CONTENTS

1 Description of the problem

Two groups have been assigned the task of developing an automated computerised poker playing system adhering to the rules of the "Texas Hold'em" variation of poker. This variation of the game is known to be quick to learn and fun for beginners but still offers great scope for advanced players to exhibit their skill. Texas Hold 'Em is now recognised as the worlds most popular Poker variant, apparent by its adoption by the World Series of Poker held in Las Vegas and on the popular "Late Night Poker" television series on Channel 4.

We must test our system by playing against opponents on-line, through a poker game-playing interface hosted by Paradise Poker [1]. There will be a one-hour trial in which our system will be run against our opposing group's system on the same online table. The winner will be the group that finishes with the most virtual money. As an added incentive, a cash prize will be awarded to the group whose system performs best on the day.

There are rules that must be adhered to by both teams and failure to comply with them may result in disqualification from the tournament and forfeiture of the cash prize. Both teams must start with the same number of chips (Virtual money) and play during the same given time period allocated on the day.

The poker system must be autonomous and the only acceptable input to our system is game status information. We may inform the system of the cards we hold, our position and number of players at the table and any other information visible on the Paradise Poker screen. At no time during the contest must we be able to intervene with our system's behaviour and we must play the suggestion decision regardless of quality. In the event of application failure, the current hand must be folded.

The final aspect of the problem is to produce a graphically pleasing solution, ensuring ease of use and clarity of presentation.

2 Relevant background information

2.1 Texas Hold 'Em

(Refer to [9] for glossary)

Texas Hold'em is a popular variation on the better-known straight poker that can involve as many as ten players in any one hand. To ensure that there is always money to be won, the two players sat to the dealer's left put an agreed amount into the pot at the start of each game. These are called the small blind and the big blind.

Each player is dealt two cards that may be examined before they are asked to make one of the three standard poker calls – to fold, call or bet. Starting to the left of the player that made the big blind, betting continues left until everyone has called, with a maximum of three raises being allowed.

There follows three rounds where new cards are dealt face up in the centre of the table. After each set of new cards there is a round of betting. The first set of cards to be dealt is called the flop and three are shown. There then follows the turn and finally the river where only one card is shown in both cases. It is the player's aim to make the best possible 5-card hand from the two cards they hold and any of the five cards on the table. The combinations to be made are standard poker hands such as a pair, a flush and a straight [10.1]

2.2 Game playing in Computer Science

Game playing has played an important role in the development of artificially intelligent systems in Computer Science. In the 1960s and 70s there were high hopes for Artificial Intelligence – many of which were overly optimistic. One realm in which this optimism has, it seems, not been misplaced is in the field of artificially intelligent game-playing, where there have been significant advances over the last thirty years, many of which have led to better designs and implementations in fields such as chip design, parallel processing and programming algorithms.

Perhaps the most well known use of Artificial Intelligence in game playing has been in chess. Claude Shannon [Shannon 1950] presented his ideas about computerised application of chess – the search size of a game and how the computer could search for the next move from within a game. These ideas were used to develop, over the following forty years a number of A.I. chess players, such as MacHack [2], which successfully competed in human tournaments and Belle, which attained expert status from the United States Chess Federation [3].

The pinnacle of computerised chess-play, and perhaps computerised game-play overall, was Deep Blue's defeat of the then Chess World Champion Gary Kasparov over a six game series. Deep Blue was developed at Carnegie Mellon University, beginning in the mid-1980s. It moved to IBM in 1989 and over the course of the next decade was steadily improved until the defeat of Kasparov in 1997. Deep Blue did, of course have millions of dollars of funding from IBM. But the techniques learned from this project have led directly to advances in chip-design and parallel processing. Indeed the project has been so successful that it has been upgraded to the $100 million Blue Gene Project which, it is hoped, will be used to develop a machine operating at 1 quadrillion floating-point operations per second [4].

Other important uses of Artificial Intelligence in game playing have been in Checkers and Backgammon. For Checkers learning techniques were being used as long ago as the early 1950s. In 1950, Arthur Samuel wrote the first checkers program on an IBM 701 computer. Samuel re-wrote the program in 1954 for an IBM 704 computer. This simple program remains one of the most important in Artificial Intelligence history, as it was the first to be able to learn it's own evaluation function. The best checkers program yet developed has been Jonathan Schaeffer's Chinook, developed in the early 1990s. In 1994 Chinook became the official world champion [G5AIAI: Game Playing].

Backgammon has been among the most successful in incorporating Artificial Intelligence techniques. The current state of the art is TD-Gammon, which was developed in the late 1980s and early 90s. It uses a neural network, which utilises a

searching algorithm to search for possible moves the opponent could make. TD-Gammon has attained expert status from the World Backgammon Federation.

(Excerpts from [5] and [6])

2.3 Poker modelling in Artificial Intelligence History

Poker has a long history in Artificial Intelligence gaming. Von Neumann and Morgenstren worked on a two-player version of poker on their research on economics in the 1940s [7]. During this research they recognised that unlike games such as chess, poker is a game of incomplete information and requires a unique approach (see table below).

General Application Problem	Problem Realisation in Poker
Imperfect knowledge	Opponent's hands are hidden
Multiple competing agents	Many competing players
Risk management	Betting strategies and their consequences
Agent modelling	Identifying and exploiting patterns in opponents play
Deception	Bluffing and varying style of play
Unreliable information	Taking into account your opponents deceptive plays

Taken from [10]

There are two ways that poker can be used as an interesting test bed for Artificial Intelligence research. One approach is to use simplified variants that are easier to analyse. For example, Findler worked on and off for 20 years on a poker-playing program for simplified 5-card draw poker [8]. He modelled human cognitive processes and built a program that could learn. The danger with this approach is that simplification can remove the challenging components of the problem that are of interest to AI researchers. A variant of this approach is to look at a subset of the game, and try to address each component in isolation. Several attempts have been made to apply machine-learning techniques to individual aspects of poker.

The second approach, and the one that is advocate by the GAMES research group, is to tackle the entire problem: choose a real variant of poker and address all the considerations necessary to build a program that performs at a level comparable to that of the best human players. Clearly this is the most ambitious approach, but also the one that promises the most exciting research opportunities.

In recent times there have been a number of research groups working on automated Poker play. One of these, the team of Luigi Barone and Lyndon While, recognised that there are four main types of poker players that can be distinctively modelled – Loose Passive, Loose Aggressive, Tight Passive and Tight Aggressive. They suggested the use of evolutionary strategies as a way of developing an adaptive poker player. This has been used to show that players that use evolutionary strategies can adapt to loose and tight play. Ultimately Barone and While developed an evolutionary, adaptive player, which reportedly outperforms component statistics-based models by a significant margin [9].

The most important of the groups researching automated poker has been the University of Alberta's GAMES (Game-playing, Analytical methods, Minimax search

and Empirical Studies) Research Group, headed by (the indomitable!) Jonathan Schaeffer. The ideas developed by this on-going research group have led to the release of Loki, arguably the strongest automated poker-playing program to date. Loki uses probabilistic knowledge to decide the action to take depending on the current game state (fold, call or raise). The group is also looking at opponent modelling in determining actions to take, based on the fact that different players have different playing-styles. Our own automated player has been heavily dependent on the work done by the GAMES group. In particular, it incorporates the essential elements of poker play [10]:

2.3.1 Hand strength

Assesses how strong your hand is in relation to the other hands. At a minimum, it is a function of your cards and the current community cards. A better hand strength computation takes into account the number of players still in the game, position at the table, and the history of betting for the hand. An even more accurate calculation considers the probabilities for each possible opponent hand, based on the likelihood of each hand being played to the current point in the game.

2.3.2 Hand potential

Assesses the probability of a hand improving (or being overtaken) as additional community cards appear. For example, a hand that contains four cards in the same suit may have a low hand strength, but has good potential to win with a flush as more community cards are dealt. At a minimum, hand potential is a function of your cards and the current community cards. However, a better calculation could use all of the additional factors described in the hand strength computation.

2.3.3 Betting strategy

Determines whether to fold, call/check, or bet/raise in any given situation. A minimum model is based on hand strength. Refinements consider hand potential, pot odds (your winning chances compared to the expected return from the pot), bluffing, opponent modelling and trying to play unpredictably.

2.3.4 Bluffing

Allows you to make a profit from weak hands, and can be used to create a false impression about your play to improve the profitability of subsequent hands. Bluffing is essential for successful play. Game theory can be used to compute a theoretically optimal bluffing frequency in certain situations. A minimal bluffing system merely bluffs this percentage of hands indiscriminately. In practice, you should also consider other factors (such as hand potential) and be able to predict the probability that your opponent will fold in order to identify profitable bluffing opportunities.

2.3.5 Opponent modelling

Makes it difficult for opponents to form an accurate model of your strategy. By varying your playing strategy over time, opponents may be induced to make mistakes based on an incorrect model.

2.3.6 Unpredictability

Allows you to determine a likely probability distribution for your opponent's hidden cards. A minimal opponent model might use a single model for all opponents in a

given hand. Modifying those probabilities based on collected statistics and betting history of each opponent may improve opponent modelling.

2.4 Loki

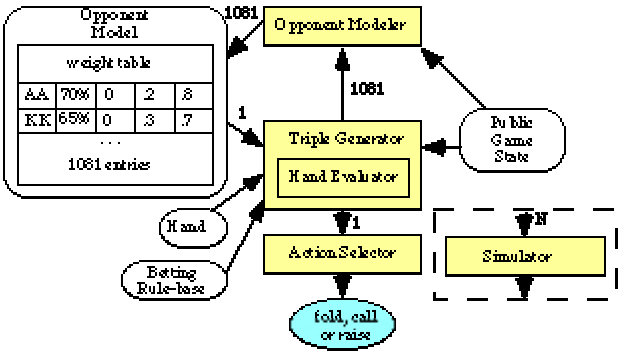


Figure 1

Of all recent academic attempts at creating an intelligent Poker player, none is greater than that of *Loki* [11]. Figure 1 illustrates how the important components of Loki's architecture interact. In the diagram, rectangles are major components, rounded rectangles are major data structures, and ovals are actions. The data follows the arrows between components. An annotated arrow indicates how many times data moves between the components for each of our betting actions.

Several components of *Loki* have been implemented in *Texem* and are covered in [The decision-making classes5.3]. The remainder of this section concentrates on those features not covered.

The architecture revolves around generating and using probability triples, which appeared in [12]. It is an ordered triple of values, PT = [f,c,r], such that f + c + r = 1.0, representing the probability distribution that the next betting action in a given context should be a fold, call, or raise, respectively. The Triple Generator contains poker knowledge, and is analogous to an evaluation function in two-player games. The Triple Generator calls the Hand Evaluator to evaluate any two-card hand in the current context. It uses the resulting hand value, the current game state, and expert-defined betting rules to compute the triple. To evaluate a hand, the Hand Evaluator enumerates over all possible opponent hands and counts how many of them would win, lose or tie the given hand.

Each time it is Loki's turn to bet, the Action Selector uses a single probability triple to decide what action to take. For example, if the triple [0.0,0.8,0.2] were generated, then the Action Selector would never fold, call 80% of the time and raise 20% of the time. A random number is generated to select one of these actions so that the program varies its play, even in identical situations. Although this is analogous to a mixed strategy in game theory, the probability triple implicitly contains contextual information.

After the flop, the probability for each possible opponent hand is different. For example, the probability that Ace-Ace hole cards are held is much higher than the cards 7-2, since most players will fold 7-2 before the flop. There is a weight table for each opponent. Each weight table contains one value for each possible two-card hand that the opponent could hold. The value is the probability that the hand would be

played exactly as that opponent has played so far. For example, assume that an opponent called before the flop. The updated probability value for the hand 7-2 might be 2% since it normally should be folded. Similarly the probability of Ace-King might be 60% since it would seldom be folded before the flop, but is often raised. After an opponent action, the Opponent Modeller updates the Weight Table for that opponent in a process called re-weighting. The value for each hand is increased or decreased to be consistent with the opponent's action. The Hand Evaluator uses the Weight Table in assessing the strength of each possible hand, and these values are in turn used to update the Weight Table after each opponent action. The absolute values of these probabilities are of little consequence, since only the relative weights affect the later calculations.

Loki has over recent years been incrementally improved. The techniques it uses for opponent modelling and betting strategies have been evolved and overhauled and are still in a continual process of improvement. More recent success has been achieved at The University of Western Australia, where the strategy for creating an adaptive poker player concentrates on evolutionary algorithms [13].

2.5 Online poker

The recent popularity of the Internet has allowed a growth in on-line gaming. More specifically there has been a recent surge in the number of websites that run on-line gambling, including poker. In these sites players can log in and exchange credit card details for virtual chips and play in multi-player poker games. There are several online poker rooms including:

- Paradisepoker.com
- Planetpoker.com
- Pokerroom.com
- Ultimatebet.com

In each case, online poker is played against people from around the world for money or just for fun. Paradise Poker is the world's largest and well-known Internet poker room. In contrast, planetpoker.com is the first website to have taught users how to play poker on the Internet.

Paradise Poker does not necessarily provide the user with the most graphically pleasing or challenging experience. Yet by using Paradise Poker in the specification we are bound to make decisions relating to choice of interface, graphical representation and style of play. Paradise Poker has a number of rooms devoted to Texas Hold 'em. Many contain real money play tables yet people can also play for virtual money. Reasons for choice could include:

a) **Popularity** – Paradise Poker is the most recognised site in online poker.

b) **Action** – It is the most active poker room in the world. It has 24-hour poker action around the world and about 2000 players are logged in during peak times. Tables are availably constantly.

c) **Security** – Often a customer's first priority. Paradise Poker offers the best security due to the SSLv3/TLSv1 encryption algorithm, collusion [9]

detection methods and random shuffling techniques. What this means is that cards, names, addresses, credit card numbers, passwords, and everything else that goes back and forth is protected using the same level of security that a bank would use. Packet sniffing by other players cannot be used to gain any sort of advantage. Numerous procedures are reviewed and updated regularly to maintain this high level of security.

d) **Connection Stability** – the game server runs for 24 hours a day, for 365 days a year and if there are any disruptions or instability Paradise Poker rectifies the problems efficiently and quickly and are up and running in no time.

e) **Support** – The support team can be reached by e-mail.

f) **Graphics** – As stated at Paradise Poker: "From the beautiful, lobby montage to the elegant table setting, our players enjoy sitting in our live, online, five-star poker paradise."

g) **Multiple Features** – One of the distinct features is that the player can play various games at one time. The various features make the playing experience enjoyable and the most memorable. It is the most easily to navigate.

h) Paradise Poker's main advantage, as we shall see later, lies in a **text box** that displays the history of the current game.

3 Design overview of the proposed system

Early on in the design process, the project took on a codename: Texem. This name has been retained in the final release version and the project may be referred by this name in the remainder of the report.
There are several alternative strategies we could use in development. We could opt for a robust deterministic set of procedures, for instance. Alternatively the development could lean towards an evolving system of play that takes the design of play strategy away from the developer. Consideration must be given to the duration of development and testing, to the ease of use, robustness and performance of each solution.

3.1 Decision-making

Crucial to our system being successful is the inclusion of playing styles and strategies. Advanced poker players learn styles of playing poker that maximise the amount of money in the pots that they win, whilst reducing the amount of their money in the pots that they lose. Our system must try and emulate a human player in this way and adopt a successful style of play. There are tricks such as bluffing and opponent hand prediction that may assist the success of the system.

After research, the team decided that the most efficient way to approach the problem would be to study research already done in this field in the past. The Computer Science department at the University of Alberta, Canada, have programmed and put together Loki, a fully functional, Artificially Intelligent learning program. We studied the implementation of this and discovered that we were able to download and

integrate some of the classes into our program. It has been decided that our program will consist of a central main method and GUI, which will call methods from the Loki classes.

The decision-making element of the program is in itself a large part of the project and needs breaking down into programming stages. The first stage is to use deterministic decision classes. These decision classes can then be tweaked to develop a unique style of play, one that is most suitable to online poker. Some of the techniques have been taken from [14].

3.2 Robustness

The system must be able to accept several types of user input and be able to cope with erroneous typing. The system must implement a series of routines that return a move suggestion. As part of this function the system must be robust and be able to provide some form of default move suggestion in the event of an error in the program. An exception, for example, could force the move suggestion to fix at "fold", thereby minimising virtual money loss. This feature does not violate the rules of competition.

3.3 GUI specification

While the solution may be complicated, the graphical user interface need only deal with a finite set of inputs and is required to return only one of three possible results. There are two major design issues that must be considered when developing the GUI:

- Allow fast and accurate input of the game state.
- Provide results as clearly and as concisely as is possible.

There are many ways of achieving fast and accurate input. One solution could be for our GUI to mimic the layout of a play table in Paradise Poker and provide suitable controls to input game state data. This has obvious advantages of clear representation, would allow accurate data input and would take less time to implement. Unfortunately, this solution may be slow to operate in the small window of time afforded to each player by Paradise Poker. The site has tight limits on the amount of time available to players to make their decisions. The GUI must provide seamless operation for input and output and not introduce unnecessary delays that prevent the team from tracking play.

During early proposals it was decided that the GUI needed six distinctive elements included. In each round of Poker, it must be possible to enter:

i. The number of players in the round, along with positions with respect to each other on the table.

ii. The position of our player on the table and that of the dealer for the current round

iii. The first two cards (pocket cards) dealt to our player.

iv. The community cards must be entered in turn (flop, river, turn).

v. The decisions of each of the players (i.e. call/check). Decision order must be maintained as to not violate rules.

vi. The decision made by our 'player' must be outputted to the user in each round and must be constantly on.

4 The Graphical User Interface

The Graphical User Interface design is an element that is imperative to the success of uniting the A.I work previously done with online poker. To best demonstrate the features that are required, this section will cover the strategies that have survived in the final product and depict the evolution of the GUI.

4.1 Visual Basic

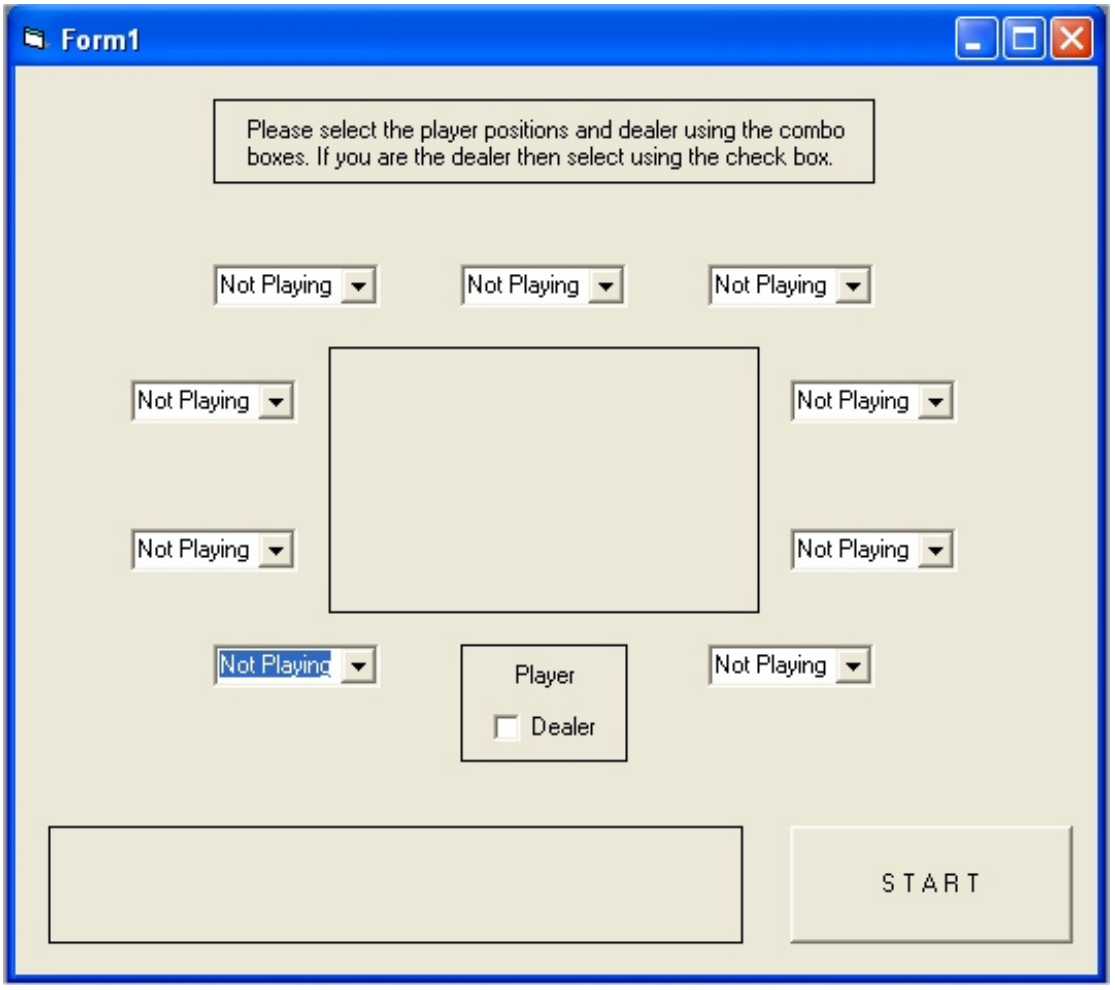


Figure 2: Visual Basic design

The team initially decided to design a template for the GUI using Visual Basic. The advantage is that the point and click GUI building ability both fast and less taxing in comparison with Java.

The first task was design of the aesthetic look of the GUI. Several possible designs were created and those with features found to best satisfy the specification have been retained in the final release. Using this design as a template a working GUI was created using Visual Basic [Figure 1]. The first screen of this GUI allows the user to

input in information for [3.3] sections (i) and (ii). The position of the players on the table was set out to match that of those on the Paradise Poker table. For each of these players there was a combo-box menu where the user chooses whether this particular player is playing in the game, if it is the position of our player, or if it is the dealer. This was later deemed a cumbersome and time-consuming entry method.

The second part of the GUI allows the entry of the first two cards dealt to our player. The initial design of this was to have four graphical select boxes. The first two of these allows the user to click on the suit of each of the first two cards (hearts, clubs, spades, diamonds). The second two allow the user to click which cards within the chosen suits have been dealt. Once this information has been selected, a 'Start play' button is pressed whereby the actual decision making process began.
Originally, the GUI was slow and made poor use of the remaining screen space once Paradise Poker was running. The Visual Basic attempt contained unnecessary circular layout of players, which was simply not practical when using limited available screen space. In addition, the use of Java would allow the team to more easily tap into the benefits offered by the Alberta code.

4.2 Swing

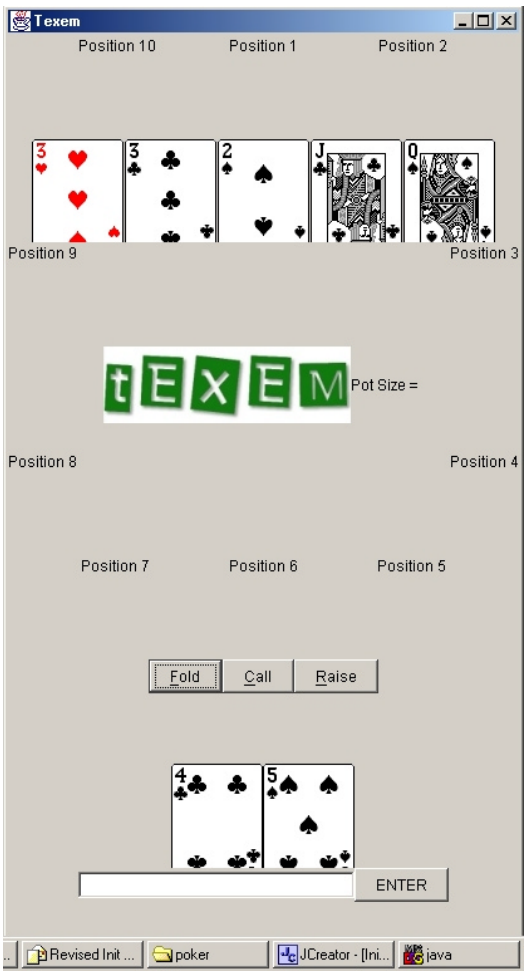


Figure 3: An attempt with Java

Figure 3 demonstrates the early Swing ideas in play. Improved ideas such as text-box processing rather than combo-boxes have survived into the final release, whereas others have been modified or removed:

- Text boxes allowed sections [3.3iii] and [3.3iv] of our GUI specification to be satisfied with simple and fast keyboard input.
- The card images on the right were GPL but looked less professional and were visually dissimilar to those in Paradise Poker. A custom made reverse of the card was created to carry the Texem logo.
- Swing's FlowLayout lead to wastage of space.
- The text box only processed card information and it was intended that we would enter player positions in here. In reflection this now seems an inefficient entry method.
- Buttons were to be used to enter each player's moves. We now realise it would have been almost impossible to keep up with the pace of the game using these very manual GUI elements.

4.3 Interfacing with Paradise Poker

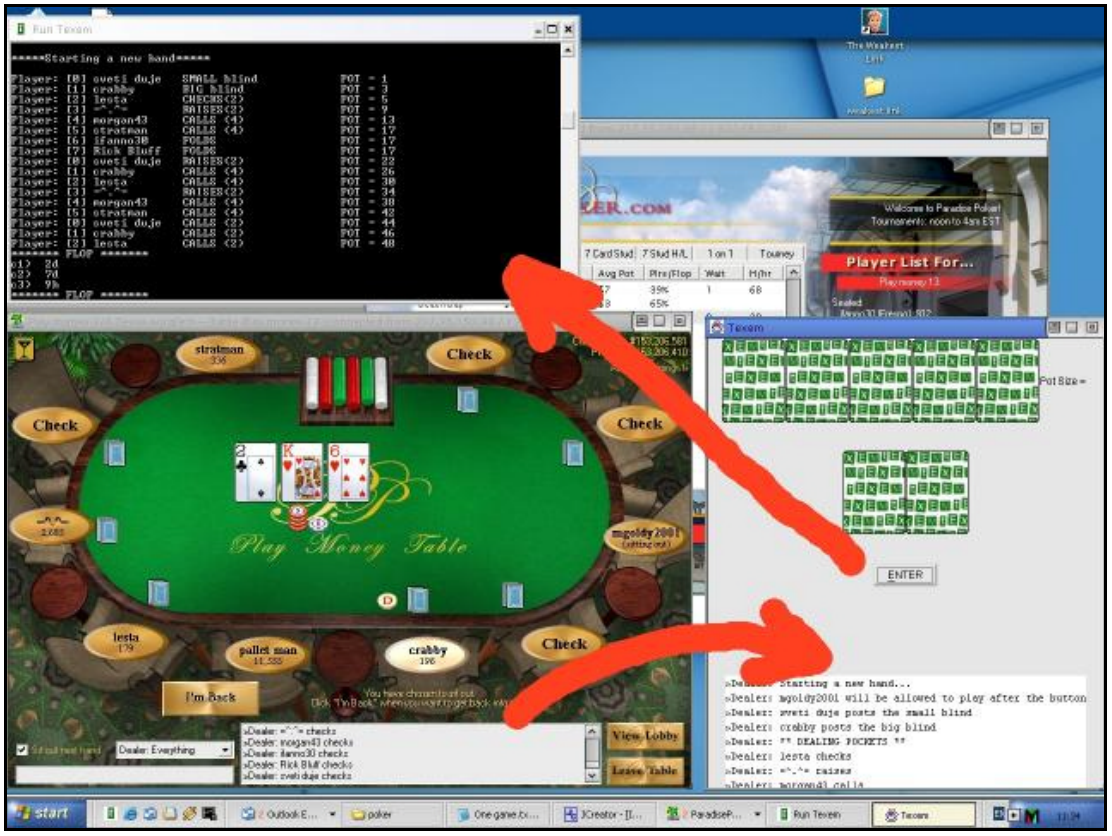


Figure 4: Tracking the game using copy and paste

Paradise Poker has a small 5-line text box that simply keeps a record of each move and stage progression in the game [Figure 4]. The next attempt required a large amount of coding to isolate specific sub strings in the output [readPPtext.java]. The screenshot shows attempts to parse the text entered into a text area.

- Code from earlier work that generated random cards was used to simulate the progression of each stage, so development was not hindered.
- A text field was used and the user would have to repeatedly paste text from Paradise Poker. This was later deemed to be too manual and highly wasteful of time.

- Code was difficult to debug as it was possible to copy arbitrarily large amounts of text and often such text was illegal (e.g. to proceed to the turn before proceeding to the flop is illegal).

4.4 SnagIt screen-grabbing technology

Figure 5: Using screen captures to automate the capture

The project took a change of direction when the team discovered a piece of software that allowed automatic capture of a selected region on screen. **SnagIt** [15] increased the efficiency of data processing to a point where the only user intervention was the input of cards. The decision to create code that processed text had been a good one, in that essentially most of the code was already complete and would need to be modified to read from a text file rather than a GUI text box. A continuous read-file loop scans a fixed filename and location and allows the project to run with minimal user intervention. The code is similar to that used in [16].

- At this stage the only buttons required were those to initialise and interrupt a continuous read-file loop and one to reset the application.
- The program was now simply mirroring and tracking events in Paradise Poker. Players' moves were now entered automatically, rather than the slow burden of pressing buttons and maintaining the order of the presses. This satisfied section [3.3v] of the GUI specification.
- Texem kept track of the number of players and their **relative** order. This satisfied section [3.3I] and [3.3ii] of our GUI specification. Texem cannot deduct which player is acting as dealer until the flop stage [6.4].

- Space was now becoming an issue and so the smaller cards used in Paradise Poker were included.
- ReadPPtext.java compares two consecutive text captures and detects overlap. It processes only the new commands.

Consult the section [10.2.3] for details on the SnagIt configuration settings.

4.5 Layout

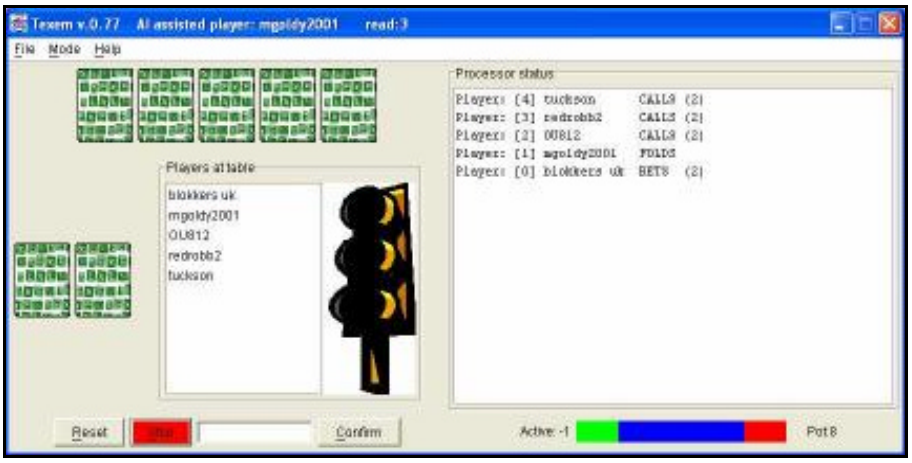


Figure 6: GridBag Layout allows efficient use of space

Using a new layout manager made it possible to minimise use of space. `GridBagLayout` is regarded as the most versatile yet complex layout manager in Java [17]. This was especially important as it allows flipping between Paradise Poker and the Texem with ease to be able to see the cards being dealt.

- At this stage the GUI incorporated a small graphical bar that represents the ratios of folds, calls and raises for each particular game. This provides a useful reference to the type of play of the game. (i.e. "tight" game would signal a large proportion of green).
- A file menu was added to allow additional features to be incorporated without compromising on low special requirement.
- Several gif images of traffic lights have been added that allow a quick and visual suggested move from the A.I engine. This image changes within the loop to give a current suggested move at all times and its accuracy improves with each piece of additional data parsed. With this, the final section of our GUI specification 3.3(vi) was satisfied. See Figure 6.
- A splash screen had been added to attach some form of identification to the developers. This used [18] as a template, which required modified dimensions and delays.

4.6 Testing and debugging with the GUI

Figure 7: The final Graphical User Interface

Our initial plans for a GUI were mainly to increase usability. We have come across several obstacles and our ideas evolved into something much more useful. The final GUI reduced space usage down to a minimum and the more important elements were moved to the top of the screen.

- Debugging the main code has been made easier by using a visual representation of events. Using minimal area also allows the file contents window (upper right hand side of Figure 7) to be viewed, which has been of incredible value when catching exceptions and skipping erroneous text captures. The text mark-up features of JTextPane were most versatile and simple. The features were modified from an example from Sun [19].
- A constantly updating selection bar represents the active players visually. This gives a quick reference as to the names and relative positions of the players and whether they are still active. The selection process was created adapting methods used in Sun's ListDemo.java [20]

4.7 GUI evaluation

Now complete, the GUI allows:

- Almost effortless program operation
- Fast and accurate input of game data.

In addition to meeting the requirements set forth in the specification:

- A fully interactive display that refreshes itself regularly with updated information.

- The display closely resembles that of the tables at Paradise Poker, making it easier to follow the game.
- Minimal screen space has been used.

For instructions on how to use the Texem's GUI, consult [10.2]

5 Implementation and class interaction

There will always be changes and decisions made that affect aspects of a program's development, requiring new time estimates and modification of modules that were previously considered complete. While the target of developing an automated computerised poker playing system has remained the same, our design for a solution has undergone many radical changes.

The May deadline of the poker tournament was the biggest limiting factor imposed on our project's development. By this stage all programming and testing was required to be finished. We began the project with many innovative ideas to improve the strategic play of our system, methods of play that we hoped would provide our system with an edge over our opponents. Although we were well aware of the time constraints placed upon the project, we decided to initially develop a basic poker playing system yet allow later extension. This section concentrates on the implementation to date.

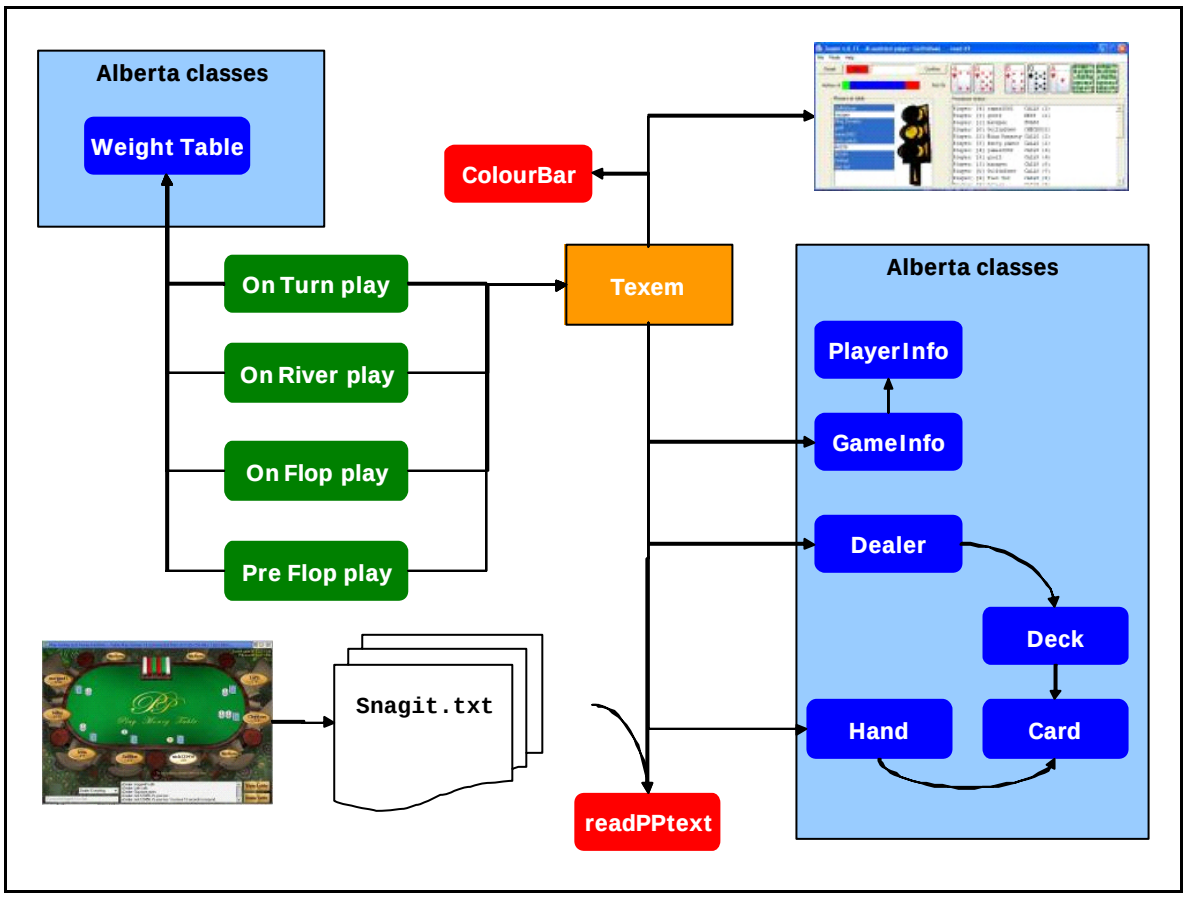


Figure 8: Interaction between Texem and Loki classes

The interaction diagram shown in [Figure 8] depicts the four major classes created by the team (green, orange and red) and the Alberta classes with which Texem classes directly interface. Essentially, the process works in the following way:

1. Any change in the state of play updates the Paradise Poker textbox
2. This is captured at a frequency of **once per second** by **SnagIt** and is saved to a file with fixed name.
3. Texem is launched and starts up the GUI.
4. A continuous read-loop is initialised in `Texem` that calls `readPPtext` each second.
5. Depending on the text, `readPPtext` calls specific methods in `Texem`.
6. `Texem`, in turn will call on the game-state classes and the statistical evaluation classes created by The University of Alberta and modified for our use.
7. The state of play is updated in the GUI, including `colourBar`
8. A move is suggested by the graphical traffic light

5.1 The Alberta classes

The Alberta classes are heavily documented and there is no need to produce extensive commentary of the code. Should this project be developed further in the future, it would be necessary to understand the interaction with the classes developed at Nottingham.

Card.java: Represents a playing card

Deck.java: A Deck of 52 Cards, which can be dealt and shuffled

Dealer.java: This is a standard dealer class for managing a Hold 'Em game.

Hand.java: Stores a Hand of Cards (up to a maximum of 7)

GameInfo.java: Useful Hold 'Em Constants

PlayerInfo.java: Stores all of the information for a player during a poker game. Also contains references to a player's past history of games.

WeightTable.java: A Weight Table of all possible 2-card combinations.

5.2 Software and hardware

5.2.1 Hardware

No special hardware was required for the project. All that is required is a PC that is connected to the Internet. This enables play on the Paradise Poker tables.

5.2.2 Java

The major reason for our choice of Java was the ability to easily interface with the existing Alberta code. Java was also the major language studied by group members.

We have chosen to use the latest version of Java: Java 2 Platform Standard Edition version 1.4. This has been improved by Java Sun Microsystems Inc in terms of its

speed performance. The new version has improved the existing API's in the previous versions of Java. It now has the ability to be used on machines that have large CPU's and have a larger memory space. One of the reasons that we used this version of Java was due to its faster and efficient way to look and call up methods, making our program faster and be run within the time limits set by Paradise Poker. After deciding to use Java's Swing early on, we are now in the position of being able to see the clear advantages of Java Version 1.4. Statistics show that the performance of the GUI has improved by 75%. This has positive implications for an application such as ours, where intensive processing operations are sustained.

5.2.3 SnagIt

This application has allows Texem to run with minimal user intervention. There are few, if any competitive products in the market place that could satisfy our very precise screen capturing requirements. However, an alternative piece of software may allow Texem to overcome the limitation specified in [4.4]

5.3 The decision-making classes

The design of the decision-making element of the program requires consideration of decision-making during a real game of poker. These stages have therefore been broken down into four main categories, which have been programmed into separate classes.

- Pre-flop (after the first two private cards are dealt)
- On the flop (after the next three community cards are dealt)
- On the turn (after the fourth community card is dealt)
- On the river (after the fifth community card is dealt).

These classes were therefore named `PreFlopPlay`, `onFlopPlay`, `onTurnPlay` and `onRiverPlay` respectively.

5.3.1 The "pre-flop play" decision class

This class decides a move based purely on the first two private cards dealt. The main Texem method calls the `PreFlopPlay` decision class using the following line of code:

```
String action =
PreFlopPlay.PlayDecision(p1,p2,gInfo.getNumActivePlayers(),
                         relPosition, gInfo.getPot(),
                         gInfo.getAmountToCall(ourPos));
```

The call to `PreFlopPlay` returns a String to the variable "action", in the format of either; "fold", "call" or "raise". The arguments passed incorporate:

- Two **card** objects p1 and p2 (these cards hold values of rank and suit)
- The number of **active** players at the table (calculated when called)
- The **relative** position (position integer relative to the dealer)
- The total amount of money in the **pot** (integer of total bets placed in hand at point of call)
- The amount of money required for a call.

According to Winning Low Limit Hold'em, 'Playing Considerations before the Flop' [21]:

"Your decision to call, raise or fold before the flop must be based on several factors. Among the most important are:
Your cards
Your position
Your relative position
How much money you must invest initially
The number of players in the hand"

A `PlayDecision` method exists within the `PreFlopPlay` class and acts as the main method inside the decision class. It is from within here that calls are made to other methods. The `PlayDecision` method is declared as below:

```
public static String PlayDecision(Card p1, Card p2, int totPlayers,
int position, int potSize, int betAmountToCall) {
```

The pre-flop decision is largely deterministic and has been extensively researched by Sklansky and Malmuth [22]. For the initial two cards, there are {52 choose 2} = 1326 possible combinations, but only 169 distinct hand types. For each one of the 169 possible hand types, a simulation of 1,000,000 poker games was done against nine random opponents. This produced a statistical measure of the approximate *income rate* for each starting hand. A pair of aces had the highest income rate; a 2 and 7 (of different suits) had the lowest income rate for a 10-player simulation. [10]

5.3.2 Hand strength

The following constructors are needed in order to evaluate the strength of the hand:

```
Hand myHand = new Hand();
Deck myDeck = new Deck();
poker.ai.WeightTable ourTable = new poker.ai.WeightTable();
double handStrength = ourTable.handStrength(p1, p2, myHand);
```

"Enumeration techniques can provide an accurate estimate of the probability of currently holding the strongest hand. For example, suppose our starting hand is A ♦-Q ♣ and the flop is 3 ♥-4 ♣-J ♥. There are 47 remaining unknown cards and there are {47 choose 2} = 1,081 possible hands an opponent might hold. To estimate hand strength, the enumeration technique gives a percentile ranking of our hand. We simply count the number of possible hands that are better than ours (any pair, two pair, A-K, or three of a kind: 444 hands), how many hands are equal to ours (9 possible remaining A-Q combinations), and how many hands are worse than ours (628). Counting ties as half, this corresponds to a percentile ranking, or hand strength (HS), of 0.585. In other words there is a 58.5% chance that our hand is better than a random hand. This measure is with respect to one opponent but can be extrapolated to multiple opponents by raising it to the power of the number of active opponents. Against five opponents with random hands, the adjusted hand strength (HS5) is .5855 = .069. Hence, the presence of additional opponents has reduced the likelihood of our having the best hand to only 6.9%." [10]

5.3.3 Hand Potential

Sometimes, cards which prove to have a low hand strength may potentially be useful later in the game. Two new calculations are useful:

- **Positive potential** (ppot) – of pulling ahead when we are behin
- **Negative potential** (npot) – of falling behind when we are ahead.

These are calculated by enumerating all combinations of cards that could leave ahead, level or behind the best hand. The method call is as follows:

```
poker.ai.HandPotential.ppot(p1, p2, myHand, ourTable, true);
```

Where true signals where we want one or two-card look ahead (slower).

5.3.4 Betting strategy

Hand strength and hand potential are further combined into the **effective hand strength (EHS):**

$$EHS = HS_n + (1 - HS_n) \times Ppot$$

Equation 1

where HS_n us the adjusted hand strength for *n* **active** opponents and Ppot is the positive potential.

The EHS can then be compared to specific thresholds to decide when to bet. However, the difference in determining if we *could* bet and if we *should* bet is substantial. According to [10] page 9, we should call when:

$$Ppot \geq pot\_odds$$

Equation 2

where the pot_odds are determined by:

$$pot\_odds = amount\_to\_call \div (pot\_size + amount\_to\_call)$$

Equation 3

5.3.5 Make the decision

Comparing the results of the previous sections to some boundaries makes the decision:

```
try{
      if(EHS > 0.70){
            decision="RAISE";
      }else if(EHS > 0.20){
            decision="CALL";
      }else if((EHS > 0.12) &&
      (ppot >= (betAmountToCall/(potSize + betAmountToCall))) ){
            decision="CALL";
      }else{
            decision="FOLD";
      }
}catch(ArithmeticException e){
                  decision = "CALL";
```

```
}


if (decision=="FOLD" && (p1Suit==p2Suit)){
      decision = "CALL";
}else if ( decision=="FOLD" && (Math.abs(p1Rank-p2Rank)==1)){
      decision = "CALL";
}
```

The latest attempts are using fixed boundaries. Later attempts may take into account the proportion of wins and losses to adjust these values.

5.3.6 The post flop decision classes

Theses decision classes are very similar to the `PreFlopPlay` class. The main differences being the way in which the probability scores are generated.

In the case of on the flop play, the Texem program calls the `onFlopPlay` decision with the following call:

```
String action =
onFlopPlay.PlayDecision(p1,p2,c1,c2,c3,gInfo.getNumActivePlayers(),
relPosition, gInfo.getPot(), gInfo.getAmountToCall(ourPos));
```

The class then adds to our 'hand' object the three community cards that have been dealt so far. The classes we call then remove these three card objects, in addition to the two private card objects from the 'deck' object. The next step of the decision class is like in the `PreFlopPlay` method, using hand strength, hand potential and explicit boundaries to return a string result.

5.3.7 Weight Tables

According to [23], there are many weak hands that would ordinarily be folded before the flop, which may form a very strong hand with a good flop. Texem was at the time of writing attempting to make use of weight tables in order to appropriately skew the probabilities to reflect the way in which an opponent would have played the hand. This is a very basic form of **opponent modelling**, which may or may not be included in the final release due to the limited number of games being played. The alternative is to compare the weights and then suits of the pocket cards:

- If the suits are the same, increase likelihood of play
- If the difference between the ranks is low, increase likelihood of play.

5.4 Debugging and testing

Debugging formed a large proportion of the total development time for Texem. It is estimated that time to debug `Texem` and `readPPtext` java files took around 60% of the total development time. Coupled with this, the team also had to contend with inconsistent text captures and modifications to the textual output from Paradise Poker.
Here we highlight some of the greatest obstacles in debugging:

5.4.1 Game entry

Texem is not guaranteed to begin capturing text exactly as a game begins and in fact will seldom do so. If the program tried to proceed to the River stage, several

exceptions would occur. This is because a Hand that holds the community cards has not been properly initialised to hold all of the earlier cards. Programming had to allow for game entry in any possible stage. This required a new solution and it was proposed that Texem have some form of **observation mode** [10.2.1]. This revealed yet more problems that required extensive debugging.

5.4.2 Player entry

The original Alberta code, which assumed that all players were entered before the game began, was not entirely suitable for online poker in which we'd have to dynamically insert players. The code was modified to stop natural player rotation and in favour, select the player by name as information fed through.

5.4.3 Player name

Clearly, Texem must be cautious when detecting keywords, such as "call". A complication lies in the fact that a player's name could in fact contain one or more of these substrings:

e.g. `>>Dealer: bob_calls raises`

Texem has been designed to overcome this problem, at the expense of more string checks.

5.4.4 Card entry

Due to time restrictions of the competition day, textual entries that are manual may be erroneous. For this reason, the code that parses data input about the cards has been made to be extremely robust. String methods and card removal methods check that both the syntax and the order of card entry are legal.

5.4.5 Chat

The code had to be rigorously checked to ensure that keywords in chat could not be accidentally (or even maliciously) used to jam our ability to track the game. For this reason the team requested that on competition day it was forbidden for either team to use chat. Under extreme circumstances, malicious use of chat can lead to data loss in our capture analysis code.

5.4.6 Exceptions

A large amount of the debugging effort focused of catching appropriate exceptions. This is due to the unpredictable nature of chat strings; player entry, play order and text capture quality.

6 Game flow and execution of code

By accurately monitoring the state of the game at Paradise Poker, we can be assured that all moves are legal. No violations can occur as Paradise Poker does not permit them. This simplifies the task enormously and reduces the problem to one of simply tracking the game accurately and suggesting appropriate moves.

6.1 Overlap detection in consecutive captures

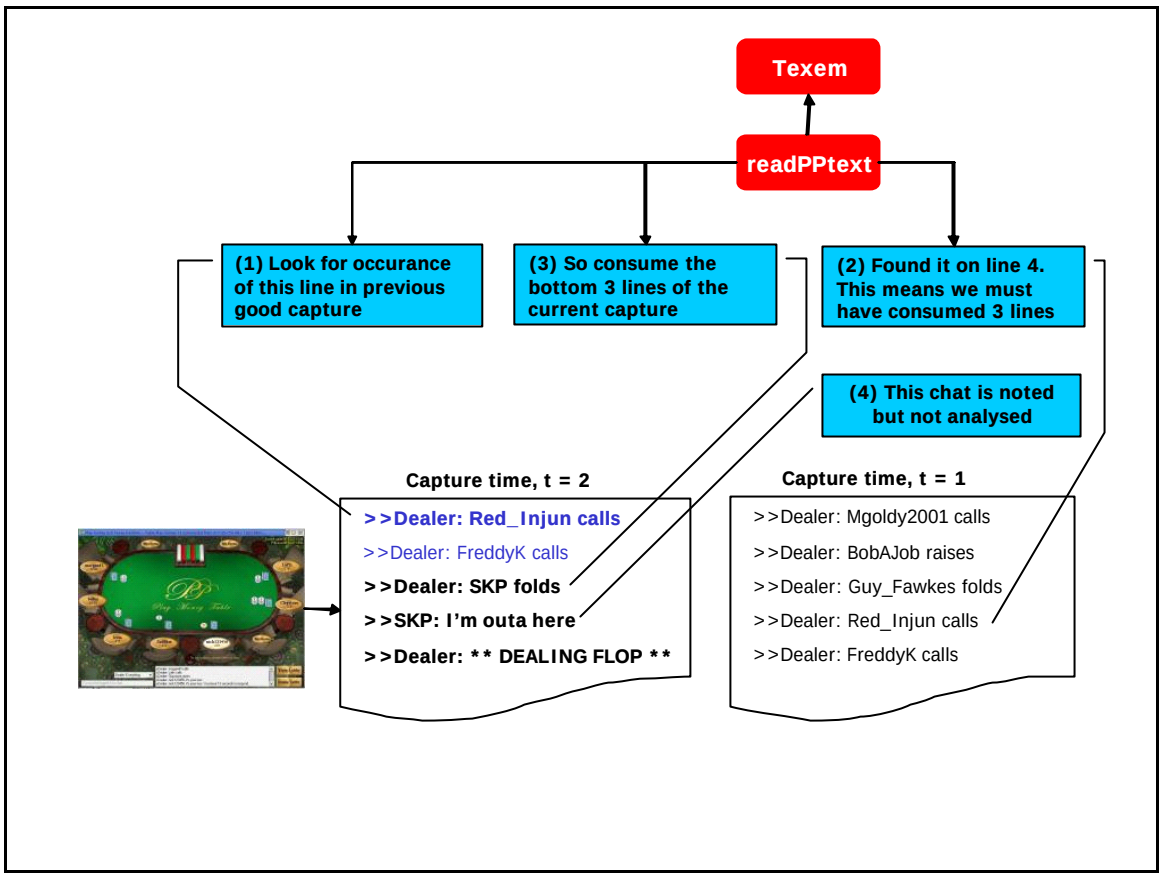


Figure 9: Detection of overlap between captures

As demonstrated in Figure 9, Texem has been designed to monitor changes in the captured text and to determine the new and unique lines that are in requirement of processing. The program retains a copy of the previous Vector (which contains the 5 lines of text). Overlap is detected by calculating the index of the new first line within this Vector. Two limitations exist; one is the maximum rate of capture [6.1.1]. The second is imperfect captures [6.1.2].

6.1.1 Rate of capture

The screen-capturing software used in conjunction with Texem is configured to capture at a rate of once per second. Texem is configured to capture at an identical rate. This rate is the fastest available setting for the screen capturing software. In criticism, this resolution of capture should be around twice the rate and would allow perfect knowledge of events. Although if we took into consideration the demands on the operating system then it would be necessary to use a high-specification machine, particularly with rapid file input-output capabilities. In very rare circumstances this rate is not sufficient to capture all lines of text and one line may be lost. However, in comparison with the manual alternative, the only suggestion for improvement could be use of a more reliable alternative screen-capturing program.

6.1.2 Imperfect captures

When concurrent reading and writing processes are demanding system time, it is unavoidable that errors occur from time to time in captures. SnagIt occasionally

performs bad captures and random text is inserted in various places. Texem is designed to skip a bad capture and retain a memory of only the last good capture providing the affected text occurs **at least** on the first line. If two bad captures occur consecutively Texem cannot guarantee that game integrity will not be violated.

6.2 Two-stage passing

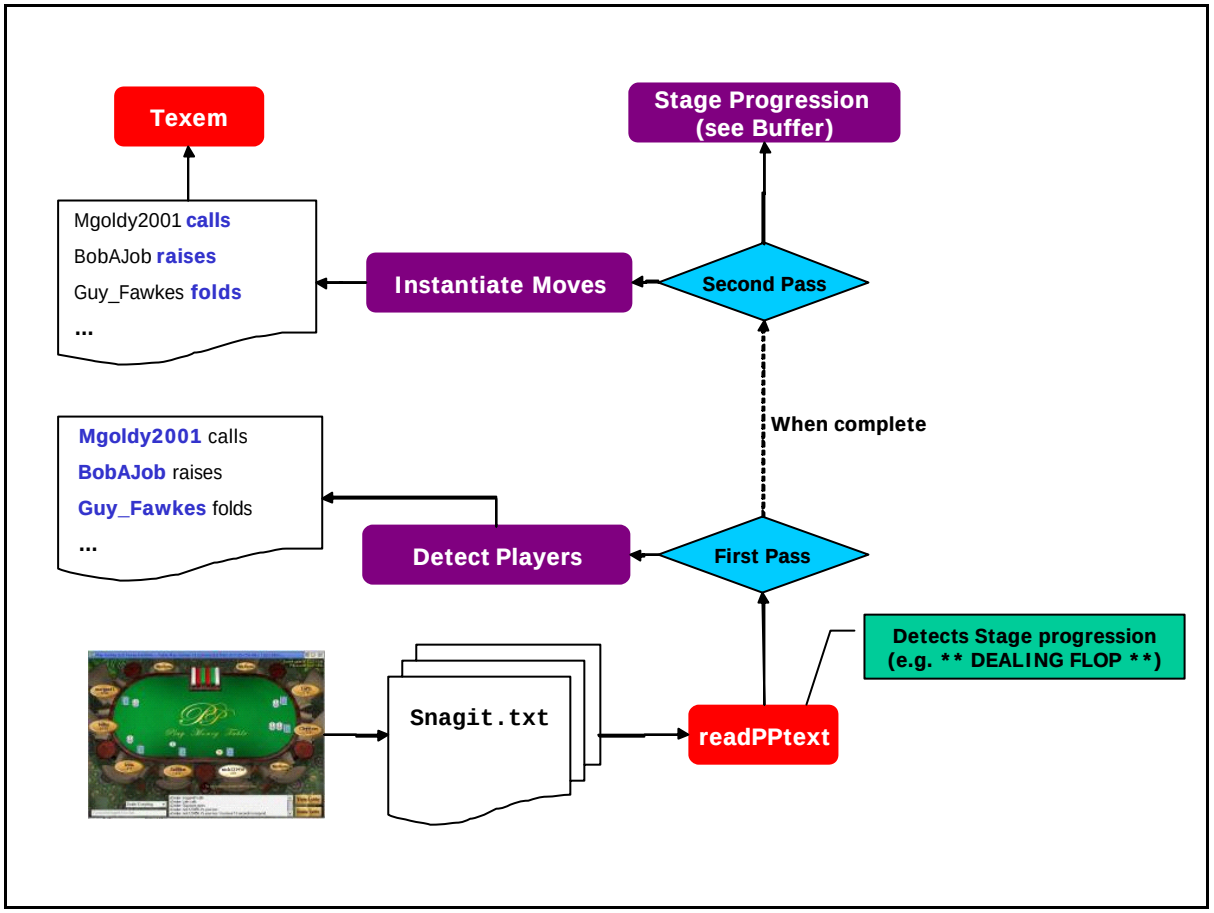


Figure 10: The process of two-stage text passing

When using the original Alberta classes, it was necessary to know the complete list of players before embarking upon a game. The Alberta classes assumed that after each player made a move, the decision would naturally rotate to the following remaining active player. In online Poker this is not possible. This rotation had to be disabled and in its place the code needed to **set** the current player to be the player we detect as a string. However, in order to enforce this **setting** action, it was necessary to add the player concerned to the game. It is this alteration that is referred to as **two-stage passing**.

Figure 7 illustrates the situation. `readPPtext.java` will process the string and detect players. Lets look at an example:

```
>>Dealer: red_injun calls
>>Dealer: Team_gxk1 raises
```

Now, "red_injun" and "team_gxk1" are checked against a temporary Vector of players to test the players have been added. If this is not the case, the player will be

created. In this case we'll assume both players are new and need adding and this is the task of the **fist pass**.

On the **second pass**, we set the current player to be equal to the position where "red_injun" is sat in our list of players. This player's move is instantiated as a "call", and we no longer allow the original code to progress to the next active player, as our order of players does not necessarily correspond to the true order. Instead we enforce the next player to be "red_injun" as we know this player has previously been added and finally activate the appropriate move. The code then proceeds to instantiate the moves made by the second player.

6.3 Text buffering

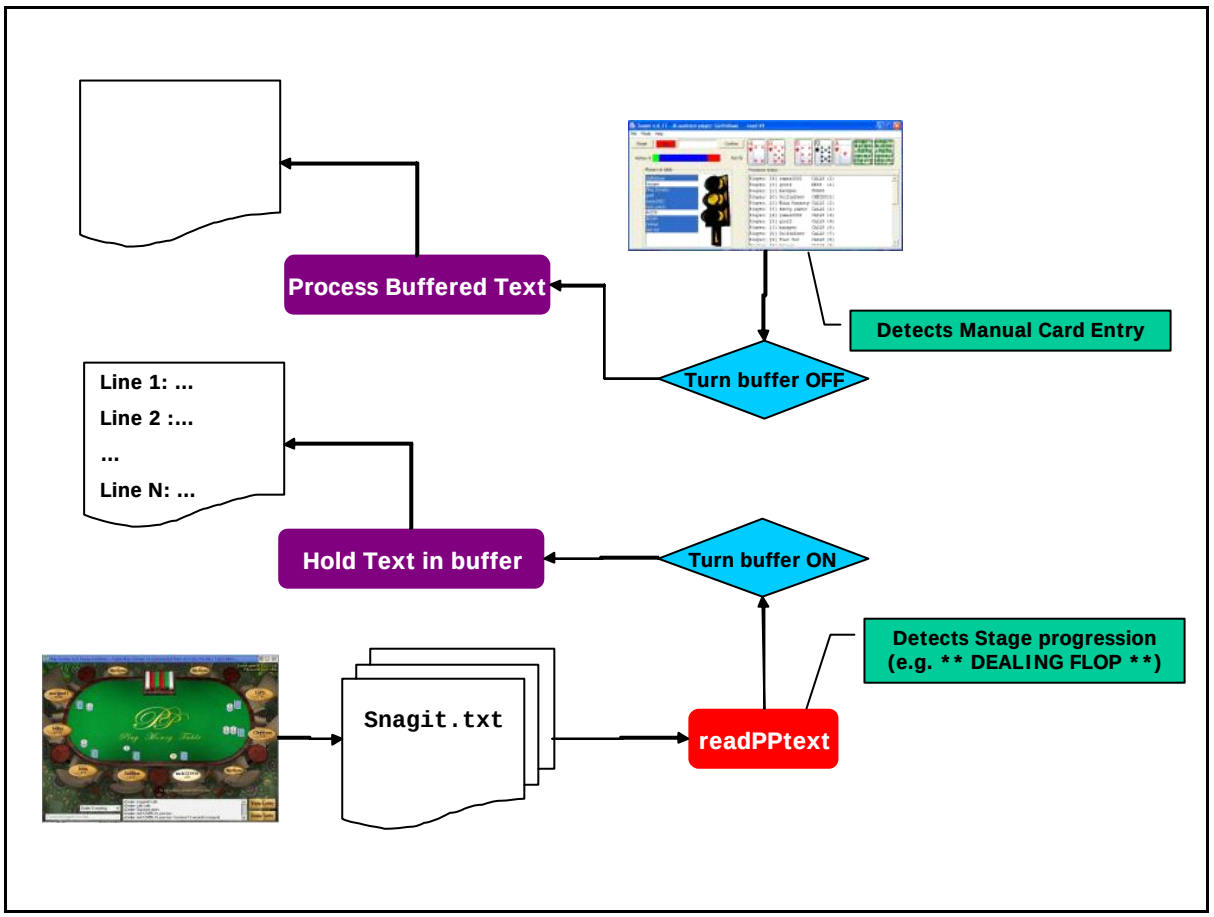


Figure 11: The use of buffers to preserve game integrity (readPPtext.java)

Consider the following screen capture from Paradise Poker:

```
>>Dealer: ** DEALING FLOP **
>>Dealer: whos_who folds
>>Dealer: FreddyK calls
>>Dealer: RCCotter calls
>>Dealer: mrTee folds
```

A problem arises if we try to blindly process each line. We cannot process dealing flop, as this in turn calls a method that expects the flop cards to be provided and as these flop cards do not appear in the text box they must be entered manually. Nor can we immediately consume any subsequent lines, as this would not preserve the order of the game and would eventually violate the program integrity (e.g. number of permissible raises would soon by exceeded). Instead, we must hold all lines between the appearance of **"Dealing Flop"** text and the **actual entry** of those stage **cards** in a temporary buffer. On manual entry of the cards, the buffer can be processed, preserving game rules. Figure 11 demonstrates this sequence.

6.4 Deductions

As mentioned, player entry is forced to be dynamic when playing online poker. This allows for several simplifications in Texem:

1. If we had opted for a static player list that updated with player entry and exit, the relative order of the players would not be maintained. We need to be concerned with this as we opted for a dynamically updating player list that refreshes with each game.
2. Player entry and exit in the game need not be accounted for. If a player does not appear to make a move, then it's obvious that they are sitting out. A player must fold before leaving the table (or is automatically made to do so), resulting in fewer complications.
3. The output box on Paradise Poker does not specify which player is the dealer forcing either manual entry or some form of deduction. A deduction trick was used: The final player to make a move must have been the dealer. By the time the flop has been made, every player is guaranteed to have made a move and hence the dealer can be calculated.

7 Problems encountered

7.1 Comprehension of the Alberta classes

The largest problem to overcome was that of understanding the University of Alberta *loki* classes. This code has been released with the accompanying Java documentation [24], which was useful for reference but did not serve well as a tutorial. Often after something had been coded, it would be found to exist within the Alberta code. This is a problem that was significantly reduced as more of members of the team became involved in programming. As we began to realise the value of the Alberta classes it became increasingly difficult to understand the code in its entirety. For simple problems, it would often take longer to work out how to use the existing code than it would to write afresh.

7.2 Compilation

We also had some difficulty in getting the Alberta classes to compile, as no one in the group had had experience in Java packages. This problem was resolved by one member of the team and the ability to compile work on home machines propagated through the group.

7.3 Communication

A major problem was communication within the group outside of the set meetings. Meeting attendance was usually high although there were parts of the project that required an incredible amount of co-ordination. Often, communication was poor and would lead to a low weekly productivity. Other times, there would be conflicting work done between members of the team and would lead to redundant code. Occasionally if someone found they were pushed for time or the task they were doing was beyond their capabilities, progression of work would be delayed until the necessary element was completed.

The way we used our time was not as efficient as it perhaps could have been. Many of the basic tasks we would set ourselves would take much longer than initially expected. A task that we aimed to do in a week could finish by taking four as slowly tasks proved to be far more complex than perceived. This repeated underestimation continued for most of the project but was identified and addressed when certain members expressed their concern for overly ambitious targets. This realisation meant that task times were revised or the task removed altogether.

The text reader code has also proved to be difficult to complete, as occasionally the text output from Paradise Poker is faster then the reader can handle and line loss occurs. However, the program decision is based upon a broad spectrum of data and the small proportion of data loss we have already witnessed is acceptable.

7.4 Debugging

Due to the size of Texem, debugging has proved to be extremely difficult. The quantity of code we were required to scrutinise is immense. There will inevitably remain many errors in the early versions although hundreds of obvious errors have found and remedied. Those members that have not been involved code development have been deliberately chosen to debug code to allow fresh ideas to circulate.

8 Extending the system

8.1 A new Alberta

Should the Alberta classes improve, or be succeeded by competing academic institution, the implications for Texem should only be positive. Providing similar signatures are used for methods and classes are written in Java, there should be minimal work required for adaptation. Admittedly within the current release Texem only exploits a small amount of the decision making code and this should be the first port of call for further work.

8.2 Porting Texem to other sites

As mentioned in section [0], Paradise Poker's major advantage over other sites is that of the text box displaying game events. This reduces the quantity of manual input to a reasonable level and could make a release version more popular. In order to achieve full product status, it would be necessary to somehow interface Texem with the alternative Poker sites mentioned.

TruePoker.com, for example, gives the user a much more graphically pleasing experience, but does not contain the same level of summary required. It does, however, execute .wav files when each character makes a move. A suggestion for further work would be to detect the sound file being executed to track the game.

8.3 Suggestion features

8.3.1 Result feedback

At present, the outcome of a game is not supplied to Texem. The program should be able to learn from its mistakes. It could be able to do this by storing statistics about decisions and their outcome. We could hold both long-term and short-term memory files. This will allow the program to analyse its own performance and improve itself accordingly.

8.3.2 Player styles

Individual styles vary considerable. Texem's failures could easily be exploited if a unique style could trick Texem's judgement. To assist decision-making and style, Texem could store statistics from players it meets in the field. When the program encounters certain players again it could reload relevant statistics. This is the problem of **opponent modelling** that has been attempted by the GAMES research team at Alberta. Texem could be adapted to "watch" players in games before actually playing and could be left gathering this data when the computer is idle.

8.3.3 Bluffing

Bluffing is an important aspect of any successful poker players' game and allows them to win money on pots that they would not have had a chance of winning in other circumstances. It is also important to vary playing style so that over the course of a game of poker, opponents find it difficult to predict a player's behaviour and guess which cards they are holding.

Bluffing is a powerful tool for a seasoned poker player, but it is a tool that should not be overused since opponents will guess your tactics more easily, and you stand to lose more money by taking risks. Should we be generous with our bluffing and bluff 3% of our hands? Given a hand lasts approximately one minute that means we would play just 60 hands during the tournament. Of these, we would be lucky to bluff once and advantages would minimal. However, even if you only break even on the bluffing plays, the false impression created about your play may improve the profitability of subsequent hands [10].

8.4 Implications for online gaming

Finally, we must assume that at some point in the near future, this application or a similar one may succeed in proliferating around the Internet and to online poker

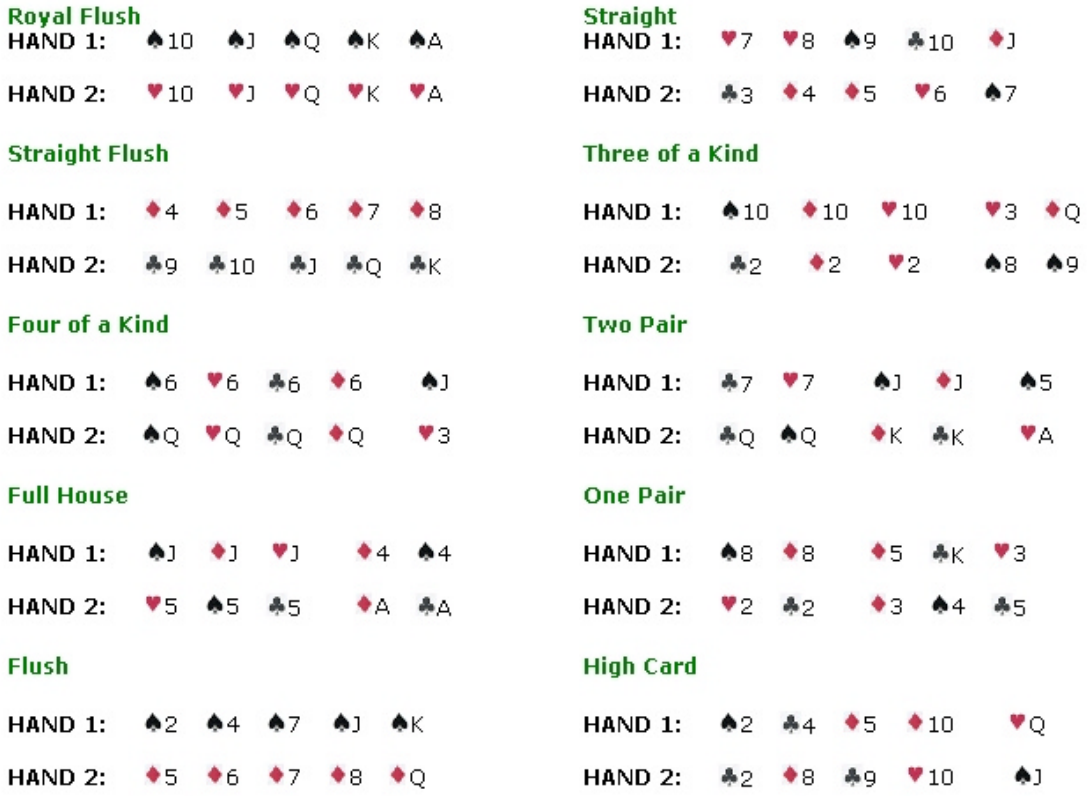
enthusiasts. This is harmless for virtual money players. However, if those people supplying such sites with custom begin to become concerned about their opposition's level of A.I assistance, then the economics of online gaming may suffer the consequences. Imagine a world where all athletes are allowed to compete using steroids. Competition in this field is then between the chemicals rather than contestants. Similarly, the future is bleak for an online poker field dominated by flawless assistants.

9 Glossary

1 **Poker** - A gambling card game of luck and skill.
2 **Texas Hold'em** - The variation of poker we are using for our program.
3 **Loki** - An AI poker playing program made at the University of Alberta
4 **Pot** - The Total Money on the table.
5 **Small Blind** - A half bet made before the cards have been dealt
6 **Big Blind** - A forced whole bet put in by a player before the cards are dealt
7 **Call** - To match the current bet adding no more than the previous.
8 **Check** – To Call, when the bet has not been increased.
9 **Raise** - To increase the bet again.
10 **Re-Raise** – To match a raise made by an opponent and to then raise the stakes again.
11 **Fold** - To retire from the current game and sacrifice all bets made in that game
12 **Bluff** - To give other players a false impression
13 **Dealer** - The person that gives each person there cards and performs the rest of the game
14 **Deck** - The stack of cards.
15 **Shuffle** - To make the cards in the deck as random as possible.
16 **Pocket Cards** – The two cards each player has that are unique to them only.
17 **Community cards** - The cards that are shared by all players, placed on the table.
18 **The Flop** - The first three community cards, put out face up, all together
19 **The Turn** - The forth community card to be shown.
20 **The River** - The turning over of the fifth and final community card.
21 **Hand** - The collection of the best five cards that you use.
22 **Collusion** is when two or more players on the same table work together as a team, unfairly knowing each other's cards, and betting with this knowledge in order to maximize their team's profits. Collusion destroys the integrity of any poker game. Any player that attempts to collude at ParadisePoker.com will be permanently banned from games.

10 Appendix

10.1 Poker hands



10.2 User instructions

Texem requires SnagIt [15]

10.2.1 Observation mode

- Start Paradise Poker and go to a table
- Do not sit down
- Open the SnagIt application
- Open Texem
- Start the record button on SnagIt with the specified configuration
- Press “Start” on the Texem interface

This mode:

- Auto-generates cards so they don’t need to be entered
- Picks one of the players at the table and pretends to be assisting them (required for table position figures among other things)
- Does not use Intelligent-Buffering

10.2.2 Participation mode

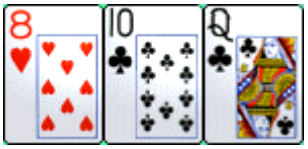
- Start Paradise Poker and go to a table
- Sit down

- Open the SnagIt application
- Open Texem
- Start the record button on SnagIt with the specified configuration
- Press “Start” on the Texem interface on the first game where Texem assistance is required.
- When entering cards, type very carefully with the correct number of spaces. For example, pocket cards would be

POCKET CARDS

```
p 2c 5d
```

FLOP CARDS



```
f 8h tc qc
```

TURN CARD

```
t 9s
```

RIVER CARD

```
r ah
```

and press Confirm or ALT+C (no leading or trailing space)

This mode:

- Takes the entered cards into account in A.I.
- Will perform Texem assistance to the player hard coded into the Java file (see Texem line: 1611)
- Does use Intelligent-Buffering to hold lines until needed.

10.2.3 SnagIt Configuration

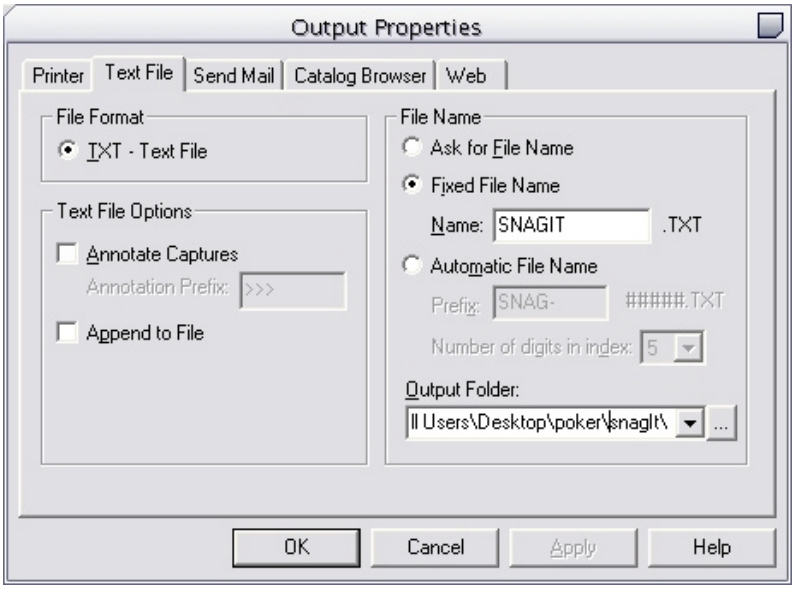


Figure 12

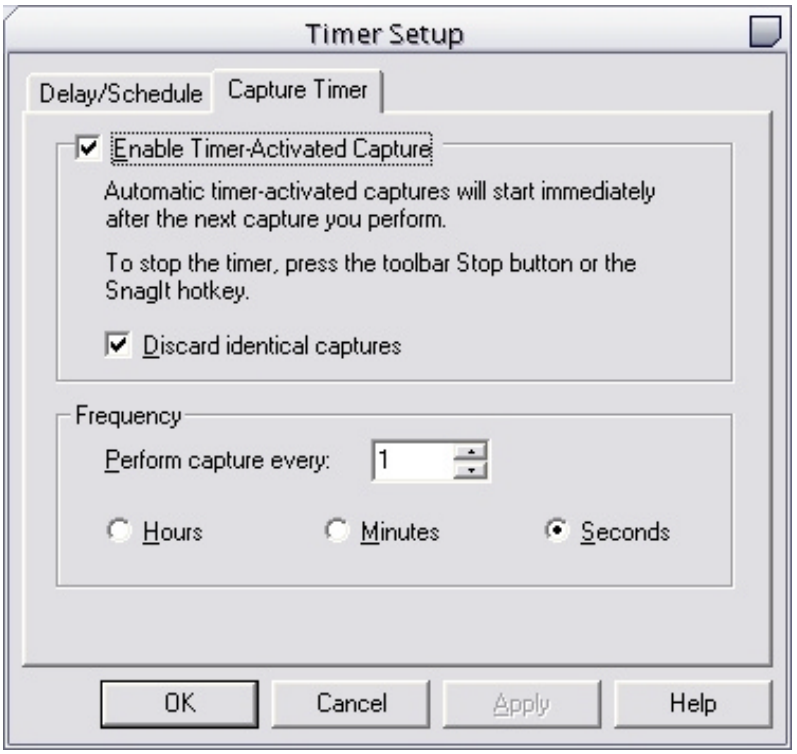


Figure 13

10.3 Colour Key

Illustrated below is the key for the file-contents window. This allows the text-capturing process to be more transparent and visual to the user.

>> Dealer: louiep calls	previously processed file-contents
>> Dealer: ThMick bets	new file contents
>> Dealer: bob raises	buffered line

11 References

1 Paradise Poker online poker forum
http://www.Paradisepoker.com

2 Phil Haley, MacHack VI
http://www.ncf.carleton.ca/~bw998/canchess.html

3 Computers, chess and Cognition, T. Anthony Marsland, Jonathan Schaeffer, Springer-Verlag 1990

4 Computer World News
www.computerworld.com/news

5 "Communications of the ACM: Temporal Difference Learning and TD-Gammon", published March 1995 by the Association for Computing Machinery

6 Gary H. Anthes, "Games computers play", published February 25 2002 Computer World Magazine [www.computerworld.com]

7 Oskar Morgenstren, John Von Neumann, "Theory of Games and Economic behaviour", Princeton University Press

8 N. Findler, 1977. "Studies in machine cognition using the game of Poker" *Communications of the ACM* 20(4):230-245

9 Barone, 1998, 1999, 2000

10 Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron, "Poker as a Testbed for Machine Intelligence Research", Proceedings of AI'98, (Canadian Society for Computational Studies in Intelligence), 1998. The University of Alberta

11 Jonathan Schaeffer, Darse Billings, Lourdes Peña, and Duane Szafron, "Learning to Play Strong Poker", ICML-99, Proceedings of the Sixteenth International Conference on Machine Learning, 1999

12 Darse Billings, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron, "Using Probabalistic Knowledge and Simulation to Play Poker", Proceedings of AAAI-99, (Sixteenth National Conference of the American Association for Artificial Intelligence), 1999.

13 Luigi Barone and Lyndon While, "Adaptive Learning for Poker"

14 Ken Warren, "Winners guide to Texas Hold Em Poker", Cardoza Publishing, 22 April, 1999.

15 SnagIt software download page
http://www.techsmith.com/products/snagit/default.asp

16 Using the Timer and TimerTask Classes: AnnoyingBeep.java
http://java.sun.com/docs/books/tutorial/essential/threads/timer.html

17 Beginning Java 2 (Using GridBagLayout pg.555), Ivor Horton, Wrox Press 2000

18 Java Tip 104: Make a splash with Swing
http://www.javaworld.com/javaworld/javatips/jw-javatip104.html

19 A visual index to the Swing components: An Example of Using a Text Pane
http://java.sun.com/docs/books/tutorial/uiswing/components/text.html

20 A visual index to the Swing components: How to use Lists
http://java.sun.com/docs/books/tutorial/uiswing/components/list.html

21 "Winning Low Limit Hold'em", Lee Jones, ConJelCo 1994

22 David Sklansky, Mason Malmuth, "Hold'Em Poker for Advanced Players", Two Plus Two Publishing, 1999.

23 Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron, "Opponent Modeling in Poker." *Proceedings of AAAI-98 (15th National AAAI Conference)*, 1998.

24 Loki Documentation, the University of Alberta
http://www.cs.ualberta.ca/~davidson/poker/src/docs/