Hold'em) find it more difficult. It has been speculated that a Nash Equilibrium for full-scale poker is unlikely to be found (Billings, 2006). This aside, it does allow some interesting analysis into optimal strategies, and is an excellent foundation strong poker play, even if it is not perfect.

Simulation can be applied to improve an agent's equilibrium strategy. Unlike the previous example of simulation predict future outcomes, it can be implemented it to simulate a game of heads-up limit poker, matching one agent against an improved version and analysing the outcomes. A developer can then effectively simulate millions of games in a few minutes, surpassing the variance, and returning results that can identify if a particular change to the agent results in an added strength or weakness. However, there exists a weakness in this approach – that self-simulation can only improve very small components of the current agent, wider problems such as exploitability from random strategies can really only be effectively tested in a real-world environment against human opposition. (Davidson, Billings, Schaeffer, & Szafron, 2002)

2.2 OPPONENT MODELLING

Opponent modelling is arguably the most important component of exploitative play and appears to possess many of the characteristics of the most difficult problems in machine learning—noise, uncertainty, an unbounded number of dimensions to explore, and a need to quickly learn and generalize from relatively small number of heterogeneous training examples. Additionally, the real-time nature of poker (a few seconds per betting decision) limits the effectiveness of most popular learning algorithms. (Finnegan Southey, 2005)

There is vast amount of data available for a player to use for his opponent modelling calculations; some will be more valuable than others, so the success of an opponent modelling component is dependent on how well an agent can decipher between the relevance of the available data. Previous hand histories, current game states and generic expectations can all be exploited to the implementation of an effective agent. (Davidson, 2002)

It would be simple to implement an opponent modelling component that relied on expert knowledge (such as "If there is a tight and weak opponent that has limped pre-flop, raise him from the button with 70% of hands"), but from a scientific standpoint, it would be more effective and ultimately accurate to develop from scratch. The reason for this is that an expertly defined rule-based approach will generally contain a much too abstracted representation of the complex scenarios that can appear in a game. (Davidson, 2004)

To implement an effective agent sequentially, a strong pre-flop strategy must be implemented. Similarly, a strong opponent modelling predictor must be applied to the pre-flop strategy to maximise its accuracy and effectiveness.

2.2.1 PRE-FLOP OPPONENT MODELLING

A popular method of determining the strength of a pre-flop hand is to use pre-flop simulated off-line roll-outs of hand value. For instance, if an opponent raises from an early position pre-flop, one approach would be to assume the range of an average player. But to maximise potential value, an agent would have to be able to adapt to opponents that can be easily exploitable. For instance, in the case that an opponent raises in a mid position, this may generally be conveyed as strength. But if specific data is analysed from the opponent, an agent may predict a much weaker range than that of an average player if the opponent is known to raise weaker hands from past data. (Jonathan Schaeffer, 2000)

It is here that the difficulty lies, one opponent may have a high contextual tendency to play suited connectors to try and catch straights and flushes, whilst another may prefer pocket pairs to try and make sets. It is using the limited data that is available to use to try and make these predictions as accurate as possible, and is a difficult task when applied to the noisy environment in which an agent will exist.

2.2.2 ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) are computational models based upon biological networks and are used to model complex relationships between inputs and outputs to find patterns in data, and are known for their effectiveness in operating in domains high in noise. (Wikipedia - Artificial Neural Network)

When implementing an ANN, first it can be structurally created in which it has no knowledge of the domain it is being applied to. The network can be *trained* on the domain using a *training set*, so that if effectively learns the importance of each contextual input that it is instantiated with (such as hand strength or board texture). The most appropriate training set to use in poker would be collected hand histories as a huge amount of variation of human play should be the most effective method of training a generic network. An implemented neural networks end nodes (named output nodes) should then represent the *effects* in which the contextual information applies to. In the domain of poker, this is a straight-forward fold, call and raise, where more output nodes could be added to modify the network to no-limit, for example an *under-bet*, *over-bet* and *value-bet*. (Aaron Davidson, 2004) (Jonathan Schaeffer, 2000)

Through action prediction, neural networks can be used to highlight exploits in an opponent's strategy. In real poker it is very common for players to play sub-optimally. A player who fails to exploit these weaknesses will not succeed in comparison to a player who does. Thus, a maximizing agent will out-perform an optimal agent against sub-optimal players, as a maximising agent will extract more expected value from the sub-optimal strategies of its opponents. (Billings, 2006)

Further modification to the variables within the network can be tuned to more accurate results by applying the trained network to more sample data with a definitive result. If a result is not what was expected from the ANN, then the nodes will be tuned to produce more accurate results in further test hands.

Fully formed and trained neutral networks have been known to produce a success rate of predictions up to 81% on independent data, making it an extremely viable solution to opponent modelling. Although the stability of predictions can be skewed against opponents who have erratic tendencies, but is reflected within the domain with human and artificial players alike. (Davidson, 1999)

2.2.3 BAYESIAN APPROACH

Other effective opponent modelling techniques include a Bayesian approach. The Bayesian approach aims to determine a player's strategy given previous observations, followed by a best response to that insight.

The 'belief factor' of an opponent's potential hole-cards (also called a behaviour strategy) are denoted by $\mathbf{P(}\boldsymbol{H}|\boldsymbol{B}\mathbf{)}$ where $\boldsymbol{H}$ is the hand in question, and $\boldsymbol{B}$ is the current *information set,* which contains the state of the table, including the opponents actions up until now. (Finnegan Southey, 2005)

To extend this calculation further, an agent can calculate a *Posterior Distribution over Opponent Strategies*. This is denoted by $\mathbf{P(}\boldsymbol{B}|\boldsymbol{O}\mathbf{)}$ where O is a set of Observations ($O_s U O_f$) (O_s being the observations of hands that led to showdowns, O_f being the observations of hands that led to folds) and B is the posterior distribution over the space of the opponent strategies.
This calculated posterior distribution can then be used to produce the *Bayesian Best Response*, calculated by creating an *Expectimax* tree of all potential observations with the bottom of the tree containing an enumeration of the potential cards that could be held. The nodes now contain

the expected value of each scenario, in which an agent can choose the biggest node to produce the best response to the observations. (Finnegan Southey, 2005)

This style of decision making is very robust and versatile given the nature of Bayes' rule, and is natural to a human style of opponent modelling decision making.

2.2.4 PARTICLE FILTERING

Finally, another effective method of opponent modelling has been documented as 'Particle Filtering', which is a type of *State Estimation.* State Estimation involves tracking a stochastic process's hidden state variables by observing noisy functions of these variables. (Nolan Bard)

Dynamic Agent Modelling using *State Estimation* involves a Bayesian style approach, much similar to the example before, where observations trees are updated after more actions are observed to produce new beliefs.

Particle Filters can then be applied to the Bayesian calculations, which approximate the probability distribution over the state using a set of samples called *particles*. These particles allow noise to be reduced within the opponent modelling environment, allowing a Monte Carlo and parameterised approach to calculating a best response of an approximated opponent strategy.

The advantage that *Particle Filtering* has over other techniques is that it allows a straight forward computation of the observation model, meaning folded hands can be used as training sample data used in decision making. It also takes into account opponents randomising their strategies, making the decision components less exploitable. This technique allows the successful exploitation of both static and dynamic opponent strategies, although poses a difficult problem when applied to a full game tree as large as Texas Hold'em. (Nolan Bard)

2.3 STRATEGY IMPLEMENTATION AND PERFORMANCE MEASUREMENT

To assess and develop a poker strategy agent effectively, performance measurement is a necessity. An intuitive performance measure of poker success is the monetary value, or stack that an agent chooses to play with when it elects to play the game. However, the stochastic and varied nature of poker implies that this performance value is usually distorted by the 'chance' variables in place, and can make performance measurement of a strategy a particularly difficult task.

It is stated that usually an agent's performance is a component of nearly all research of sequential decision making and the degree of the sample is dependent on the stochasticity of the agent and corresponding environments. A perfect agent performance calculator would encompass **all** the variables of the game, including the varied results, and their repercussions. (Michael Bowling, 2006)

The effects of variance are still present in a game of poker even after 40,000 hands. Take for example a match where one agent has performed particularly well against the other, and is 2500 small blinds ahead of the other player. One agent employs an 'always-call' strategy, whilst the other has employed an 'always raise' strategy (to heighten variance). As neither strategy is dominant than the other, the *true* outcome should be completely break-even. However, with the effects of variance applied, this is rarely the case. (Kan, 2007)

2.3.1 DIVAT TOOL

A proposed solution to this problem is an application that is implemented to remove the chance and luck outcomes, revealing the *true* winner of the game. A tool such as this has been created at the University of Alberta, and has been named DIVAT. It is applicable to the limit poker variant and is optimized to heads up poker. (Michael Bowling, 2006)

The tool is comprised of several modules, working together to result in a non-bias outcome.

One module operates by reducing the variance of the importance sampling, by adding synthetic data that is consistent with the sample data. For example, in the scenario where one agent makes a mathematical error by drawing to a flush, but draws to one of its flush cards to win the pot, the DIVAT analysis tool will un-bias the game by allowing the other player to have an equal mathematical edge when in the same scenario. The tool will punish the agent who plays badly but wins due to luck, and will reward the agent who loses but makes better decisions.

Another technique that the DIVAT tool employs is to compare the sample data against a baseline policy which reflects a straight forward strategy. For this strategy to be generated, an approximation of a NASH equilibrium strategy is generated for every scenario. When an agent derives from this optimal decision making, it is punished – thus effectively punishing an agent who plays erratically but is stochastically lucky.

The last module in the tool is applied to the strategies indirectly, and operates by 'capturing' the strategies. A strategy could be captured by employing a rule based predication algorithm to extract key properties of the game in the form of statistics for each play in the game. The tool will then apply the two player strategies to a game of extremely low statistical variance. No 'chance' cards will fall, and the players will have to adopt a very straightforward strategy in order to win the game, thus punishing a lucky player. (Michael Bowling, 2006)

When these modules are combined, a tool is created that can effectively determine a strategies performance measurement over another, by reducing the non-deterministic aspects of the game.

2.3.2 LIMITATIONS OF DIVAT

One of the limitations of DIVAT analysis is that it can unjustifiably punish a player in an unlikely circumstance that could arise. In the example of both agents have a flush on the river, one having the very best hand (Ace high flush) while the other having the second best hand (King high flush) but raising, the losing player will be punished by DIVAT analysis, as he is raising into the best hand and cannot possibly win. The problem with this approach is that possessing the second best hand in poker is usually a hand the player would wish to raise with, even in the event that the opposition is beating the current hand [2]. Therefore, losing to the best hand when a player has an extremely strong holding is variance in itself, but is not addressed with DIVAT analysis. One solution to this problem, introduced by the 'UoA' University is LFAT analysis (Luck Filtering Analysis Tool) which complements DIVAT's short-comings by calculating mathematical algorithms in turn with the player's decisions. (Kan, 2007)

When a comparison is made between a normal game and a game that has been modified through DIVAT analysis in a graph form, it can be observed that the *actual* result is distorted and one agent has measured a lot better than another, whilst DIVAT analysis now portrays the *real* result, where both agents have broken even. It is shown that DIVAT is not perfectly unbiased – but extremely effective at analyzing small-samples of heads up poker. (Michael Bowling, 2006)

2.4 DATA ANALYSIS

In the implementation of an AI poker agent, the agent can potentially implement several advantages over any human opponent. In the domain of internet poker, computer autonomy can be applied for accurate mathematical calculations to aide decision making.

2.4.1 DATA MINING

In this domain, poker clients offer their users the ability to observe several poker tables at once. By using a 'Data Mining' approach, a potential observer can use external software to record results and statistics of several (up to 16) tables at once, without playing on the tables themselves.

This opens up strong possibilities for data analysis; a user wishing to record and analyze current opponent data can do so very swiftly, often logging several million poker hands per month. (Sakai, 2005)

Texas Hold'em poker is an extensive game in the realm of internet poker. Several million users play poker every day offering a huge selection of varied opponents from which to model data.

The differences involved in sub-domains of poker can be correlated to statistical information gathered using a data mining approach. This proves extremely useful to for analyzing both sub-domains (such as stake amounts) and opponent modeling data before an agent begins to play poker.

It is generally assumed that lower stake poker games contain less skilled players than higher stake games, which can be confirmed by data analysis over a large sample of hand recordings, as seen in Harayoshi Sakai's thesis, 'Internet Poker: A Data Mining Approach'.

2.4.2 IMPORTANT STATISTICS

The paper lists the three most important correlating statistics which correspond to the limit (or strength) of a poker table as being '% of players to the flop', '% of flops seen' and 'average bet-sizing'.

For the '% of players to the flop' it can be assumed that there is a larger percentage of players to the flop in a smaller staked game than that of a higher one. Since smaller-staked games usually consist of weaker players who do not apply the dynamics or mathematical foundation of the game, players tend to play much more passively and will often call bets or raises with hands that do not hold a positive expected value. In a higher staked game players tend to have higher skill, and recognize that the loose-passive strategy style is an overall losing strategy.

This assumption is proven by data analysis statistics (over 100,000 hands) which dictate that in a small staked game around 40% of players will reach the flop, whilst in higher staked games around 20% of players will reach the flop. (Ulf Johansson, 2006)

The percentage of flops seen can also be used as an indication of strength of the table. Generally in smaller-staked games, weak players will raise less pre-flop, and frequently call. As the stakes are raised, the strength of the players increase, and the game becomes more aggressive with more raising and less calling being observed. Playing strong hands by raising and betting is a proven style of winning play, thus it can be derived that a player or table who shows statistical information showing a high % of flops is generally weak, and will be found in lower-staked games.

This assumption is confirmed through data analysis showing that in a small staked game, the average % of flops seen is ~40%, whilst a higher staked game is ~20%. (Sakai, 2005)

Finally, average bet sizing (only applicable to the no-limit poker variant) can be used to indicate a player's strength. Stronger players will have a tendency to *value bet* their hands (bet around 70% the size of the pot) because it prevents drawing hands from possessing mathematical odds to continue, whilst weaker players will play much more erratically by under-betting (~20% of the pot) or over-betting (> 90%) the pot. It is true that a player must mix styles to introduce deception to a strategy, but it is generally a mathematical error of judgment to under or over-bet in comparison to the pot-size. (Sakai, 2005)

An agent could use this statistical information to place assumptions on an opponent's style of play and employ a counter exploitive strategy, before ever confronting them in a hand.

3 DESIGN

3.1 APPROACH

In order to demonstrate how varied approaches and implementations of current Artificial Intelligence techniques can affect the performance of an AI agent, the project is split into two '*Phases'* of implementation.
The design of the solution reflects the iterative nature of the project and the increments of AI computation for both solutions.

Based upon the background research of the domain, the design of the agent is based on using neural networks to predict the hidden elements of the game, particularly opponent strategy tendencies. The predictions are then applied to a constructed game tree simulation which implements the *Expectimax* algorithm to determine its decision making. Neural Networks were chosen for their documented effectiveness in operating in domains high in noise and statistical variance. The game tree simulation is a commonly documented game theory solution, suitable for analyzing multiple outcomes and is highly applicable to poker.

A decision was made to apply the solution to the heads-up no-limit variant of poker. *Heads-up* poker restricts the number of opponents to one, as the *no-limit* property enables a non-capped betting amount at any stage of the game. The no-limit format was chosen to extend on its current limited research, and *heads-up* was chosen to simplify the game tree implementation.

To ensure *scalability* and *code re-use* throughout the projects, the solution is abstracted into components which interact with each-other using the *Observer* design pattern. The *Observer* design pattern possesses advantages such as scalability and reliability across concurrent implementations, and is suitable for a large-scale project such as this. Additionally, the *Observer* design pattern is highly suited to .NET solutions through the *event* data structures available within the .NET package. (MSDN)

PHASE ONE

'Phase One' depicts a very simple AI agent incorporating expert knowledge in the form of user-defined rules. The rules conform to a very straightforward, tight aggressive pre-flop strategy, as defined in Ed Millers poker strategy book, "Getting Started in Hold'em" and uses rules incorporating hand evaluation algorithms for a post-flop strategy. (Miller, 2005)

PHASE TWO

'Phase Two' removes the rules-based component, and implements several commonly used AI techniques for non-deterministic stochastic domains such as Poker. A *Game Tree* component constructs an abstracted representation of each possible scenario from the current state of the hand to showdown in which equity and estimated value calculations can indicate the correct actions that the agent should make based on the *Expectimax* algorithm.

An *Opponent Modeling* component is also implemented in Phase Two which employs *Artificial Neural Networks* to predict opponent actions based on previous hand data. The opponent models are applied to the game tree in order to increase the accuracy of the computed leaf nodes. The Opponent Modelling components can also be used to predict and narrow a weighted distribution of an opponent's possible hole cards resulting in more precise calculations throughout the model.

3.2 REQUIREMENTS

Based on the aims and objectives of the project, the requirements of the system are as follows:

I. The solution will be implemented in two phases - one based on expert knowledge, the other on documented Artificial Intelligence techniques.

II. The AI Agent will interface with a desktop online poker application.

III. The solution will implement tools for player database clustering.

IV. The solution will implement tools for the creation and training of neural networks as both action and hand distribution prediction components.

V. The solution will implement tools for hand evaluation algorithms to determine the value of hands under varying circumstances.

VI. The AI agent must communicate with a database adapter for hand history retrieval.

VII. A Game Tree must be implemented to allow simulation of future hand possibilities in order to aid decision making.

VIII. Functionality to identify a new opponent with a stored data cluster will be required.

IX. Both offline general opponent models and online specific opponent models in the form of neural networks will be required to allow the agent to adapt to changing strategies.

3.3 TECHNOLOGIES

The solution has been implemented in the newest programming technologies available to fulfill the robustness, performance and parallel computing requirements of the project.

C#.net 3.5 Framework base language
C# is a multi-paradigm programming language developed by Microsoft as part of the .NET initiative which places an emphasis on durability and programmer productivity. It interfaces with Microsoft's SQL Server product to allow fast data retrieval and manipulation. (Wikipedia - C Sharp)

LINQ.NET Query Language
Language Integrated Query is a .NET component that adds native data querying capabilities to .NET languages. It offers object-based mapping to SQL, XML or Collections to allow advanced data manipulation. (MSDN - Linq.net)

WPF Graphical Subsystem
Windows Presentation Foundation provides rich user interface development for 2D and 3D rendering, vectors and animation to build upon the usual .NET *Winforms* UI platform.

SQL Server 2008
Microsoft SQL Server is a relational model database server produced by Microsoft which offers fast, concurrent transactions and Visual Studio integration.

3.4 RESOURCES

ONLINE POKER CLIENT

There are numerous online poker clients available for the solution. For the scope of this thesis, the selected client will be *PartyPoker* ©, because of its relative ease of interfacing through *Dll Injection* hooking components.

FIGURE 2 – PARTYPOKER© CLIENT RUNNING ON A DESKTOP

COMPUTATIONAL POWER

The computational power available to the project is limited to the desktop computer on which the solution is developed. As demonstrated in the design in later pages, the platform will need to be extremely computationally efficient to provide effective calculations.

The system's components are as follows:

- CPU – **Intel Q6600 Core 2 Quad 2.40Ghz**
- GPU – **NVidia Geforce 8800GT 512MB PCI-E**

The CPU is generally quite powerful and the quad core architecture of the hardware allows efficient concurrent computation.

The GPU can also offer further computational ability. A new form of programming (coined GPGPU - General-Purpose Computation Using Graphics Hardware has recently arisen (GPGPU Homepage), and the NVidia card described above offers CUBA (NVidia's API to GPGPU programming) allowing fast, but memory-limited computation. (CUDA.NET 2.0 Homepage)

SAMPLE HAND DATA

In order for the implemented agent to reach skill above that of average human ability it will require maximum exposure to sample poker hands for analysis and learning. The Hand History Exchange (hosted on pokerai.org) offers over 50 million recorded hands of around 1 million online poker players of varying skill and strategy.

```
#Game No : 7706905183
***** Hand History for Game 7706905183 *****
$50 USD NL Texas Hold'em - Thursday, January 15, 08:19:41 ET 2009
Table Jackpot #1314207 (Real Money)
Seat 4 is the button
Total number of players : 6
Seat 5: BellHat ( $49.50 USD )
Seat 1: Juvald ( $65.54 USD )
Seat 3: MrDarcy777 ( $51.07 USD )
Seat 4: adoxxl ( $3.87 USD )
Seat 2: jin2811 ( $10.76 USD )
Seat 6: sx619 ( $13.44 USD )
BellHat posts small blind [$0.25 USD].
sx619 posts big blind [$0.50 USD].
** Dealing down cards **
Dealt to BellHat [  Js 4s ]
Juvald folds
jin2811 folds
MrDarcy777 raises [$1.50 USD]
adoxxl is all-In  [$3.87 USD]
BellHat folds
sx619 folds
MrDarcy777 calls [$2.37 USD]
** Dealing Flop ** [ 2h, Ac, 7c ]
** Dealing Turn ** [ Ts ]
** Dealing River ** [ 7h ]
MrDarcy777 shows [ 5c, Kc ]a pair of Sevens with King kicker.
adoxxl doesn't show [ 8s, 9c ]a pair of Sevens.
The time at which hand ended:Jan 15 2009 08:20 ET
MrDarcy777 wins $7.57 USD from the main pot with a pair of Sevens with King kicker.
```

FIGURE 3 – HAND HISTORY FROM PARTYPOKER

Figure 3 demonstrates a hand history for one hand in *PartyPoker*. These histories are stored in both a human and computer readable format when partaking in poker, and are regularly shared online for opponent modeling purposes in collaboration with third party products such as *PokerTracker*, as described below.

PROPRIETARY PRODUCTS/LIBRARIES

TextCaptureX

TextCaptureX is a COM library that allows screen text capture in Windows application. It is not a font-based OCR, but instead uses its own internal C++ hooking methods to extract text from other processes, given some X/Y co-ordinates. *TextCaptureX* is currently compatible with *PartyPoker* allowing the game state to be recorded. (TextCaptureX homepage)

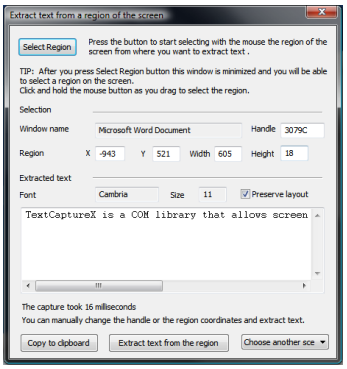

FIGURE 4 – A GUI REPRESENTATION OF TEXTCAPTUREX'S FUNCTIONALITY

This library is beneficial to this solution, as hooking Windows Text API methods from external processes is non-trivial, and outside of the scope of this project.

PokerTracker3

PokerTracker is the most popular poker tracking and analysis software currently available. It imports hand histories from poker clients in real time and clusters the data allowing opponent modeling, self analysis and extremely fast access to hand data. (PokerTracker 3 Homepage)

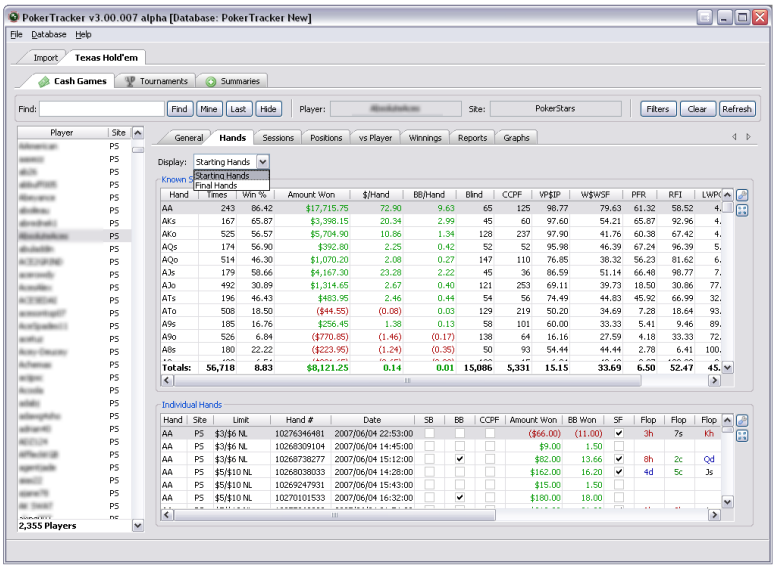

FIGURE 5 – POKERTRACKER GUI INTERFACE

PokerTracker is useful to this solution, as it clusters the data in a well structured style and allows access to its internal database. The self-analysis functions of the product also allow improvements or problems to be recognized.

PokerEval C to C# Port

For complex poker hand comparisons (such as *Hand Strength* and *Potential*), a poker hand evaluation library will be required. One of the fastest libraries currently available is the *pokereval* library, written in C.

The C# port of this library is slightly slower, but still offers around 16 million hands evaluations per second (PokerEval C# Port). The reason that this solution will not implement the original poker-eval C library (using interoperability calls) is that it needs to be heavily customized to the needs of this project (including optimization through exploiting the multi-core architecture of the test system). The C# library enables code to be altered to interface to this solution.

3.5 ARCHITECTURE

The architecture of the system is described in phases. Two phases of implementation will take place, starting with an expert knowledge based solution, followed by a solution featuring common AI algorithms for stochastic domains. This is to demonstrate how different modules affect the performance of the agent and the level of computation incorporated in each solution.

3.5.1 PHASE ONE

'Phase One' demonstrates the functionality of the most basic AI agent for poker. The table state information is passed from the *Scraping Manager* to the *Rules Manager* where the state is compared against a set of expertly-defined rules, in order to come to the action decision for the current situation.

Selective rules can incorporate evaluation results based on the *Hand Strength*, *Hand Potential* and *Effective Hand Strength* algorithms which the *Hand Evaluator* component supplies on demand given a table state.

3.5.1.1 PHASE ONE - HIGH LEVEL ARCHITECTURE

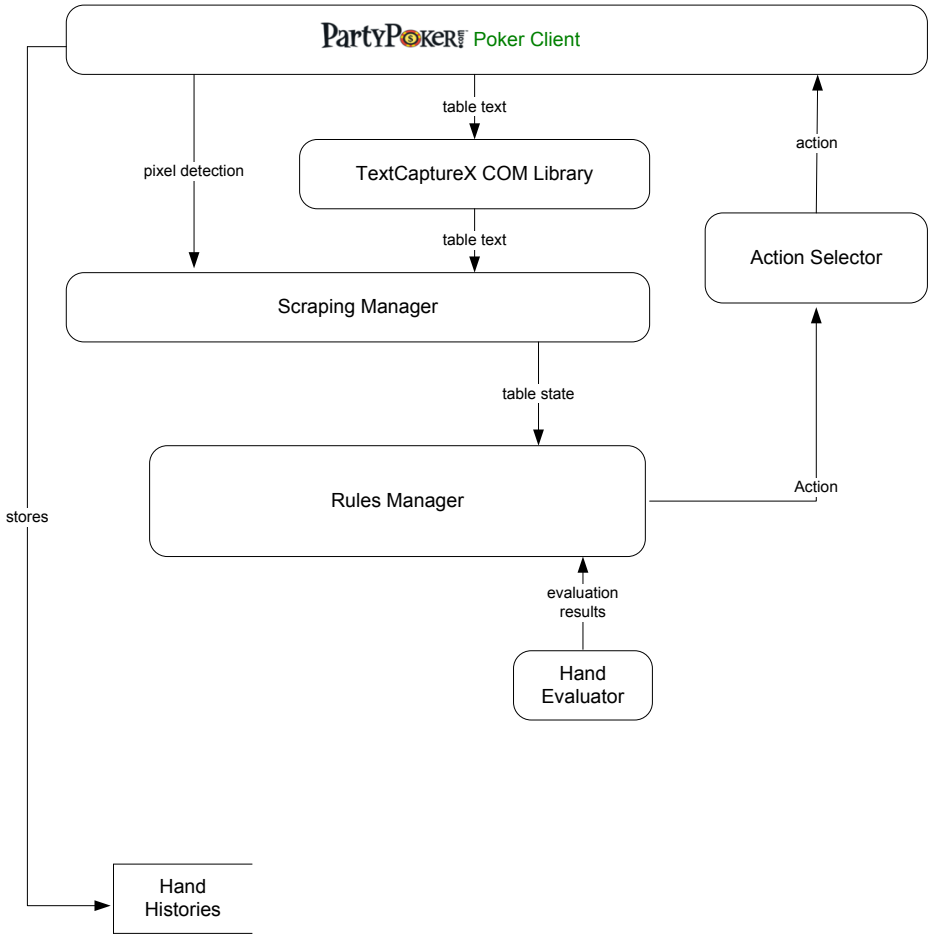

FIGURE 6 – 'PHASE ONE' HIGH LEVEL ARCHITECTURE

3.5.1.2 PHASE ONE DESIGN COMPONENTS – SCRAPING MANAGER

The *Scraping Manager* is the key to providing a representative *'Table'* object (Stored in a *'Client'*) that can be consumed and manipulated in other processes.

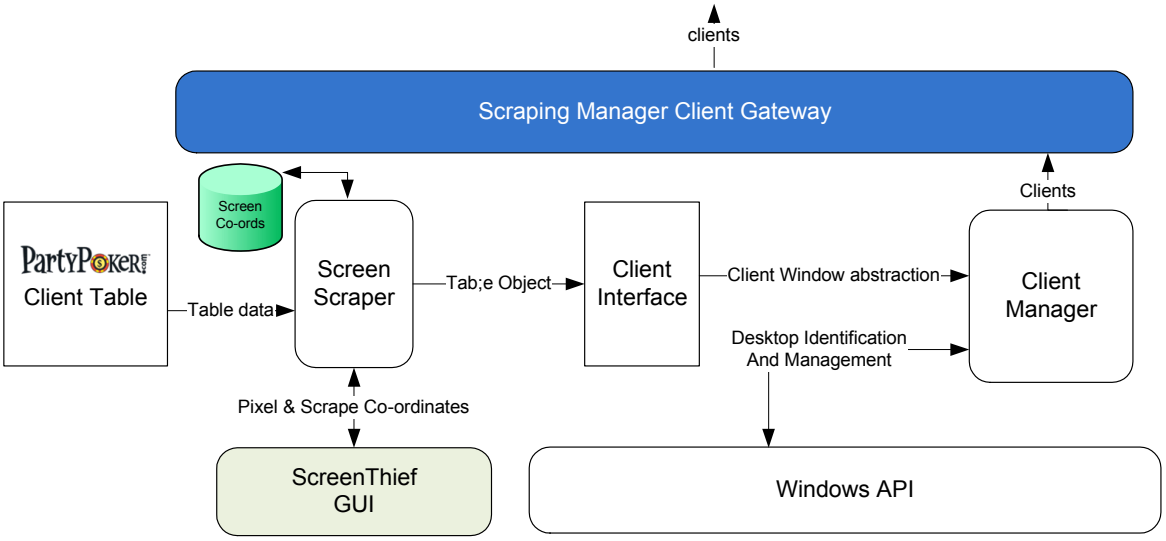

FIGURE 7 – THE SCRAPER MANAGER ARCHITECTURE

The *Screen Scraper* sub-component listens to the *Client Table* for a new hand to begin. It will then access the *Screen-coordinate* database, to retrieve the relevant pixel and x/y boundaries and forward them to a text scraping COM library.

The interface to *Scraping Manager* will allow an external component to extract the running representational client windows (containing the *Table* objects). *PartyPoker* allows multi-tabling (playing several poker tables at once), so the scrape manager will need to support multiple table interaction.

The Windows API allows low level interoperability (P/Invoke) calls to manipulate external process windows (such as the *PartyPoker* client windows) which are contained in *Windows API* sub-component. (P/Invoke Interop .net Wiki)

3.5.1.3 PHASE ONE DESIGN COMPONENTS - HAND EVALUATOR

The *Hand Evaluator* is a component that given a table state, delivers hand evaluation algorithm results which are crucial to both the decision making and opponent prediction functions of the solution. The component uses the *C to C# PokerEval Library* (described in **3.3.4**) for poker related base comparisons which are heavily used in all evaluations.

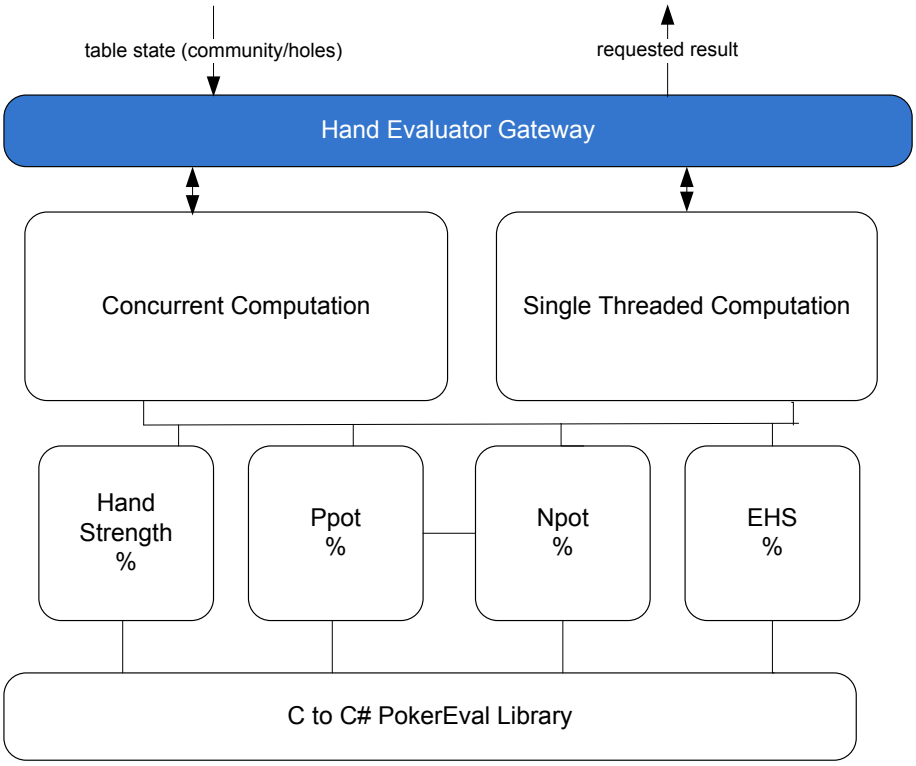

FIGURE 8 – THE HAND EVALUATOR ARCHITECTURE

The algorithms employed are as follows:

Hand Strength is the probability that a given hand is better than that of an active opponent at the current table state.

Hand Potential - After the flop, there are still two more board cards to be revealed. On the turn, there is one more card to be dealt. Hand Potential can be used to determine the chance that the agents hand will be the best when all remaining community cards have been dealt.

Positive Potential is the chance that a hand that is not currently the best improves to win at the showdown.

Negative Potential is the chance that a currently leading hand ends up losing.

It can be noted that the *Hand Evaluation* component offers the ability to evaluate the algorithms asynchronously (by calling *Concurrent Computation*). This allows the solution to consume multi-core processors for a much faster evaluation speed.

3.5.2 PHASE TWO

'Phase Two' removes the *Rules Manager* to be replaced with the *Game Tree Simulator*. The table state information in passed from the *Scraping Manager* to the *Game Tree Simulator* where a game tree is constructed of an abstracted version of possible situations that can occur from the current snapshot of the hand. Nodes in the tree are constructed for many possible agent actions, allowing an evaluation (tree searching) algorithm using *Estimated Value* calculations to decide on the action to take with the best aggregated result.

Leaf nodes are computed for hands where the agent is simulated to have folded, won the current pot, or showed down with an opponent. The leaf nodes return a value representing how many chips are predicted to be won, and are return a value defined as *Estimated Value* ($EV).

Offline hand histories (as described in 3.3.3) are parsed, clustered used by an opponent modeler to supply predictions to the simulation making $EV calculation more accurate. Finally, the action with the highest $EV is chosen from the tree, and passed to the *Action Selector*.

3.5.2.1 PHASE TWO - HIGH LEVEL ARCHITECTURE

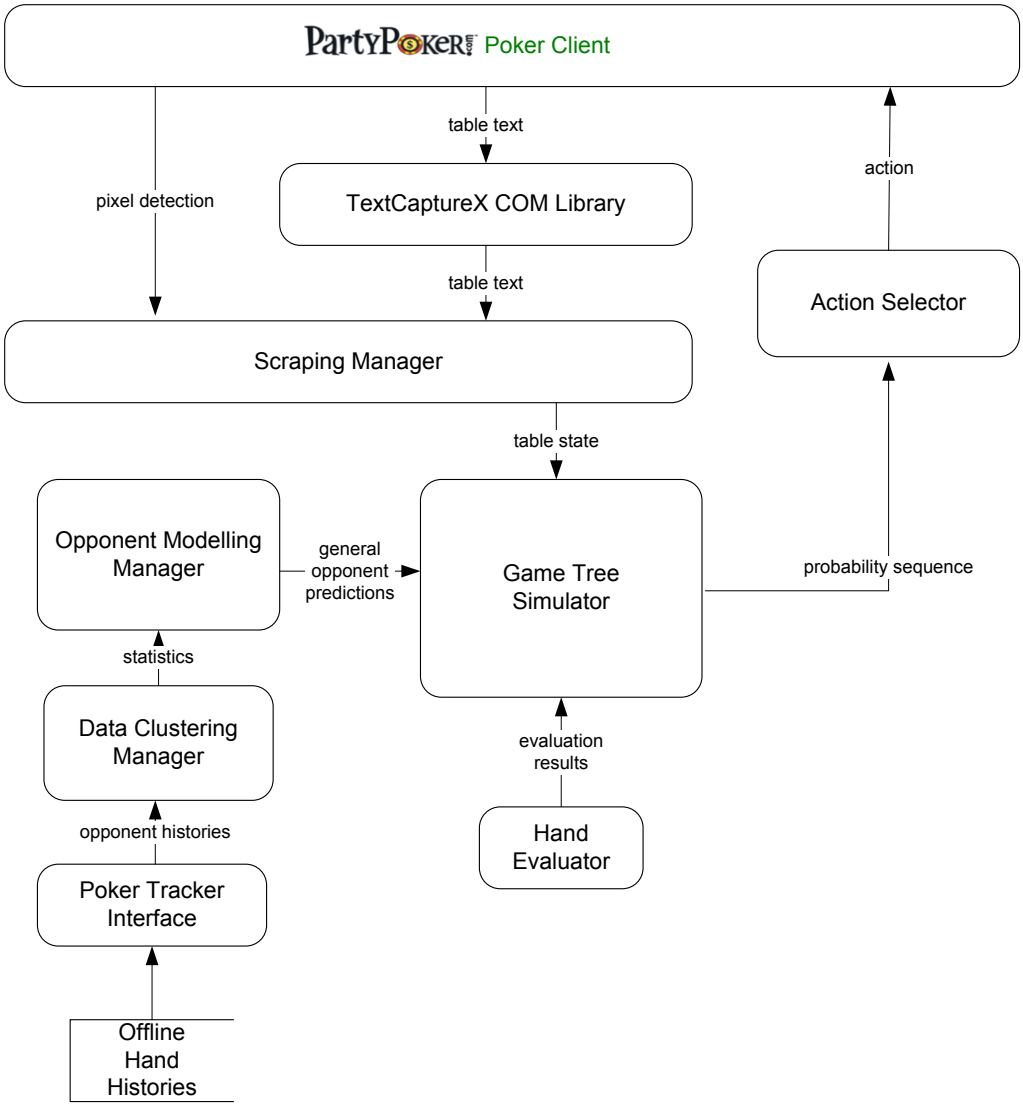

FIGURE 9 - PHASE TWO HIGH LEVEL ARCHITECTURE

3.5.2.2 GAME TREE SIMULATOR

The *Game Tree Simulator* is the most important component of the 'Phase Two' solution, and is the tallest in the hierarchy of control. The *Game Tree Simulator* operates by taking a game state of the table and constructing a game tree of abstracted possibilities that remain in the hand in order to determine the best course of action.

The following game tree example is heavily simplified to demonstrate its operation, and in practical use the game tree will usually consist of many more nodes depending on the game state. The sample tree is constructed on the river (no more board cards dealt) with the following game context:

The community cards are **A♠ T♥ J♣ 2♣ 5♦** with a current pot of $35. There are two players in the hand including the agent. The agent's hole cards are **Q♣ 9♦** and its oppositions are **J♦ Q♦**. Most poker players will be quick to observe here that the agent has a losing hand (as it has missed its draw and would have needed an **8** or a **K** to make a straight) with **Q** high and its opposition currently has the winning hand, with a pair of Jacks.

This is unlikely to be a profitable situation for the agent given that its current hand is beaten. If the game tree is abstracted to allowing $20 bets, and only allowing a maximum of 2 actions per player, the tree could be visualized as follows:

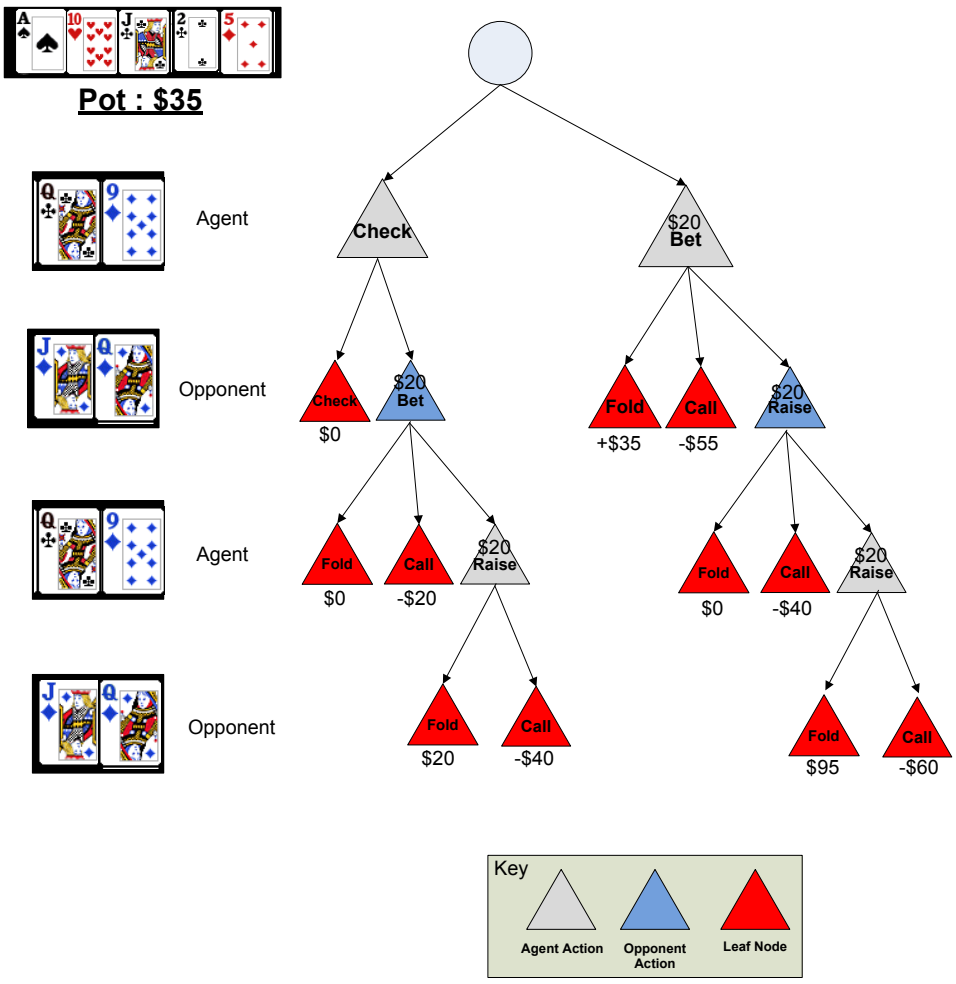

FIGURE 10 - A SIMPLIFIED 'OPEN' GAME TREE

By calculating the leaf nodes the agent can view what the most profitable line of action entails.

As there are no supplied opponent action predictions at this point, the action to raise, and re-raise (hoping the opponent will fold) can be identified as the highest valued option (at **$95**).

It can also be noted that the game tree does not calculate the current pot in the evaluation function, as the pot is treated as external value rather than something into which the agent has already committed value. Thus, the tallest agent 'Fold' leaf nodes are valued at **$0**, rather than the loss of the pot size (**-$35**).

The game tree is an extremely powerful tool in deciding the correct actions in given situations. Algorithms such as *expectimax* or *miximix* (a stochastic variation of the popular *minimax* algorithm (Billings, 2006)) can be applied to ensure that the agent plays pseudo-optimally. However, without estimations of potential hands and weighting certain action predictions for opponents the extra equity in exploiting and adapting to opponent strategic mistakes cannot be extracted.

3.5.2.3 OPPONENT MODELLING MANAGER

The *Opponent Modelling Manager* uses a variety of methods to try and accurately predict imperfect information within the domain.

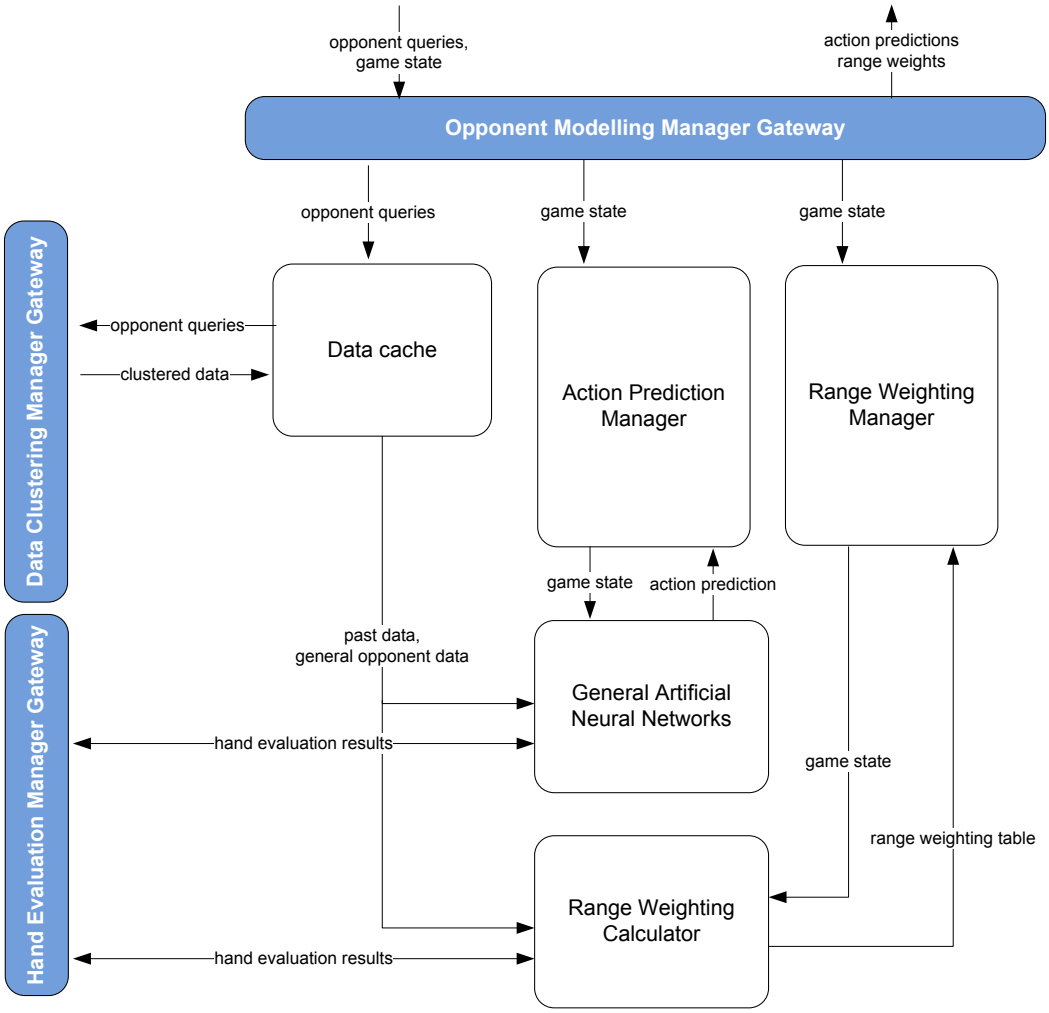

FIGURE 11 - OPPONENT MODELLING MANAGER ARCHITECTURE

There are three main elements of imperfect information within poker.

- Un-dealt board cards
- The opponents potential card holdings (range)
- The opponents strategy (which depicts how he likely to play his range)

Un-dealt board cards are uniformly random allowing accurate calculations providing probabilities for drawing hands. However, human opposition will be expected to employ certain strategies and therefore assumptions can be derived from observed past and current actions of how the opponents are likely to act.

Therefore, potential card holdings and how each hand is likely to be played needs particular attention so models and predictions can influence how each opponent should be played against.

It would be a naïve to make an assumption that an opponent has one particular hand, as various hands will be played in similar manner. Therefore, an opponent is assigned a *range* of hands

that can be weighted in terms of probability, which is created by the *Range Weighting Calculator*.

Take for example a flop of **A♠ T♥ 2♣** where the agent is re-raised by a single opponent. The re-raise indicates that the opponent holds a strong hand, but there are several strong hands that are likely to be played in this way.

Possible weighted holdings (in order of strength) could be:

{AA, TT, 22, AT, A2, T2, AK & AQ}

In which probabilities can be assigned to each hand given the raise that the agent has just observed.

A potential weighting would be:

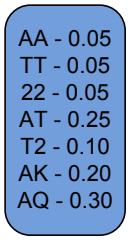

FIGURE 12 - AN OPPONENT RANGE PREDICTION

It can be observed that all probabilities sum to 1. In this example scenario, the opponent modeler may have predicted **AK** and **AQ** to be highest probability of hand as it has assumed that should this particular opponent hold extremely strong hands such as **AA**, **TT** or **22** that they would be unlikely to raise, and would introduce deception by *slowplaying* their hand.

This component allows the agent to adapt its strategy against varied opponents. In the above scenario the agent would most likely fold all but the strongest hands. If the scenario was changed where the agent was playing a very loose opponent, then a much wider range would be assigned, and the agent would adapt its strategy to meet these predictions.

The second component of the *Opponent Modelling Manager* is the *Action Predictor*. The *Range Weighting Calculator* creates the range, but this component predicts how an opponent may play each holding for various game contexts. As seen in the next chapter, this allows the agent to perform estimated value calculations to decide how to play the hand by exploiting the opponent.

For each possible hand in the range, a *probability sequence* is assigned. The sequences can then be summed for an overall prediction of how the opponent may play his range, as defined in Figure 13 where a prediction is made of how the opponent will play after the agent further re-raises.

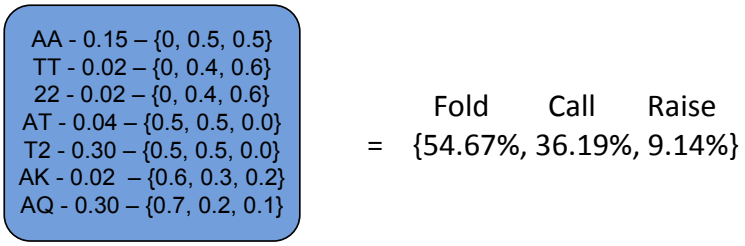

FIGURE 13 - AN OPPONENT RANGE WITH ACTION PREDICTIONS

Based on the probability sequences (simplified for this example) of {fold%, call%, raise%} it can be observed that the opponent modeler has predicted that the opponent would be most likely

fold to a re-raise. In this case the combination of range and action predictions has identified a flaw in the opponent's strategy – he folds too much.

In this scenario, assuming the range and action predictions are accurate, the agent should re-raise the opponent the majority of the time as the agent will win the pot immediately.

The action predictions are maintained by the *Generic Artificial Neural Networks*. An artificial neural network is a computational model based on biological neural networks. (Artificial Neural Networks) These models support re-enforcement learning that is highly appropriate to the domain of machine learning for poker.

Given that the agent has access to millions of hand histories, this data can be used to train the agent on the opponents it is likely to be playing and how they are most likely to act. The knowledge gained from training the networks on these histories should be sufficient to act as the main tool in supporting decision making components in the agent's architecture.

3.5.2.4 COMBINING THE OPPONENT MODELLING MANAGER WITH THE GAME TREE

Applying the opponent modeling elements to the game tree creates a much more practical depiction of the final constructed simulation.

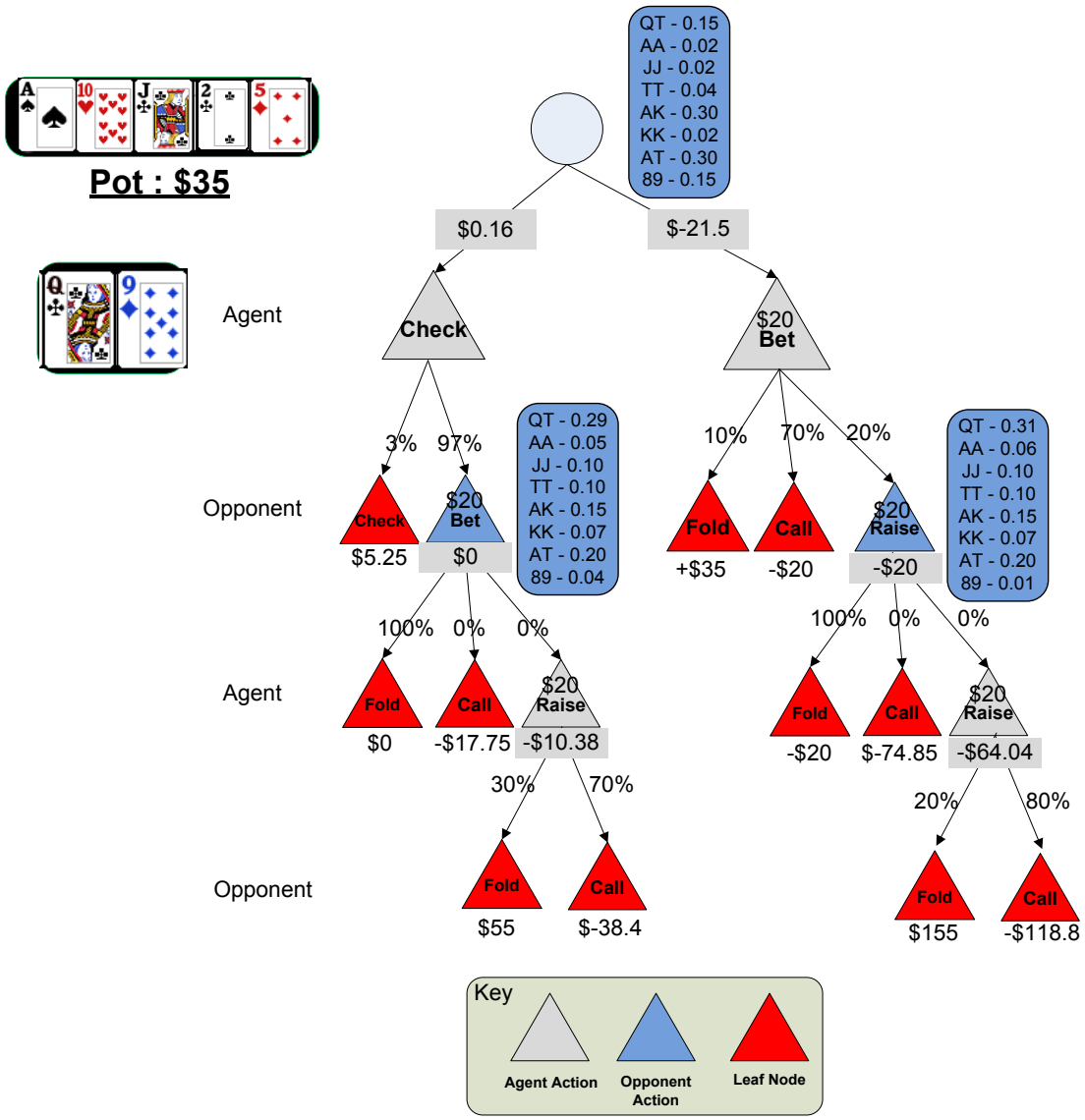

FIGURE 14 - OPPONENT MODELLING APPLIED TO A 'CLOSED' GAME TREE

The game tree is re-evaluated as shown in section 3.4.2.2 but the opponents specific hand has been removed (as in almost all situations, the opponents range can only be estimated). The opponent modeling manager up to this point has assigned the opponent the range of

{JJ+,AKs,AJs-ATs,98s,AKo,AJo-ATo,98o}

and has weighted the holdings in correlation to its predictions.

This adds a further layer of complexity in the evaluation functions that are calculated throughout the simulation, as the solution now has to deal with multiple holdings to calculate the equity. After each opponent action, the ranges are re-weighted by the modeling predictions, and further predictions are applied to each opponent action.

98 is currently the only hand in the opponent's range that the agent can currently beat, but has been assigned very low probability by the opponent modelling manager. Thus, for calculated

nodes where the agent either calls or checks, the majority of the pot value is won by the opponent.

Applying opponent modeling to the game tree provides a much more accurate simulation of the scenarios that are likely to occur. Leaf nodes are calculated for equity and passed up the game tree to provide an *Estimated Value* for the current agent action. It can be observed that checking (with the intention to fold to a bet) is the action with the best value, which is accurate to the situation that has occurred and is how poker players with an understanding of the game would be most likely to act.

3.5.2.5 DATA CLUSTERING MANAGER

The data clustering manager clusters players found in the *PokerTracker* database into groups that represent different playing styles.

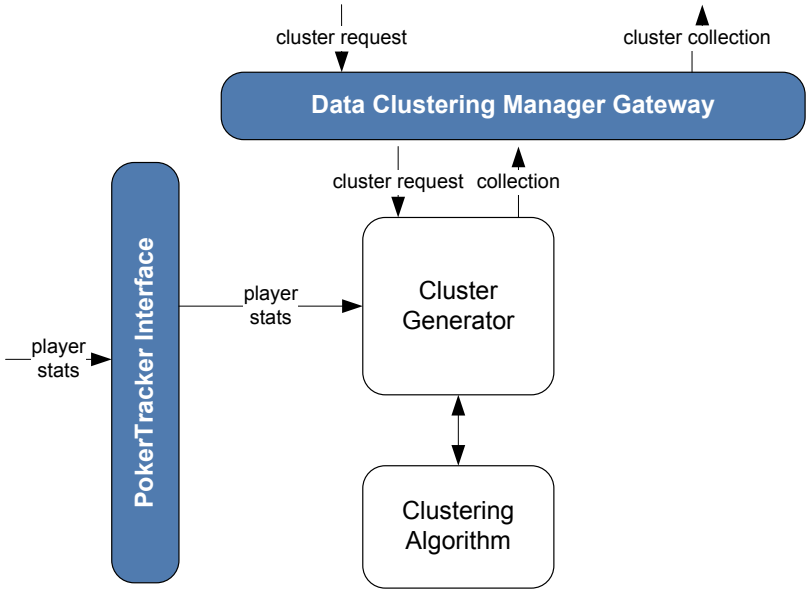

FIGURE 15 - DATA CLUSTERING MANAGER ARCHITECTURE

The data clustering algorithm will implement a K-Means data clustering algorithm to determine the distance measure between groups of players.
The two statistics that will be based on are as follows:

1. **VPIP** (Voluntarily Put In Pot) – this statistic quantifies how *loose* a player is, a higher percentage of VPIP indicates a player who calls a large range of hands.
2. **PFR** (Percentage of Pre-flop Raise) – this statistic quantifies how many hands a player is prepared to raise with. A player with a higher percentage of PFR will be known to raise many marginal hands.

These two statistics are central to quantifying an opponent's strategy and will allow new players to be categorized into their respective playing styles for analysis.

The implementation of the K-means algorithm is described in section 4.2.1 and the results of the player data clustering are listed in section 5.1.

4. IMPLEMENTATION

4.1 PHASE ONE COMPONENTS

4.1.1 SCRAPING MANAGER

The purpose of the Scraping Manager is to retrieve a game state snapshot of the online poker client when it is the agent's turn to act. This populates a *Table* object which is representational of the poker table in which the agent is playing.

SCRAPING THE POKER CLIENT

The *Table* object is populated by one of two ways. Firstly, text can be extracted from the poker client by passing screen co-ordinate vectors to the external **TextCaptureX** library (3.3.4). Secondly, tests for specific pixel color can extract the rest of the game state properties.

The process is visualized in Figure 16:

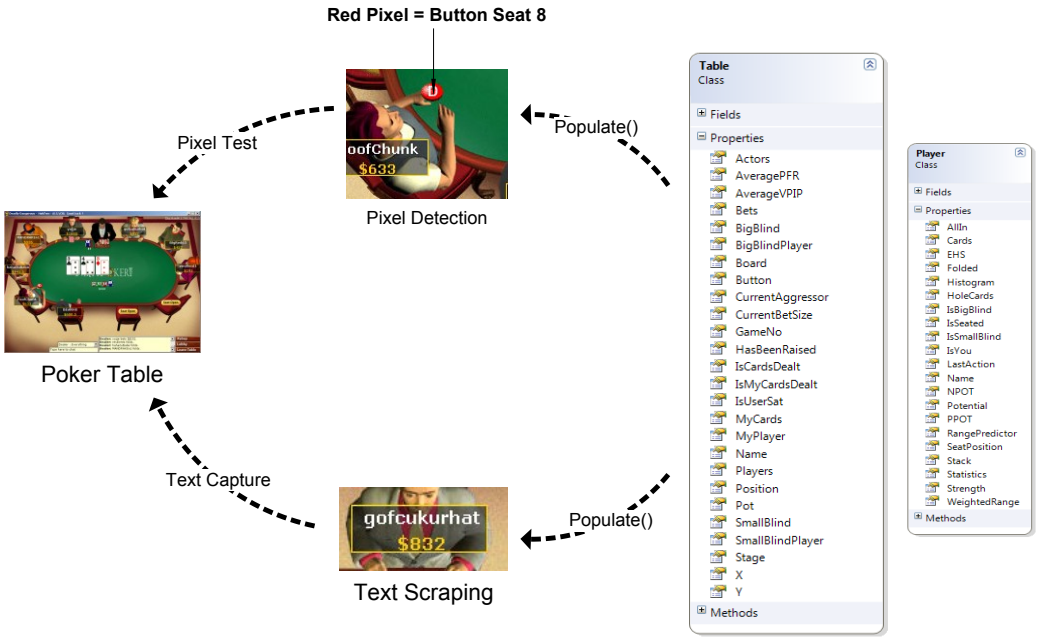

FIGURE 16 - TABLE POPULATION THROUGH THE SCRAPING MANAGER

Coordinates of both pixel and text vectors are maintained via a simple relational SQL database named *Scrape Boundaries.* The database structure is defined in Figure 17.

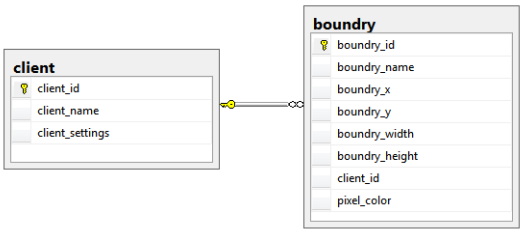

FIGURE 17 – SCRAPE-BOUNDARIES DATABASE STRUCTURE

The scraping manager is implemented as an event-based architecture. This means that external processes can *subscribe* to the table object and are informed each time the table is populated with a new hand. An overview of the scraping process is visualized in Figure 18.

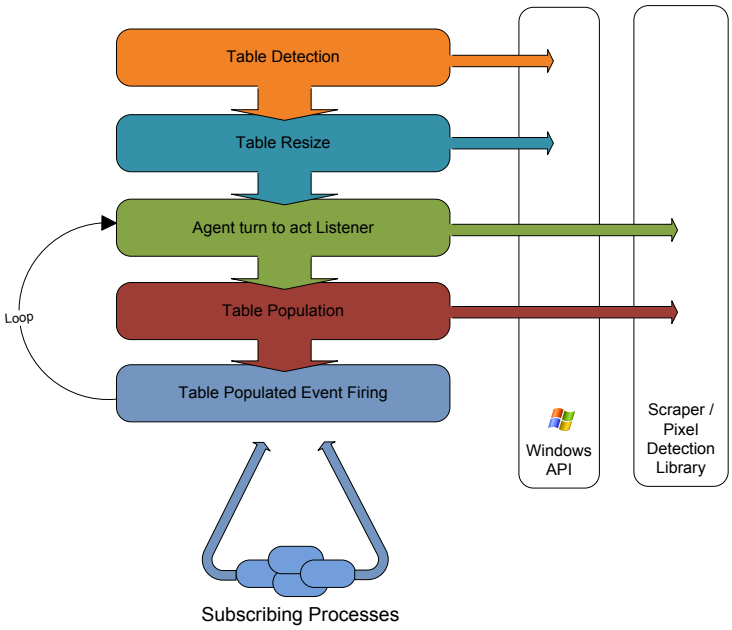

FIGURE 18 - THE SCRAPING PROCESS

HOW TO USE THE SCRAPER

Once the scraper is initialised, a *ClientDetected* event must be subscribed to that adds the logic of how to handle a new poker client. This will reveal the table object to the subscribing process to allow more logic to be added.

```
pokerInterface.ClientManager.ClientDetected += new ClientManager.ClientDetectedHandler(ClientManager_ClientDetected);
```

Once the client is detected – there are several events within the *ClientWindow* that can be subscribed to:

New hand has started

When a new hand has started, the scraper fires this event. This can be useful for subscribing processes to gauge whether to initialise or reset their state.

New board Cards dealt

When the flop, turn or river is dealt an event is fired. This is useful for subscribing processes that rely on using hand-evaluation to quantify visible cards. This event will allow better efficiency as the algorithms can be applied as soon as the cards are made visible and before the agent must make its decision.

Agents cards have been Dealt

As with the case in the new board cards being dealt, this event is fired when the hole cards are dealt to the agent. Subscribing processes could use this for pre-flop hand evaluation algorithms.

Agents Turn to Act

The most important event fired within the scraper is the agent's turn to act. Subscribing processes can then use this to apply their own logic to the decision making elements of the solution.

MAINTENANCE

It is common for a poker client to regularly change the layout of their tables, which poses a problem for our database of state vectors.

To enable efficient maintenance of the scraper, the screen co-ordinate database is available to the environment through an implemented application named *ScreenThief,* demonstrated in Figure 19.

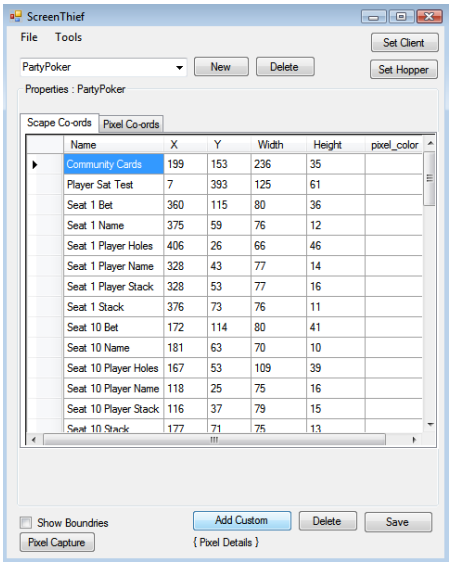

FIGURE 19 - SCREENTHIEF GUI

The purpose of the *ScreenThief GUI* is firstly to enable easy accessibility to the *ScrapeBoundaries* database for fast and efficient changes, and secondly as an effective debugging tool should the scraping of the poker client fail. Stored co-ordinates can be tested by clicking the respective boundaries in UI and new clients and boundaries can be added through the front end interface.

4.1.2 RULES MANAGER

The purpose of the rules manager is to supply expertly encoded rules in which the *Phase One* agent will play by. As the single objective of the rule based system is to contrast the difference between typical AI approaches and an expert system - the rules are hard-coded into the system via a *Strategy* object which can be plugged into the *Table* object.

Strategy rules make use of both table state data and hand evaluation algorithm results to decide how to act on the table. The strategies assume a very simple strategy of betting or raising strong hands and folding the rest.

4.1.3 HAND EVALUATION

A strong poker strategy depends on firstly measuring the value of the cards dealt to the player, then comparing those values to the community cards as and when they are dealt. Human poker players can measure strength by comparing the cards in their hands with their general awareness of the poker rules (for example an Ace is higher than that of a King) and their past experiences at the poker table (for example a three of a kind is a very strong hand and is it usually strategically correct to bet to gain value).

Implementing the AI agent to act in this way would be both non-trivial and non-optimal. Instead two functions have been devised to quantify hand strength – one for pre-flop and one for when community cards have been dealt.

PRE-FLOP

For measuring the strength of a pre-flop hand a simple poker game simulator was constructed in which ten simulated players were dealt random cards and *check/called* the hand down to showdown. The winning hand would be recorded to memory and the simulation would repeat. To mitigate the high variance associated with poker this simulation was repeated 1,000,000 times. The results show win percentages ordered by the best hand to the worst and are shown in Figure 20. The full results are included in Appendix D.

```
Title : Call Only Preflop Sim
Date : 03/12/2008 20:24:29
Description : Call-only showdown equity simulation

Pockets : AAo 85.14%
Pockets : KKo 82.22%
Pockets : QQo 79.63%
Pockets : JJo 77.2%
Pockets : TTo 74.57%
Pockets : 99o 71.08%
Pockets : 88o 68.24%
Pockets : AKs 67.67%
Pockets : AQs 66.41%
Pockets : AKo 66%
Pockets : 77o 65.74%
Pockets : AJs 65.72%
```

FIGURE 20 – PRE-FLOP ROLLOUT SIMULATION SNIPPET

The results were as expected giving a mathematically proven insight into the most profitable poker hands. The pre-flop iterated roll-out simulation can also be used for comparing opponent tendencies with pre-flop holding likelihoods within the opponent modeling manager. For instance, if it is known that an opponent calls 10% of all hands in certain situation, then his hand distribution can be reweighted by the top 10% of the roll-out results. This would result in a range of **{ 88+,A9s+,KTs+,QTs+,AJo+,KQo }**

POST-FLOP

Quantifying the strength of an agents hand when community cards have been dealt is a less trivial task. Varying community board values and card textures can significantly impact the strength of a hand. For example, in the instance where the agent holds A♠ K♠ (a very strong pre-flop starting hand) and the board is dealt as 9♥ T♥ J♥ the agent now has to deal with the possibility of an opponent already holding a better hand or drawing to a straight or a flush.

The post-flop *Hand Evaluator* makes use of the C to C# pokereval library in which a framework has been supplied for very fast card and hand comparisons.

As described in the design, the four algorithms for quantifying a poker hand are Hand Strength, Hand Potential, Positive Potential and Negative Potential which are described in more detail in this chapter.

Hand Strength (HS)

To recap, Hand Strength is an algorithm that produces a representing the probability that the current hand is the best, with no more community cards being dealt.

Example : A♥ T♥ on a board of T♠ 9♠ 6♠ holds an **89.64%** equity of the pot against a random hand.

The algorithm is computed by iterating over all possible opponent hands and using the *pokereval* library to determine whether the agents hand is better(+1), tied (+1/2) or worse (0). Taking the sum and dividing by the total number of possible opponent hands returns the resulting *Hand Strength*.

This is a useful function for determining the current absolute value of a hand for deciding whether or not to bet.

Hand Potential

Hand Potential produces a percentile result of how likely it is that the current hand will win after all hands have been dealt.

Using the same example, A♥ T♥ on a board of T♠ 9♠ 6♠ holds an **63.45%** equity of the pot against a random hand (note the result difference between potential and strength).

This algorithm is computed by enumerating possible opponent hands and all possible community cards that can be dealt until the end of the hand. The *pokereval* library can then identify when the agents hand was better, tied, or worse. Taking the sum and dividing by the total number of possible opponent hands gives the *Hand Potential*.

This algorithm is useful for determining the chance that the hand will win on showdown, but also for calculating how much equity a certain drawing hand holds. For instance, 4♥ 5♥ on a board of 6♥ 7♥ A♠ holds a very low hand strength of **0.42%** but a hand potential of **62.16%** which could be a deciding factor on how the agent chooses to act with the hand.

Positive Potential

Positive Potential produces a percentile result of how likely it is that the hand is the currently the worst, but will improve to be the best when the remaining community cards have been dealt.

The algorithm is computed in a similar manner to Hand Potential but is modified using the counting variables to determine the result.

Using the same example, A♥ T♥ on a board of T♠ 9♠ 6♠ holds an **18%** chance that it will *improve* to be the best hand.

Negative Potential

Negative Potential produces a percentile result of how likely it is that the hand is the currently the best, but will be the worst when the remaining community cards have been dealt.

Using the same example, A♥ T♥ on a board of T♠ 9♠ 6♠ holds a **35%** chance that it will be the best hand now, but will be beaten by a *drawing hand*.

This is useful when combined with *Positive Potential*. In the given example, Ah Th has a small positive potential and a relatively large negative potential, therefore it would be beneficial for the agent to bet appropriately to force opponents with drawing hands into making both strategic and mathematical mistakes when opting to continue in the hand.

THE EVALUATION WORKBENCH

To efficiently test the Hand Evaluation components, an evaluation workbench application was implemented – which is demonstrated in Figure 21.

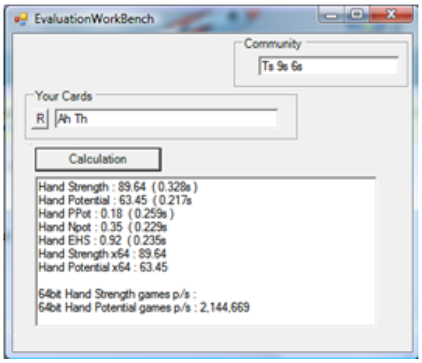

FIGURE 21 - THE HAND EVALUATION WORKBENCH

The workbench calculates the results for all algorithms against a random hand. When 'R' is pressed, the *Range Chooser* is shown. (Figure 22)

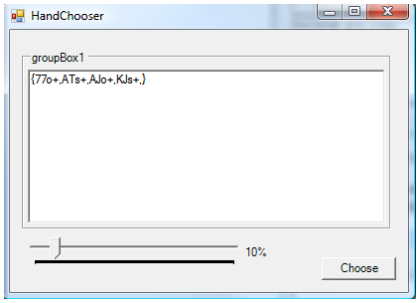

FIGURE 22 - THE HAND RANGE CHOOSER

Since it is very unlikely that an agents opponent has an equal chance of holding a uniform distribution of hands, the hand chooser allows the range to be narrowed for increased accuracy in evaluation calculations. In Figure 22 the range of the opponent has been narrowed from a random distribution to the top 10% of all hands as determined by the pre-flop iterated roll-outs.

Now, instead of A♥ T♥ on a board of T♠ 9♠ 6♠ holding a **63.45%** potential, the result has been reduced to **48.31%** which holds a more likely equity estimate against a tight opponent.

The Hand Evaluation library also supports *Weighted Ranges* in which hands held within the narrowed range can explicitly weighted to provide more precise equity calculations.

For example, in the case where the agent is raised by an opponent who is known to deceptively *limp* extremely strong hands such as AA or KK, the opponents range of **{ 99+, AJs+, AKo }** will not be uniformly weighted. If the hand breaks down as shown in Figure 23, then this will further affect the actual equity calculations which are encapsulated within the component.

```
AA :  4%     AKo : 14%
KK :  4%     AKs : 14%
QQ :  7%     AQs : 14%
JJ : 10%     AJs : 12%
TT : 10%     99 :  12%
```

FIGURE 23 - A SIMPLE WEIGHTED RANGE

4.2 PHASE TWO COMPONENTS

4.2.1 DATA CLUSTERING MANAGER

Because of the large data requirement for training neural networks, an agent cannot wait until it has collected thousands of scenarios before it trains on a specific player. Instead, new players are identified and compared against the database of observed players in PokerTracker so *similar* data can be extracted which is enough for the networks to train on.

The Data Clustering Manager provides the functionality to cluster the *PokerTracker* database into the respective groups of strategies on which the networks be identified and trained.

K-MEANS CLUSTERING ALGORITHM

K-Means Clustering is a method of cluster analysis which aims to partition *n* partitions into *k* clusters in which each observation belongs to the cluster with the nearest mean. (Wikipedia - K-means)

Clusters are initialised with a random *Centriod*. A Centriod represents the middle of the data cluster. The observations (statistics) are then iterated, assigning the observations to their nearest cluster using the allocated centriod. Centriods are then recomputed as the mean of their observations. These steps are repeated until the observation allocations no longer change, in which the algorithm has converged.

The algorithm implementation is applied to both the VPIP and PFR statistics to produce 9 different strategy data clusters, which are listed in Figure 53.

4.2.2 OPPONENT MODELLING MANAGER

The opponent modeling manager meets two very specific aspects that are critical to very strong poker play:

1. What an opponent's likely hand distribution will contain.
2. What action the opponent is likely to take with each of those hands.

By answering these questions accurately the agent is able to model an opponent's strategy. By quantifying likely player actions across a range of potential hands this will also highlight weaknesses that exist within a sub-optimal strategy which then can be exploited.

At the core of the Opponent Modeling Manager is the Neural Network Manager which provides a likely action prediction named a *probability triple* (Billings, 2006). A probability triple contains a normalized distribution over action possibilities such as *{20% Fold, 50% Call, 30% Raise}* which is useful for modeling *mixed strategies* under the assumption that an opponent may not always play the same way under similar scenarios. Additionally, the resulting predictions are not always 100% accurate but accurate enough to highlight the tendencies embedded within an opponent's strategy.

ARTIFICIAL NEURAL NETWORK IMPLEMENTATION

An *Artificial Neural Network* (ANN) was chosen as the medium in which to provide the probability distributions. ANN's are computational models based upon biological neural networks and are used to model complex relationships between inputs and outputs to find patterns in data, and are known for their effectiveness in operating in domains high in noise. (Wikipedia - Artificial Neural Network)

The neural networks have been implemented using an external library called *Aforge* which supports pluggable learning algorithms, network architectures and provides a flexible, re-usable and robust solution to creating neural networks for any domain. An example of a neural network can be seen in Figure 24.

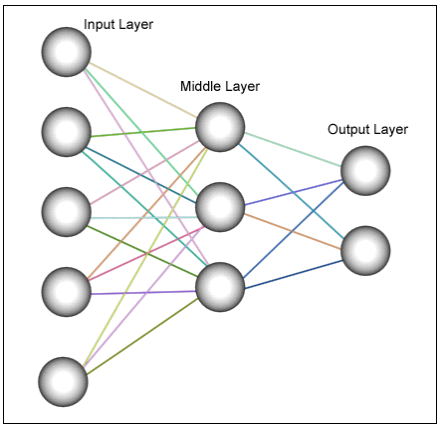

FIGURE 24 - A VISUALIZED NEURAL NETWORK

As with any neural network, there are several parameters that can affect how the networks reach their outputs. There is no correct way to configure neural networks as the parameters are dependent on the training samples and the amount of noise in the domain.

The dynamic parameters are defined as follows:

Learning Rate

The learning rate defines how neurons are re-weighted during each training iteration. A higher learning rate will result in a network that is trained faster (ideal in a real-time network) but will have more chance of reaching local minima. (WISC - Neural Network Configurations)

Momentum

The momentum rate allows the networks to be *throttled* into pushing past the local minima. However, a momentum rate that is set too high will skew the results as they will not settle at their correct threshold values. (WISC - Neural Network Configurations)

Sigmoid Alpha Value

The Sigmoid alpha parameter affects how the neurons are weighted based on their inputs and how their weights are calculated.

Iterations

The number of iterations defines how many times the training datasets are pushed through the network. More iterations will generally result in the network being trained more precisely, but will eventually settle at the correct threshold values, so too many iterations will result in unnecessary computation and time.

Number of Hidden Layer Nodes

The more hidden layer nodes of a network there are the greater number of different input combinations that the network is able to recognise.

A *Back-Propagation* learning algorithm was chosen to assist the network in training based on domain data. The algorithm operates by randomly initializing network node weight values, presenting a training sample to the network, comparing the networks prediction to the actual input output and re-weighting the neurons relative to the calculation of error. In order to train a network efficiently, many samples of data are passed through the network whilst in training mode, dependent on the noise or variance of the domain which in this case is very high. For this implementation 80,000 training samples were found to be sufficient for the training sample size.

The inputs of a neural network are the key properties on which the neural network trains on. When modeling different players it is common for varied network inputs to be more predominant in the decision making process for different strategies. In this manner it is vital that the correct inputs are chosen that will encompass all ranges of strategies as well as staying proportional to the data training samples available, as more inputs will generally require a larger sample size.

The inputs that have been proven to be the most efficient for the training of the networks are outlined in Figure 25.

Hand Rank	int	VPIP	real
Betting Round	int	Aggression Factor	real
Hand Potential	real	Street Aggression Factor	real
Hand Strength	real	Ace Count on Community	int
Negative Potential	real	King Count on Community	int
Positive Potential	real	Number of potential better kickers	int
Percentage of Stack Commited	real	Last Action To Bet or Raise	bool
Pot-Odds	real	Raised pre-flop	bool

FIGURE 25 - NEURAL NETWORK INPUTS

DATA EXTRACTION FOR TRAINING SAMPLES

The training sample data for the neural networks are created from hands stored in the *PokerTracker3* database, sub-clustered by the *DataClusteringManager* and extracted from hand history text files (Figure 3). When poker hand samples are imported into the PokerTracker database they are stored as one single hand. This poses a problem, as there are several actions players may take within a single poker hand that must be extracted for training the networks. A solution was created to parse the hands stored in the PokerTracker database into an array of actions named *scenarios*. The scenarios are stored in a SQL database in which the neural networks can assign predictions. Actual hand and board cards are also parsed into their respective strength and potential values for neural network consumption. The data flow model is shown in Figure 26 whilst the *NeuralNetworkDataStore* database schema is shown in Figure 27.

Sample scenarios are then pushed through the networks to act as debugging data to test the accuracy of the newly trained networks and to store the prediction results for manual testing.

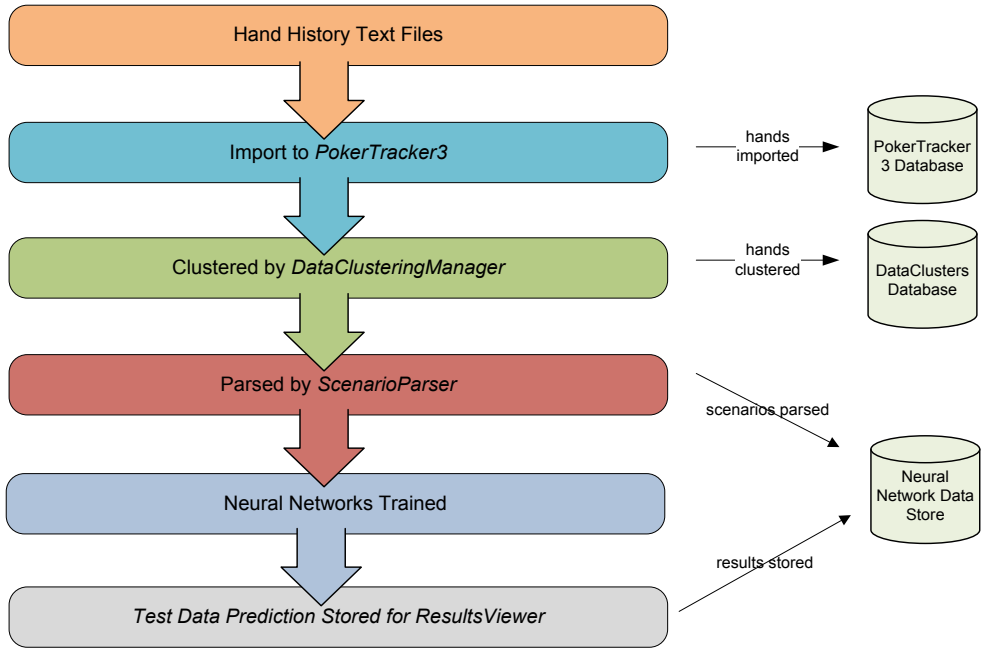

FIGURE 26 - DATA FLOW VISUALISATION

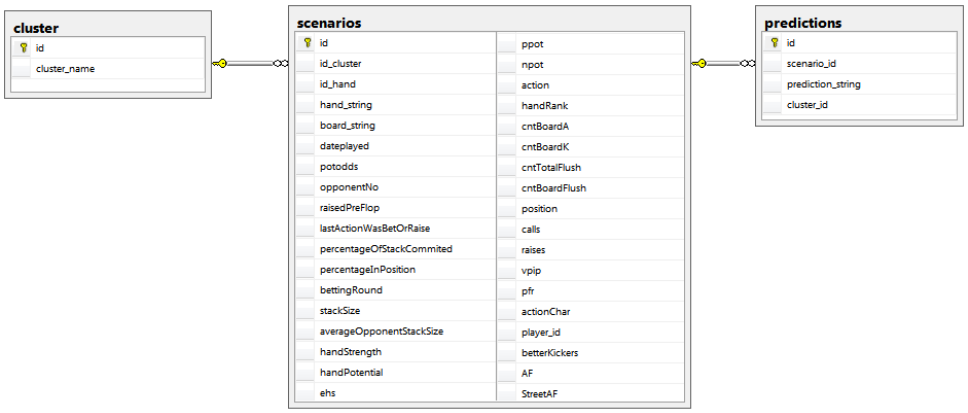

FIGURE 27 - NEURAL NETWORK DATA STORE (PART 1)

FINDING THE OPTIMAL NETWORK PARAMETERS

For the optimum parameters of the neural networks to be found and to maximize the accuracy of the network, a testing suite was implemented to exhaustively iterate possible parameters. The results are shown in Figure 28, 29, 30 & 31.

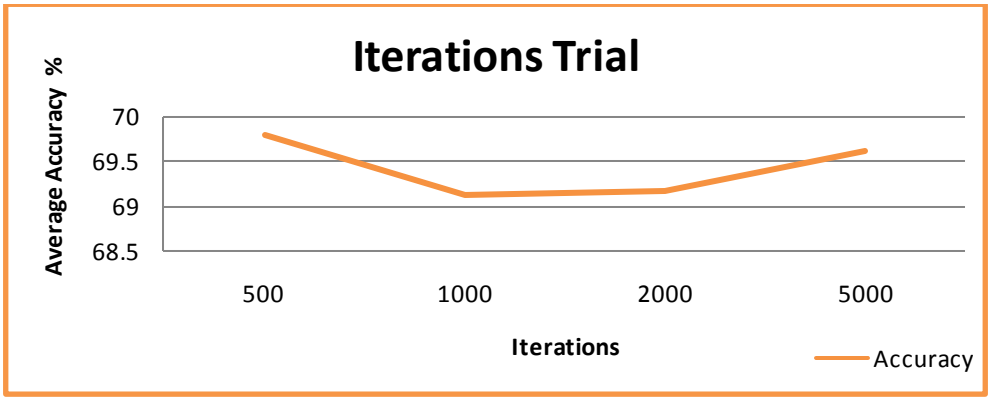

FIGURE 28 - NEURAL NETWORK ITERATION ANALYSIS

Iterations Trial

The number of iterations affected the accuracy of the network by around one percent – the graph shows the accuracy to have an incline with more iterations, with the exception of the '500' iterations test, which given that the neurons start with random threshold weights could be down to a lack of training skewing the results, rather than a more accurate network.

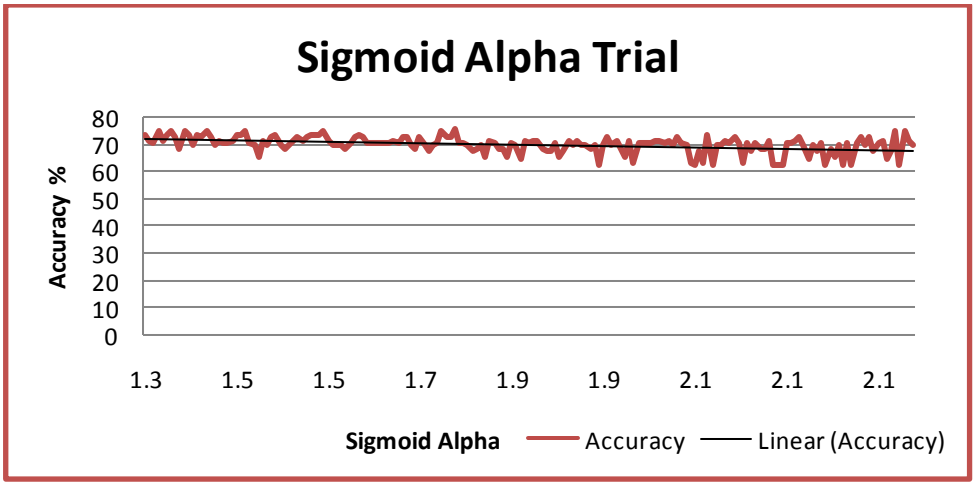

FIGURE 29 - NEURAL NETWORK SIGMOID ALPHA ANALYSIS

Sigmoid Alpha Trial

Sigmoid Alpha had a higher affect on the accuracy with a steady decline being observed from a higher sigmoid value. It is interesting to note that a higher sigmoid value affected the stability of the results as shown with the distorted accuracy line continuing through the graph.

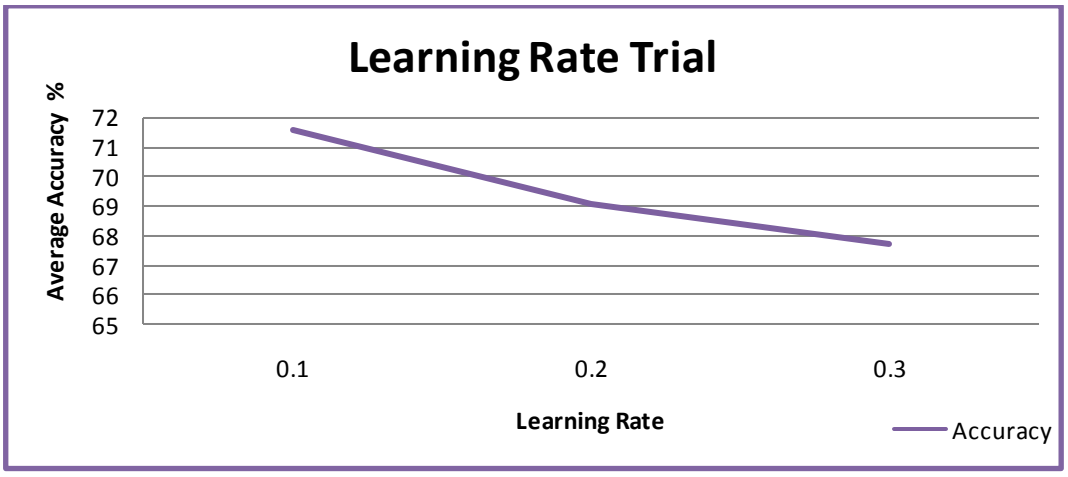

FIGURE 30 - NEURAL NETWORK LEARNING RATE ANALYSIS

Learning Rate Trial

The learning rate had the greatest effect on the accuracy of the results. Increasing the learning rate steadily decreased the accuracy of the networks.

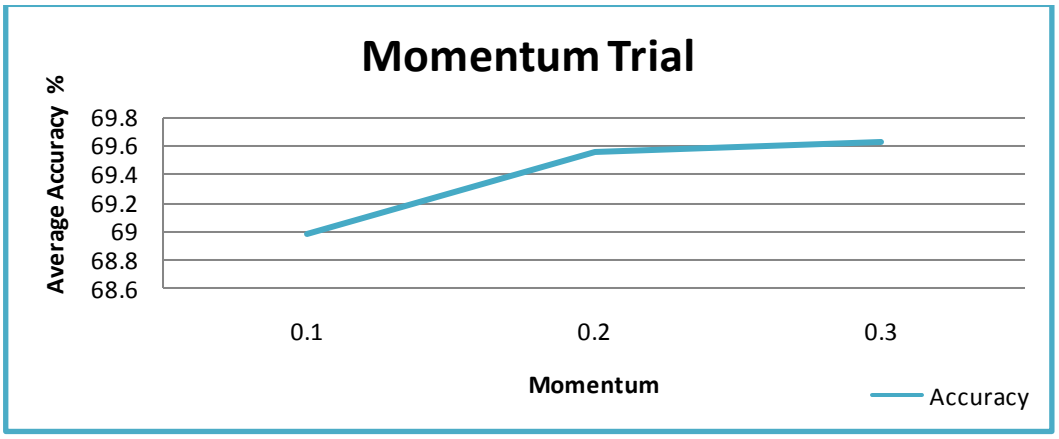

FIGURE 31 - NEURAL NETWORK MOMENTUM ANALYSIS

Momentum Trial

The momentum affected the accuracy by around one percent. It was shown that a higher momentum value generally gave more accurate results although this would be dependent on the learning rate parameter.

It can be concluded that the optimum parameters for the networks include high iterations, low sigmoid alpha, a low learning rate and a high momentum.

HOW THE NETWORKS ARE USED

After the networks have been trained to a satisfactory level of accuracy they can be stored in several different ways.
Firstly, the networked can be serialised either to an *XML* or *Binary* file. This allows the networks to be de-serialised for usage at any time, from any component who wishes to obtain either action or hand distribution predictions.

```
Component 1 : trainedNetwork.Serialise(@"C:\TestANN.dat");
Component 2 : SingleNetwork network = SingleNetwork.DeSerialise(@"C:\TestANN.dat");
```

Secondly, the networks can be stored to a SQL database (schema shown in Fig. 30). This can be beneficial to the primary networks used throughout the implementation as SQL Server offers sufficient redundancy and robustness in long term storage.

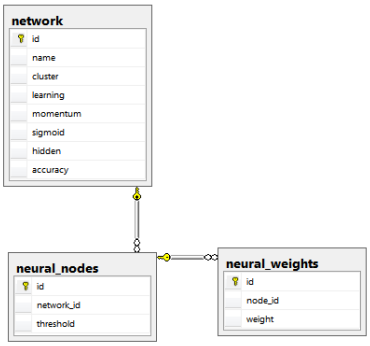

FIGURE 32 - NEURAL NETWORK DATA STORE (PART 2)

Prediction of an opponent's action is achieved by compiling the same input array as noted in Figure 22 and running it through the network using the *Predict()* method. The input array can be created manually or by extracting the scenario out of the poker table (provided by `GenerateAction(Player player)` in the *Table* object).

Now that a function has been provided to predict how a player will act under a certain circumstance it is beneficial to maintain a weighted distribution of likely hands an opponent may hold. This is achieved by re-weighting a player's distribution every time he makes an action on the poker table.

When a player makes an action, a scenario can be extracted from the table which contains the inputs needed to make a prediction (Fig. 24). To re-weight the range with the new action each possible player hand is iterated, calculated for potential and strength (inputs 3, 4, 5 & 6) and is passed to the network for prediction. The prediction is then returned, and the hand is weighted by the action (for example, if the action was to raise, then the likelihood of raising is the value used for weighting). The process can be visualised in Figure 33.

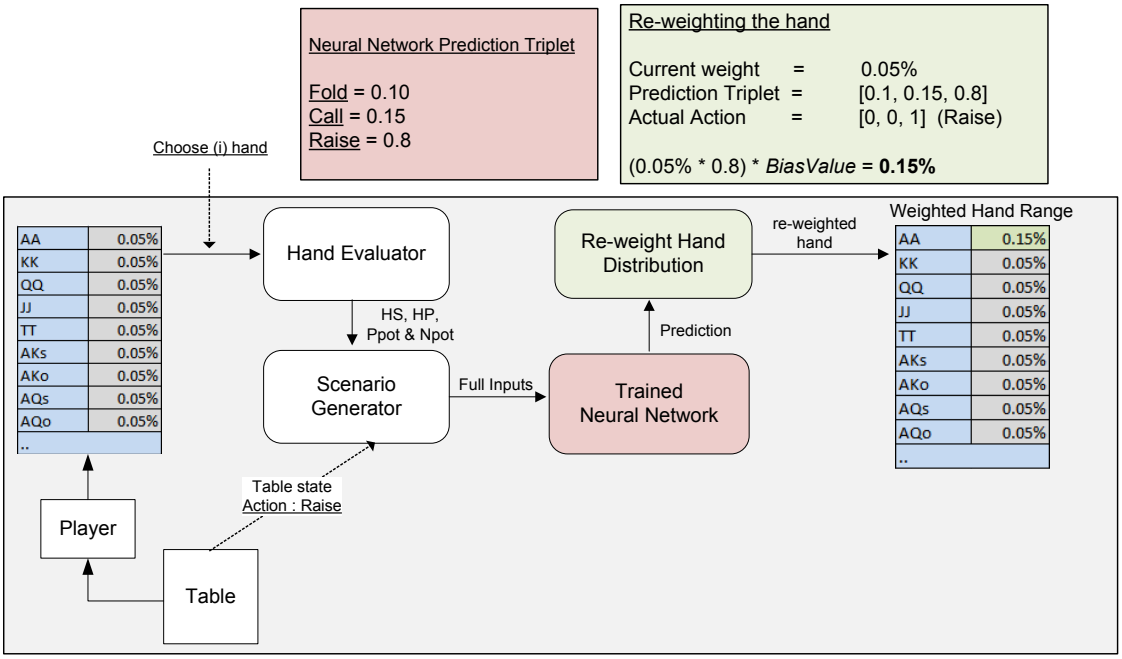

FIGURE 33 - USING NEURAL NETWORKS TO REWEIGHT A HAND DISTRIBUTION

A bias value is calculated to normalise the new hand distribution weights table. This is because a networks output values are real numbers greater than 0 but less than 1 – therefore a very strong prediction of 0.90% would decrease the weight in the weights table without the bias.

This process is encapsulated within the RangePredictor component and can applied by calling the `RangePredictor.ReWeight(WeightedRange range,ScenarioAction action)` method.

NEURAL NETWORK MANAGER

To add the functionality of creating, tweaking, saving and debugging the networks – a neural network managing component was implemented (Figure 34).

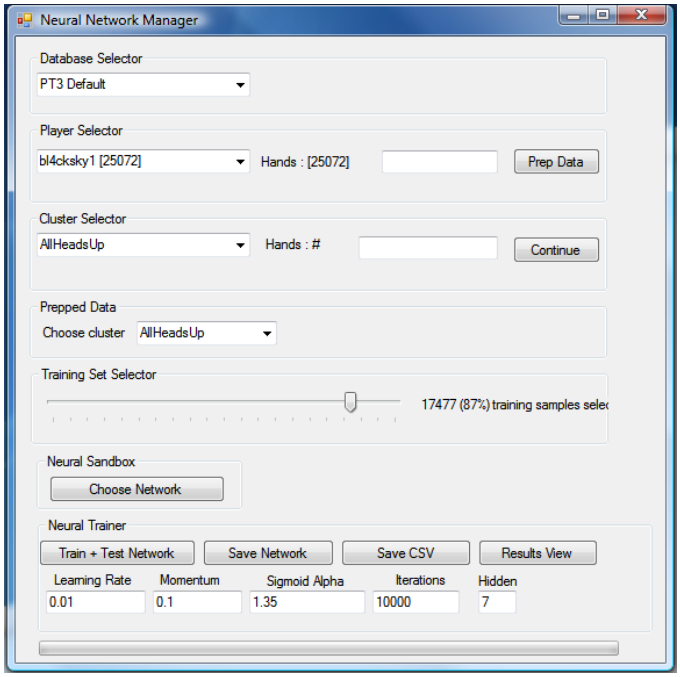

FIGURE 34 - NEURAL NETWORK MANAGER

Firstly, the developer can decide which *PokerTracker3* database they would like to extract data from (as PT3 allows multiple databases). This now enables players to be chosen from the database to cluster data from. When a player is selected, prepping the data will parse the single hand clusters into an array of scenarios for usage in training the neural networks. (See Figure 25). Clusters of player hands created by the *Data Cluster Manager* can also be used for network training.

Once the data is parsed into the neural network format, the training of the data can begin. Developers can use the *Training Set Selector* to determine how much of the data should be used for training data and how much for test data. Generally around 80% for training data is sufficient to test the network.

As the time-duration of training is dependent on the supplied parameters, the progress-bar (Figure 35) will indicate the progress of training the network.

FIGURE 35 - PROGRESS INDICATOR OF NETWORK TRAINING

When the networks' training is complete, the application will indicate the accuracy of the resulting network. The developer can then choose to re-train the network with different parameters, save the network to XML/Binary or view the results in the *Results Viewer*.

RESULTS VIEWER

The Results Viewer allows developers to use their domain expert knowledge to validate trained neural networks. Scenarios that were not used for training data are predicted and are viewable in the Results Viewer (Figure 36).

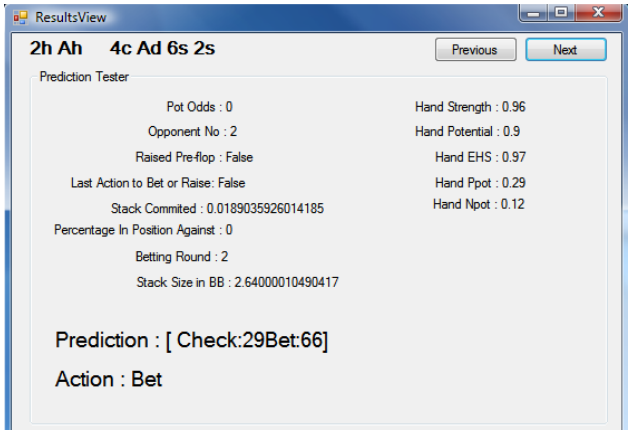

FIGURE 36 - RESULTS VIEW SHOWING PREDICTION THAT THE PLAYER WILL BET

The results viewer allows navigation through the predicted scenarios using the 'previous' and 'next' buttons. The player's hole-cards are viewed at the top of the UI panel, with the board cards on the right. The neural network inputs are viewable in the center with the prediction and actual action at the bottom.

In the example shown in Figure 36 the player in the scenario holds A♥ 2♥ with a board of 4♣ A♦ 6♠ 2♠ and has just turned the 2♠ to make the player two pair, a relatively strong hand but vulnerable to draws and better two pairs. The Pot Odds are at '0' so no player has bet yet at this stage and it can be observed that the player is out of position as the 'Percentage In Position' statistic is at '0'. The neural network has predicted that under these conditions, the player is more likely to bet, and has predicted correctly.

The Results Viewer also offers the ability to test the range predicting ability of the network. The predicted weighted range of the example scenario above is shown below in Figure 37.

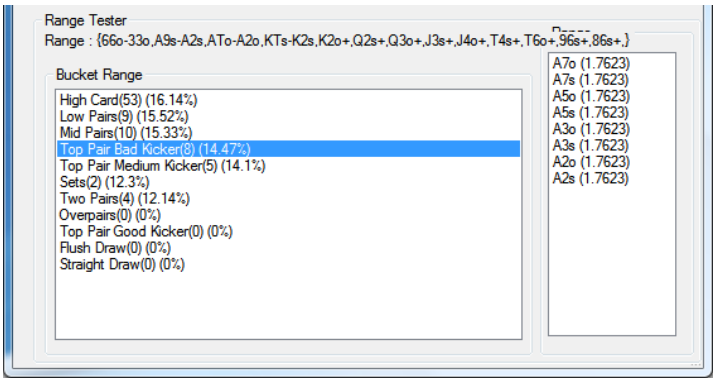

FIGURE 37 - A RANGE PREDICTION FOR THE FIG 35 EXAMPLE

An observant reader would notice that the range prediction in Figure 37 is much different to the proposed weighted range in Figure 22.

The reason for this is that each hand in the entire hand range must be pushed through the trained network. For this to happen, first the hand must be evaluated for hand strength, hand potential, positive potential and negative potential. This limits the effectiveness of the hand

range predictor, as hand evaluation takes around 300ms. Given that there are 169 possible hand combinations this would result in a single range prediction taking 50 seconds.

To solve this problem, a *BucketCollection* was implemented. Since we can assume that players will play some hands in a very similar manner (for example A7 and A8 with a board of A T 2) it is not necessary to evaluate every hand for strength. Instead, *bucket*s were created to encapsulate groups of hands that are likely to be played in similar ways. To achieve better efficiency, one hand in each bucket is calculated for strength and pushed through the network reducing the total time cost to approximately 3 seconds.

In the implementation, buckets only need to be evaluated once per board card drawn – it can preferable to bind a component to 'board card drawn' event thrown in scraper to avoid a three second delay before the agent's decision.

NEURAL SANDBOX

Finally, a component was implemented to aid the developer in debugging, testing and identifying which inputs are most reliable within the neural networks. This component is named the *Neural Sandbox* and allows custom scenarios to be created and pushed through the network (Figure 38).

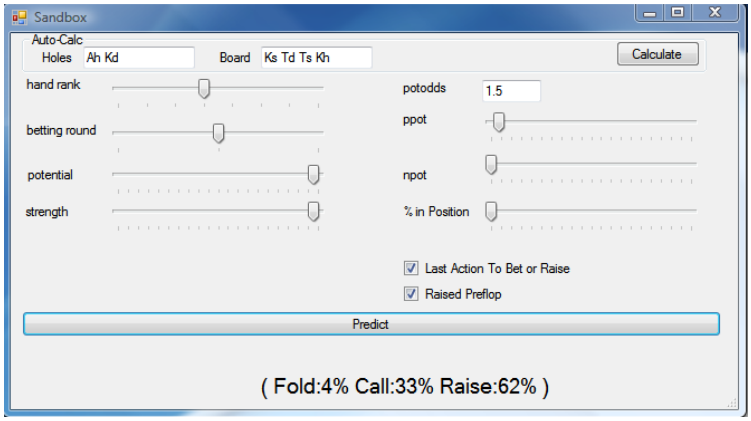

FIGURE 38 - THE NEURAL SANDBOX PREDICTING A RAISE

The developer can explicitly input the hand evaluation values by tweaking the track-bars or can input the hole-cards and board-cards and let the component set them automatically.

In the example shown in Figure 38 a scenario has been created which gives the player a strong hand (full house). The network predicts that the player will Call 33% (for deception) and Raise 62% (for value).

4.2.3 GAME TREE SIMULATOR

The opponent modeler allows the hidden elements of the game to be estimated, but this will not alone support the agent to play a strong game of poker.

The Game Tree Simulator was implemented to allow possible *future* scenarios to be created and simulated for analysis within a current stage of a hand. Predictions of opponents hand distribution and how he is likely to act with those hands at each stage of future action allows equity calculations to be applied to leaf nodes. Equity calculations can then be passed up the tree to give every agent action a value. This value is known as *Estimated Value* (EV) as it is only as accurate as the opponent modeling predictions upon which they are based.

CREATING THE TREE

Firstly, the tree is recursively populated with nodes which are checked to ensure they are within the rules of the game. For example, a node which implies all players have called up to that point is set as a leaf node, as it marks the final action of that stage in the game. A segment of the tree at this stage is demonstrated in Figure 39.

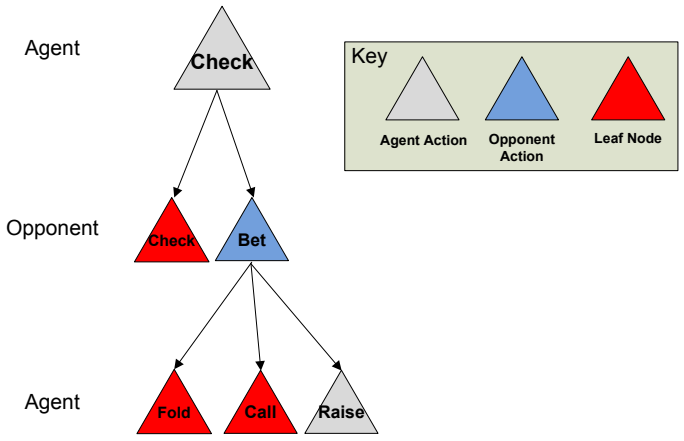

FIGURE 39 – A SEGMENT OF THE POPULATED GAME TREE

Next, opponent nodes within the tree are assigned their weighted hand distribution. Agent nodes are not assigned the distribution as the exact hole-cards are known. This is demonstrated in Fig. 39.

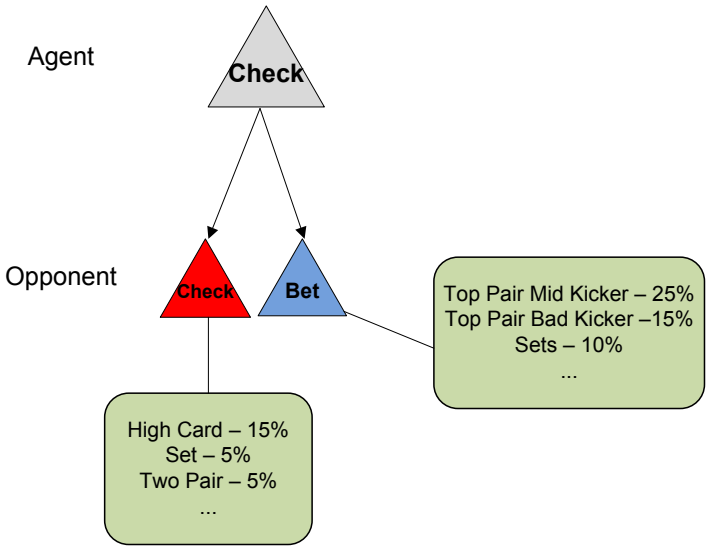

FIGURE 40 – APPLYING WEIGHTED RANGE DISTRIBUTION

Now that the opponent range at each increment of the game tree is assigned, leaf nodes can be calculated for equity. This is achieved by matching the opponent weighted range distribution against the known agent's hole-cards using the hand evaluation algorithms (Figure 41).

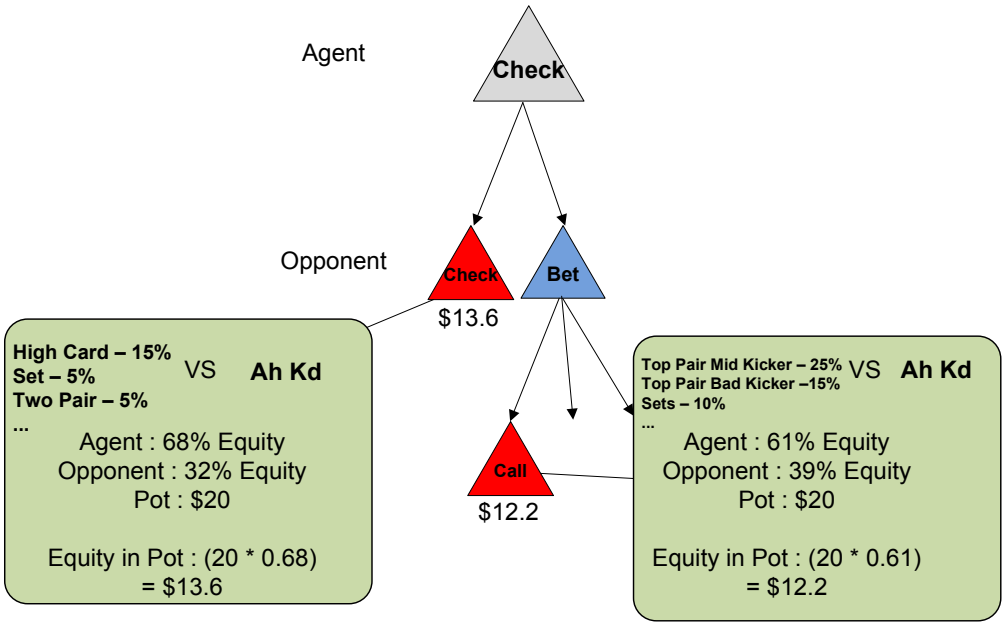

FIGURE 41 - APPLYING EQUITY CALCULATIONS TO LEAF NODES

Penultimately, an action distribution is assigned to each opponent node which is calculated by pushing the opponent's weighted range distribution through the neural networks to determine the likely action of the overall range (Figure 42). For estimated value predictions within agent nodes it is always assumed that the agent will pick the action with the most EV.

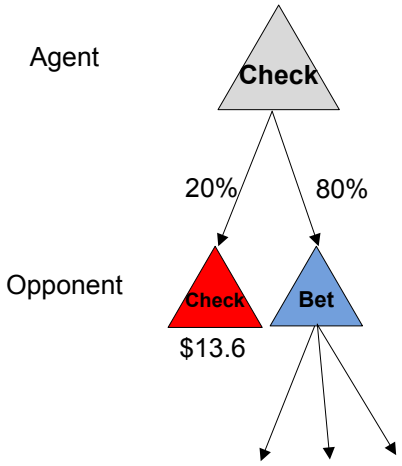

FIGURE 42 - APPLYING ACTION PREDICTIONS TO OPPONENT NODES

Finally, Estimated Value calculations are applied to all the nodes in the tree. This is calculated by weighting the leaf nodes equity calculations with the action distribution. The Estimated Value is passed up the tree for the use of parent node EV calculations (Figure 43).

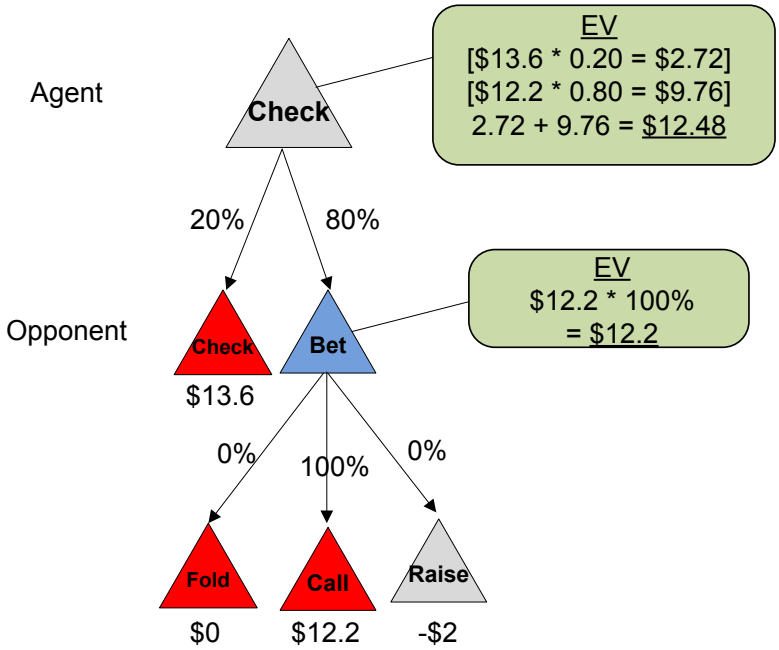

FIGURE 43 -APPLYING ESTIMATED VALUE CALCULATIONS TO PARENT NODES

By constructing the game tree in this way, the Estimated Value calculations can now be used to inform the agent how to act. In the examples given above, if the agent checks – it will expect to obtain a $12.48 value in the hand.

HOW TO USE THE TREE

The referenced game tree library can be initialised by creating a **GameTreePopulator** and applying it to a **GameTree**. The poker table is then passed in to populate the tree, and the resulting EV calculations for each action can be extracted.

```
this.Tree = new GameTree();
GameTreePopulator populator = new GameTreePopulator();
populator.Populate(this.Tree, table);
double[] actionEVs = this.Tree.ActionEVs;
```

The game tree can also be serialised to *Binary* or *XML*. This is useful for examining important hands throughout a poker session. If the agent loses a large hand, the tree can be dumped and de-serialised so the developer can check that the agent made no strategic errors.

VISUALISATION CONTROL

To aid in the debugging of the Game Tree, a visualisation control was produced. The control is demonstrated in Figure 44.

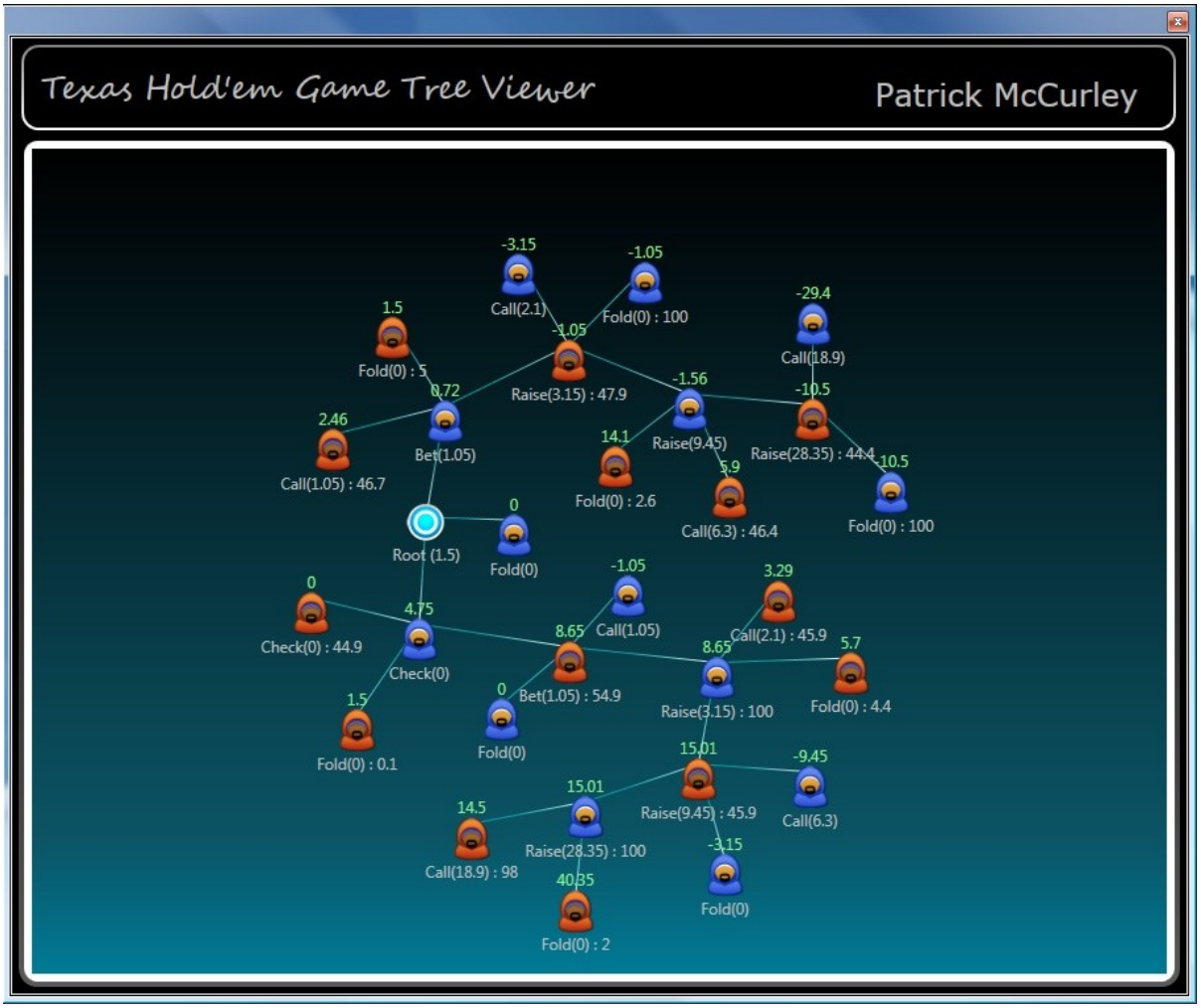

FIGURE 44 - THE GAME TREE VIEWER

The Game Tree Viewer was implemented in WPF using the **Graphite** library, created by Orbifold (Orbifold.net). Graphite was modified to produce the game tree viewer shown above. The game tree works by initialising the library and passing in the game tree in an XML format.

There are several key nodes throughout the visualisation:

Root Node

The root node (Figure 44) is at the root of the tree. In brackets it shows the pot's current value.

FIGURE 45 - THE ROOT NODE

Agent Nodes

The agent nodes (Figure 46) represent the actions that the agent can make throughout the simulation. The green figure above the avatar represents the Estimated Value node, whilst the Action and Action Amount are shown below.

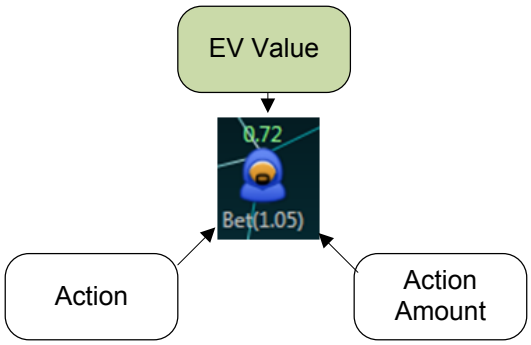

FIGURE 46 - THE AGENT'S NODES

Opponent Nodes

The opponent nodes (Figure 47) represent the actions taken by the agent's opponent throughout the simulation. The action probability is assigned to the left of the action and action amount.

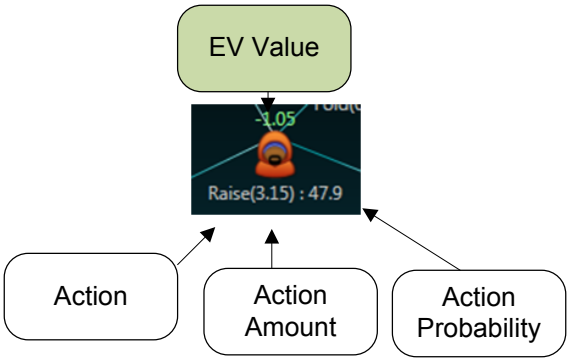

FIGURE 47 - THE OPPONENT'S NODE

When the mouse hovers over an opponent node, the predicted weighted range for that action is displayed as shown in Figure 48.

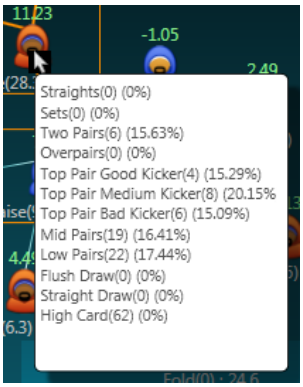

FIGURE 48 - VIEWING AN OPPONENTS NODE WEIGHTED RANGE

GAME TREE SANDBOX

The last debugging tool for the Game Tree is the Game Tree Sandbox which allows custom scenarios to be built for visualisation through the Game Tree Viewer. The Sandbox is demonstrated in Figure 49.

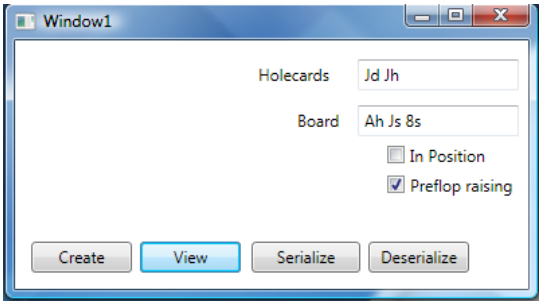

FIGURE 49 - THE GAME TREE SANDBOX

The developer can construct game trees by entering the board, hole-cards, other parameters and clicking 'create'. The tree will then be populated which can then be saved to a Binary or XML format using the 'Serialise' button.

Stored Game Tree's can then be reconstructed by de-serialising through this component.

5. RESULTS

5.1 DATA CLUSTERING RESULTS

FINDING THE CENTRIODS

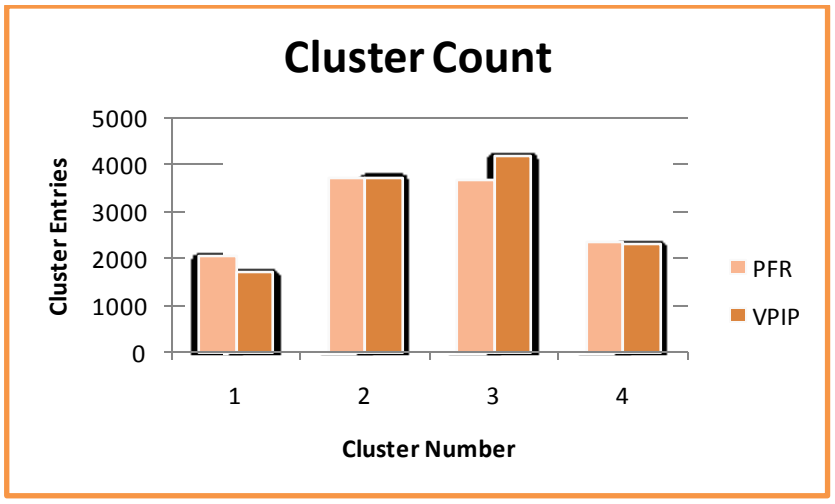

FIGURE 50 - NUMBER OF PLAYERS IN EACH CLUSTER

Figure 50 denotes the number of players in each cluster after the k-means clustering algorithm has been applied to the data,

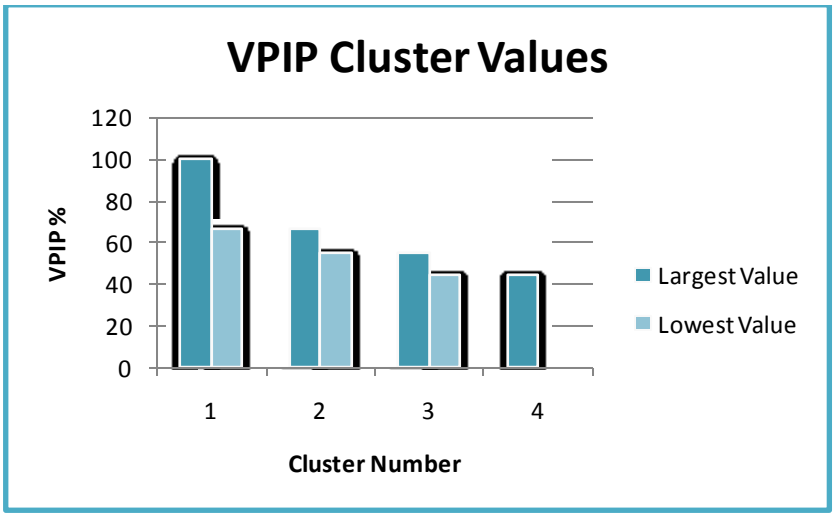

FIGURE 51 - VPIP VALUES IN EACH CLUSTER

Figure 51 denotes the range of 'VPIP' values in each cluster.

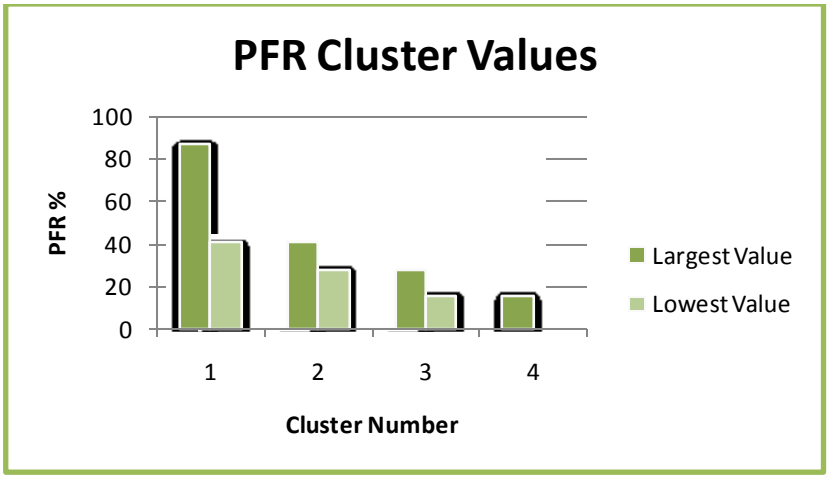

FIGURE 52 - PFR VALUES IN EACH CLUSTER

Figure 52 denotes the range of 'PFR' values in each cluster.

COMBINING INTO PLAYER CATEGORISATION CLUSTERS

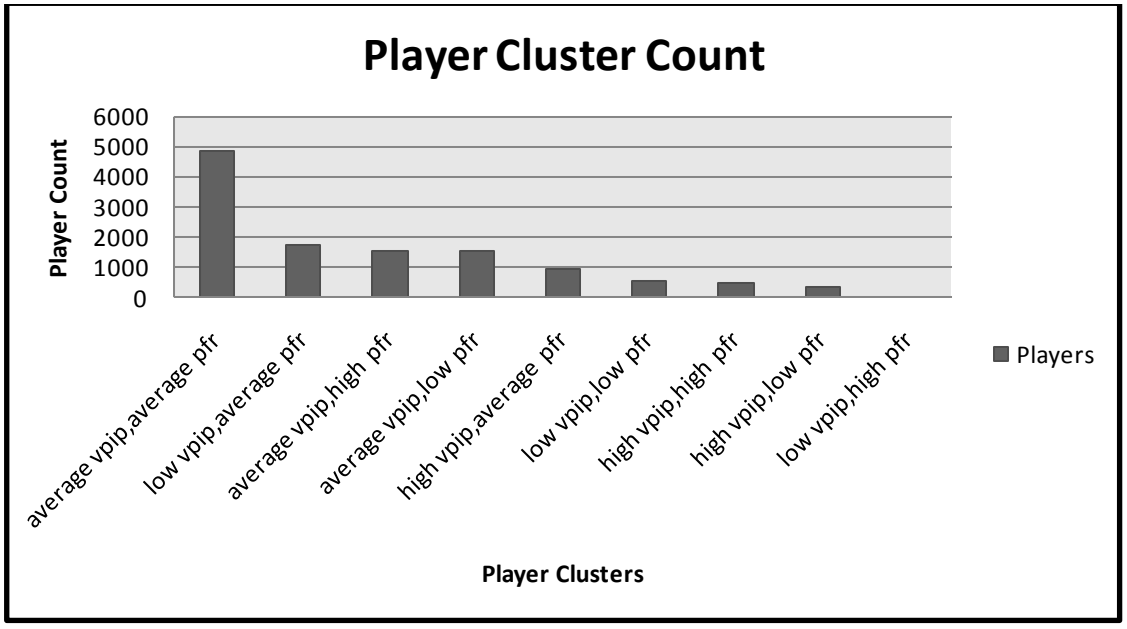

FIGURE 53 - COMBINED VPIP & PFR CLUSTERS

Once the VPIP and PFR Clusters have been combined, the player categorisation clusters have been completed. Figure 53 denotes the number of players in each cluster.

5.2 NEURAL NETWORK RESULTS

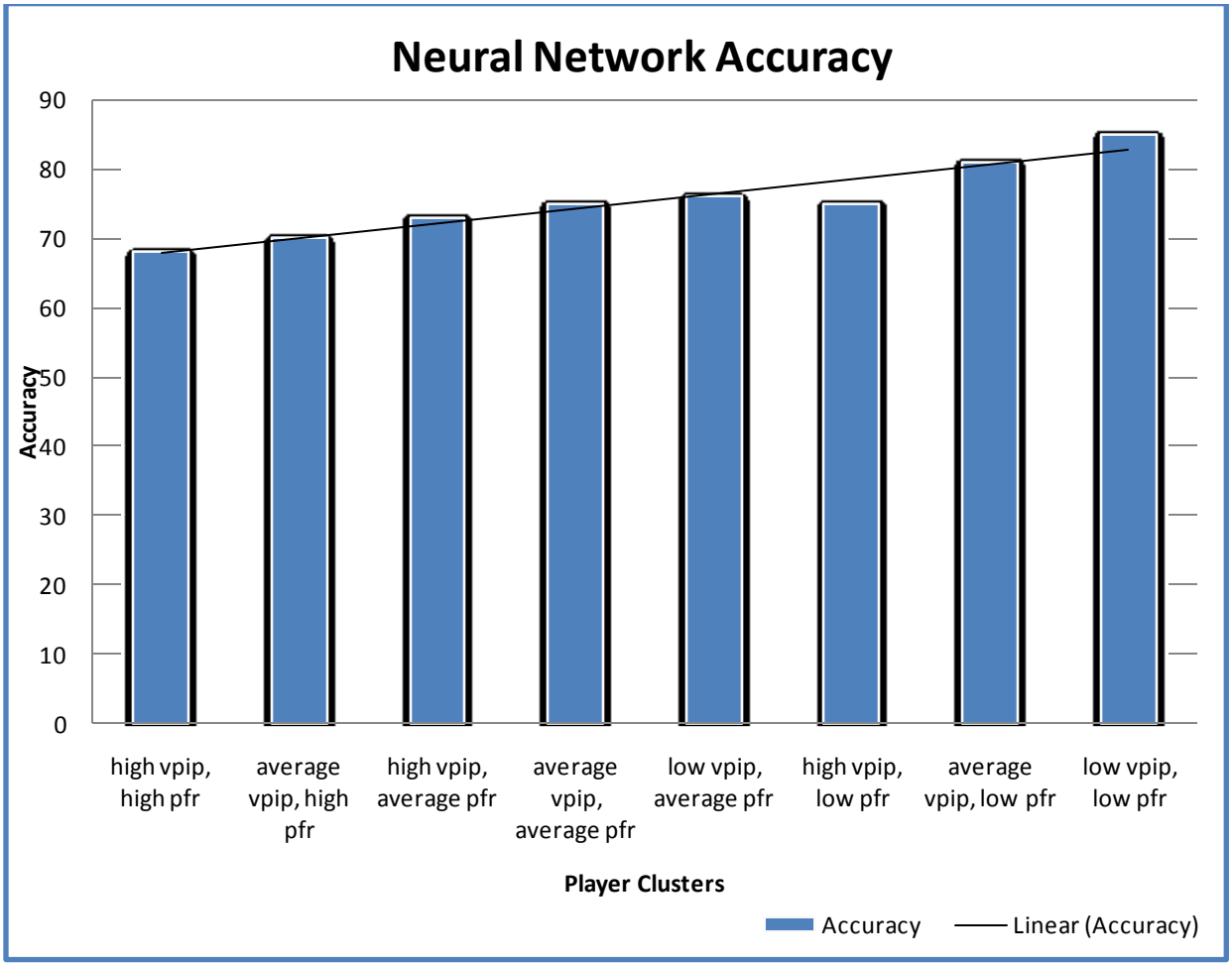

FIGURE 54 - NEURAL NETWORK ACCURACY OF PLAYER CLUSTERS

Figure 54 denotes the accuracy of the neural networks trained against the clustered player types.

5.3 PHASE ONE AGENT RESULTS

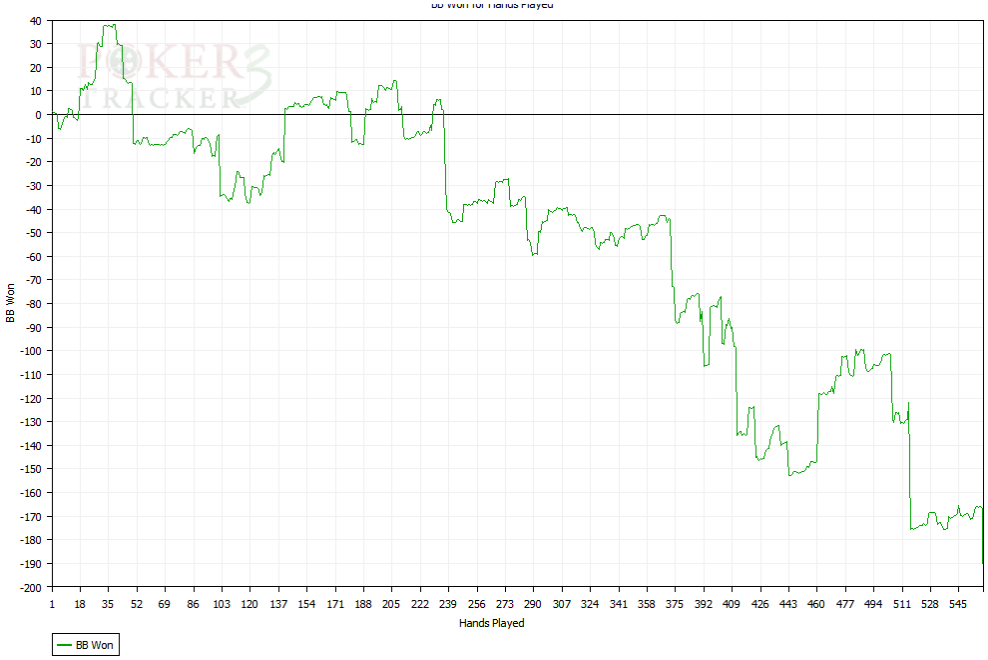

FIGURE 55 - RULE BASED AGENT RESULTS

It can be observed in Figure 56 that for the first 35 hands, the rules based agent started to make a profit. However, this initial gain was countered by a steady decline in the rest of the hands. This can be attributed to the variance associated with the domain.

5.4 PHASE TWO AGENT RESULTS

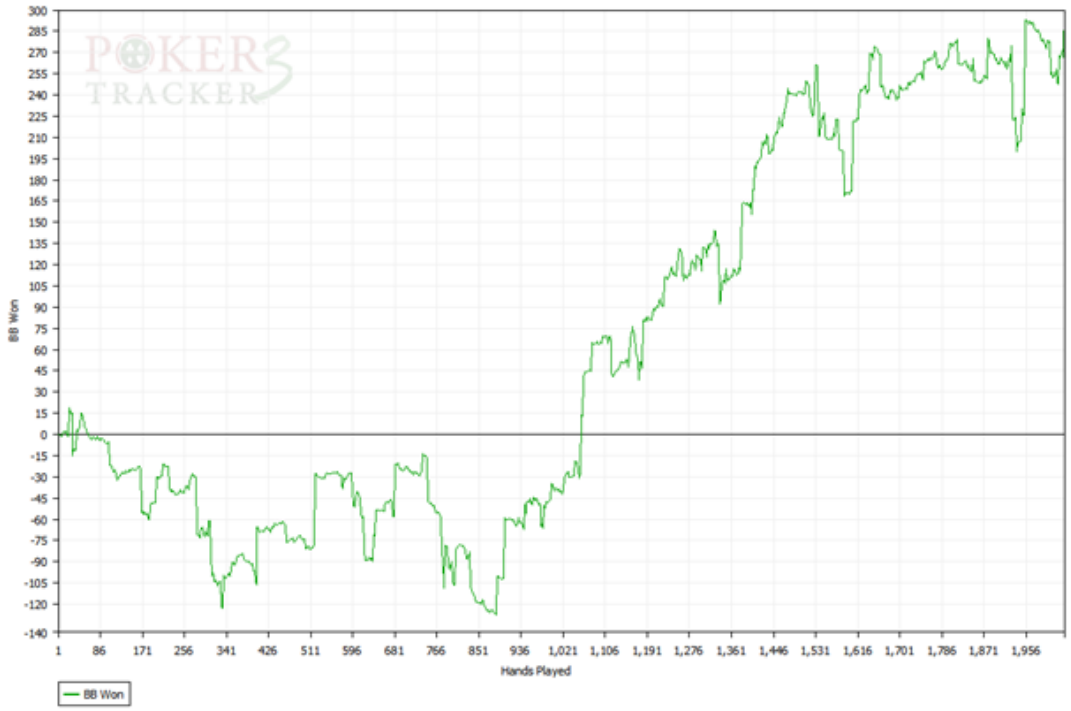

FIGURE 56 - AI AGENT RESULTS

Figure 57 describes the performance of the Phase Two AI agent. The agent started to make a loss over the first 1,000 hands, but ended up making a considerable profit. The initial decline can be attributed to variance associated within the domain.

6. EVALUATION

6.1 RESULTS EVALUATION

6.1.1 DATA CLUSTERING RESULTS

The data clustering manager aimed to provide the solution with a player categorisation function, both for identification and self-training.

The results in Figure 53 confirm that the process was successful, with nine categories of player strategies being created. Figures 50, 51 and 52 demonstrate the results of the K-means algorithm, and how they created the clusters in the data.

Some interesting observations can be made of the results of the clustering.

Firstly, Clusters 2 and 3 represent the average values of both VPIP and PFR. It can be derived from Figure 50 that they hold the most players of the clusters, and Figure 51 and 52 indicates that they also holds the least range from smallest value to biggest.

Secondly, when analysing the results of Figure 53, it can be observed that most of the players fall into the average VPIP and average PFR cluster. A player categorised into this cluster will be treated as possessing the least amount of expected exploitation opportunity in its strategy. However, the other player clusters can be assumed to have a higher potential of being strategically exploitable. 59% of all clustered players fall into this category.

6.1.2 NEURAL NETWORK RESULTS

The neural network results demonstrate the degree of accuracy among the networks trained for each player cluster. Several observations can be made.

Firstly, the least amount of accuracy was attained in clusters regarded as holding volatile strategies. The lowest accuracy was attributed to a cluster with statistics observed as high in both VPIP and PFR. This can be explained by players in this category holding tendencies to play a lot of marginal hands, but by raising more than calling. This type of strategy is hard to model as there will be a much higher percentage of bluffs. In contrast, a strategy low in VPIP and PFR can be modeled more easily, as the players will hold a tendency to only play strong hands, and play them by checking and calling, rather than betting or raising. This would explain the higher network accuracy in this cluster.

6.1.3 RULE-BASED AGENT RESULTS

The rules-based agent performed as expected. The results showed in Figure 56 show that the agent made a steady loss over its testing period, with little variance in the decline.

This can be explained by the very nature of the approach, in which rules can often fail to address the complex scenarios that can arise throughout a poker game. By applying rules to situations that occur, an agent essentially abstracts the game too far to address opponent characteristics efficiently enough to make a consistent or meaningful profit.

That said, it has been known for rule-based strategies to make a small edge in low-stake games, where players make mistakes continuously. Therefore, expanding the rule-based engine to encompass more opponent modeling and complex rules could be worth investigation in further work.

Additionally, the agent was run over 600 hands of testing to provide the results. As discussed in section 2.3 under 'Strategy Implementation and Performance Measurement', the variance associated with poker implies that the results cannot be used an explicit decision to determine if the rule-based strategy is indeed profitable. At this level, the results only *indicate* that the agent is unprofitable.

6.1.4 AI AGENT RESULTS

In contrast to the Phase One Rule-Based agent's results, the Phase Two AI Agent was profitable throughout the testing hands. The results ran over 2,000 testing hands in which agent made a profit of 285 big-blinds.

This can be explained by the approach that was taken in the AI implementation. By applying accurate opponent predictions to the game-tree, the agent was able to evaluate possible outcomes in the hand to reach its decision. In contrast to the rule-based approach, the AI agent could use the information at hand to play strong exploitive poker without abstracting the poker scenarios into more human-digestible terms. This is in exception to the weighted hand distribution bucketing described in Section 4.2.2 under 'Neural Network Manager' in which hand collections were abstracted to provide a low cost overhead.

Although the results were run over 2,000 hands, this is still not enough to surpass the stochastic variance that is applied to the game. Therefore, all indications point towards a profitable implementation of the AI agent, but this cannot be taken for granted.

6.2 PROJECT EVALUATION

6.2.1 ARCHITECTURAL IMPLEMENTATION

The implementation of both solutions was designed through following the *Observer* design pattern to a high degree of success.

The nature of the solution allows plug-ability and scalability through adding additional subscribing processes, reliability through appropriate internal fault tolerance at a component level and enabled data centralisation through the SQL Server 2008 service in which the data components rely on.

The components in the implementation are de-coupled to the extent that they can be easily applied to other domains. Take for instance the scraping manager, in which game-state snapshots are forwarded to the subscribing AI processes. Many other applications could make use of this component in which game-state information can be used, such as real-time poker calculators, or poker advisors such as *Hold'em Inspector*. The game-tree could be applied to other games such as chess or backgammon with little modification and the opponent modeling manager could be applied to any domain requiring predictions in non-deterministic environments.

6.2.2 CALCULATION PERFORMANCE

There are several processes throughout the solution that are costly in terms of CPU consumption and time. This can be critical to the agent operation on the poker client, as there is a usual time-restriction applied to decision making of 15 seconds.

Scraping the poker client

Due to many requests being sent through the external *TextCaptureX* (Section 3.4) library, populating a snapshot of the game state takes an average duration of **488** milliseconds. This is acceptable when the agent is applied to only one client, but in a multi-tabling environment there is a potential for a backlog of scraping requests to slow the response time of subscribing processes.

Hand Evaluation

Hand Potential, *Positive Potential* and *Negative Potential* algorithms rely on large amounts of calculations to return their results. The average duration for each of these algorithms is **320** milliseconds, which can again complicate the solution for a multi-tabling environment.

Game Tree Population

Several processes that are high in computational cost take place when the game tree is populated. Firstly, the hand evaluation algorithms are applied to the opponent weighted range buckets. Secondly, the agents hand is evaluated. Thirdly, the nodes are populated for each action throughout the scenario. This makes the average duration of population **5** seconds per decision which makes multi-tabling impractical. In the scenario where an agent must make decisions on multiple tables, the backlog of computation required for each decision will run over the allocated time to make an action, and will ultimately lose the agent value.

6.2.3 FINAL IMPLEMENTATION

The final implementation of the agent demonstrates the power of Artificial Intelligence techniques within game domains and beyond.

The hand history database initially clustered by *PokerTracker* allowed the agent to learn from domain specific data. By mitigating the expert knowledge throughout the Phase Two solution, the agent effectively learned poker independently and applied its knowledge to make strong poker decisions. This resulted in a more interesting computer science application, but also a stronger agent that is only affected by the evidence to which it is exposed.

In the scenario where player strategies evolve over time, in which new players become more aware of game theory – the agent will be able to adapt and maintain its level of play, as long as neural networks are updated with freshly data-mined and clustered hand samples to construct its beliefs in the game tree.

In light of the success of the implementation, there are still some limitations that exist in the final solution.

Data Clustering Manager

The data clustering manager currently only uses the VPIP and PFR statistic to determine how to create the player clusters. Although these are two good statistics when applied to differentiating a strategy, other statistics could be incorporated to enhance the categorisations. Statistics such as *Aggression Factor* which measures an opponent's aggression, and *Won When Showed Down* which can indicate how strong an opponent is likely to be in an all-in scenario could model the player strategies more effectively.

Hand Evaluation

The hand evaluation algorithms currently deliver exactly what they set out to achieve. This said, the time taken for some of the calculations to complete could be improved significantly. The current multi-threaded implementation allows approximately 2 million hands comparisons per second. Other documented implementations have been known to achieve over 100 times that of an unoptimised hand evaluation solution. (PokerStove Website)

By enhancing the speed of the hand evaluation algorithms, other components that rely on its results will in turn be more efficient, allowing more calculations, leading to more accurate results.

Opponent Modeling

When predicting an opponent's weighted hand distribution, the opponent modeling manager currently *buckets* the hands into collections that can be played in similar ways. (Section 4.2.2) This is implemented to reduce the time taken for the predictions, the main bottleneck coming from the hand evaluation manager. A certain amount of accuracy is lost when abstracting the hand collection as hands at the edges of the cutoff point (for example between top-pair-top-kicker and top-pair-middle-kicker) may be played slightly differently depending on the opponent's strategy.

By evaluating each hand individually, both the weighted hand collection predictions and action predictions over the range will be more accurate, resulting in more accurate EV calculations when applied to the game tree.

Additionally, the neural networks currently incorporate several defined inputs in the predictions. This has lead to an average accuracy across all networks of **75%**. By tweaking the

neural networks architecture and experimenting further with different network inputs the accuracy has the potential to increase, leading to more accurate predictions.

Finally, neural networks were only implemented from the player categories clustered by the Data Clustering Manager. This prevents the agent from adapting to an opponent's changing strategy. In the scenario where an opponent changes strategy throughout the hand, the opponent modeling manager may assign the opponent a new network based on the PFR and VPIP statistics, but this is not sufficient for the agent to adapt to less obvious strategic tendencies in which to exploit.

Ideally, a specific neural network should be trained for every new player that the agent faces. Therefore, when enough hands have been observed, the networks can be trained in real-time to give action and hand distribution predictions specific to that opponent. However, the complications in maintaining a real-time neural network implementation would require substantial investigation.

Game Tree Construction

The current game tree population only evaluates the current point in the hand. This limits the agent to making its decisions based on one round of betting, which is naïve to future actions in the rest of the hand. Take for example a scenario where the agent is constructing a game tree for the flop. The game tree constructs all possible actions that can take place on the flop, but not after the flop. For opposition who may like to call the flop and raise the turn with a weak hand, this will not be addressed in the analysis. Additionally, advanced moves that the agent could make in actions for the rest of the hand will not be evaluated.

By implementing the game tree to take into account *all* possible scenarios until the end of the hand, the agent will be able to make stronger decisions. However, the speed of the game tree would have to be increased dramatically and complications of storing more nodes in memory would need to be addressed.

Additionally, the game tree abstracts betting sizes to fixed amounts. A simulated re-raise will only raise three times the previous bet, a standard assumption, but too abstracted to maximise the value of a decision. Ideally, several different betting size nodes should be created such as *under-bet, value-bet* and *over-bet* to represent better abstractions of predicted bet-sizing. The complication with this is that it would increase the nodes in the game tree exponentially, which could inflict performance implications.

Results

The final results of both the Phase One and Phase Two agent are useful, but not definitive in terms of conclusions. 2,000 hands recorded for testing is not enough to surpass the variance associated within the domain.

To arrive at definitive results, an approach could be taken in which both agents are tested over around 100 sessions, containing 1,000 hands each. The average and standard deviation of the scores could then be calculated and recorded and a *Student t-test* could quantify whether these values were significantly different or not.

This would enable the results to measure the profitability of both agents, surpassing the variance, but also to measure just how much more effective an AI based agent was in comparison to the rule-based agent.

7. CONCLUSION

7.1 OBJECTIVES

The following section provides analysis into which objectives were met, and to what degree.

1. **To investigate the effectiveness of neural networks for opponent modeling predictions applied to this domain.**

   The *Opponent Modeling Manager* created neural networks for opponent action and range predictions successfully in the final solution. Networks were created and trained on thousands of observed players with an average prediction success rate of 75%.
   The neural networks were proven to be very effective when applied to the domain of poker.

2. **To evaluate what factors predominantly affect opponent modeling predictions.**

   The factors which have been shown to affect opponent modeling predictions include a combination of hand evaluation results, game-state properties and table histograms (as shown in Figure 25).
   It is felt that although these properties produced good accuracy throughout the networks, further experiments in input selection could potentially yield better results.

3. **To evaluate the effectiveness of a simulated game tree in modeling the domain to provide quantified decision making values.**

   The *Game Tree Simulator* was implemented to allow the agent to analyse potential outcomes in the current hand. The component provided the agent with decision making values using the *ExpectiMax* algorithm to a high degree of success, as shown in the results.

4. **To contrast the difference in performance between a rule-based and AI-based approach.**

   Figure 56 and 57 contrast the results between the rule-based and AI approach. It is clear that although a rule-based agent is trivial to implement, an AI approach is superior, which is reflected in the results.

5. **To produce a finalised agent, that produces positive results over at least 10,000 hands.**

   The finalised agent produced positive results. However, the agent was only tested over ~2000 hands. This does not surpass the variance associated in the environment, but indicates that the agent is most likely profitable.

7.2 PROJECT REFLECTION

Positive Observations

Firstly, the background literature surrounding the artificial intelligence poker domain is extensive. The University of Alberta has a strong foothold in leading poker AI research, allowing a great deal to be learned from their publications. In respect to this project, a great deal of theory was extracted from background knowledge and applied to this solution. Concepts such as game-tree searching algorithms, hand evaluation algorithms and various opponent modeling concepts are all previously documented research, but nevertheless interesting to implement in the no-limit variant, where little research has been applied.

The implementation of neural networks for opponent modeling predictions was an interesting one. The adaptive nature of the component during training demonstrates its power in operating in noisy environments. There are many domains in which neural networks could be deployed with success, in which they demonstrate the power of computer science solutions for solving complex real-world problems.

By approaching the implementation of this solution in the same way, many skills can be attained.

Firstly, the implementation of the screen scraping component involves low-level Windows API programming which is a necessity for accessing external processes in which the internal structures or processes are unknown.

Secondly, the implementation of hand evaluation algorithms provides an insight into algorithm design in a multithreaded environment where efficiency and optimisation are crucial to providing effective results.

Thirdly, the implementation of the opponent modeling manager gives a great insight into the theory of applying Artificial Intelligence techniques to handle various forms of uncertainty, which is very applicable to other domains.

Finally, the entire solution is a large scale project, complex in various areas of software engineering, whilst specific in the area of Artificial Intelligence. By implementing this project, one can only assume to become a better software developer.

Negative Observations

One of the negative observations in the project was the cost-overheads applied to some processes. The hand evaluation algorithms were effective but non-optimal, which in turn slowed components relying on the evaluation. The time-taken to populate a game-tree also incurred a large cost and will pose problems should the component attempt to be extended. In part this comes down C-Sharp's characteristics, which falls short in comparison to highly optimised programming languages such as C or C++ (Onur Gumus Website). However, there is a large quantity of code in the solution which is unoptimised and if implemented properly would provide more efficient results.

Secondly, another negative observation falls at the game tree. It was originally intended that the game tree would consist of thousands of nodes and would evaluate a hand from the point of play, to the very end in which showdowns occurred. When the game tree was beginning to be implemented it became clear that this would be not possible within the allocated time limit and the game tree design was cut short to only evaluating one betting round per action.

7.3 FURTHER WORK

Data Clustering Manager

In further work, the data clustering manager could be extended to incorporate more statistics in player categorisations. This would allow the neural network implementation to be based on more precise strategies in the environment.

Additionally, the manager could be extended to support automated data-mining of hand samples to keep the player categorisation clusters up to date, and to allow the neural networks to adapt to changing traits in the game.

Opponent Modeling

The opponent modeling manager could be improved by investigating further approaches for predictions in stochastic non-deterministic environments. A Bayesian decision network implementation would provide this level of analysis in which the model determines probabilistic relationships between cause and effect. The network could provide an enhancement over artificial neural networks as beliefs of the domain can be extracted after the training phase. This is in contrast to neural networks, which do not allow the knowledge gained in training cycles to be extracted for further analysis or usage.

The improvement to the opponent modeling would be more substantial if specific opponent modeling models were implemented in real time. Currently, neural networks are trained on generic player categorisations. Specific neural network predictions would provide more accurate results as specific opponent tendencies can be analysed so the agent could adapt to mixed strategies.

Game Tree

As discussed in section 6.2.3, the game tree component could be improved by expanding the number of scenarios that the tree constructs for analysis, which would allow the agent to evaluate more advanced actions within the scenario.

An investigation of additional game theoretic tree algorithms could also be beneficial for an extension of the project. Algorithms such as *Minimax* (where expected value is not maximised as that of *ExpectiMax*, but potential loss is minimised to result in a pseudo optimal solution) could be evaluated and integrated into the solution.

Entire Solution

The entire solution would benefit from optimising the existing code to provide more efficient calculations. By providing hand evaluation results faster, the entire agent would be speeded up, allowing CPU and memory consumption to focus on additional calculations or models.

The solution could also be extended to incorporate more variants of poker, such as “No-limit” and “Pot-limit Omaha”, or extending player number limitations to enable six or ten player tables. Additionally more poker clients could be interfaced, so the solution is exposed to the maximum amount of opponents possible.

Additional Domains

The approach taken in the implementation of this solution is highly applicable to other non-deterministic, stochastic domains high in noise. The game tree implementation is applicable to other game-theoretic solutions, whilst components implemented in the opponent modeling manager could be used predictions in domains such as financial markets, weather or sport.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my dissertation supervisor **Dr. Peter Andras** for his support in the duration of the project. Secondly I would like to thank my second marker **Dr. Jason Steggles** in taking the time to mark the paper.

One of my biggest sources of inspiration and ideas came from regulars of the '**PokerAI**' forums (www.pokerai.org). Not only did they provide a centralised library of useful material and offer an incredible amount of hand history samples in which to apply my solution to, I always felt assured that I could post my ideas or questions knowing that they would be answered by a group of highly intelligent experts in the domain.

A special thanks goes out to **Francios Vanderseypen**, the developer of the Orbifold Graphical Solutions (www.orbifold.net). Francios gave me full access to his commercial Graphite WPF product free of charge, in which I could develop the Game Tree Visualisation control demonstrated in this paper.

Thanks to my relatives and friends for taking the time to proof read the material prior to submission, in particular **Karen Sleat** and **Tom McKee** who provided me with extensive grammatical corrections and computer science inspiration.

Finally, thanks to the **University of Newcastle Upon Tyne** for providing an excellent education and degree in computer science.

REFERENCES

Aaron Davidson, D. B. (2004). Improved Opponent Modelling in Poker. 7.

Billings, D. (2006). *Algorithms and Assessment in Computer Poker.* Ph. D.

CUDA.NET 2.0 Homepage. (n.d.). Retrieved from GASS Ltd Website: http://www.gass-ltd.co.il/en/products/cuda.net/

Davidson, A. (2002). Opponent Modeling in Poker : Learning and Acting in a Hostile and Uncertain Environment.

Davidson, A. (1999). Using Artificial Neural Networks To Model Opponents In Texas Hold'em.

Davidson, A., Billings, D., Schaeffer, J., & Szafron, D. (2002). The Challenge Of Poker. *Artificial Intelligence Journal* , 40.

Finnegan Southey, M. B. (2005). Bayes Bluff : Opponent Modelling in Poker.

GPGPU Homepage. (n.d.). Retrieved from GPGPU: http://www.gpgpu.org/

Johanson, M. (2007). *Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player.* Msc.

Jonathan Schaeffer, D. B. (2000). Learning to play strong Poker.

Kan, M. H. (2007). Postgame Analysis of Poker Decisions. 109.

Michael Bowling, M. J. (2006). Strategy Evaluation In Extensive Games with Importance Sampling. 8.

Miller, E. (2005). *Getting Started in Hold 'em.* Two Plus Two Publishing.

MSDN - Linq.net. (n.d.). Retrieved from http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

Nolan Bard, M. B. (n.d.). Particle Filtering for Dynamic Agent Modelling in Simplified Poker. 7.

Official Cuda Homepage. (n.d.). Retrieved from Nvidia: http://www.nvidia.com/object/cuda_home.html

Onur Gumus Website. (n.d.). Retrieved from http://reverseblade.blogspot.com/2009/02/c-versus-c-versus-java-performance.html

P/Invoke Interop .net Wiki. (n.d.). Retrieved from Pinvoke.net: http://www.pinvoke.net/

Papp, D., Billings, D., Schaeffer, J., & Szafron, D. (1998). Poker as a Testbed for Machine Intelligence Research. *AI'98, The Twelfth Sanadian Conference on Artificial Intelligence* (p. 14). University of Alberta.

PokerEval C# Port Website. (n.d.). Retrieved from CodeProject: http://www.codeproject.com/KB/game/pokerhandevaldoc.aspx

PokerStove Website. (n.d.). Retrieved from http://www.pokerstove.com/

PokerTracker 3 Website. (n.d.). Retrieved from PokerTracker: http://www.pokertracker.com/

Sakai, H. (2005). Internet Poker : Data Collection and Analysis. 52.

TextCaptureX Webpage. (n.d.). Retrieved from Deskperience:
http://www.deskperience.com/screen-scraper/textcapturex.html

Ulf Johansson, C. S. (2006). Explaining Winning Poker : A Data Mining Approach. 97.

Wikipedia - Artificial Neural Network. (n.d.). Retrieved from
http://en.wikipedia.org/wiki/Artificial_neural_network:
http://en.wikipedia.org/wiki/Artificial_neural_network

Wikipedia - Artificial Neural Networks. (n.d.). Retrieved from Wikipedia:
http://en.wikipedia.org/wiki/Artificial_neural_network

Wikipedia - C Sharp. (n.d.). Retrieved from
http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Wikipedia - C Sharp. (n.d.). Retrieved from Wikipedia:
http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Wikipedia - Deep Blue. (n.d.). Retrieved from Deep Blue IBM:
http://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

Wikipedia - Observer Pattern. (n.d.). Retrieved from Wikipedia:
http://en.wikipedia.org/wiki/Observer_pattern

WISC - Neural Network Configurations. (n.d.). Retrieved from
http://pages.cs.wisc.edu/~bolo/shipyard/neural/tort.html

APPENDICES

A. POKER GLOSSARY

***Action*.** Refers to a players action on the poker table. Actions may include folding, checking, calling, betting or raising.

***All-in*.** When a player commits his entire stack of chips or money to the pot.

Aggressive. Refers to a player's style of play. An aggressive player will tend to raise and re-raise instead of calling. These players predominately raise or fold rather than call.

Betting for Value. Betting with the expectation that if the opponent calls you will win more. To bet for value, your hand needs to compare favourably with the probable opponent holdings.

Big Blind. In Texas Hold'em, the player two places to the left of the dealer posts a forced bet called the big blind. This amount is set depending on the stakes being played. The amount of the big blind is usually twice that of the small blind, and equal to a small bet.

Bluff. Making a bet when holding a weak hand that has little chance to win if called. Bluffs are used in balance with bets that are intended to be for value to create uncertainty about the strength of their hand. If a player does not bluff, they will be too predictable and therefore be exploitable to observant opponents.

Board Card(s) Also known as community cards. Refers to the open cards dealt to the table in which players can all observe to make their hands.

Button Also refered to as the 'Dealer marker'. The player with the button is the player just before the small-blind, and has positional advantage over the entire table for the duration of the hand.

Check-raise. A trap play in which a player checks (feigning weakness) and encouraging an opponent to bet, and then raising at the next opportunity. Most often, this indicates a strong hand, but can also be used as a bluff.

Poker Client – The software which hosts the simulated representation of the game in an online format.

Community Card. (AKA a 'Board' card) A public card that all players can use to make their best poker hand.

Flop. (a) The first three community cards dealt in Texas Hold'em. (b) The betting round that commences after the first three community cards have been dealt.

Flush. A poker hand ranked higher than a straight, but lower than a full house. Consists of five cards of the same suit.

***Fold*.** Also known as '*mucking*'. Refers to when a player forfeits the rest of the betting in the hand after a raise or bet has been placed.

Full House. A poker hand ranked higher than a flush, but lower than a four of a kind. Consists of three of kind and a pair (eg. A-A-A-3-3).

Heads-up. To play poker with just two players (either from the outset of the game or at the late stages of a multiplayer game).

Hole Card. A player's private card in poker, unknown to the opponents.

Implied Odds. The expectation of winning bets in future rounds. It considers situations after the future cards are dealt and shows how the situation is better than the immediate pot odds indicate.

Limp. Refers to when a player chooses to call the pre-flop big blind amount without the pot being raised up to that point.

Loose. Refers to a player's style. A loose player plays comparatively more hands than a tight player.

Passive. Refers to a player's style of play. A passive player will tend to check or call other player's bets rather than raise themselves. These players predominately call or fold rather than raise.

Pot. The collection of all the betting amounts on the table at that point in time.

Pot Equity. A measure of how valuable a current hand is. It is a share of the current pot with regard to the likelihood that the player wins the pot. To calculate pot equity, we determine the percentage chance that the player wins the pot against the opponent, and assign that player the same percentage of the pot as his pot equity.

Pot Odds. The ratio of money in the pot to the money required to call. This ratio is often used to decide whether it is correct to call a bet with a weaker hand. If the hand can improve to the likely best hand with probability better than the pot odds offered, then the call is possibly correct. Pot odds are often influenced by implied odds and reverse implied odds.

Pre-flop. Refers to the betting round before the flop has been dealt.

***Post-flo*p.** Refers to the bettings rounds that extend during and after the flop has been dealt. This includes the flop, turn and river.

Hand Protection When a player has a strong hand that is vulnerable to the opponent making a stronger hand when future cards are dealt, it is usually a good idea to bet to give the opponent the chance to fold, rather than giving them a free card.

Raise. When a player increases the current bet size for that betting round.

Range. A collection of hands usually consisting of one or more hole-cards. This term is commonly used when an oppositions hands are not known, and can only be estimated.

Reverse Blinds. A fairly common way of playing heads-up Texas Hold'em. The dealer posts the small blind and the other player posts the big blind. In the pre-flop betting round, the dealer acts first and then acts last in all remaining rounds. This is reversed from the normal method of having the player to the left of the dealer post the small blind.

Reverse Implied Odds. The expectation of losing bets in future rounds. It considers situations after the future cards are dealt and shows how the situation is worse than the immediate pot odds indicate.

River. (a) The fifth community card dealt in Texas Hold'em. (b) The final betting round after the fifth community card is dealt.

Slowplaying. To play a strong hand as though it were weak by checking or calling until a later round. Used to mix up a strategy, or as a value play with the expectation of winning more later in the hand.

Small Blind. In Texas Hold'em, the player immediately to the left of the dealer (unless playing heads-up with the reverse-blinds format) posts a forced bet called the small blind. This amount is set depending on the stakes being played. The amount of the small blind is typically half of the big blind.

Starting hand. In Texas Hold'em, the first two hole cards a player is dealt.

Straight. A poker hand ranked higher than three of a kind, but lower than a flush. Consists of five cards of differing suits in sequential order (eg. 3-4-5-6-7).

Straight Flush. The highest ranking poker hand. Consists of five cards of the same suit in sequential order (eg. 3-4-5-6-7, all hearts).

Tight. Refers to a player's style. A tight player plays comparatively fewer hands than a loose player.

Trips. When a player makes three of a kind. In Texas Hold'em, trips is when the player makes three of a kind using two community cards. In contrast, making a ***set*** refers to only using one community card to make three of a kind (when holding a matching pair in their hole cards).

Turn. (a) The fourth community card dealt in Texas Hold'em. (b) The betting round after the fourth community card is dealt.

Weak. Refers to a player's style in which they are very *tight* and have a low tendency to commit chips to the pots they are involved in.

B. POKER RULES

The following poker rules are provided by Wikipedia under 'Texas Hold'em Poker'. If required, additional resources may be helpful such as:

- Poker-Listings website - http://www.pokerlistings.com/poker-rules
- Flop-Turn-River website : http://www.flopturnriver.com/

OBJECTIVE

In Texas hold 'em, like all variants of poker, individuals compete for an amount of money contributed by the players themselves (called the pot). Because the cards are dealt randomly and outside the control of the players, each player attempts to control the amount of money in the pot based on the hand the player holds.

The game is divided into a series of hands or deals; at the conclusion of each hand, the pot is typically awarded to one player. A hand may end at the showdown, in which case the remaining players compare their hands and the highest hand is awarded the pot; that highest hand is usually held by only one player, but can be held by more in the case of a tie. The other possibility for the conclusion of a hand is when all but one player have folded and have thereby abandoned any claim to the pot, in which case the pot is awarded to the player who has not folded.

The objective of winning players is not winning every *individual* hand, but rather making mathematically correct decisions regarding when and how much to bet, raise, call or fold. By making such decisions, winning poker players maximize long-term winnings by maximizing their *expected utility* on each round of betting.

BETTING STRUCTURES

Hold 'em is normally played using small and big blind bets – forced bets by two players. Antes (forced contributions by all players) may be used in addition to blinds, particularly in later stages of tournament play. A dealer button is used to represent the player in the dealer position; the dealer button rotates clockwise after each hand, changing the position of the dealer and blinds. The *small blind* is posted by the player to the left of the dealer and is usually equal to half of the big blind. The *big blind*, posted by the player to the left of the small blind, is equal to the minimum bet.

When only two players remain, special 'head-to-head' or 'heads up' rules are enforced and the blinds are posted differently. In this case, the person with the dealer button posts the small blind, while his/her opponent places the big blind. The dealer acts first before the flop. After the flop, the dealer acts last and continues to do so for the remainder of the hand.

The three most common variations of hold 'em are *limit* hold 'em, *no-limit* hold 'em and *pot-limit* hold 'em. In no-limit hold 'em, players may bet or raise any amount over the minimum raise up to all of the chips the player has at the table (called an all-in bet). The minimum raise is equal to the big blind. If someone wishes to re-raise, they must raise at least the amount of the previous raise. For example, if the big blind is $2 and there is a bet of $6 to a total of $8, a raise must be at least $6 more for a total of $14. If a raise or re-raise is all-in and does not equal the size of the previous raise, the initial raiser cannot re-raise again. This only matters of course if there was a call before the re-raise.

PLAY OF THE HAND

Play begins with each player being dealt two cards face down, with the player in the small blind receiving the first card and the player in the button seat receiving the last card dealt. (Like most poker games, the deck is a standard 52-card deck, no jokers.) These cards are the player's *hole* or *pocket cards*. These are the only cards each player will receive individually, and

they will only (possibly) be revealed at the showdown, making Texas hold'em a closed poker game.

The hand begins with a "pre-flop" betting round, beginning with the player to the left of the big blind (or the player to the left of the dealer, if no blinds are used) and continuing clockwise. A round of betting continues until every player has folded, put in all of their chips, or matched the amount put in by all other active players.. Note that the blinds are considered "live" in the pre-flop betting round, meaning that they contribute to the amount that the blind player must contribute, and that, if all players call around to the player in the big blind position, that player may either check or raise.

After the pre-flop betting round, assuming there remain at least two players, the dealer deals a flop, three face-up community cards. The flop is followed by a second betting round. This and all subsequent betting rounds begin with the player to the dealer's left and continue clockwise.

After the flop betting round ends, a single community card (called the turn or fourth street) is dealt, followed by a third betting round. A final single community card (called the river or fifth street) is then dealt, followed by a fourth betting round and the showdown, if necessary.

C. PRE-FLOP SIMULATED ROLL-OUTS RESULTS

Title : Call Only Preflop Sim
Date : 03/12/2008 20:24:29
Description : Call-only showdown equity simulation

Pockets : AAo 85.14%	Pockets : K5s 55.55%	Pockets : 95s 45.63%
Pockets : KKo 82.22%	Pockets : K7o 55.33%	Pockets : 87o 45.45%
Pockets : QQo 79.63%	Pockets : A3o 55.28%	Pockets : J3o 45.26%
Pockets : JJo 77.2%	Pockets : K4s 54.71%	Pockets : J2s 45.09%
Pockets : TTo 74.57%	Pockets : Q7s 54.57%	Pockets : 22o 44.98%
Pockets : 99o 71.08%	Pockets : T9s 54.48%	Pockets : 85s 44.79%
Pockets : 88o 68.24%	Pockets : K6o 54.47%	Pockets : Q2o 44.79%
Pockets : AKs 67.67%	Pockets : J8s 54.45%	Pockets : 96o 44.6%
Pockets : AQs 66.41%	Pockets : K3s 54.03%	Pockets : T5o 44.11%
Pockets : AKo 66%	Pockets : Q8o 53.69%	Pockets : 75s 43.88%
Pockets : 77o 65.74%	Pockets : J9o 53.58%	Pockets : 86o 43.57%
Pockets : AJs 65.72%	Pockets : K5o 53.33%	Pockets : 94s 43.4%
Pockets : AQo 64.79%	Pockets : Q6s 53.21%	Pockets : 65s 43.35%
Pockets : ATs 64.79%	Pockets : A2o 52.89%	Pockets : T4o 43.17%
Pockets : AJo 63.85%	Pockets : J7s 52.87%	Pockets : 93s 42.86%
Pockets : KQs 63.84%	Pockets : K4o 52.65%	Pockets : 84s 42.86%
Pockets : KJs 63.22%	Pockets : 33o 52.4%	Pockets : 76o 42.69%
Pockets : A9s 63.04%	Pockets : T8s 52.34%	Pockets : 95o 42.58%
Pockets : ATo 62.73%	Pockets : Q5s 52.22%	Pockets : T3o 42.54%
Pockets : A8s 62.51%	Pockets : Q7o 51.83%	Pockets : T2s 42.48%
Pockets : 66o 62.39%	Pockets : Q4s 51.63%	Pockets : 74s 41.8%
Pockets : KTs 61.72%	Pockets : K2s 51.48%	Pockets : J2o 41.52%
Pockets : KQo 61.42%	Pockets : J8o 51.46%	Pockets : 85o 41.29%
Pockets : KJo 61.05%	Pockets : 98s 51.45%	Pockets : 54s 41.07%
Pockets : A7s 60.92%	Pockets : K3o 51.44%	Pockets : 94o 40.92%
Pockets : A9o 60.88%	Pockets : T9o 51.36%	Pockets : 75o 40.91%
Pockets : QJs 60.82%	Pockets : Q3s 51.2%	Pockets : 64s 40.9%
Pockets : KTo 60.13%	Pockets : Q6o 50.73%	Pockets : 83s 40.58%
Pockets : A8o 59.95%	Pockets : J6s 50.56%	Pockets : 92s 40.14%
Pockets : A6s 59.92%	Pockets : T7s 50.5%	Pockets : 73s 39.86%
Pockets : K9s 59.89%	Pockets : Q5o 50.1%	Pockets : 65o 39.79%
Pockets : QTs 59.7%	Pockets : J7o 50.05%	Pockets : 93o 39.71%
Pockets : A5s 59.51%	Pockets : J4s 49.42%	Pockets : 84o 39.4%
Pockets : 55o 59.15%	Pockets : T8o 49.41%	Pockets : 63s 39.15%
Pockets : A7o 59.08%	Pockets : Q4o 49.41%	Pockets : 53s 39.07%
Pockets : A4s 58.36%	Pockets : J5s 49.36%	Pockets : T2o 38.69%
Pockets : QJo 58.34%	Pockets : 97s 49.16%	Pockets : 74o 38.47%
Pockets : A3s 58.21%	Pockets : T6s 48.58%	Pockets : 43s 38.21%
Pockets : K9o 58.17%	Pockets : K2o 48.36%	Pockets : 82s 37.92%
Pockets : K8s 58.02%	Pockets : 98o 48.08%	Pockets : 64o 37.65%
Pockets : JTs 57.99%	Pockets : T7o 48.08%	Pockets : 54o 37.49%
Pockets : QTo 57.69%	Pockets : 87s 47.82%	Pockets : 83o 37.12%
Pockets : K7s 57.62%	Pockets : Q3o 47.69%	Pockets : 73o 36.44%
Pockets : Q9s 57.59%	Pockets : J6o 47.63%	Pockets : 92o 36.12%
Pockets : A5o 57.44%	Pockets : J3s 47.5%	Pockets : 63o 35.9%
Pockets : A6o 57.23%	Pockets : T5s 47.29%	Pockets : 52s 35.69%
Pockets : K6s 56.41%	Pockets : J5o 47.22%	Pockets : 53o 35.25%
Pockets : A4o 56.31%	Pockets : 96s 47.05%	Pockets : 62s 34.74%
Pockets : K8o 56.24%	Pockets : Q2s 47.02%	Pockets : 72s 34.72%
Pockets : 44o 55.98%	Pockets : 86s 46.47%	Pockets : 43o 34.65%
Pockets : J9s 55.86%	Pockets : T4s 46.45%	Pockets : 82o 33.67%
Pockets : Q8s 55.75%	Pockets : T6o 46.32%	Pockets : 42s 33.65%
Pockets : Q9o 55.75%	Pockets : J4o 46.26%	Pockets : 32s 32.44%
Pockets : JTo 55.67%	Pockets : 97o 46.21%	Pockets : 72o 31.4%
Pockets : A2s 55.67%	Pockets : T3s 45.86%	Pockets : 62o 30.89%
	Pockets : 76s 45.81%	Pockets : 52o 30.81%
		Pockets : 42o 29.98%
		Pockets : 32o 28.98%