2\textsuperscript{nd} Year Software Engineering Project
‘Final Group Report’

automated \textit{poker player}

Supervisor: Graham Kendall
Group: gp-gxk2

Group Members:

Michael James Pope \hspace{1cm} mjp01u
Neil Oscar Collins Tippett \hspace{1cm} not01u
Oliver Philip Turley \hspace{1cm} opt01u
Louis Wan \hspace{1cm} lxw01u
<table>
<thead>
<tr>
<th>Contents Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Abstract</strong></td>
</tr>
<tr>
<td><strong>2. Introduction</strong></td>
</tr>
<tr>
<td>2.1 Automated Poker Playing</td>
</tr>
<tr>
<td>2.2 Updated and Expanded Description of the problem Objective</td>
</tr>
<tr>
<td><strong>3. Texas Hold' em</strong></td>
</tr>
<tr>
<td><strong>4. Background Information</strong></td>
</tr>
<tr>
<td>4.1 Context</td>
</tr>
<tr>
<td>4.1.1 Game Playing</td>
</tr>
<tr>
<td>4.1.2 Online Poker Playing</td>
</tr>
<tr>
<td><strong>5. Implementation</strong></td>
</tr>
<tr>
<td>5.1 Program Interaction</td>
</tr>
<tr>
<td>5.2 Entering A Game</td>
</tr>
<tr>
<td>5.2.1 Loading The Client And Choosing A Table</td>
</tr>
<tr>
<td>5.2.2 Paradise Poker Client Window Position</td>
</tr>
<tr>
<td>5.3 Setting Up and Sitting Down</td>
</tr>
<tr>
<td>5.4 Commentary</td>
</tr>
<tr>
<td>5.4.1 Capturing the text</td>
</tr>
<tr>
<td>5.4.2 Repeat Checking</td>
</tr>
<tr>
<td>5.4.3 Parsing</td>
</tr>
<tr>
<td>5.5 Card Recognition</td>
</tr>
<tr>
<td>5.6 Making Decisions</td>
</tr>
<tr>
<td>5.6.1 Hand Strength</td>
</tr>
<tr>
<td>5.6.2 Decision Making Class</td>
</tr>
<tr>
<td>5.7 Making Plays</td>
</tr>
<tr>
<td>5.8 Graphical User Interface</td>
</tr>
<tr>
<td>5.9 Combining Everything</td>
</tr>
<tr>
<td>5.10 Problems Encountered</td>
</tr>
<tr>
<td>5.10.1 Colours</td>
</tr>
<tr>
<td>5.10.2 Options Window</td>
</tr>
<tr>
<td>5.10.3 Paradise Poker Client</td>
</tr>
<tr>
<td><strong>6. Summary</strong></td>
</tr>
<tr>
<td><strong>7. Bibliography</strong></td>
</tr>
<tr>
<td><strong>8. Appendices</strong></td>
</tr>
<tr>
<td>8.1 Paradise Poker Windows</td>
</tr>
<tr>
<td>8.2 Legal Hands</td>
</tr>
<tr>
<td>8.3 Hand Rankings</td>
</tr>
<tr>
<td>8.4 GUI</td>
</tr>
</tbody>
</table>
1. Abstract

This paper describes the development of our automated poker-playing agent. The playing agent will have to connect to an online poker site, and take part within a poker game observing the game and interacting in relation to the knowledge it has obtained. The difficulty of developing such a player for an on-line gaming site is that each of the large number of client software has a number of intricate steps involved in participating in a game. Selecting one particular client we developed an agent to suit that client and its play is entirely automated.
2. Introduction

2.1 Automated Poker Playing

The group was handed the following description as our project brief:

Last year, two of my group projects had to develop an automated poker player and then compete against each other on an on-line poker site.

One of the most challenging parts of the project turned out to be either keeping up with the game in real time (i.e. looking at one screen and typing in the instructions onto "their" screen or trying to automate the process).

The problem is that there is no well defined interface (API) into the on-line gaming sites so they either had to do everything manually or try to automate the process by interpreting whatever information is on the screen and making sense of it.

Your task is two-fold

1) Write a poker playing "agent" that is able to observe an on-line game and produce a full record of what went on. This record should be output in a structured format so that it can be used by other programs to analyse the game but should also be understandable to humans.

2) Your "agent" should be able to play in real-time, having access to the information it is capturing. That is, your agent should have access to the current game state and base its decisions on that data.
2.2 Updated and Expanded Description of the problem Objective

Firstly to write the poker-playing agent our group had to determine which of the online poker-playing clients to use, and what variation of poker to play. With the vast number of clients available on the Internet the group had decided to pursue Paradise Poker [1], a client used in the previous years group project [2]. Importantly the automated process will be strongly related to the layout of the client, and for that reason the practical procedure of registering with the client would underline a set of processes the agent will have to employ see section 8.1. The actions highlighted are features of Paradise Poker that will have to be interpreted to automate the process and enter an online game at that particular client.

With numerous variations of poker available at Paradise Poker the group decided to use the Texas Hold’em play money tables. Importantly one group member understands the rules of Texas Hold’em, to help the group learn the rules and techniques, though Texas Hold’em was equally elected because it is a strategic and fast-paced variant of poker, which presents a variety of obstacles particularly keeping track of the large number of players.

The second part of the project is to write an "agent" to play in real-time. The complexity of real time is that there will be a number of steps the agent will have to record, as each betting round will change the nature of the game. As a group we felt we would benefit from the text box provided by the client, as it provides real time commentary of what is going on in the game (e.g. who post small blind/big blind, and which player bets/folds). Therefore to produce a full record of what went on and to have access to the current game state most of the game status could be obtained from that message box. It is vital that the agent captures the information at a high rate to keep in sync with the game state otherwise the server may time-out before a decision can be made.

To conclude the aims of the project are to automate each process, effectively interpreting, evaluating and outputting the necessary game information:

Aims:

- Automatically load the client Paradise Poker
- Select and enter an available game
- Sit down at the table
- Capture the game information
- Interpret the game information
- Select a play
3. Texas Hold’em

In a hand of Texas Hold’em each player is dealt only two private cards, relative to five public cards. As fewer cards are dealt to each player as many as ten players maybe be involved in any one hand. The Involvement of such a large number of participants in a game of Texas Hold’em has led to its popularity, hence it is used in many major tournaments including ‘The World Series of Poker’ and channel 4’s TV show ‘Late Night Poker’.

To start a hand, before any cards are dealt, the two players to the left of the dealer place forced bets known as the small blind and big blind, which ensures at all times there is money to be won. Each player is then dealt two private cards, known as the pre-flop followed by the first round of betting, a player then has the option to check, raise or fold. The player to the left of the player who placed the big blind is first to go, continuing round clockwise until the last to go, which is the player who placed the big blind.

If any player has decided raise, then an opportunity is offered for each player to check, re-raise or fold. Then three community cards are dealt face up on the middle of the table, called the flop. A second round of betting then takes place.

On the turn, a fourth community card is dealt face up, with another round of betting.

The river is the final card to be dealt and a final round of betting then occurs between the remaining players.

The players still active in the game at that time reveal their two private cards, and the player with the best five-card poker hand wins the pot. A hand may be formed from any of the player’s two private cards and the five public community cards. In a tie, the pot is split.

A ruling of Texas Hold’em is a player is never obliged to drop out of contention because they have insufficient funds. Therefore a player has the opportunity to call the amount of chips in front of them, called ‘all in’. The excess bet is then placed in a side pot for the players matching the amount called [3].
4. Background Information

4.1 Context

4.1.1 Game Playing

The further development of an artificial intelligent poker player is unnecessary for this particular project, however the group deemed the study of related artificial intelligence classic game playing appropriate as automating the playing process and the thought processes behind it are closely linked. The field of Artificial Intelligence has played witness to much advancement like the recent Deep Blue victory over World Chess Champion Garry Kasparov in 1997 [4]. With a game of chess, there has been little need to automate the playing process and for that reason research has centered on the game state, in contrast an online game such as poker proposes two tasks the ability to select actions and the interpretation of the game state to formulate the decisions behind each action.

As we develop our agent to play in real-time, the information of the game state is incomplete [5] in contrast chess is visible to both participants providing the developers with perfect game information [6]. With that game information Deep Blue had developed an excessive search engine, which analysed roughly 250 million chess positions per second [7] their accomplishment was seen as a milestone for the Computer Science community. Programmers now strive to implement alternative methodology other than ‘brute-force’ and ‘deep searches’, which has led to the examination of games involving imperfect information such as poker. An automated poker player could well be the front end of the latest game playing artificial intelligent poker player, to have both an agent able to perform the operations of human action, selecting valid actions for play with the artificial intelligence to be that of a high performance player. Although Artificial intelligence has stretched far beyond the scope of our project, the group has a view of the importance for automated applications and how such an agent will be a fresh domain in the outsized Artificial Intelligence research, which has developed copious intelligent game playing programs.

4.1.2 Online Poker Playing

The world of online poker has been the interest of recent Computer Science research, as clients such as Paradise Poker host a large online catalogue of card rooms with such games like that of Texas Hold’em. Poker has a broad appeal due to the number of poker game variations and their complex properties including incomplete knowledge, multiple competing agents, opponent modeling, deception, and dealing with unreliable information [8]. An additional difficulty of producing an agent for an online client arises as the client constantly upgrades and alters the poker software, this may alter the way in which the program interacts with the client.

To help the group approach the problem of developing the automated processes of playing poker online we visited the work of The University of Alberta Computer Poker Research Group [9] and their development of an automated Texas Hold’em agent. Their ongoing research has developed a program called Loki, and more recently an updated release of Loki, named Poki; importantly the program is implemented in Java a language all the group has had experience with. The Poki player is constructed through a number of complex classes and commands, the source code for much of the agent has been released [10], briefly reading through the classes we discovered that the classes may well play relevance to the decision making. Poki does not provide a foundation for the automated processes but reflects advanced Artificial Intelligence methods to learn and observe opponents, construct opponent models and dynamically adapt play to exploit patterns in the opponents’ play [8].
5. Implementation

5.1 Program Interaction

One of the main objectives of the brief is to implement a way to make the groups program interact online with the ‘Paradise Poker’ client. Through our research we have discussed two methods with which we can do this using java, our elected programming language. The first of these is to use ‘SnagIt,’ a commercial program developed by TechSmith™ [11], and secondly a class in the Java class libraries called Robot [12].

‘SnagIt’ allows the user to capture text, pictures or movies being displayed to the screen by another program, and export it to a file, to clipboard or to the screen. It has a further feature, which repeatedly captures the specified media with a user specified time interval between captures, one second being the minimum.

The Robot class in java is the second way in which our poker player program will interact with ‘Paradise Poker.’ This class mimics user input to a computer, allowing control of mouse features such as movement and clicking. The Robot class allows keystrokes to be sent to programs and the operating system and also can get the pixel colour at a specified point on the screen and capture pictures of areas of the screen.

5.2 Entering A Game

5.2.1 Loading The Client And Choosing A Table

The first thing that our program needed to do was to load the ‘Paradise Poker’ client. If the client were already loaded, executing the client’s file would simply bring the client to top of the window stack. Once the client was loaded and in the lobby screen, the next task is to choose and enter a game.

One thing that we did not cover when we were planning the program was how we would know which game to join from the ‘Paradise Poker’ lobby window. Fortunately when it came to addressing this task we found that we could reuse the OCR class [13] written for the commentary box in section 1.4.1, as the lobby window uses the same font and font size. The only alteration needed was to add an extra method to the OCR class to allow for eleven lines, rather then the five found in the commentary box.

Deciding which game to join was similarly easy to implement. Using the Robot class in java, we could send keystrokes to the lobby window to tell the scroll bar of the games list to move down to the ‘Play Money’ games. The game chosen from this point was one which had less then ten players, and hence had room for us to sit down, and a table that had at least one player sat at it, as playing against nobody would be pointless.

5.2.2 Paradise Poker Client Window Position

Finding the position of the ‘Paradise Poker’ client on the screen of the computer executing it is critical to the successful running of our automated poker player. All the user input to the client that our program mimics, such as button clicks, and screen capturing using the Robot class, requires the program to know where to click or where to capture. In the plan for our project we planned and later implemented a way of searching for a specific turquoise pixel in the top left of the screen, a pixel that appears in the icon of the clients window. The need for searching stems from Java’s lack of Operating-System interaction due to it’s cross platform nature.

Having tested this searching method we realised that it was unsuitable for two reasons, firstly it took over fifteen seconds to search an area of about three hundred pixels square. The second problem was that if the ‘Paradise Poker’ client window was not on top, the program would fail to find the client window and would require the user to bring the client to the top of windows display stack.
Our solution to this problem involves using the ‘Java Natural Interface’ to communicate with a program written in another language. This was researched briefly before the interim report, but not followed up until we realised that our previous ideas were not suitable. The language that we have written this program in is ‘Microsoft Visual C++ (VC++) as it has very close ties to the ‘Windows’ operating system and provides us with many system call that are unavailable in Java. With VC++ we created a Dynamically-Link-Library (DLL) file that contains methods that we are able to call from java. These methods include returning the screen coordinates of the top most window, returning the title bar name of the top window and manipulating the windows stack so that the client can be moved to the top. All of these functions use basic built in method calls in VC++.

5.3 Setting Up and Sitting Down

Once ‘Paradise Poker’ has been manipulated to get it onto a table, we need to set a few things in the table window in preparation for playing. The first of which is to ensure that the commentary box displays all the information we require. The commentary box has different settings so that it can show more or less information. Our program requires that the information setting be set to its maximum so that our program has all the commentary it requires to play. In order to do this we used the Robot class to make two mouse clicks on the combo box that is used to adjust the setting. The first click brings down the options and the second makes the selection.

The second stage of getting into a game involves login-in to the ‘Paradise Poker’ server, done so using the Robot class to operate the menu.

The final stage requires the program to find a free seat at the table and to then select to sit down in it. Pixel colour grabs using the Robot class allows the program to find a free seat. A seat that is occupied to has a larger gold oval encompassing the nickname of the player compared to a free seat, which has a smaller oval. By grabbing a pixel for each seat at the table using coordinates that would be gold if a person is sat in the seat, or the dark background colour if the seat is unoccupied will determine if it is free. Once a free seat has been found, the Robot class can be used to mimic a mouse click on the seat to indicate to the client that we wish to sit there.

5.4 Commentary

5.4.1 Capturing the text

One of the major alterations we had to make from our original plan was the way in which we captured the text from the ‘Paradise Poker’ client’s commentary box. The commentary provided allows our program to understand much of what is going on in the game.

We had planned to use ‘SnagIt’ [11] for this, which would output the dialog to a text file for our program to read. We planned to automate our program with ‘SnagIt’ by means of the Robot class, and use this product’s repeated capture function to continuously capture the text in the commentary box, output it to a text file for our program to read.

When we tested this, however, we noticed that the one second capture interval, the lowest ‘SnagIt’ allowed, caused certain lines of text to be lost between captures. Lost lines occur when more then five lines of text appear in the commentary box within the interval, the number five due to there being five lines with which commentary can be displayed. This led our prototype to think the game was at a different stage in the game then it really was. What was needed was a way of capturing the text with a much smaller interval then one second, more along the lines of half a second or less, hence reducing the chances of lost lines.

To lower the interval rate to less then a second would mean that we could no longer use ‘SnagIt’ as our capture method. What we decided was that the creation of a very basic character recognition class could solve our problems, if it would capture the text very quickly. Early indications through testing suggested that a character recognition class would reduce the capture interval to under half a second, so we set about writing the class in full.
When the class was finished it was tested using the ‘Paradise Poker’ client and the capture interval ranged from about 0.3 – 0.7 seconds, depending on the processor’s load at the time.

The ‘Optical-Character-Recognition’ (OCR) class [13] that we developed is fed a partial screen capture, created using the Robot class, of the dialog box for it to decipher. Firstly it takes the picture of the text box and splits it up into the five individual lines of text. Then each line is split up into characters by looking for a complete vertical line of white pixels, these being the spaces between the characters. Space characters are noticed as being more then one vertical line of white pixels. Finally each character is examined to identify it. This final stage first works out the horizontal length in pixels of the character, how high the letter appears and how low it appears. These three values help to split up the different characters into groups for further testing. The final stage is to do simple pixels captures to work out the exact character being examined. The only problem encountered while doing this was that certain characters didn’t have a vertical line of white pixels between it and the previous character, this can be solved by looking for characters that have an abnormally large length due to it actually being two characters together, and then splitting them.

Not only did this solution reduce the capture interval, it also came closer to solving the problem encountered when we were using ‘SnagIt,’ as three less lines were lost on average by this class compared to ‘SnagIt’ when tested on one hundred lines of dialog with both running concurrently. This might not seem perfect as it implies that some lines were lost with the character recognition class, however there is no way to tell if this is so as only the human eye could be used as a comparison, for no better method is available.

5.4.2 Repeat Checking

Although the OCR class outlined in the above section captures the text in the commentary box successfully, all it is doing is constantly taking pictures of the box. If our program were to just parse straight from the captured text, it could repeat instructions that were parsed from the previous capture. This would lead to a loss in synchronisation with the poker client and cause our program to run into errors.

What was needed was an algorithm that would compare the current commentary capture with the previous, looking for series of repeated lines, and therefore only parse the unique lines that were left. As there are only at most five lines of text in the commentary box at any one time, this means that this algorithm only has to check for the five possible repeated lines scenarios; all lines repeated, four lines repeated, three lines repeated and so on. The main point to note here is that the lines will not just be repeated randomly in the current capture but will be successive lines from top to bottom, as a new line of commentary appears at the bottom of the dialog.

5.4.3 Parsing

Writing the parser was a simple case of ignoring any lines of commentary that our program did not need for its successful execution, and only looking for lines that we needed. Lines such as chat, players joining or leaving the table, players sitting out from a hand and so on are not required. Whereas commentary concerning the dealing of cards, the winner of hands, increases in betting stakes and who players the blinds are required information.

The parser looks for these required lines of text by searching for specific keywords within the captured string. These unique words or series of words allow the parser to recognise the line exactly. A problem that could have occurred was that if a player wrote a line using the chat feature, that satisfied the parser criteria for a no chat line, the parser may have made an error. Thankfully, however, a chat line can be easily differentiated from a commentary line, as a commentary line starts with “Dealer;” and a player chat line starts with their nickname.
5.5 Card Recognition

Unfortunately the commentary box does not provide all the information necessary for our program to follow the poker game. The main thing that it leaves out is telling the user what cards appear in the hand as well as the ‘flop,’ ‘turn’ and ‘river.’ For the values of these cards to be found, so that an informed judgement on the current game decision can be made, some sort of card recognition software was necessary.

Our original idea for this card recognition was to use ‘SnagIt’ to capture the picture of the card and then to compare the picture with the picture file stored in the ‘Paradise Poker’ directory from which the card is displayed by the client. This seemed however be an unwise decision, bearing in mind the search time needed for covering a large number of pixels outlined in section 1.2.2. So then we started to look at working out the value of the card by looking at the pixel colours on the cards in the client’s window to deduce the value. This would involve capturing pixels at predefined locations on the card to narrow down, using the colour of the pixel, which of the 52 available cards was being displayed by comparing them against known colour values.

This is almost exactly what we have implemented, however with a slight change. Rather then making the pixel captures directly to the client window, we took a picture of the card using the Robot class, and sent this picture to a separate class called ‘CardRecog’ for the value of the card to be found. The reason for this was to ensure that the program was better structured then just putting all the code into one class.

5.6 Making Decisions

5.6.1 Hand Strength

Writing the agent to interpret and participate within a hand of Texas Hold'em forms an integral part of the project. One of the proposed ideas for how we would approach the problem of playing in real-time was to examine the available source code for Poki, an intelligent Texas Hold'em poker-playing program, which was notably implemented in Java our elected language. However, although the prospects of using the Poki classes [8] seemed sensible we discussed the length of time that would be necessary to study the implementation of each class and how to integrate the classes into our program. As developing a good quality player wasn’t the main aim of the project we decide we could use the time more effectively to develop our own decision class, as the decision-making element of the program is not an outsized part of the project.

As the card recognition determines the cards dealt to the player and the shared table cards the agent has to interpret what the cards mean, that is whether the cards form any sensible hand to therefore supply a decision to either call/check, raise or fold. This process will therefore have to be applied to each betting round:

- **Pre-flop:** after each player is then dealt two private cards
- **Flop:** after the next three community cards are dealt
- **Turn:** after the fourth community card is dealt
- **River:** after the fifth community card is dealt

After each consecutive betting round the ‘CardRecog’ will be called to pass the information of what private and community cards are held using a two element ID defining the number and suit of each card. The hand evaluator therefore uses two data structures to store the identities of the hands in each game to then repeatedly check the validity of each hand using the methods flush, straight and repetition evaluator. The evaluator methods determine whether the hand is legal (see section 8.2) and importantly returns an integer ranking (see section 8.3) of the hand to be passed to the decision class.
5.6.2 Decision Making Class

A concern over of the hand evaluation class outlined in the above section is the time that will be available to evaluate each hand before a play decision is required. If the decision class does not make a decision within a set time the synchronisation of the agent will suffer as the Paradise Poker client will then ask for a response before it declares our agent “timed out”. The problem here is that the agent will be required to act in response to a new window to acknowledge we wish to continue to play. To handle such an exception we intend for the “CardRecog” class to now pass the card ID immediately to the hand evaluator to formulate a decision in time.

As the hand evaluator passes an integer hand ranking to the decision class we then implement a sequence of decision methods for the corresponding betting round, Pre-flop, Flop, Turn or the River. Although the agent’s decision is determined as a result of the validity and strength of our hand, we also have to consider whether an opponent has chosen to raise the bet. Although we initially were interested in developing a thorough strategy for different situations the resulting decision class implements two straightforward decision strategies.

Firstly if an opponent has raised the bet we need to decide whether to call, raise or fold, while the agent sits with a hand ranking of 1 we select to call, when the hand ranking is greater than 1 the agent will raise. The option to fold is only executed if the agent is in the Flop holding a hand ranking less than 1. If no opponent has raised the bet, the agent will always seek to raise the bet when the hand ranking is greater than 1 otherwise it simply takes the free opportunity to check as there is no cause to fold. The options to check/call, raise or fold are simply methods, which classify the movement of the robot class to click the corresponding option on the client window.

5.7 Making Plays

Although this particular part of the program was not covered in the interim report, we had discussed how the program would decide when to make plays. A play is made by using the Robot class to mimic a mouse click to select the course of action when it is their turn.

Our original plan involved working out where players were sat at the poker table. With the knowledge of nickname of the player that precedes us at the table, the program would simply wait upon the commentary box to display a line that stated that they had made their move. This would indicate that it was our turn and we would make ours.

Finding the name of the player that precedes us at the table, however, is not very easy. The client uses a different font and a different font size for the player names then on the commentary box, hence ruling out our current OCR. Not only that, but if the player has a long nickname it will use a smaller font to fit the name on the gold disk where the nickname appears. To read the nickname on the gold disk would require a more complex OCR at the very least. We couldn’t even use ‘SnagIt’ to grab the nicknames, as even it is unable to grab the text.

A further problem with our planned approach was that if a player was sitting out from a hand at the table, then we needed to ensure that we knew this. Similarly if a new player had just sat down between the previous preceding player, and us, it might cause our program to get confused unless we were very careful about checking where new players sat.

An alternative solution to this is to wait for the commentary box to prompt us that it is our turn. This happens when the player has not responded to their go for roughly thirteen seconds. This would be a perfect solution except that the other players at the table would quickly get very annoyed if they had to wait thirteen seconds every time it was our turn at the table. This would however solve the problem of players sitting out and of new players being seated.
The solution the group came up with is actually very simple. It relies on waiting until the buttons through which we make our move appear by testing if they are visible every time the commentary box OCR makes a grab. This means that we will always know when it is our go and we will not have to work out the nicknames of the players on the table. This also solves the problem of players sitting out and of new arrivals at the table. The button test is completed using a simple pixel grab and comparing the colour to a known value.

5.8 Graphical User Interface

Although user interfaces are a critical aspect of software development, throughout the project the value of the GUI for the agent has been omitted as the processes of the agent are entirely automated. Alternatively with view of demonstrating the software, the GUI will form a prominent identifying feature of the agent and provide a useful mechanism to assist illustrating the information that the agent is interpreting, and to demonstrate how the agent is playing in real-time see section 8.3.

The user interface that has been written for the project adequately provides a menu with the functions to start and stop the program, with additional settings for the time synchronisation with the client. The use of two text windows both illustrate the commentary that has been captured from the clients’ text box, e.g. the details of the next betting round and the identification of the cards that have been dealt, with the actions of each corresponding player to that hand. The second window denotes the user the actions of our poker-playing program, e.g. the actions such as buying in with chips and similarly the agent’s evaluation of the current hand. We had decided the use of showing cards as the game ran inadequate, although it looked useful and portrayed the real time interpretation of the game we had to appreciate the limited space that was available on the screen due to the size of the paradise poker window.

5.9 Combining Everything

Once the main interaction code was written and the code for decision-making was complete, the final task was to bring all these components together with the GUI to create the final program.

All parts of the program such as decision making, card recognition and making plays run directly from the commentary capture loop. This loop has two tasks, firstly to capture the text that appears in the commentary box and to secondly check to see if is decision is necessary.

Card Recognition runs directly from the parser, which interprets lines from the commentary box text. When a line such as “>>Dealer: **Dealing Pockets***” the parser will know to invoke the card recognition class upon the locations defined by the position the player is sat at the table.

If the commentary loop realises that a decision is necessary then the decision class is invoked to decide what action to take. The decision class in turn invokes the making plays methods contained within the main program class.

The parser also has further functions such as working out how much money is involved in the current hand, working out who has won at the table and knowing when a new hand has started. Although this information is not necessary for the program to play poker as defined in our brief, as our poker player does not need to win games, it would be helpful for this information to be stored if this project were to be taken further by any members of the group. If a successful poker player were to be created, some of this information, especially the money involved would be used to aid the decisions made by the program.
5.10 Problems Encountered

5.10.1 Colours
When implementing the plans we had for the card recognition and finding a seat at the poker table, outlined in the interim report, we noticed that colour was going to cause a problem. This problem arises when we tried to compare a colour at a specific pixel location on the screen with what we expected to be there. When using the same machine for the testing this all worked out fine, however when it was tested on a different machine the poker player didn’t seem to thing that the colour that was clearly there to the user was actually there. We eventually traced this problem down to different colour depths being used on different machines, the program had been written assuming that it would only be run on a machine with 32-bit colour in use.

To the user the difference in the colours on the ‘Paradise Poker’ client when run under 32-bit colour or 16-bit colour was nominal, to the computer the Red, Green and Blue (RGB) values were completely different. To solve this problem we wrote a short method that would compare the colour expected with the colour found with a predefined margin of error. The difference between the 16-bit and 32-bit RGB values was included in this margin.

5.10.2 Options Window
Our program is completely reliant on the ‘Paradise Poker’ client being at a stage in the game or even loading that is where our program believes it is. If our program loses synchronisation with the poker client, it might try and generate user input that was not required at the time or not generate input that the client is expecting. Our program, in some of the cases, can deal with loss of synchronisation where it might arise. This cannot be guaranteed in every case, however, so we decided that to safe guard against this would allow a certain amount of alteration at run time to try and combat this. The two main causes of loss of synchronisation are the speed of the computer that is being used to run the poker client and our program, and secondly the Internet connection being used to access the ‘Paradise Poker’ servers.

If the computer is having trouble running both the poker client and our poker player simultaneously, we need to make sure that the poker player will wait longer for the computer accept input and to update the poker client window. Similarly if the Internet connection to the game server is not receiving the play information quickly enough, especially when a table has been selected from the lobby and the window is loading, our program might try and capture a part of the screen that is not the commentary box.

For these problems to be addressed, all of the time intervals being allowed by our poker player can be adjusted by as whole using an ‘Options Window,’ accessible from the poker players’ menu bar. This window will consist of two slider bars that can be adjusted, one for the computer’s speed and the second for the Internet connection speed. From the values that the user adjusts these slider bars to, the time intervals used for synchronisation can be calculated depending on their particular use. The inclusion of a settings window allowed us to also request from the user what the location of the ‘Paradise Poker’ client is on the computer the poker player is being run upon. Storing this information in a simple text file also allows the program to remember this information for future executions.

5.10.3 Paradise Poker Client
One problem we encountered towards the end of the implementation was to do with the layout of the Paradise Poker interface. Periodically the client program may change appearance. These appearance updates can often be minor but can cause problems with the agent collecting information, particularly when concerned with capturing text. A seemingly insignificant movement of the text commentary box by a few pixels, for example, can cause the OCR class to incorrectly capture the text, therefore it is unable to follow the game commentary and unable to follow the game. Also movement of the cards and movement of table selection screen will cause errors within the program resulting in it being unable to continue. Upon noticing these changes in the interface we are able to rectify the problem by changing the co-ordinates at which it captures the information.
6. Summary

As a group we have achieved to a good level what we set out to achieve. We have successfully managed to develop an agent that automatically loads the Paradise Poker client, to search the relevant play money tables and select an available table, subsequently joining the table by selecting an unoccupied seat. The agent has been written to observe the on-line game and keep track of that game in the form of monitoring the text dialog box, recognising what cards are held and acknowledging when it is our turn. The player is also able to make its own decisions based on the cards at the table and the state of the game it is in, for example, when someone has raised. The poker player is not sophisticated enough to be a serious rival to a human player, this wasn’t the scope of our project, however it was made clear that the player needs to make its own decisions, rather than playing randomly making use of and interpreting the information it has gathered in order to appear to act intelligently.
7. Bibliography

[8]  Schaeffer, J., Billings, D., Peña, L and Szafron, D Learning to Play Strong Poker
8. Appendices

8.1 Paradise Poker Windows

Introduction: Lobby Screen

Seat: Table screen
Log In:  Member's log in screen

Account:  Member's account screen
8.2 Legal Hands

Four of a kind: All four cards of the same kind

Full House: A full house is a combination of three of a kind and a pair

Flush: A flush is a hand where all the cards are of the same suit

Straight: A straight is a hand in which all of the cards are consecutive, but need not to be of the same suit

Three of a kind: All three cards of the same kind

Two Pair: Two separate pairs

Pair: One pair of two equal value cards constitutes a pair

8.3 Hand Rankings

```java
public static int handStrength = 0; //hand evaluator evaluates hand to
/*nothing = 0
pair = 1
*2 pair = 2
*3 of kind = 3
*straight = 4
*flush = 5
*full = 6
*4 of kind = 7
*/
```
8.4 GUI