Class Dissertationintroduction {

Public static void main(String[] args){

System.out.println(

“Computer poker – from precise probabilities to calculated bluffing.

\n

Student number: 292418

\n

Ceri De Lloyd

\n

Supervisor: Dr.Kullmann”

)

}

}

Contents

[Introduction 3](#_Toc302932009)

[Poker: a brief history: 5](#_Toc302932010)

[The poker program: 8](#_Toc302932011)

[Poker Games: 11](#_Toc302932012)

[Implementing the poker program: 17](#_Toc302932013)

[Storage of the poker game files in C++ and Java: 20](#_Toc302932014)

[Changes: 22](#_Toc302932015)

[Poker algorithms: 24](#_Toc302932016)

[Morton’s Theorem: 27](#_Toc302932017)

[M-Ratio: 31](#_Toc302932018)

[Q-Ratio: 32](#_Toc302932019)

[Mathematical probabilities and possibilities: 33](#_Toc302932020)

[Fold Equity: 35](#_Toc302932021)

[Probabilities in Poker: 37](#_Toc302932022)

[Alternative software and poker programs: 38](#_Toc302932023)

[CS.Alberta.ca and poker programming: 40](#_Toc302932024)

[Cs.mcgill.ca and poker programming: 49](#_Toc302932025)

[Coding the wheel poker programming: 50](#_Toc302932026)

[Caspercomsci.com and poker programming: 52](#_Toc302932027)

[My adapted poker code in C++: 53](#_Toc302932028)

[Conclusion: 54](#_Toc302932029)

[Appendices 56](#_Toc302932030)

[Bibliography: 57](#_Toc302932031)

[Books and printed resources: 57](#_Toc302932032)

[Online resources: 58](#_Toc302932033)

[Poker code in java: 60](#_Toc302932034)

[Tables of probability for Texas Hold ‘em poker: 88](#_Toc302932035)

## Introduction

The project that this paper will deal with is based upon a poker program created by Dr. Oliver Kullmann in the programming language, Java. The aim of this project will primarily be to transfer the poker program from Java into the C++ language. Whilst the primary aims of the project are of paramount importance attempts will also be made to find better and more efficient ways of creating the program in C++ and running it, effectively creating a new poker program in C++. In order to transfer the program from Java into C++ the program must first be understood in terms of its functions and what it achieves. Thankfully, Dr. Kullmann, in creating the program in Java has neatly split the program up into different parts and has provided footnotes in the program to explain what each part of the program does. In this manner it is easier to understand the poker program and make a transfer to C++. The poker program may be included in part in this section of the project and will be displayed later in the project as the transfer to C++ progresses. The poker program written in Java will be included as a full document in the appendices of this paper.

In order to understand Dr. Kullmann’s Java poker program more fully it is necessary to examine the literature behind such a program. To this end it would be a prudent course of action to examine any dissertations that deal with poker programs in any computer language as well as looking at various poker algorithms and even internet software such as that used on PKR.com, a site that allows people to play poker online[[1]](#footnote-1). There are a wide variety of poker games available to play and due to this the programming environment may need to be changed. Each poker game plays differently so it is not unreasonable to assume that a different programming environment would benefit different poker games. Obviously an important factor in moving the poker program from Java to C++ or creating my own poker program in C++ would be to get the probabilities right (poker is a game of probabilities) and to have good algorithms.

This preliminary part of the paper will deal mainly with the research behind the poker program and the software used to move it into C++ and to augment it to suit the needs of the poker game being played. Since the poker program will be implemented in the C++ programming language from the Java programming language, a number of books and sources dealing with algorithms in C++ and Java will be used[[2]](#footnote-2).

### Poker: a brief history:

In order to explain the game that is being created in C++ code, the history behind the game would be the best place to begin. Poker has been played in a variety of forms for nearly two hundred years with evidence to this coming from the English actor Joseph Crowell. Cromwell reported that poker was played in New Orleans in 1829, with a deck of 20 cards. This game had four players betting on which player's hand was the most valuable[[3]](#footnote-3). The game of poker then appears to have spread to the rest of the United States by way of riverboats on the Mississippi on which gambling was a common pastime. The gambling culture of the Mississippi and the spread of the poker game are documented well by Jonathan H. Green[[4]](#footnote-4). After the initial spread of the game, Poker began to adapt to the areas it was being played in and new rules began to come into use. The full 52-card deck was used and the concept of the flush was introduced with the draw rule being added prior to 1850[[5]](#footnote-5). It was during the American Civil War that many of the additions to the original poker game were made. These included the five card variant, also known as stud poker as well as the straight further variants to the game such as the wild card followed with others following as the years went by[[6]](#footnote-6). The poker game is responsible for a large number of phrases in the English language which are used by people in simple conversation today even by people who are unaware of the origins of these sayings at the poker tables. Some of these phrases include: *ace up one's sleeve*, *beats me*, *call one's bluff*, *cash in*, *high roller*, *poker face*, *stack up*, *up the ante*, *when the chips are down,* amongst others all began in the game of poker[[7]](#footnote-7).

The modern game of poker stems from its origins on the Mississippi and became popular during the 1970’s after a series of developments. These developments came in a variety of forms, much like the game of poker itself[[8]](#footnote-8):

* The Modern tournament style of play became popular after the World Series of Poker which began in 1970. This made modern tournament play very popular especially in American casinos.
* By the mid 1980’s, poker was being depicted in various forms of popular culture as a commonplace game. It was featured in at least 10 episodes of Star Trek: The Next Generation, for example, as a weekly event of the senior staff of the fictional ship's crew.
* During the early to mid 1990’s, casino gambling spread across the United States, taking the poker game along with it.
* The first real money online poker game was realised in 1998 when Planet Poker opened its website.
* Finally, in 1999, Late Night Poker debuted on British and later European television. This introduced poker for the first time to many Europeans and the game was taken up with gusto.

Poker's popularity experienced a major and unexpected increase yet again at the beginning of the new millennium. This was largely because of the introduction of new online poker games as well as hole-card cameras. The hole-card cameras turned the televised games into a spectator sport[[9]](#footnote-9). Viewers were now not only able to watch the tense drama of the poker game they could now play from the comfort of their own home. The Broadcasting of poker tournaments such as the World Series of Poker and also World Poker Tour brought in huge audiences for viewers of cable and satellite television distributors such as Virgin™ and Sky™. Because of this large increase in the coverage of poker events, the professional poker players became celebrities with poker fans all over the world. The fans would even enter into tournaments for the chance to compete with them thus increasing the popularity of the game even more and also the amount the game was played worldwide[[10]](#footnote-10). Television coverage also added an important new dimension to the poker professional's game. Since any given hand could now be shown on air later, even if the hand had not been revealed to those playing the game at the time, it revealed information about how particular players played the game. This information could be seen not only by those who had played at the table, but also to anyone who cared to view the broadcast[[11]](#footnote-11).

In early 2003, major poker tournament fields grew dramatically. This was due in part to people playing online or via satellite feed and winning games where the main prize was an entry into a major tournament. The 2003 and 2004 World Series of Poker champions, Chris Moneymaker and Greg Raymer won their seats to the main event by winning online games[[12]](#footnote-12). Attendance at live tournaments as well as participation in live and online cash games initially slowed due somewhat to the passage of the UIGEA in October 2006. However attendance is still growing and is far more popular today than it was prior to 2003. The growth and popularity of poker can be seen in the WSOP, which during its 2010 main even had a record 7,319 entrants. Poker has come a long, long way from its origins as a simple pastime. For further information on the WSOP and on poker in general, visit: <http://www.pokerstars.com/wsop/>.

### The poker program:

The program written by Dr. Kullmann has various notes included along with the programming to help in understanding what is going on[[13]](#footnote-13). In the main program the program requires the StdIn.java file to be read properly. This file then sets up suites, card ranks and hand ranks as well as hand size and the number of hand ranks via integers which are then made final. The suites are represented in the program by the numbers 0 to 3, card ranks by integers 0 to 12, and cards by integers 0 to 51. The integer-representation of suites and ranks are then given as indices of arrays which are created and contain the various suites, card ranks and hand ranks. The hand ranks are then ensured by inclusion of explicit constants for the 9 hand ranks which are numbered from 1 to 9, highest to lowest. Two for loops are then included which do two things; one converts a string into a suit and returns -1, if the string doesn't represent a suit whilst the other converts a string into a card-rank and returns -1, if the string doesn't represent a card-rank. A cards suit, rank and corresponding card are then calculated and the program is then asked to read a hand (and an array of cards) from standard-input which returns null if there is a parsing error but returns the hand if all is well. The program makes sure that no two cards can be the same card (no two Ace of spades for example) and then checks the hands produced against the hand ranks. A large for loop with if statements checks the hands via the 9 different ranks. Dr. Kullmann has noted that there is an alternative algorithm to use: Remark: An alternative algorithm is to first sort the hand by ranks, using the sort-algorithm from the Java-library. This would yield more compact code, however in this module (CSM-41) we use only elements from the Java-library as discussed in the lectures[[14]](#footnote-14). The program concludes with checking the various hands available and then printing various error messages to ensure the game is played correctly with no two cards being the same card or there being no players.

The next two programs provide additional testing for hand ranks and hands but are not entirely required for the poker program as they are covered by the main program just discussed. The storage program provides storage for the results of all (different) strategies against each other which are usable for the single results as well as for the statistics. The strategies covered appear to be for the Texas hold ‘em style of poker. The strategy program is utilised by the storage program to implement correct strategies. The code in the strategy file also accounts for apparent bluffing with the code instructed to choose the best hand available while the suit program. Some of these files can be run alone as they contain the public static void main(String[] args) code. One of the programs explicitly deals with the wide variety of hands that can result in differing ranks. This program assigns a rank to a (Poker) hand such that a hand wins over another hand if and only if its rank is strictly lower (in case of equality the result is a draw). The ranks are subdivided into 9 major ranks, where each is subdivided further into minor ranks. All three kinds of ranks are enumerated starting with value 1 for the highest possibility[[15]](#footnote-15). The number of minor ranks, the number of ranks between minor and major ones (set by the types of cards in a players’ hand) and the number of major ranks are all accounted for and set up via integers which are rendered final to prevent any tampering. The probability that a (strictly) better hand than the given hand occurs for a random hand is calculated to aid in the strategy of the game where the computer players are concerned. A human player will have to choose the best scenario themselves.

The next set of programs beginning with hand.java appears to deal with a completely different poker game and accordingly have a different set of algorithms to go with it. These programs do appear to be able to be used with the rest of Dr. Kullmann’s poker programs however. The programs that seem to deal with a different type of poker game being played have similar functions to the other poker programs. They store the same variables as integers; hand rank, hand, card suites etc and also shuffle the cards for randomisation. The problem with using a sorting algorithm from the Java library is that we would need to be able to "tell" that algorithm the sorting criterions; the various hand ranks, card suites and so on. In this game of poker the number of cards in a persons’ hand can be exchanged for the same number of cards to optimise the hand that the player has. The ExchangeRequest program enables this and limits the number of times a person can exchange cards (since the deck of cards is finite) and also ensures that when the player exchanges N amount of cards that they receive a randomised N amount of cards back from the deck. The last few programs deal with determining the hands of players as well as the outcomes of each round of the game, who won and with what hand before moving onto the next round of the game, gathering the discarded cards and then reshuffling and distributing them to the players.

Overall there are two distinct poker games to be seen here the Texas hold ‘em style which comprises the first 9 programs seen in the appendices The second type of poker game is more in line with standard poker and comprises the final 8 programs. Interestingly it seems that some of these programs can be run in either type of game which is not surprising as a number of functions are the same. The different styles of poker seen here in these Java programs as well as other possible poker games that can be implemented will be examined and a definitive game of poker will be chosen to be written in C++. Other poker algorithms may also be included along with the large variety of poker games that exist.

### Poker Games:

As stated there are a number of different poker algorithms based upon which type of poker game you want to play. All poker algorithms are essentially “poker calculators” which through probabilistic and statistical means determines a player's chance of winning, losing, or tying a poker hand or game[[16]](#footnote-16). Poker can be a very complex game and given the variety of poker games available to play, there are a great number of complexities to account for. Rules change, so algorithms that deal with poker games must change too. To this end most algorithms or “poker calculators” are statistical machines that deal with probabilities. Card counting is rarely used in such an algorithm although in the case of Dr. Kullmann’s Java program, a certain amount of card counting does occur[[17]](#footnote-17). There are three main styles to a poker algorithm/calculator. These are: poker advantage calculators, poker odds calculators and poker relative calculators[[18]](#footnote-18). With a poker odds calculator the player’s winning ratio is calculated. The winning ratio is defined as: the number of games won divided by the total number of games simulated in an algorithm. The poker advantage calculator does things a little differently. It calculates a player's winning ratio and then normalizes the winning ratio relative to the number of players[[19]](#footnote-19). The final type of poker algorithm that may be seen in use, an advantage calculator, provides a normalized value between -100% and +100%. This describes a player's winning change in a locked domain. A locked domain means that if a player's result is -100%, regardless of how many players there are in the game then the player will certainly lose the game. Similarly if a player's advantage is +100%, regardless of the number of players, the player will certainly win the game[[20]](#footnote-20). As stated the game changes the algorithm used and game scenario variables may include: the number of players, the game type being played (which will be discussed later), the hand a player has or may have and the cards available for any of the players in question. There is an alternative to poker calculators which are known as poker relative calculators. These display a player's winning chance relative to another player's winning chance[[21]](#footnote-21). The poker relative calculator algorithm is not normally used in a single game and they tend to be displayed on poker shows and tournaments. This is mainly done for an audience because they can provide an accurate assessment of a player's winning chance. The main algorithms that would be used in a normal poker scenario are therefore the three mentioned above and would be implemented for the variety of games that exist.

The poker calculator is also used to compute the probability of each type of 5-card hand in a poker game. This is done by calculating the proportion of hands of a single type among all possible hands. The probability of drawing a hand is then calculated by dividing the number of ways of drawing the hand by the total number of 5-card hands[[22]](#footnote-22). The following quote displays how the odds are calculated:

The odds are defined as the ratio (1/p) - 1 : 1, where p is the probability. Note that the cumulative column contains the probability of being dealt that hand or any of the hands ranked higher than it. (The frequencies given are exact; the probabilities and odds are approximate.)[[23]](#footnote-23)

The odds are calculated based upon the type of poker game and there are a large number of probabilities for each type of game. These probabilities may be discussed later in the paper. The variety of poker games include: Texas Hold ‘Em, five card stud and Omaha rules to name but a few.

In Texas Hold 'em, a player is dealt two pocket cards which are also known as down or cards. The first card can be any one of 52 playing cards in the deck and the second card can be any one of the 51 remaining cards. Therefore the possible starting hand combination for a player is 52 × 51 ÷ 2 = 1,326 combinations. The order of the cards is not important since the deck is shuffled so there are in actuality 2,652 combinations divided by two since the player holds two cards[[24]](#footnote-24). The probability of 1,326 possible starting hands can be reduced since card suits have no relative value in poker and so many hands included in the 1,326 combinations for starting hands are identical in value before the flop. The flop is dealt when all players have made their bets and is three cards dealt face up from the deck. The only factors that may determine how good a player’s starting hand is are the ranks of the cards and whether the cards share the same suit. Whilst suits have no intrinsic value in poker, having five cards of the same type (Clubs, Diamonds, Hearts, Spades) results in a flush[[25]](#footnote-25). Given that the probability of being dealt a hand of each given shape is different then out of 1,326 combinations, there can be 169 distinct starting hands. This is possible if the hands are grouped into three shapes. This would give:

13 pocket pairs, 13 × 12 ÷ 2 = 78 suited hands and 78 unsuited hands; 13 + 78 + 78 = 169.[[26]](#footnote-26)

Here are the probabilities and odds of being dealt various other types of starting hands.

Taken from [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011.

|  |  |  |
| --- | --- | --- |
| Hand | Probability | Odds |
| AKs (or any specific suited cards) | 0.00302 | 331 : 1 |
| AA (or any specific pair) | 0.00452 | 220 : 1 |
| AKs, KQs, QJs, or JTs (suited cards) | 0.0121 | 81.9 : 1 |
| AK (or any specific non-pair incl. suited) | 0.0121 | 81.9 : 1 |
| AA, KK, or QQ | 0.0136 | 72.7 : 1 |
| AA, KK, QQ or JJ | 0.0181 | 54.3 : 1 |
| Suited cards, jack or better | 0.0181 | 54.3 : 1 |
| AA, KK, QQ, JJ, or TT | 0.0226 | 43.2 : 1 |
| Suited cards, 10 or better | 0.0302 | 32.2 : 1 |
| Suited connectors | 0.0392 | 24.5 : 1 |
| Connected cards, 10 or better | 0.0483 | 19.7 : 1 |
| Any 2 cards with rank at least queen | 0.0498 | 19.1 : 1 |
| Any 2 cards with rank at least jack | 0.0905 | 10.1 : 1 |
| Any 2 cards with rank at least 10 | 0.143 | 5.98 : 1 |
| Connected cards (cards of consecutive rank) | 0.157 | 5.38 : 1 |
| Any 2 cards with rank at least 9 | 0.208 | 3.81 : 1 |
| Not connected nor suited, at least one 2-9 | 0.534 | 0.873 : 1 |

When calculating the probabilities for Texas Hold 'Em, there are two ways to implement the calculation. The first way is to evaluate a given condition and determine the number of outcomes that satisfy that condition. Then divide the number of outcomes being evaluated by the total number of possible outcomes. For example, there are six outcomes for being dealt a pair of aces in Hold' em: (Ace of clubs and Ace of hearts), (Ace of Spades and Ace of diamonds), (Ace of Spades and Ace of clubs), (Ace of hearts and Ace of diamonds), (Ace of hearts and Ace of Spades) and (Ace of diamonds and Ace of clubs)[[27]](#footnote-27). Since two cards are being picked the odds are the same as when a player’s starting hand is dealt which means that there are 1,326 possible outcomes. The second way of implementing the calculation for probability is to use conditional probabilities. In other more complex situations a decision tree may be used[[28]](#footnote-28). More often than not, the key to determining the probability is selecting the best approach for a given problem.

In Texas Hold ‘Em, after the flop, the number of possible hands any player or opponent can have is reduced by three. The three cards dealt from the deck are also known as community cards which are revealed on the flop. This reduces the probability of the player’s hands to 47 × 46 ÷ 2 = 1,081 when the flop comes down. In this game of poker, each round is known as a match up where each player competes against the next and bets on the outcome. The total number of match ups in a game of Texas Hold ‘Em can is divided by the number of ways that hands between two players can be distributed. This then gives the number of unique match ups. Since the number of distinct starting hands is limited to 169 versions then there can only be so many distinct match ups or match ups where two players go head to head:

169 × 1,225 = 207,025 distinct head-to-head match ups.[[29]](#footnote-29)

Before the flop comes down the players may bet and a number will fold. It is generally assumed that not everyone will fold and there will be at least two players left when the flop is drawn. The problem of computing the odds of head to head match ups is simple when using brute force searches via computer although there are severe limitations on brute force searches. There are a large number of software programs available for purchase or on open-source that can compute the odds in seconds. A full analysis of the head to head match ups is exhaustive and each game with a player winning must be evaluated which gives a large number of results:

1,712,304 × 207,025 = 354,489,735,600 (≈354 billion) results.[[30]](#footnote-30)

Depending on the number of players in the game the results will be even higher, up to 21 Octillion! If one trillion combinations were evaluated every second, it would take over 670 million years to evaluate all of the combinations for hands and for games where a player wins[[31]](#footnote-31). It is possible to significantly reduce the total number of combinations by deleting combinations that are identical but the total number of situations is still too great to be evaluated by brute force alone. Since brute force is not enough, most software programs calculate the probabilities and expected values for Texas Hold 'Em hands against multiple players. This is done by simulating the play of thousands or even millions of hands to determine statistical probabilities which is similar to what the first of Dr. Kullmann’s poker programs does. The full array of probabilities for Texas Hold ‘Em can be seen in tables in the appendices and this was the game I would initially have coded the program for in C++ simply because I have played it and enjoy it. As stated previously, it is prudent to remain with the game that the code plays for simplicity.

Other types of poker games exist such as five card stud or Omaha rules poker. These games use a similar type of probability to that seen in Texas Hold ‘Em particularly for dealing the players their starting cards. In five card stud however different algorithms are required since the number of cards in a players hand increases up to five and the players may change the cards they have for ones in the deck (at random and unseen). Omaha rules are different again and since I have little to no experience playing these poker games I would prefer to implement one that I have experience with.

Poker games are not just played in casinos and clubs for money but are also played on computers as standalone games. They are also played online for money much as in any casino. Sites such as PKR.com which is a dedicated online poker website and others like Sky.com have poker games available. These online sites operate on the same algorithms we have been discussing. Probabilities of hands and cards are calculated and the algorithm must run correctly since human interaction is as high as any normal game of poker played in reality. The players interact with the online game in relatively the same way that they would interact with a game played in a casino. This is especially true on PKR.com where the player creates an avatar of themselves to sit around a 3-dimensional table where the poker game is played. The avatar has a number of realistic responses that the player can perform when they are receiving cards (in five card stud), bluffing, betting or winning/losing a hand or game and in this manner the player feels more drawn into the virtual world. In years to come poker websites may offer even more services to the players on their sites such as team-speak which would allow the players to communicate with each other. This could be useful for parties of friends who want to play but are a large distance from each other. Poker games online are becoming more and more connected with the players on their systems. The algorithms employed ensure a fully realistic game and aid in protecting a player from cheaters[[32]](#footnote-32).

## Implementing the poker program:

In implementing the poker program in C++ from Java a number of software programs have been considered. One of the main considerations is Doxygen. Doxygen is a documentation generator created mainly for the programming languages C, C++, C#, Fortran, Java, Objective-C, PHP and Python[[33]](#footnote-33). Since it supports C++ easily then it was a good choice to generate the poker program. Doxygen runs on most UNIX style systems, including the Apple Mac Snow Leopard system, and on Microsoft Windows operating systems. Doxygen is primarily a tool for writing software reference documentation[[34]](#footnote-34). The documentation of the program is written within the code, this makes it quite easy to keep up to date and makes it easy to understand what the program is trying to accomplish. Doxygen can make cross references to documentation and code at the same time, so the reader of a document can easily refer to the code that has been written. The operating system KDE, a Linux based operating system uses Doxygen for certain parts of its documentation. The software KDevelop has built-in support for it[[35]](#footnote-35). Doxygen is free software, released under the terms of the GNU General Public License and is readily available online for download at various websites.

Taken from <http://en.wikipedia.org/wiki/Doxygen> accessed 19th April 2011.

The following illustrates how a C++ source file can be documented. Make sure the parameter EXTRACT\_ALL in the Doxyfile is set to YES.

/\*\*

\* @file

\* @author John Doe <jdoe@example.com>

\* @version 1.0

\*

\* @section LICENSE

\*

\* This program is free software; you can redistribute it and/or

\* modify it under the terms of the GNU General Public License as

\* published by the Free Software Foundation; either version 2 of

\* the License, or (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful, but

\* WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

\* General Public License for more details at

\* http://www.gnu.org/copyleft/gpl.html

\*

\* @section DESCRIPTION

\*

\* The time class represents a moment of time.

\*/

class Time {

public:

/\*\*

\* Constructor that sets the time to a given value.

\*

\* @param timemillis Number of milliseconds

\* passed since Jan 1, 1970.

\*/

Time (int timemillis) {

// the code

}

/\*\*

\* Get the current time.

\*

\* @return A time object set to the current time.

\*/

static Time now () {

// the code

}

};

An alternative approach that is preferred by some for documenting parameters is shown below. It will produce the same documentation.

/\*\*

\* Constructor that sets the time to a given value.

\*

\*/

Time (int timemillis ///< Number of milliseconds passed since Jan 1, 1970.

)

{

// the code

}

Richer markup is also possible. For instance, add equations using LaTeX commands:

/\*\*

\*

\* An inline equation @f$ e^{\pi i}+1 = 0 @f$

\*

\* A displayed equation: @f[ e^{\pi i}+1 = 0 @f]

\*

\*/

The KDE Linux operating system may be used in place of a Windows based operating system to implement Doxygen and the java program in C++.

### Storage of the poker game files in C++ and Java:

Since there will be a large amount of code and a large number of files written for various rules and probabilities then a place to store them all so that they can easily be called on and compiled is needed. For this purpose I chose the GitHub software. Git (the software GitHub uses) is a distributed revision control system with an emphasis on speed[[36]](#footnote-36). The Git software was initially designed and developed by Linus Torvalds for Linux kernel development which is one reason that the KDE Linux Operating system may be used instead of a Windows based OS. In GitHub you can create working directories which are full-fledged repositories which also store a complete history and also enable full revision tracking capabilities. These abilities are not dependent on network access or a central server[[37]](#footnote-37). GitHub is free software that is distributed under the terms of the GNU General Public License version 2 which enables it to be made available via open source on various websites. When the Git software was being written, its creator Linus Torvalds wanted to make a distributed system similar to BitKeeper but with key differences. None of the available free systems met his needs so he decided to create his own similar to James Gosling who wrote Java after becoming frustrated with C++[[38]](#footnote-38). Several of the key aspects of Git included: Very strong safeguards against corruption, either accidental or malicious and very high performance. On April 3rd 2005 Torvalds began developing Git and by April 29th he had realised his goals.

The Git software was benchmarked and recorded patches to the Linux kernel tree at the rate of 6.7 per second.[[39]](#footnote-39)

While the main influence for Git was BitKeeper, Torvalds deliberately avoided more conventional approaches which led to the creation of a unique design. The Git software engine supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

Like BitKeeper, Git can give each developer or each user a local copy of the entire development history of the programs stored in its repositories. All changes are copied from one repository to another while the main files remain unchanged and stored[[40]](#footnote-40). There are a large number of compatibilities between the Git software and various other systems/protocols. The Git software also enables the use of IDE plug-ins to access other Git repositories. The software is useful for the efficient handling of large projects and is fast and simple to use with a useful feature being that Git does not get slower as the project history and the project itself grows larger[[41]](#footnote-41). The software also runs on Microsoft Windows and there is a port to Microsoft Windows available which is called msysgit. This is somewhat slower than the Linux version but is still acceptably fast with minor awkwardness’s and is reported to be usable in production. Some of these awkwardness’s are that some commands are not yet available from the GUIs, and must be invoked from the command line[[42]](#footnote-42). The GitHub source is the one I have decided to use for the purposes of this project.

Taken from <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

The following websites provide free source code hosting for Git repositories:

BerliOS

GitHub

Gitorious

Sourceforge

GNU Savannah

Project Kenai

GitEnterprise

### Changes:

Whilst undertaking this project I stated previously that it was my intent to utilise a number of software pieces in order to implement the code as well as store it. The software that I intended to use to store the project files so that they could be seen and used by others interested in creating a computer poker program was GIThub. As previously mentioned in this paper, GIThub is an open source online storage website where users can upload their data and files to a repository they have created. Users of the website can then see the uploaded files and view/download them for free to use as part of their own projects. GIThub was created by the same people who made the Linux operating system and so functions in tandem with it. Unfortunately the main operating system I have at my disposal for this project is Windows as attempting to implement Linux on any of my home computers has caused them to crash. This is unfortunate as the Linux system is easily obtained (it is open source) and GIThub would have been useful for allowing others to view the project files. For the simple reason that I am essentially unable to run Linux and therefore GIThub I have decided against using GIThub to store the files online. I have made it known on the GIThub repository that I managed to create that if anyone wishes to view my files they may email me for them. The files are all included on the CD that accompanies this document.

I also initially intended to use the compiler known as Doxygen to run my programs in C++. Doxygen is also open source software and is easily made available on the internet. In this case I chose against Doxygen and have instead used Microsoft Visual C++ 2005 Express Edition to compile and create the files. The reasoning behind it is quite simple and possibly a little childish; I was allowed to download for free the Microsoft compiler by the MSDNAA and decided to give it a try before using Doxygen. I managed to get to grips with the functionality of Microsoft Visual C++ 2005 quite easily and decided not to complicate matters by using Doxygen as well.

As the last change made to the project, I had initially expressed an interest in creating a more functional poker program which dealt with the Texas Hold ‘Em version of the game as this is the version most people are playing these days and is the version of poker that I am most familiar with and enjoy playing. Since then I have decided to simply stick with the original game that the code by Dr. Kullmann plays for matters of simplicity. Creating a probability and odds calculator to deal specifically with a single game of poker just because I happen to enjoy it is not sensible when the code already deals with a given style of poker play. Despite this however it may be possible to alter the code in later versions to deal with any and all styles of poker game available.

## Poker algorithms:

In this chapter of the project, various Poker algorithms will be examined to help in understanding what Dr. Kullmann’s Java code does and what it will do once it is placed into the C++ language. Examining other algorithms may also aid in developing a better poker program or it may help the reader in creating a poker program suited to their needs.

The fundamental theorem of poker is a principle first articulated by David Sklansky. Sklansky believes that this theorem expresses the essential nature of poker as a game of decision-making in the face of incomplete information[[43]](#footnote-43).

Taken from <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011:

|  |  |  |
| --- | --- | --- |
| “ | *Every time you play a hand differently from the way you would have played it if you could see all your opponents' cards, they gain; and every time you play your hand the same way you would have played it if you could see all their cards, they lose. Conversely, every time opponents play their hands differently from the way they would have if they could see all your cards, you gain; and every time they play their hands the same way they would have played if they could see all your cards, you lose.* | ” |

The fundamental theorem is written in plain English or common language but its formulation is based on mathematical reasoning. Each and every decision that is made in a poker game can be analyzed in terms of the expected value of the payoff of any given decision[[44]](#footnote-44). The correct decision to make in any given situation should be the decision that has the largest expected value. If you were able to see all your opponents' cards, you would always be able to calculate the correct decision with mathematical certainty since you would be able to accurately predict how your hand would stand up to theirs. The Fundamental Theorem implies that the less you deviate from these correct decisions, the better your expected long-term results. This is the basic mathematical expression of this theorem[[45]](#footnote-45).

Taken from <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011:

* 1. ***An example:***

*“Suppose Alice is playing limit Texas hold 'em and is dealt* ***9♣ 9♠*** *under the gun before the flop. She calls, and everyone else folds to the big blind who checks. The flop comes* ***A♣ K♦ 10♦****, and the big blind bets.*

*She now has a decision to make based upon incomplete information. In this particular circumstance, the correct decision is almost certainly to fold. There are too many turn and river cards that could kill her hand. Even if the big blind does not have an* ***A*** *or a* ***K****, there are 3 cards to a straight and 2 cards to a flush on the flop, and he could easily be on a straight or flush draw. She is essentially drawing to 2 outs (another* ***9****), and even if she catches one of these outs, her set may not hold up.*

*However, suppose she knew (with 100% certainty) the big blind held* ***8♦ 7♦****. In this case, it would be correct to raise. Even though the big blind would still be getting the correct pot odds to call, the best decision is to raise. (Calling would be giving the big blind infinite pot odds, and this decision makes less money in the long run than raising.) Therefore, by folding (or even calling), she has played her hand differently from the way she would have played it if she could see her opponent's cards, and so by the Fundamental Theorem of Poker, her opponent has gained. She has made a "mistake", in the sense that she has played differently from the way she would have played if she knew the big blind held* ***8♦ 7♦****, even though this "mistake" is almost certainly the best decision given the incomplete information available to her.”*

This example illustrates one of the most important tactics of playing poker; to make your opponents make mistakes in betting or playing their hand. In the hand given in the example, the person with the big blind has used a semi-bluff — he has bet a hand, hoping Alice will fold, but he still has outs even if she calls or raises. He has induced her to make a mistake through deception and should be able to capitalise on this mistake[[46]](#footnote-46).

The Fundamental Theorem of Poker applies to all heads-up decisions; heads-up being a poker term, but it does not apply to all multi-way decisions. The Theorem cannot do this because each opponent of any player can make an incorrect decision, but the "collective decisions" of all of the opponents works against the singular player. This type of situation occurs mostly in games with multi-way pots, when a player has a strong hand, but several opponents are chasing the player with weaker hands[[47]](#footnote-47). A good example is a player with a deep stack, that is a player with a large number of poker chips. The deep-stacked player then makes a play that favours a short stacked opponent because the deep stacked player can extract more expected value from the other opponents who are deep stacked. This type of situation can sometimes be referred to as implicit collusion[[48]](#footnote-48). The Fundamental Theorem of Poker is simply expressed but is quite difficult to understand especially in terms of its application to a game that has an almost countless variety of circumstances that each poker player may encounter. Applying the Theorem in order to maximise its effectiveness and ensure victory would require a great deal of patience, experience and skill.

### Morton’s Theorem:

Morton's theorem is a poker principle expressed by Andy Morton. Morton is credited with having done this in a Usenet poker newsgroup which is cited below in the given example. The Theorem states that in multi-way pots, a player's expectation of an outcome may be maximized by an opponent making a correct decision as opposed to a wrong one[[49]](#footnote-49). The most common application of Morton's theorem seems to occur when one player holds the best hand with two or more opponents on draws. Following the Theorem, the player with the best hand might make more money in the long run when an opponent folds to a bet. The player with the best hand might make more money,

*“even if that opponent is folding correctly and would be making a personal mistake to call the bet[[50]](#footnote-50)*.”

This type of situation can be referred to as implicit collusion.

Morton's theorem can and should be contrasted with the fundamental theorem of poker discussed above. The discrepancy between the two "theorems" occurs because of the presence of more than one opponent which is the very nature of the poker game, except perhaps at the end where there may only be one opponent left. The Fundamental Theorem always applies heads-up and does not always apply in multi-way pots[[51]](#footnote-51). The scope of Morton's theorem in multi-way situations appears to be a subject of controversy. In his post on the Usenet group, Morton apparently expressed the belief that his theorem is generically applicable in multi-way pots, so that the fundamental theorem rarely applies except for heads-up situations[[52]](#footnote-52).

* 1. Taken from <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011:
  2. **An example**

“The following example is credited to Morton, who first posted a version of it on the Usenet newsgroup rec.gambling.poker.

Suppose in limit hold'em a player named Arnold holds **A♦K♣** and the flop is **K♠9♥3♥**, giving him top pair with best kicker. When the betting on the flop is complete, Arnold has two opponents remaining, named Brenda and Charles.

Arnold is certain that Brenda has the nut flush draw (for example **A♥T♥**, giving her 9 outs), and he believes that Charles holds second pair with a random kicker (for example **Q♣9♣**, 4 outs — not the **Q♥**). The rest of the deck results in a win for Arnold. The turn card is an apparent blank (for example **6♦**) and the pot size at this point is *P*, expressed in big bets.

When Arnold bets the turn, Brenda, holding the flush draw, is sure to call and is almost certainly getting the correct pot odds to do so (note that, due to large reverse implied pot odds, this would not be true in a no limit game).

Once Brenda calls, Charles must decide whether to call or fold. To figure out which action he should choose, we calculate his expectation in each case.

This depends on the number of cards among the remaining 42 that will give him the best hand, and the current size of the pot. (Here, as in arguments involving the fundamental theorem, we assume that each player has complete information of their opponents' cards.) Charles doesn't win or lose anything by folding.

When calling, he wins the pot 4/42 of the time, and loses one big bet the remainder of the time. Setting these two expectations equal and solving for *P* lets us determine the pot size at which he is indifferent to calling or folding.

When the pot is larger than this, Charles should continue; otherwise, it's in his best interest to fold.

To figure out which action on Charles' part Arnold would prefer, we calculate Arnold's expectation the same way.

Arnold's expectation depends in each case on the size of the pot (in other words, the pot odds Charles is getting when considering his call). Setting these two equal lets us calculate the pot size *P* where Arnold is indifferent whether Charles calls or folds.

When the pot is smaller than this, Arnold profits when Charles is chasing, but when the pot is larger than this, Arnold's expectation is higher when Charles folds instead of chasing.

Hence, there is a range of pot sizes where both:

(a) it's correct for Charles to fold, and (b) Arnold makes more money when Charles (correctly) folds, than when he (incorrectly) chases.

This can be seen graphically below.

|

C SHOULD FOLD | C SHOULD CALL

|

v

|

WANTS C TO CALL | WANTS C TO FOLD

|

v

+---+---+---+---+---+---+---+---+---> pot size *P* in big bets

0 1 2 3 4 5 6 7 8

XXXXXXXXXX

^

"PARADOXICAL REGION"

The range of pot sizes marked with the X's is where Arnold wants Charles (C) to fold correctly, because he loses expectation when Charles calls incorrectly.”

In the above example, when Charles calls in the "paradoxical region", he is paying too high a price for his weak draw. Arnold is no longer the sole benefactor of that high price as Brenda is now taking Charles' money when Brenda makes her flush draw. Arnold still stands the risk of losing the whole pot, but compared to the case where he is heads up with Charles, he is no longer getting 100% of the compensation from Charles' loose calls[[53]](#footnote-53).

“The existence of a middle region of pot sizes, where a player wants at least some of their opponents to fold correctly, seems to explain the standard poker strategy of thinning the field as much as possible whenever a player thinks they hold the best hand. Even opponents with incorrect draws cost a player money when they call their bets, because part of these calls end up in the stacks of other opponents drawing against them[[54]](#footnote-54).”

Because Arnold is losing expectation from the call made by Charles, it makes sense that the total of all the other opponents (Brenda, Charles and anyone else left) must be gaining from that call made by Charles. Simply put, if Brenda and Charles were to meet outside in a car park after the game and split their profits, they would have been colluding against Arnold. This is the situation that is sometimes referred to as implicit collusion[[55]](#footnote-55). Implicit Collusion should be contrasted with the term called *schooling*. Schooling can occur when many opponents correctly call against a player with the best hand, whereas implicit collusion occurs when an opponent incorrectly calls against a player with the best hand[[56]](#footnote-56). One conclusion of Morton's theorem is that, in a loose hold 'em game, (A version of Texas Hold ‘Em) the value of all suited hands goes up because they are the type of hand that will benefit from implicit collusion.

### M-Ratio:

The M-Ratio term was invented and named by Paul Magriel. In no-limit or pot-limit poker, a player's M-ratio (also called "M number", "M factor" or just simply "M") is a measure of the health of their chip stack as a function of the cost to play each round. In simple terms, a player can sit passively in the game, making only compulsory bets, for *M* laps of the dealer button before running out of chips. A high *M* means the player can afford to wait a number of rounds before making a move. The concept of the M-ratio applies primarily in tournament poker. In a game that is played with and for money, a player could in principle manipulate their M-ratio at will, simply by purchasing more chips[[57]](#footnote-57). A player with a low M-ratio must act soon or be weakened by the inability to force other players to fold with aggressive raises. Another term for this is bullying the pot, where a player with the most chips uses their monetary status to force large bets and make other players fold.

Taken from <http://en.wikipedia.org/wiki/M-ratio> accessed 19th April 2011:

Dan Harrington studied the concept in great detail in Harrington on Holdem*: Volume II The Endgame*, defining several "zones" in which the M-ratio may fall:

|  |  |  |
| --- | --- | --- |
| M-ratio | Zone name | "Optimal" strategy |
| M ≥ 20 | Green zone | Most desirable situation, freedom to play conservatively or aggressively as you choose. |
| 10 ≤ M < 20 | Yellow zone | Must take on more risk, hands containing small pairs and small suited connectors lose value |
| 6 ≤ M < 10 | Orange zone | Main focus is to be first-in whatever you decide to play, important to preserve chips |
| 1 ≤ M < 6 | Red zone | Your only move is to move all-in or fold |
| M < 1 | Dead zone | You are completely dependent on luck to survive, the only strategy is to push all-in into an empty pot |

A player’s M-ratio is multiplied by the percentage of players remaining at the table and assumes a ten-player table to be "full". Therefore, for a player with a "simple M ratio" of 9 at a five player table, the effective M is 4:5. This would mean that although the player's simple M value places them in the orange zone (as seen in the table above), their effective M value indicates a shift in playing style that would be appropriate for the red zone. In essence, ten times the effective M denotes the expected number of hands a player can let pass before running out of chips[[58]](#footnote-58).

### Q-Ratio:

The Q-ratio (also known as Q number or simply just Q) is used in various poker tournament strategies. It is also known by some as the "weak force." The Q-ratio describes the relation of the player's stack of chips to the tournament players' average stack of chips. A low Q-ratio (for example less than 1) indicates a below-average chip stack, implying disadvantage against opponents. The Q-ratio can and should be used as an addition to the M-ratio (sometimes known as the"strong force") and usually doesn't play a large role in tournament decision-making.

Q-ratio on freezeouts (a poker term) is calculated using the following method. For example, in a tournament starting with fifty players who have 10,000 chips, of which thirty players have been eliminated, one player has 20,000 chips. This player's accumulation of chips has not kept pace with the elimination of players, and their chip stack is now below average[[59]](#footnote-59).

## Mathematical probabilities and possibilities:

Due to the nature of poker, a great number of different possibilities arise when playing the game. These probabilities can take the form of what hands the opponents around the table may have, what cards will give the best outcomes, what player has the best chances of winning given the amount of chips they have left as well as the odds that someone will fold before the next turn (fold equity), to name but a few. Poker is a game that is normally played with a deck of 52 cards. Sometimes, in certain cases such as variant games or in some casino rules, additional cards known as "jokers" can also be used. In the straight or draw games of poker, all players are dealt a hand of five cards. Depending on the variant of game played, players may discard and then redraw cards, trying to improve their hands. Bets are placed at each discard step until the display round where each player shows their hand. The number of possible distinct five-card hands is therefore equal to the number of possible ways of picking five cards out of a deck of 52[[60]](#footnote-60).

It is also possible to calculate the frequency of specific hand types occurring using binomial co-efficiency. In poker there are typically a series of special names for specific hand types. These hand types are placed into a ranking system. A royal flush is an ace, king, queen, jack, and 10, all of one suit and is the highest possible hand to obtain(also one of the most difficult). A straight flush is five consecutive cards all of the same suit (but not a royal flush), where an ace may count as either high or low. A full house is three-of-a-kind and a pair. A flush is five cards of the same suit (but not a royal flush or straight flush). A straight is five consecutive cards in order of number (but not a royal flush or straight flush), where an ace may again count as either high or low[[61]](#footnote-61).

The probabilities of a player being dealt five-card poker hands of any given type (before discarding and with no jokers) on the initial deal (the start of the game) are given in Packel 1981. Other games are also discussed[[62]](#footnote-62).

A discussion of the variety of poker games along with the large number of probabilities and mathematical problems that go along with these games may be found at: <http://mathworld.wolfram.com/Poker.html>. This site allows the reader to accurately examine mathematical possibilities of poker games. Another site that helps in examining the mathematical probabilities of various poker hands may be seen at <http://people.math.sfu.ca/~alspach/comp18/>. Both sites may aid in creating computer poker programs and the Alspach website has downloadable PDF files that contain the mathematical probabilities of poker hands. These will be included on the CD provided with this document.

### Fold Equity:

Fold equity is a poker strategy concept. It is especially important when a player becomes short-stacked (has few chips left) in a no limit or pot limit tournament. It is the degree of “even-handedness” a player can expect to gain due to the opponents folding to the bets of the player. Fold Equity relies on a two part formula, the first part can be estimated based on reading opponents and their previous/current actions. The second part of the equation is the equity obtained when the opponent(s) fold to the players’ raise (i.e. the total current pot). The current pot is then calculated minus the equity resulting if the opponents call the raise made by the player[[63]](#footnote-63).

The post-raise pot should be larger than the current pot and so fold equity can be positive and also negative. The concept of fold equity becomes important for short stacks for the following reason: Opponents can be considered likely to call all-ins with a certain range of hands (or in certain cases will go all-in without looking at their hands should they be so short stacked as to almost be out of the game). When the player has the opportunity to use a large percentage of their stack to make the call for all-in or raise, the range of hands that a player can be expected to go all-in on or make a raise can be expected to be quite narrow (the range should include all the hands the caller expects to win in an all-in situation against the bettor).

As the percentage of the stack needed to be called by the player becomes increasingly lower, the range of cards the caller will need to win against the bettor becomes wider, and so they should become less likely to fold (although since they are human they may fold anyway. This situation is only absolutely true if the player is playing a game against AI bots). As the range of cards needed to win decreases, so too does the fold equity. At some point the caller should only need a sufficiently small percentage of their remaining stack to call the all-in that they will do so with any two cards. This covers the concept of “bullying the pot” whereby the caller has such a large amount of chips compared to the opponent that they can make the opponent fold by raising the pot as much as needed. It is at that point that the all-in bettor has no fold equity[[64]](#footnote-64). The below example displays Fold Equity and the way in which it functions:

Taken from <http://en.wikipedia.org/wiki/Fold_equity> accessed 19th April 2011:

An Example

“Alice holds **A♣ 6♥** playing against one opponent, Brian, who holds **2♥ 2♦**. The flop is **9♠ 7♣ 3♦**.

At this point, Alice has a pot equity of 31.5% and Brian has a pot equity of 68.5%. In other words, if there were no further betting and both players simply turned up their hands and were dealt the turn and river cards, Alice would be 31.5% likely to win the pot.

Because Brian's hand is so weak, though, and many hands that Alice might be playing can beat him easily, he may be 70% likely to fold facing a pot-sized bet. As such, Alice's fold equity is 70% \* 68.5% = 48.2925%. Consequently, Alice can consider that her hand equity *if she bets* will equal 31.5% + 48.2925% = 79.7925%.”Probabilities in Poker:

In any poker game the probability of many events can be determined by direct calculation. The mathematical qualities discussed above are used to determine these calculations. In most cases of poker, the probabilities and odds seen in the game are similar since each game lends itself to its variants. When calculating the probabilities for a card game such as Omaha or Texas Hold ‘Em, there are two basic approaches:

1. Determine the number of outcomes that satisfy the condition being evaluated and divide this by the total number of possible outcomes.
2. Use conditional probabilities, or in more complex situations, a decision graph[[65]](#footnote-65).

The key to determining probability can be done by simply selecting the best approach for any given problem to which the problems and possibilities of poker are no different.[[66]](#footnote-66)

Poker probabilities have been discussed previously for the Texas Hold ‘Em game as well as the Omaha game and the tables in the appendices display the large variety of probabilities for each of these games. The main problems of working out probabilities in poker are to calculate them via mathematical algorithms as seen above and in tables in the appendices. Utilising these mathematical aids a reader should find them useful for implementing such probability in a computer poker program.

## Alternative software and poker programs:

Whilst I have elected to remain relatively true to the poker program created by Dr. Kullmann whilst transferring it from Java to C++, a number of alternative poker programs do exist as open-source software which, given more time, may have been applied to this project. Such programs and software exist in the form of poker calculators as well as in the form of actual games. Many of these currently existing programs are written in Java or Visual Basic but a number are written in C, C# and also in C++. These programs could have been implemented as part of this project in order to make the code applicable to all poker games and also to perhaps provide a Graphical User Interface or GUI so that actual poker games could be played and probabilities calculated at the same time. A number of the sources that display such programs offer the user the possibility of downloading free demonstration packages that show what the programs are capable of. Unfortunately due to the extremely lucrative nature of online poker programs, the demo versions of these programs and software are the only things that are free[[67]](#footnote-67). Some software and programs are available for free however and these will be examined shortly. A number of websites are available that provide lists of websites that contain such poker software and programs, open-source or otherwise.

These “list” websites include: <http://www.pagat.com/poker/software.html> which provides a long list of websites that enable a user to find Analysis software - which can be used for off line analysis poker probabilities and strategies, Online assistants - programs that you can run locally to provide help while you play online, Performance trackers - programs to help you record and keep track of your poker results, League and tournament software - programs for organisers of home poker leagues and tournaments and also software dealing specifically with Computer poker - programs with which you can play poker against a computer. The following programs allow a user to analyse probabilities in various situations, or input hands to obtain a recommendation on how they should be played[[68]](#footnote-68). A similar website is: <http://www.dmoz.org/Games/Gambling/Poker/Software_and_Tools/> although as stated previously a number of links to websites from this site do not provide open-source software. The majority of tools found through these links are unfortunately not free. Many of them are only obtained by paying a fee (usually in dollars $). The software made available by these websites is also mainly for use in tandem with online poker games and so would not have been useful for this project[[69]](#footnote-69).

### CS.Alberta.ca and poker programming:

As mentioned a number of software and programs are available for free and have been created by both users interested in the game as well as by academic sources. The University of Alberta Computer Poker Research Group is aiming to create computer programs that play poker better than any human being and are doing so as a test-bed for doing good science.

“There are many core artificial intelligence and computer science problems that need to be solved to make an excellent player, and games like poker are a fun and controllable way to examine these problems[[70]](#footnote-70).”

A more in depth view of why this University is attempting this is given in a dissertation written by one of their colleagues, Darse Billings and is made available in PDF format on the website[[71]](#footnote-71). The University website supplies a number of other PDF files that document the advancement made in creating their poker program. Also given are a series of links to source code as well as a number of text document files also in the form of PDF files that show what the university is currently looking at in terms of other poker programs and software[[72]](#footnote-72). Of major interest on the University of Alberta website is a .tar file that contains the university’s efforts to create their poker program. The program is predominantly written in C and attempts to play a poker game as accurately as possible[[73]](#footnote-73). An example of the code written is given below. This example was taken from the game.c file given in the .tar files.

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <assert.h>

#include "game.h"

#include "rng.h"

#include "evalHandTables"

static enum ActionType charToAction[ 256 ] = {

/\* 0x0X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x1X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x2X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x3X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x4X \*/

invalid, invalid, raise, call, invalid, invalid, fold, invalid,

invalid, invalid, invalid, call, invalid, invalid, invalid, invalid,

/\* 0x5X \*/

invalid, invalid, raise, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x6X \*/

invalid, invalid, raise, call, invalid, invalid, fold, invalid,

invalid, invalid, invalid, call, invalid, invalid, invalid, invalid,

/\* 0x7X \*/

invalid, invalid, raise, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x8X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0x9X \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xAX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xBX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xCX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xDX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xEX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

/\* 0xFX \*/

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid,

invalid, invalid, invalid, invalid, invalid, invalid, invalid, invalid

};

static char actionChars[ invalid ] = "fcr";

static char suitChars[ MAX\_SUITS ] = "cdhs";

static char rankChars[ MAX\_RANKS ] = "23456789TJQKA";

static int consumeSpaces( const char \*string, int consumeEqual )

{

int i;

for( i = 0; string[ i ] != 0

&& ( isspace( string[ i ] )

|| ( consumeEqual && string[ i ] == '=' ) );

++i ) {

}

return i;

}

/\* reads up to numItems with scanf format itemFormat from string,

returning item i in \*items[ i ]

ignore the '=' character if consumeEqual is non-zero

returns the number of characters consumed doing this in charsConsumed

returns the number of items read \*/

static int readItems( const char \*itemFormat, const int numItems,

const char \*string, const int consumeEqual,

void \*items, const size\_t itemSize,

int \*charsConsumed )

{

int i, c, r;

char \*fmt;

i = strlen( itemFormat );

fmt = malloc( i + 3 );

strcpy( fmt, itemFormat );

fmt[ i ] = '%';

fmt[ i + 1 ] = 'n';

fmt[ i + 2 ] = 0;

c = 0;

for( i = 0; i < numItems; ++i ) {

c += consumeSpaces( &string[ c ], consumeEqual );

if( sscanf( &string[ c ], fmt, items + i \* itemSize, &r ) < 1 ) {

break;

}

c += r;

}

free( fmt );

\*charsConsumed = c;

return i;

}

Game \*readGame( FILE \*file )

{

int stackRead, blindRead, raiseSizeRead, boardCardsRead, c, t;

char line[ MAX\_LINE\_LEN ];

Game \*game;

game = malloc( sizeof( \*game ) );

stackRead = 4;

for( c = 0; c < MAX\_ROUNDS; ++c ) {

game->stack[ c ] = INT32\_MAX;

}

blindRead = 0;

raiseSizeRead = 0;

game->bettingType = limitBetting;

game->numPlayers = 0;

game->numRounds = 0;

for( c = 0; c < MAX\_ROUNDS; ++c ) {

game->firstPlayer[ c ] = 1;

}

for( c = 0; c < MAX\_ROUNDS; ++c ) {

game->maxRaises[ c ] = UINT8\_MAX;

}

game->numSuits = 0;

game->numRanks = 0;

game->numHoleCards = 0;

boardCardsRead = 0;

while( fgets( line, MAX\_LINE\_LEN, file ) ) {

if( line[ 0 ] == '#' || line[ 0 ] == '\n' ) {

continue;

}

if( !strncasecmp( line, "end gamedef", 11 ) ) {

break;

} else if( !strncasecmp( line, "gamedef", 7 ) ) {

continue;

} else if( !strncasecmp( line, "stack", 5 ) ) {

stackRead = readItems( "%"SCNd32, MAX\_PLAYERS, &line[ 5 ],

1, game->stack, 4, &c );

} else if( !strncasecmp( line, "blind", 5 ) ) {

blindRead = readItems( "%"SCNd32, MAX\_PLAYERS, &line[ 5 ],

1, game->blind, 4, &c );

} else if( !strncasecmp( line, "raisesize", 9 ) ) {

raiseSizeRead = readItems( "%"SCNd32, MAX\_PLAYERS, &line[ 9 ],

1, game->raiseSize, 4, &c );

} else if( !strncasecmp( line, "limit", 5 ) ) {

game->bettingType = limitBetting;

} else if( !strncasecmp( line, "nolimit", 7 ) ) {

game->bettingType = noLimitBetting;

} else if( !strncasecmp( line, "numplayers", 10 ) ) {

readItems( "%"SCNu8, 1, &line[ 10 ], 1, &game->numPlayers, 1, &c );

} else if( !strncasecmp( line, "numrounds", 9 ) ) {

readItems( "%"SCNu8, 1, &line[ 9 ], 1, &game->numRounds, 1, &c );

} else if( !strncasecmp( line, "firstplayer", 11 ) ) {

readItems( "%"SCNu8, MAX\_ROUNDS, &line[ 11 ],

1, game->firstPlayer, 1, &c );

} else if( !strncasecmp( line, "maxraises", 9 ) ) {

readItems( "%"SCNu8, MAX\_ROUNDS, &line[ 9 ],

1, game->maxRaises, 1, &c );

} else if( !strncasecmp( line, "numsuits", 8 ) ) {

readItems( "%"SCNu8, 1, &line[ 8 ], 1, &game->numSuits, 1, &c );

} else if( !strncasecmp( line, "numranks", 8 ) ) {

readItems( "%"SCNu8, 1, &line[ 8 ], 1, &game->numRanks, 1, &c );

} else if( !strncasecmp( line, "numholecards", 12 ) ) {

readItems( "%"SCNu8, 1, &line[ 12 ], 1, &game->numHoleCards, 1, &c );

} else if( !strncasecmp( line, "numboardcards", 13 ) ) {

boardCardsRead = readItems( "%"SCNu8, MAX\_ROUNDS, &line[ 13 ],

1, game->numBoardCards, 1, &c );

}

}

/\* do sanity checks \*/

if( game->numRounds == 0 || game->numRounds > MAX\_ROUNDS ) {

fprintf( stderr, "invalid number of rounds: %"PRIu8"\n", game->numRounds );

free( game );

return NULL;

}

if( game->numPlayers < 2 || game->numPlayers > MAX\_PLAYERS ) {

fprintf( stderr, "invalid number of players: %"PRIu8"\n",

game->numPlayers );

free( game );

return NULL;

}

if( stackRead < game->numPlayers ) {

fprintf( stderr, "only read %"PRIu8" stack sizes, need %"PRIu8"\n",

stackRead, game->numPlayers );

free( game );

return NULL;

}

if( blindRead < game->numPlayers ) {

fprintf( stderr, "only read %"PRIu8" blinds, need %"PRIu8"\n",

blindRead, game->numPlayers );

free( game );

return NULL;

}

for( c = 0; c < game->numPlayers; ++c ) {

if( game->blind[ c ] > game->stack[ c ] ) {

fprintf( stderr, "blind for player %d is greater than stack size\n",

c + 1 );

free( game );

return NULL;

}

}

if( game->bettingType == limitBetting

&& raiseSizeRead < game->numRounds ) {

fprintf( stderr, "only read %"PRIu8" raise sizes, need %"PRIu8"\n",

raiseSizeRead, game->numRounds );

free( game );

return NULL;

}

for( c = 0; c < game->numRounds; ++c ) {

if( game->firstPlayer[ c ] == 0

|| game->firstPlayer[ c ] > game->numPlayers ) {

fprintf( stderr, "invalid first player %"PRIu8" on round %d\n",

game->firstPlayer[ c ], c + 1 );

free( game );

return NULL;

}

--game->firstPlayer[ c ];

}

if( game->numSuits == 0 || game->numSuits > MAX\_SUITS ) {

fprintf( stderr, "invalid number of suits: %"PRIu8"\n", game->numSuits );

free( game );

return NULL;

}

if( game->numRanks == 0 || game->numRanks > MAX\_RANKS ) {

fprintf( stderr, "invalid number of ranks: %"PRIu8"\n", game->numRanks );

free( game );

return NULL;

}

if( game->numHoleCards == 0 || game->numHoleCards > MAX\_HOLE\_CARDS ) {

fprintf( stderr, "invalid number of hole cards: %"PRIu8"\n",

game->numHoleCards );

free( game );

return NULL;

}

if( boardCardsRead < game->numRounds ) {

fprintf( stderr, "only read %"PRIu8" board card numbers, need %"PRIu8"\n",

boardCardsRead, game->numRounds );

free( game );

return NULL;

}

t = game->numHoleCards \* game->numPlayers;

for( c = 0; c < game->numRounds; ++c ) {

t += game->numBoardCards[ c ];

}

if( t > game->numSuits \* game->numRanks ) {

fprintf( stderr, "too many hole and board cards for specified deck\n" );

free( game );

return NULL;

}

return game;

}

void printGame( FILE \*file, const Game \*game )

{

int i;

fprintf( file, "GAMEDEF\n" );

if( game->bettingType == noLimitBetting ) {

fprintf( file, "nolimit\n" );

} else {

fprintf( file, "limit\n" );

}

fprintf( file, "numPlayers = %"PRIu8"\n", game->numPlayers );

fprintf( file, "numRounds = %"PRIu8"\n", game->numRounds );

for( i = 0; i < game->numPlayers; ++i ) {

if( game->stack[ i ] < INT32\_MAX ) {

fprintf( file, "stack =" );

for( i = 0; i < game->numPlayers; ++i ) {

fprintf( file, " %"PRId32, game->stack[ i ] );

}

fprintf( file, "\n" );

break;

}

}

fprintf( file, "blind =" );

for( i = 0; i < game->numPlayers; ++i ) {

fprintf( file, " %"PRId32, game->blind[ i ] );

}

fprintf( file, "\n" );

if( game->bettingType == limitBetting ) {

fprintf( file, "raiseSize =" );

for( i = 0; i < game->numRounds; ++i ) {

fprintf( file, " %"PRId32, game->raiseSize[ i ] );

}

fprintf( file, "\n" );

}

for( i = 0; i < game->numRounds; ++i ) {

if( game->firstPlayer[ i ] != 0 ) {

fprintf( file, "firstPlayer =" );

for( i = 0; i < game->numRounds; ++i ) {

fprintf( file, " %"PRIu8, game->firstPlayer[ i ] + 1 );

}

fprintf( file, "\n" );

break;

}

}

for( i = 0; i < game->numRounds; ++i ) {

if( game->maxRaises[ i ] != UINT8\_MAX ) {

fprintf( file, "maxRaises =" );

for( i = 0; i < game->numRounds; ++i ) {

fprintf( file, " %"PRIu8, game->maxRaises[ i ] );

}

fprintf( file, "\n" );

break;

}

}

fprintf( file, "numSuits = %"PRIu8"\n", game->numSuits );

fprintf( file, "numRanks = %"PRIu8"\n", game->numRanks );

fprintf( file, "numHoleCards = %"PRIu8"\n", game->numHoleCards );

fprintf( file, "numBoardCards =" );

for( i = 0; i < game->numRounds; ++i ) {

fprintf( file, " %"PRIu8, game->numBoardCards[ i ] );

}

fprintf( file, "\n" );

fprintf( file, "END GAMEDEF\n" );

}

uint8\_t bcStart( const Game \*game, const uint8\_t round )

{

int r;

uint8\_t start;

start = 0;

for( r = 0; r < round; ++r ) {

start += game->numBoardCards[ r ];

}

return start;

}

uint8\_t sumBoardCards( const Game \*game, const uint8\_t round )

{

int r;

uint8\_t total;

total = 0;

for( r = 0; r <= round; ++r ) {

total += game->numBoardCards[ r ];

}

return total;

}

static uint8\_t nextPlayer( const Game \*game, const State \*state,

const uint8\_t curPlayer )

{

uint8\_t n;

n = curPlayer;

do {

n = ( n + 1 ) % game->numPlayers;

} while( state->playerFolded[ n ]

|| state->spent[ n ] >= game->stack[ n ] );

return n;

}

uint8\_t currentPlayer( const Game \*game, const State \*state )

{

/\* if action has already been made, compute next player from last player \*/

if( state->numActions[ state->round ] ) {

return nextPlayer( game, state, state->actingPlayer[ state->round ]

[ state->numActions[ state->round ] - 1 ] );

}

/\* first player in a round is determined by the game and round

use nextPlayer() because firstPlayer[round] might be unable to act \*/

return nextPlayer( game, state, game->firstPlayer[ state->round ]

+ game->numPlayers - 1 );

}

uint8\_t numRaises( const State \*state )

{

int i;

uint8\_t ret;

ret = 0;

for( i = 0; i < state->numActions[ state->round ]; ++i ) {

if( state->action[ state->round ][ i ].type == raise ) {

++ret;

}

}

return ret;

}

uint8\_t numFolded( const Game \*game, const State \*state )

{

int p;

uint8\_t ret;

ret = 0;

for( p = 0; p < game->numPlayers; ++p ) {

if( state->playerFolded[ p ] ) {

++ret;

}

}

return ret;

}

uint8\_t numCalled( const Game \*game, const State \*state )

{

int i;

uint8\_t ret, p;

ret = 0;

for( i = state->numActions[ state->round ]; i > 0; --i ) {

p = state->actingPlayer[ state->round ][ i - 1 ];

if( state->action[ state->round ][ i - 1 ].type == raise ) {

/\* player initiated the bet, so they've called it \*/

if( state->spent[ p ] < game->stack[ p ] ) {

/\* player is not all-in, so they're still acting \*/

++ret;

}

/\* this is the start of the current bet, so we're finished \*/

return ret;

} else if( state->action[ state->round ][ i - 1 ].type == call ) {

if( state->spent[ p ] < game->stack[ p ] ) {

/\* player is not all-in, so they're still acting \*/

++ret;

}

}

}

return ret;

}

uint8\_t numAllIn( const Game \*game, const State \*state )

{

int p;

uint8\_t ret;

ret = 0;

for( p = 0; p < game->numPlayers; ++p ) {

if( state->spent[ p ] >= game->stack[ p ] ) {

++ret;

}

}

return ret;

}

return c;

}

The above code has been largely modified (chunks of code have been removed) as it is very long but it is easy to see the amount of work that has gone into such a program given that this is only a single file and there are at least five others used to run the University’s poker program. To see the full version of the code please see the CD provided along with this paper or visit <http://poker.cs.ualberta.ca/acpc_code/project_acpc_server.tar.bz2>. The university of Alberta website is useful for this project since it is attempting to do a similar job even though the code used is C and not C++.

### Cs.mcgill.ca and poker programming:

Another website that has roots in the academic circle is <http://www.cs.mcgill.ca/~gkazam/cs303/index.html> which provides a .jar file containing a large amount of code in Java. The program appears to play a complete poker game, providing code that deals with AI, player interactions, game strategies, bets placed, tournaments, scoring amongst others whilst it also provides a full GUI. This website and its owner appear to have attempted to create a fully realised poker game that can be played online or offline. To run the game the available .Jar file needs to be downloaded then at the command line the user must type "java -jar poker.jar". The game has so much code that it will not be given here, except a brief overview of the code and what they deal with in the program is given below:

* .settings
* game
* AI
* GUI
* images
* META-INF
* money
* players
* scoring
* util

These are the file names given in the .jar file and each file contains at least ten to twenty Java programs. The file is very useful for users wishing to create their own poker program as it essentially displays a complete poker program. The file is available for download here: <http://www.cs.mcgill.ca/~gkazam/cs303/poker.jar> and will also be included on the CD made available with this paper.

The last two programs that will be examined here were obtained from open-source websites where the users were attempting to create their own poker programs simply because they love the game or were attempting to create their own program for later use in a project that would net a monetary gain.

### Coding the wheel poker programming:

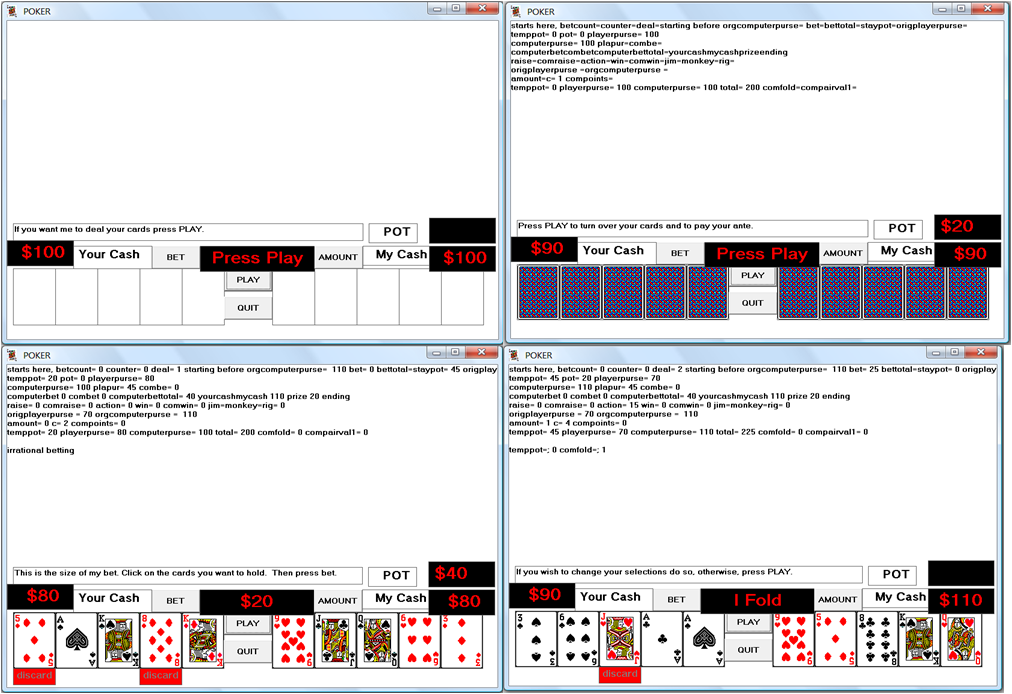
One of the most useful sites in terms of creating a poker hand evaluator for any game of poker is to be found at <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> where the users of the site have all endeavoured to create their own program for evaluating poker hands in a variety of languages from Java to Visual Basic and also C++. The website is a collection of previously available open-source poker software that began on a previous website. A useful feature of the [codingthewheel](http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup) website is that it notes how difficult it is for a reader to come along, take a piece of open-source software from the site then build and run it. To this end the website creator has placed the complete source code for every evaluator into a single, easy-to-build library: [XPokerEval](http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup#xpokereval)[[74]](#footnote-74). As the website so eloquently describes itself:

“The [Pokersource poker-eval library](http://pokersource.sourceforge.net/) is probably **the most widely-used poker hand evaluator in the galaxy**.  It's open-source, it's fast, it's over ten years old, and it can not only analyze your poker hands but make you a cup of coffee, interpolate the doomsday trajectory of near-Earth asteroids, and fix the grease-clot in your kitchen sink[[75]](#footnote-75).”

There is no definitive software available from this site that proves the most useful as they all borrow from one another. Some of the programs appear quite sophisticated in examining poker hands. For example, one of Cactus Kev’s evaluators orders hands from 1 (strongest) to 7,462 (weakest) so a hand with value 2302 beats a hand with value 3402[[76]](#footnote-76). Some of the other programs used this as a template and then made programs that worked faster and were more efficient. These programs are all part of the [XPokerEval](http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup#xpokereval) library and are included on the CD provided with this paper. The best thing about the library is that is predominantly written in C and C++ so a comparison with the goals of this project can be easily made in terms of what the program achieves when written in C++. The [codingthewheel](http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup) site describes in detail each of the programs included in the library and allows the reader to make a choice about which program in particular to use depending on what they are creating their own program for. Another excellent feature of this site is that it provides links to other sites in order to explain the vast array of poker terms being used. This helps the reader understand what is going on and as a final note the website also gives a thorough discussion on arguably the best poker hand evaluator created; the two plus two evaluator. Not only is the discussion on [codingthewheel](http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup) thorough and detailed but this site also gives links to the website that the two plus two evaluator discussion and implementation was posted on[[77]](#footnote-77). As a final note the website allows anyone to view its forum and the users on the forum are quite vocal about the usefulness of any given poker program and provide snippets of code (and in some cases whole programs) to prove their point[[78]](#footnote-78). Quite simply this site is very useful for those who are veterans in programming poker hand evaluators and is also a detailed and useful tool for those who are just starting out. For the source code for the library of all the files see: <http://www.codingthewheel.com/file.axd?file=XPokerEval.zip>.

### Caspercomsci.com and poker programming:

The final website I would like to mention is <http://www.caspercomsci.com/> which has an interesting little program written in Visual Basic. This program, when run, creates a small two player poker game where the player is pitted against the computer. A number of screenshots can be seen below:



The game created is simple but interesting and actually quite entertaining to play. On-screen instructions are given to the player so that playing the game is quite easy. Whilst the game itself is simplistic it does show that a poker game can be implemented with thought and work in any programming language[[79]](#footnote-79). The program file is open-source and can be downloaded from: <http://www.caspercomsci.com/pages/vbsource.htm#5>.

## My adapted poker code in C++:

Dfghj

## Conclusion:

In the course of this paper we have taken a brief look at the poker programs created in Java by Dr. Kullmann and have also discussed various implementations of poker algorithms based upon the type of poker game being played. The various poker games have been investigated and the main choice for the implementation of the poker program from Java into C++ was decided upon and was revealed as the game played by the original Java code by Dr. Kullmann. I would have liked to have attempted to implement a code based solely upon the Texas Hold ‘Em game simply because I am familiar with the rules and tenets of this game and enjoy playing it. Using the game played by the original Java code was more practical, however, and the code displayed probabilities and statistical outcomes for hands, betting and the odds of a player winning a game based on both. Using the original Java code to implement the program in C++ was chosen due to the fact that coding for a singular game such as Texas Hold ‘Em would have proven too difficult and time consuming. Instead of writing code in C++ that dealt with a singular game, other sources in C++ as well as Java and Visual Basic have been provided that deal with various other poker games. These programs are primarily open source projects done by others and are included as compressed files (.tar, .zip, etc) on the CD that is included along with this paper. The ways in which people interact with poker online have also been examined as have the ways in which the C++ poker program will be implemented, namely via the Doxygen software or possibly with another C++ compiler such as Microsoft Visual C++ which is the software that was used to compile the project code. The Git software and the Github site has also been displayed and discussed and whilst it was not used, its usefulness has been documented and I feel that any wishing to share files utilise this site in future (but only if you have a Linux or Linux based Operating System).

A CD containing the poker programs, both Dr. Kullmann’s, my own and the open-source programs will also be included.STUFF TO ADD/LOOK OVER

Need to mention mathematical equations and stuff. Use that website you found that deals with the mathematics of poker. See the Wikipedia poker articles document too. DONE Include some of the stuff on this site in the final version.

Need to mention about GIThub being useless since the platform you used wasn’t Linux but Windows (say why you changed!)DONE.

Need to mention why you didn’t use Doxygen but Microsoft Visual C++ 2005 instead! Mostly because you downloaded Visual C++ first and got to grips with it. DONE.

Mention that you stuck with the original game that the code deals with but other games have been implemented by others then put in the other C++ code from those other sources and also say what they do. DONE. Maybe display their output as well, use cs.alberta for this as well as cactus kevs stuff amongst others.

Finally display the code you have done, mention how you’ve changed it from Kullmann’s Java code to your C++ code and also mention that in the C++ code you have left various comments to show any other users what the code is attempting to do.

Remember when you are fine-tuning this to go back through the Wikipedia articles you have used and possibly put some of the books that they mention into your bibliography as well as some of the equations that are mentioned (might need to go back online for this as word has deleted some of the equation pics). Also attempt to put in the output from other poker programs!

Make sure that in the intro you say what the aims of the project are, namely to implement Kullmann’s code in C++ and possibly improve on it. Make sure to say in the conclusion that you have no prior training with C++! Put the other code from the various sources on the CD too!

Add a contents page too! Will need to save this in word 2007 to have word make the contents list!

## Appendices

## Bibliography:

### Books and printed resources:

* Barclay, K.A. and Gordon, B.J. (1994) C++: problem solving and programming. Hemel Hempstead: Prentice Hall.
* Botermans, J. (2008) [translated from the Spanish by Fankbonner, E.L.].The book of games: strategy, tactics & history. New York: Sterling.
* Brassard, G. and Bratley, P. (1988) Algorithmics: theory and practice. Englewood Cliffs, N.J.: Prentice-Hall.
* Cadenhead, R. (2010) Sams Teach Yourself Java in 24 Hours. Sams Publishing.
* Donath, T. (2004) Hiding and revealing in online poker games. In: *CSCW '04 Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM New York, NY, USA.
* Ettinger, J. (1994) Programming in C++. London: M. Macmillan.
* Fethi A. Rabhi, F.A. and Lapalme, G. (1999) Algorithms: a functional programming approach. Harlow: Addison-Wesley.
* Gilpin, A. and Sandhome, T. (2007) Better Automated Abstraction Techniques for Imperfect Information Games, with Application to Texas Hold’em Poker. In *The Sixth Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 07)* *Honolulu, Hawai’i, USA.*
* Golder, S.A. and Donath, T. (2004) Hiding and Revealing in Online Poker Games. In *CSCW '04 Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM New York, NY, USA.
* Hahn, B.D. (1993) C++: a practical introduction. Manchester: N. C. C. Blackwell.
* Hortsmann, C. (2008) Big Java: Third Edition. San Jose State University; John Wiley & Sons, Inc.
* Journal of algorithms. San Diego [etc.] Academic Press. Volume 1 -41.
* K.L. Man, M. Mercaldi and H.L. Leung. (2009) SpeedyDes3 - Speedy Des3 Encryption for Video Poker Game Machines Licensed in Italy. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol II IMECS 2009, March 18 - 20, 2009.* Hong Kong.
* Liberty J. (2001) Sams Teach Yourself C++ in 21 Days. Sams Publishing.
* McDonald, J. (1996) Strategy in poker, business & war. New York; London: Norton.
* Packel, E. (2006) The Mathematics of Games and Gambling, 2nd e. Washington, DC: Math. Assoc. Amer.
* Paull, M.C. (1988) Algorithm design: a recursion transformation framework. New York: Wiley.
* Petriv, M. (1996) *Hold'em Odds Book*. Objective Observer Press.
* Sedgewick, R and Wayne, K. (2008) Introduction to programming in Java: an interdisciplinary approach. Princeton University, Pearson, Addison-Wesley.

### Online resources:

* <http://archives1.twoplustwo.com/showflat.php?Cat=0&Number=8513906&page=0&fpart=1&vc=1> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Decision_tree> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Doxygen> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Fold_equity> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/M-ratio> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Normalization_(statistics)> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Poker_calculator> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011.
* [http://en.wikipedia.org/wiki/Poker\_probability\_(Texas\_Hold\_’em)](http://en.wikipedia.org/wiki/Poker_probability_(Texas_Hold_'em)) accessed 19th April 2011.
* <http://en.wikipedia.org/wiki/Q-ratio> accessed 19th April 2011.
* [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011.
* <http://mathworld.wolfram.com/Poker.html> accessed 19th April 2011.
* <http://people.math.sfu.ca/~alspach/comp18/> accessed 19th April 2011.
* <http://poker.cs.ualberta.ca/acpc_code/project_acpc_server.tar.bz2>.
* <http://poker.cs.ualberta.ca/publications/abourisk.msc.pdf> accessed 19th April 2011.
* <http://poker.cs.ualberta.ca/publications/IJCAI03.pdf> accessed 19th April 2011.
* <http://poker.cs.ualberta.ca/publications/NIPS07-cfr.pdf> accessed 19th April 2011.
* <http://poker.cs.ualberta.ca/publications/schauenberg.msc.pdf> accessed 19th April 2011.
* <http://www.caspercomsci.com/> accessed 19th April 2011.
* <http://www.cigital.com/papers/download/developer_gambling.php> accessed 19th April 2011.
* <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> accessed 19th April 2011.
* <http://www.cs.mcgill.ca/~gkazam/cs303/index.html>
* <http://www.cs.swan.ac.uk/~csoliver/ProgrammingJava201011/CS-M41_Programs/Courseworks/200910/Two/> accessed 19th April 2011.
* <http://www.dmoz.org/Games/Gambling/Poker/Software_and_Tools/> accessed 19th April 2011.
* <http://www.pagat.com/poker/software.html> accessed 19th April 2011.
* <http://www.pokerlistings.com/poker-software-other> accessed 19th April 2011.
* <http://www.pokerstars.com/wsop/> accessed 19th April 2011.
* <http://www.stack.nl/~dimitri/doxygen/index.html> accessed 19th April 2011.
* <https://github.com/> accessed 19th April 2011.
* [www.**pkr**.com](http://www.pkr.com) accessed 19th April 2011.
* <http://www.cigital.com/papers/download/developer_gambling.php> accessed 19th April 2011.

## Poker code in java:

// Oliver Kullmann, 5.11.2009 (Swansea)

/\*

To be compiled by

javac Poker.java

(needs StdIn.java), and to be run by

java Poker N

where N >= 0 is an integer. For N = 0, a hand of five cards is read from

standard input, and its hand rank (from straight flush to high card) is

output, while for N > 0 this number of random hands are drawn, and

the statistics on the relative frequency of the nine hand ranks is

output.

The format of a card to to be read is for example "King of Clubs" (that is,

card-rank "of" suit), where space-symbols are irrelevant (except of the

separating space-symbols). See below for the exact spelling of card-ranks

and suites.

\*/

public class Poker {

// Suites are represented by integers 0,...,3,

// card ranks by integers 0,...,12, and cards by integers 0,...,51.

// And hand ranks are represented by integer 1,...,9 (from highest

// to lowest).

public static final int num\_suites = 4;

public static final int num\_ranks = 13;

public static final int num\_cards = num\_suites \* num\_ranks;

public static final int hand\_size = 5;

public static final int num\_hand\_ranks = 9;

// The integer-representation of suites and ranks are given as indices of

// the following arrays:

public static final String[] suit\_names = {"Clubs", "Diamonds", "Hearts", "Spades"};

public static final String[] rank\_names = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};

// The string representation of hand ranks:

public static final String[] hand\_rank\_names = {"Straight flush", "Four of a kind", "Full house", "Flush", "Straight", "Three of a kind", "Two pairs", "One pair", "High card"};

// Explicit constants for the 9 hand ranks:

public static final int straight\_flush = 1;

public static final int four\_of\_a\_kind = 2;

public static final int full\_house = 3;

public static final int flush = 4;

public static final int straight = 5;

public static final int three\_of\_a\_kind = 6;

public static final int two\_pairs = 7;

public static final int one\_pair = 8;

public static final int high\_card = 9;

// Converts a string into a suit; returns -1, if the string doesn't

// represent a suit:

public static int parse\_suit(final String s) {

for (int i = 0; i < num\_suites; ++i)

if (s.equals(suit\_names[i])) return i;

return -1;

}

// Converts a string into a card-rank; returns -1, if the string doesn't

// represent a card-rank:

public static int parse\_rank(final String s) {

for (int i = 0; i < num\_ranks; ++i)

if (s.equals(rank\_names[i])) return i;

return -1;

}

// For a card, compute its suit:

public static int suit(final int card) {

return card / num\_ranks;

}

// For a card, compute its rank:

public static int rank(final int card) {

return card % num\_ranks;

}

// Given card-rank and suit, compute the corresponding card:

public static int card(final int rank, final int suit) {

return suit \* num\_ranks + rank;

}

// Read a hand (and array of cards) from standard-input; returns null

// if some parsing error occurs:

public static int[] read\_hand() {

int[] hand = new int[hand\_size];

for (int i = 0; i < hand\_size; ++i) {

final int rank = parse\_rank(StdIn.readString());

if (rank == -1) return null;

if (! StdIn.readString().equals("of")) return null;

final int suit = parse\_suit(StdIn.readString());

if (suit == -1) return null;

hand[i] = card(rank, suit);

}

return hand;

}

// Creates a random hand:

public static int[] random\_hand() {

int[] hand = new int[hand\_size];

int[] cards = new int[num\_cards];

for (int i = 0; i < num\_cards; ++i)

cards[i] = i;

for (int i = 0; i < hand\_size; ++i) {

int random\_index = i + (int) (Math.random() \* (num\_cards - i));

// i <= random\_index < num\_cards - i

hand[i] = cards[random\_index];

cards[random\_index] = cards[i];

}

return hand;

}

// Check that the cards of a hand are really all different:

private static boolean check\_all\_different(final int[] hand) {

for (int i = 0; i < hand\_size; ++i)

for (int j = i+1; j < hand\_size; ++j)

if (hand[i] == hand[j]) return false;

return true;

}

// Check whether a hand is a flush:

private static boolean is\_flush(final int[] hand) {

final int first\_suit = suit(hand[0]);

for (int i = 1; i < hand\_size; ++i)

if (suit(hand[i]) != first\_suit) return false;

return true;

}

// Check whether a hand is a straight:

private static boolean is\_straight(final int[] hand) {

boolean[] ranks = new boolean[num\_ranks+1];

// shifting ranks 0..12 to 1..13, and adding new rank 0 for card "1"

for (int i = 0; i < hand\_size; ++i)

ranks[rank(hand[i])+1] = true;

if (ranks[num\_ranks] & ranks[1]) ranks[0] = true;

int first = 0;

while (! ranks[first]) ++first;

final int num\_remains = hand\_size - 1;

if (first + num\_remains > num\_ranks) return false;

for (int i = first + 1; i <= first + num\_remains; ++i)

if (! ranks[i]) return false;

return true;

}

/\* Remark: An alternative algorithm is to first sort the hand by

ranks, using the sort-algorithm from the Java-library. This would

yield more compact code, however in this module we use only elements

from the Java-library as discussed in the lectures.

\*/

// Determine the hand-rank of a hand:

public static int hand\_rank(final int[] hand) {

int[] rank\_count = new int[num\_ranks];

for (int i = 0; i < hand\_size; ++i)

++rank\_count[rank(hand[i])];

int[] count\_of\_counts = new int[num\_suites+1];

for (int i = 0; i < num\_ranks; ++i)

++count\_of\_counts[rank\_count[i]];

if (count\_of\_counts[4] == 1) return four\_of\_a\_kind;

if (count\_of\_counts[3] == 1)

if (count\_of\_counts[2] == 1) return full\_house;

else return three\_of\_a\_kind;

if (count\_of\_counts[2] == 2) return two\_pairs;

if (count\_of\_counts[2] == 1) return one\_pair;

final boolean is\_flush = is\_flush(hand);

final boolean is\_straight = is\_straight(hand);

if (is\_flush && is\_straight) return straight\_flush;

else if (is\_flush) return flush;

else if (is\_straight) return straight;

else return high\_card;

}

public static void main(String[] args) {

if (args.length == 0) {

System.err.println("ERROR[Poker]: One argument N is required.");

return;

}

final int N = Integer.parseInt(args[0]);

if (N < 0) {

System.err.println("ERROR[Poker]: N < 0.");

return;

}

if (N == 0) {

final int[] hand = read\_hand();

if (hand == null) {

System.err.println("ERROR[Poker]: Incorrect hand of cards.");

return;

}

if (! check\_all\_different(hand)) {

System.err.println("ERROR[Poker]: Two cards coincide.");

return;

}

System.out.println("Hand rank = " + hand\_rank\_names[hand\_rank(hand)-1]);

}

else {

int[] counts = new int[num\_hand\_ranks+1];

for (int i = 1; i <= N; ++i)

++counts[hand\_rank(random\_hand())];

for (int hand\_rank = 1; hand\_rank <= num\_hand\_ranks; ++hand\_rank)

System.out.println(hand\_rank\_names[hand\_rank-1] + ": " + (double) (counts[hand\_rank]) / N \* 100 + "%");

}

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Poker {

// Simulating the behaviour of the Poker-program from the first

// coursework (with additional information when evaluating a single hand):

public static void main(String[] args) {

if (args.length == 0) {

System.err.println("ERROR[Poker]: One argument N is required.");

return;

}

final int N = Integer.parseInt(args[0]);

if (N < 0) {

System.err.println("ERROR[Poker]: N < 0.");

return;

}

if (N == 0) {

Hand h = new Hand(new In());

final HandRank hr = new HandRank(h);

System.out.println(hr);

System.out.println("The number of (strictly) better hands: " + hr.cumulated\_count());

System.out.println("The probability of a (strictly) better hand: " + (100 \* hr.cumulated\_probability()) + "%");

}

else {

int[] counts = new int[HandRank.num\_major\_hand\_ranks+1];

for (int i = 1; i <= N; ++i) {

final Bank b = new Bank(1);

final HandRank hr = new HandRank(b.orig\_hand(1));

++counts[hr.major\_rank];

}

for (int r = 1; r <= HandRank.num\_major\_hand\_ranks; ++r)

System.out.println(HandRank.major\_hand\_rank\_names[r-1] + ": " + (double) (counts[r]) / N \* 100 + "%");

}

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class PokerExtended {

// Simulating the behaviour of the Poker-program from the first

// coursework (with additional information when evaluating a single hand):

public static void main(String[] args) {

if (args.length == 0) {

System.err.println("ERROR[Poker]: One argument N is required.");

return;

}

final int N = Integer.parseInt(args[0]);

if (N < 0) {

System.err.println("ERROR[Poker]: N < 0.");

return;

}

if (N == 0) {

Hand h = new Hand(new In());

final HandRank hr = new HandRank(h);

System.out.println(hr);

System.out.println("The number of (strictly) better hands: " + hr.cumulated\_count());

System.out.println("The probability of a (strictly) better hand: " + (100 \* hr.cumulated\_probability()) + "%");

System.out.println("Analysis of the exchange possibilities:");

final ExchangeRequest[] all = ExchangeRequest.all\_requests();

final Evaluation eval = new Evaluation(h);

for (int i = 0; i < all.length; ++i) {

System.out.println("Exchange " + all[i] + " yields:");

final EvaluationResult result = eval.evaluate(all[i]);

System.out.println("Best: ");

System.out.println(result.get(1));

System.out.println("Worst: ");

System.out.println(result.get(result.length));

}

}

else {

int[] counts = new int[HandRank.num\_major\_hand\_ranks+1];

for (int i = 1; i <= N; ++i) {

final Bank b = new Bank(1);

final HandRank hr = new HandRank(b.orig\_hand(1));

++counts[hr.major\_rank];

}

for (int r = 1; r <= HandRank.num\_major\_hand\_ranks; ++r)

System.out.println(HandRank.major\_hand\_rank\_names[r-1] + ": " + (double) (counts[r]) / N \* 100 + "%");

}

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Storage {

// Provides storage for the results of all (different) strategies against

// each other (for a fixed deck), usable for the single results as well as

// for the statistics.

// Remark: One could call this class also "UpperTriangularMatrix".

/\*

Via Storage(5) for example storage for 5\*4/2 = 10 integers is

provided. Via set(i,j,x) then for the pairs 1 <= i < j <= 5

the value x is stored, and retrieved by get(i,j).

\*/

public final int number\_strategies;

public final int number\_results;

public Storage(final int num\_strategies) {

assert num\_strategies >= 0;

number\_strategies = num\_strategies;

number\_results = (number\_strategies \* (number\_strategies-1)) / 2;

results = new int[number\_results];

}

public int get(final int i, final int j) {

return results[index(i,j)];

}

public void set(final int i, final int j, final int x) {

results[index(i,j)] = x;

}

public String toString() {

String result = "";

for (int i = 1; i < number\_strategies; ++i) {

for (int j = i+1; j <= number\_strategies; ++j)

result += get(i,j) + " ";

result += "\n";

}

return result;

}

public boolean equals(final Storage S) {

if (S.number\_strategies != number\_strategies)

return false;

for (int i = 1; i < number\_strategies; ++i)

for (int j = i+1; j <= number\_strategies; ++j)

if (S.get(i,j) != get(i,j))

return false;

return true;

}

private int[] results;

// Using colexicographical order for linearisation:

private int index(final int i, final int j) {

assert i >= 1;

assert i <= number\_strategies;

assert j >= 1;

assert j <= number\_strategies;

assert i < j;

return ((j-2)\*(j-1))/2+i-1;

}

// Tests:

public static void main(String[] args) {

final Storage S = new Storage(5);

assert S.number\_strategies == 5;

assert S.number\_results == 10;

assert S.equals(S);

for (int i = 1; i < 5; ++i)

for (int j = i+1; j <= 5; ++j)

S.set(i,j,i\*j);

assert S.equals(S);

for (int i = 1; i < 5; ++i)

for (int j = i+1; j <= 5; ++j)

assert(S.get(i,j) == i\*j);

final Storage S2 = new Storage(5);

assert ! S.equals(S2);

for (int i = 1; i < 5; ++i)

for (int j = i+1; j <= 5; ++j)

S2.set(i,j,i\*j);

assert S.equals(S2);

S2.set(1,2,77);

assert ! S.equals(S2);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Strategy {

// For an object s of type Strategy by s.exchange(i) we obtain

// the exchange request for the strategy with index i.

public static final int number\_strategies = 5; // to be updated

public Strategy(final Hand h\_) {

h = h\_;

}

public ExchangeRequest exchange(final int index\_strategy) {

assert valid\_index(index\_strategy);

assert index\_strategy <= number\_strategies;

switch (index\_strategy) {

case 1 : return no\_exchange();

case 2 : return always\_first\_two();

case 3 : return always\_last();

case 4 : return always\_last\_two();

case 5 : return minimise\_expected\_number\_better\_hands();

case 6 : return risky();

case 7 : return cautious();

}

return no\_exchange(); // for the compiler; does not occur for valid indices

}

public static boolean valid\_index(final int i) {

return i >= 1 && i <= number\_strategies;

}

private Hand h;

private static ExchangeRequest no\_exchange() {

return new ExchangeRequest();

}

private static ExchangeRequest always\_first\_two() {

int[] e = new int[2];

e[0] = 1;

e[1] = 2;

return new ExchangeRequest(e);

}

private static ExchangeRequest always\_last() {

int[] e = new int[1];

e[0] = 5;

return new ExchangeRequest(e);

}

private static ExchangeRequest always\_last\_two() {

int[] e = new int[2];

e[0] = 4;

e[1] = 5;

return new ExchangeRequest(e);

}

private ExchangeRequest minimise\_expected\_number\_better\_hands() {

final Evaluation eval = new Evaluation(h);

final ExchangeRequest[] all\_requests = ExchangeRequest.all\_requests();

ExchangeRequest best\_exchange\_request = null;

double best\_expected\_value = Double.POSITIVE\_INFINITY;

for (int i = 0; i < all\_requests.length; ++i) {

final ExchangeRequest new\_exchange\_request = all\_requests[i];

final double new\_expected\_value = expected\_number\_better\_hands(eval.evaluate(new\_exchange\_request));

if (new\_expected\_value < best\_expected\_value) {

best\_expected\_value = new\_expected\_value;

best\_exchange\_request = new\_exchange\_request;

}

}

return best\_exchange\_request;

}

private static double expected\_number\_better\_hands(final EvaluationResult E) {

double sum = 0;

for (int i = 1; i <= E.length; ++i) {

final EvaluatedOutcome e = E.get(i);

sum += e.hand\_rank.cumulated\_count() \* e.prob;

}

return sum;

}

// Go for the best hand achievable:

private ExchangeRequest risky() {

// XXX

return new ExchangeRequest(); // temporarily

}

// Choose only amongst choices which can not impair the original hand:

private ExchangeRequest cautious() {

// XXX

return new ExchangeRequest(); // temporarily

}

// Possibly further strategies to be implemented XXX

// Tests:

public static void main(String[] args) {

{

final Hand h = new Hand(new Card(0), new Card(1), new Card(2), new Card(3), new Card(25));

final Strategy s = new Strategy(h);

final ExchangeRequest e = s.exchange(5);

assert (e.equals(new ExchangeRequest(4,5)));

}

final Bank b = new Bank(1);

final Hand h = b.orig\_hand(1);

final Strategy s = new Strategy(h);

System.out.println(h);

for (int i = 1; i <= number\_strategies; ++i) {

System.out.println(i + ": " + s.exchange(i));

}

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Suit {

// Wrapper class around suites as integer indices.

public static final int num\_suites = 4;

public static final int clubs = 0;

public static final int spades = 1;

public static final int hearts = 2;

public static final int diamonds = 3;

public final int index;

public Suit(final int i) {

assert(i >= 0);

assert(i < num\_suites);

index = i;

}

public Suit(final String str) {

index = determine\_index(str);

}

public String toString() {

return suit\_names[index];

}

public boolean equals(final Suit s) {

return s.index == index;

}

private static final String[] suit\_names = {"Clubs", "Spades", "Hearts", "Diamonds" };

private static int determine\_index(final String str) {

for (int i = 0; i < num\_suites; ++i)

if (str.equals(suit\_names[i]))

return i;

throw new RuntimeException("Does not represent a suit: " + str);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Tournament {

// Read the number N of rounds from the command-line, and for each

// pair 1 <= i < j <= Strategy.number\_strategies of strategies from Strategy,

// play N rounds, showing the difference of the number of games won by

// strategy i minus the number of games won by strategy j.

public static void main(String[] args) {

if (args.length == 0) {

System.err.println("ERROR[Tournament]: One parameter is neded, the number N of games.");

return;

}

final long N = Long.parseLong(args[0]);

final Storage S = new Storage(Strategy.number\_strategies);

for (long t = 1; t <= N; ++t) {

final Storage R = TwoPlayers.play();

for (int i = 1; i < Strategy.number\_strategies; ++i)

for (int j = i+1; j <= Strategy.number\_strategies; ++j)

S.set(i,j, S.get(i,j)+R.get(i,j));

}

System.out.println(S);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class TwoPlayers {

// Plays a single game between two players, using all available strategies

// from the Strategy class, and returns an object of type Storage with

// entries from {-1,0,+1}, where an entry -1 means player 2 wins, +1 means

// player 1 wins, and 0 means draw:

public static Storage play() {

final Storage R = new Storage(Strategy.number\_strategies);

final Bank b = new Bank(2);

final ExchangeRequest[] ep1 = new ExchangeRequest[Strategy.number\_strategies-1];

{

final Strategy p1 = new Strategy(b.orig\_hand(1));

for (int s1 = 1; s1 < Strategy.number\_strategies; ++s1)

ep1[s1-1] = p1.exchange(s1);

}

final ExchangeRequest[] ep2 = new ExchangeRequest[Strategy.number\_strategies-1];

{

final Strategy p2 = new Strategy(b.orig\_hand(2));

for (int s2 = 2; s2 <= Strategy.number\_strategies; ++s2)

ep2[s2-2] = p2.exchange(s2);

}

final ExchangeRequest[] E = new ExchangeRequest[2];

for (int s1 = 1; s1 < Strategy.number\_strategies; ++s1)

for (int s2 = s1+1; s2 <= Strategy.number\_strategies; ++s2) {

E[0] = ep1[s1-1];

E[1] = ep2[s2-2];

b.perform\_exchanges(E);

final HandRank hr1 = new HandRank(b.new\_hand(1));

final HandRank hr2 = new HandRank(b.new\_hand(2));

if (hr1.rank < hr2.rank)

R.set(s1,s2,+1);

else if (hr2.rank < hr1.rank)

R.set(s1,s2,-1);

else

R.set(s1,s2,0);

}

return R;

}

public static void main(String[] args) {

final Storage S = TwoPlayers.play();

System.out.println(S);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class HandRank {

// Poker has 3614 ranks, where a rank is a number from 1 to 3614 assigned

// to a (Poker) hand such that a hand wins over another hand if and only if

// its rank is strictly lower (in case of equality thus we have a draw).

// The ranks are subdivided into 9 major ranks, where each is subdivided

// further into minor ranks.

// All three kinds of ranks are enumerated starting with value 1 for the

// highest possibility.

public static final int num\_major\_hand\_ranks = 9;

public static final int straight\_flush = 1;

public static final int four\_of\_a\_kind = 2;

public static final int full\_house = 3;

public static final int flush = 4;

public static final int straight = 5;

public static final int three\_of\_a\_kind = 6;

public static final int two\_pairs = 7;

public static final int one\_pair = 8;

public static final int high\_card = 9;

// The number of minor ranks in a major rank:

public static final int num\_straight\_flush\_ranks = 10;

public static final int num\_four\_of\_a\_kind\_ranks = 13;

public static final int num\_full\_house\_ranks = 13;

public static final int num\_flush\_ranks = 1277;

public static final int num\_straight\_ranks = 10;

public static final int num\_three\_of\_a\_kind\_ranks = 13;

public static final int num\_two\_pairs\_ranks = 858;

public static final int num\_one\_pair\_ranks = 2860;

public static final int num\_high\_card\_ranks = 1277;

// The number of ranks up to a major rank:

public static final int cum\_straight\_flush\_ranks = num\_straight\_flush\_ranks;

public static final int cum\_four\_of\_a\_kind\_ranks = cum\_straight\_flush\_ranks + num\_four\_of\_a\_kind\_ranks;

public static final int cum\_full\_house\_ranks = cum\_four\_of\_a\_kind\_ranks + num\_full\_house\_ranks;

public static final int cum\_flush\_ranks = cum\_full\_house\_ranks + num\_flush\_ranks;

public static final int cum\_straight\_ranks = cum\_flush\_ranks + num\_straight\_ranks;

public static final int cum\_three\_of\_a\_kind\_ranks = cum\_straight\_ranks + num\_three\_of\_a\_kind\_ranks;

public static final int cum\_two\_pairs\_ranks = cum\_three\_of\_a\_kind\_ranks + num\_two\_pairs\_ranks;

public static final int cum\_one\_pair\_ranks = cum\_two\_pairs\_ranks + num\_one\_pair\_ranks;

public static final int cum\_high\_card\_ranks = cum\_one\_pair\_ranks + num\_high\_card\_ranks;

// Access for a given major rank to the cumulated ranks via an array:

public static final int[] cumulated\_num\_ranks = {0, cum\_straight\_flush\_ranks, cum\_four\_of\_a\_kind\_ranks, cum\_full\_house\_ranks, cum\_flush\_ranks, cum\_straight\_ranks, cum\_three\_of\_a\_kind\_ranks, cum\_two\_pairs\_ranks, cum\_one\_pair\_ranks, cum\_high\_card\_ranks};

public static final int num\_hand\_ranks = cum\_high\_card\_ranks; // = 6331

// For a given rank, how many hands of that rank are there (this depends

// only on the major rank):

public static final int size\_straight\_flush\_rank = 4;

public static final int size\_four\_of\_a\_kind\_rank = 48;

public static final int size\_full\_house\_rank = 288;

public static final int size\_flush\_rank = 4;

public static final int size\_straight\_rank = 1020;

public static final int size\_three\_of\_a\_kind\_rank = 4224;

public static final int size\_two\_pairs\_rank = 144;

public static final int size\_one\_pair\_rank = 384;

public static final int size\_high\_card\_rank = 1020;

// Access for a given rank to the sizes via an array (through the associated

// major rank):

public static final int[] size\_ranks = {0, size\_straight\_flush\_rank, size\_four\_of\_a\_kind\_rank, size\_full\_house\_rank, size\_flush\_rank, size\_straight\_rank, size\_three\_of\_a\_kind\_rank, size\_two\_pairs\_rank, size\_one\_pair\_rank, size\_high\_card\_rank};

// The number of hands of a given major rank:

public static final int num\_straight\_flushes = size\_straight\_flush\_rank \* num\_straight\_flush\_ranks;

public static final int num\_four\_of\_a\_kinds = size\_four\_of\_a\_kind\_rank \* num\_four\_of\_a\_kind\_ranks;

public static final int num\_full\_houses = size\_full\_house\_rank \* num\_full\_house\_ranks;

public static final int num\_flushes = size\_flush\_rank \* num\_flush\_ranks;

public static final int num\_straights = size\_straight\_rank \* num\_straight\_ranks;

public static final int num\_three\_of\_a\_kinds = size\_three\_of\_a\_kind\_rank \* num\_three\_of\_a\_kind\_ranks;

public static final int num\_two\_pairs = size\_two\_pairs\_rank \* num\_two\_pairs\_ranks;

public static final int num\_one\_pairs = size\_one\_pair\_rank \* num\_one\_pair\_ranks;

public static final int num\_high\_cards = size\_high\_card\_rank \* num\_high\_card\_ranks;

// Access for a given major rank to its size via an array:

public static final int[] size\_major\_ranks = {0, num\_straight\_flushes, num\_four\_of\_a\_kinds, num\_full\_houses, num\_flushes, num\_straights, num\_three\_of\_a\_kinds, num\_two\_pairs, num\_one\_pairs, num\_high\_cards};

// The number of hands up to a given major rank:

public static final int cum\_num\_straight\_flushes = num\_straight\_flushes;

public static final int cum\_num\_four\_of\_a\_kinds = cum\_num\_straight\_flushes + num\_four\_of\_a\_kinds;

public static final int cum\_num\_full\_houses = cum\_num\_four\_of\_a\_kinds + num\_full\_houses;

public static final int cum\_num\_flushes = cum\_num\_full\_houses + num\_flushes;

public static final int cum\_num\_straights = cum\_num\_flushes + num\_straights;

public static final int cum\_num\_three\_of\_a\_kinds = cum\_num\_straights + num\_three\_of\_a\_kinds;

public static final int cum\_num\_two\_pairs = cum\_num\_three\_of\_a\_kinds + num\_two\_pairs;

public static final int cum\_num\_one\_pairs = cum\_num\_two\_pairs + num\_one\_pairs;

public static final int cum\_num\_high\_cards = cum\_num\_one\_pairs + num\_high\_cards;

// Access for a given major rank to the cumulated sizes via an array:

public static final int[] cumulated\_size\_major\_ranks = {0, cum\_num\_straight\_flushes, cum\_num\_four\_of\_a\_kinds, cum\_num\_full\_houses, cum\_num\_flushes, cum\_num\_straights, cum\_num\_three\_of\_a\_kinds, cum\_num\_two\_pairs, cum\_num\_one\_pairs, cum\_num\_high\_cards};

public static final String[] major\_hand\_rank\_names = {"Straight flush", "Four of a kind", "Full house", "Flush", "Straight", "Three of a kind", "Two pairs", "One pair", "High card"};

public static boolean valid\_hand\_rank(final int r) {

return r >= 1 && r <= num\_hand\_ranks;

}

public static int major\_rank\_(final int r) {

assert valid\_hand\_rank(r);

for (int i = 1; i <= num\_major\_hand\_ranks; ++i)

if (r <= cumulated\_num\_ranks[i]) return i;

return -1; // for the compiler; won't be executed for valid r

}

// Analyses a hand and returns an array with major-rank and minor-rank:

public static int[] hand\_rank(final Hand h) {

int[] result = new int[2];

int[] rank\_count = new int[CardRank.num\_ranks];

for (int i = 1; i <= Hand.hand\_size; ++i)

++rank\_count[h.get(i).rank.index];

int[] count\_of\_counts = new int[Suit.num\_suites+1];

for (int i = 0; i < CardRank.num\_ranks; ++i)

++count\_of\_counts[rank\_count[i]];

if (count\_of\_counts[4] == 1) { // four of a kind

result[0] = four\_of\_a\_kind;

final int rank\_quad = h.get(2).rank.index;

result[1] = rank\_quad + 1;

return result;

}

if (count\_of\_counts[3] == 1) {

final int rank\_triple = h.get(3).rank.index;

result[1] = rank\_triple + 1;

if (count\_of\_counts[2] == 1) { // full house

result[0] = full\_house;

return result;

}

else { // three of a kind

result[0] = three\_of\_a\_kind;

return result;

}

}

if (count\_of\_counts[2] == 2) { // two pairs

result[0] = two\_pairs;

int[] ranks = new int[3];

transfer\_ranks(rank\_count, ranks);

final int remaining\_ranks = CardRank.num\_ranks - 2;

result[1] = (lex\_order\_13(ranks[0],ranks[1])-1)\*remaining\_ranks+adjusted\_rank(ranks[0],ranks[1],ranks[2])+1;

return result;

}

if (count\_of\_counts[2] == 1) { // one pair

result[0] = one\_pair;

int[] ranks = new int[4];

transfer\_ranks(rank\_count, ranks);

final int num\_remaining\_triples = 220; // = binom(13-1,3)

result[1] = ranks[0]\*num\_remaining\_triples+lex\_order\_12(adjusted\_rank(ranks[0],ranks[1]),adjusted\_rank(ranks[0],ranks[2]),adjusted\_rank(ranks[0],ranks[3]));

return result;

}

if (is\_straight(h)) {

if (h.get(1).rank.index == CardRank.ace && h.get(2).rank.index == CardRank.five)

result[1] = CardRank.five + 1;

else

result[1] = h.get(1).rank.index + 1;

if (is\_flush(h))

result[0] = straight\_flush;

else

result[0] = straight;

return result;

}

else {

int[] ranks = new int[5];

transfer\_ranks(rank\_count, ranks);

result[1] = lex\_order\_13(ranks[0],ranks[1],ranks[2],ranks[3],ranks[4]);

result[1] -= h.get(1).rank.index+1;

if (h.get(1).rank.index != CardRank.ace) // subtracting the low-ace-case

--result[1];

if (is\_flush(h))

result[0] = flush;

else

result[0] = high\_card;

return result;

}

}

HandRank(final int r) {

assert valid\_hand\_rank(r);

rank = r;

major\_rank = major\_rank\_(rank);

minor\_rank = r - cumulated\_num\_ranks[major\_rank-1];

}

HandRank(final Hand h) {

final int[] hr = hand\_rank(h);

major\_rank = hr[0];

minor\_rank = hr[1];

rank = cumulated\_num\_ranks[major\_rank-1] + minor\_rank;

}

public final int rank;

public final int major\_rank;

public final int minor\_rank;

public String toString() {

return "Major rank: " + major\_hand\_rank\_names[major\_rank-1] + "\nMinor rank: " + minor\_rank + "; total rank: " + rank;

}

public boolean equals(final HandRank hr) {

return hr.rank == rank;

}

// The probability that a (strictly) better hand than the given hand

// occurs for a random hand:

public double cumulated\_probability() {

return (double) cumulated\_count() / Hand.num\_hands;

}

// The number of (strictly) better hands (than the given hand):

public int cumulated\_count() {

return cumulated\_size\_major\_ranks[major\_rank-1] + (minor\_rank-1) \* size\_ranks[major\_rank];

}

public static boolean is\_flush(final Hand h) {

final Suit first = h.get(1).suit;

for (int i = 2; i <= 5; ++i)

if (! h.get(i).suit.equals(first))

return false;

return true;

}

public static boolean is\_straight(final Hand h) {

if (h.get(1).rank.index == CardRank.ace && h.get(2).rank.index == CardRank.five && h.get(5).rank.index == CardRank.two)

return true;

else if (h.get(1).rank.index + 5 - 1 == h.get(5).rank.index)

return true;

else

return false;

}

// Transfer the ranks from rank\_count to ranks, where rank\_count[j] > 0

// means that rank j is present, and will be entered into the ordered list

// "ranks" of ranks, where pairs come first (no rank occurs more than

// twice):

private static void transfer\_ranks(final int[] rank\_count, final int[] ranks) {

int i = 0;

for (int j = 0; j < CardRank.num\_ranks; ++j)

if (rank\_count[j] == 2)

ranks[i++] = j;

for (int j = 0; j < CardRank.num\_ranks; ++j)

if (rank\_count[j] == 1)

ranks[i++] = j;

}

// Determine the adjusted rank of "rank" when p1, p2 are not taken into

// account:

private static int adjusted\_rank(final int p1, final int rank) {

if (rank <= p1) return rank;

return rank - 1;

}

private static int adjusted\_rank(final int p1, final int p2, final int rank) {

assert p1 < p2;

if (rank <= p1) return rank;

if (rank <= p2) return rank - 1;

return rank - 2;

}

// Functions for ranking subsets S of {0,1,...,12} resp. {0,1,...,11}

// for set-sizes 2,3,5; the elements of S are given by x1 < ... < x5:

private static int lex\_order\_13(final int x1, final int x2) {

return 66-(11-x1)\*(12-x1)/2+x2;

}

private static int lex\_order\_12(final int x1, final int x2, final int x3) {

return 209+(-(9-x1)\*(10-x1)\*(11-x1))/6-(10-x2)\*(11-x2)/2+x3;

}

private static int lex\_order\_13(final int x1, final int x2, final int x3, final int x4, final int x5) {

return 1275+(-(8-x1)\*(9-x1)\*(10-x1)\*(11-x1)\*(12-x1))/120+(-(9-x2)\*(10-x2)\*(11-x2)\*(12-x2))/24+(-(10-x3)\*(11-x3)\*(12-x3))/6-(11-x4)\*(12-x4)/2+x5;

}

// Tests (run by "java -ea HandRank", enabling assertions):

public static void main(String[] args) {

assert num\_hand\_ranks == 6331;

assert num\_straight\_flushes == 40;

assert num\_four\_of\_a\_kinds == 624;

assert num\_full\_houses == 3744;

assert num\_flushes == 5108;

assert num\_straights == 10200;

assert num\_three\_of\_a\_kinds == 54912;

assert num\_two\_pairs == 123552;

assert num\_one\_pairs == 1098240;

assert num\_high\_cards == 1302540;

assert cum\_num\_high\_cards == Hand.num\_hands;

// Testing the various functions for ranking subsets according to

// lexicographical order:

assert lex\_order\_13(0,1) == 1;

assert lex\_order\_13(11,12) == 78;

assert lex\_order\_12(0,1,2) == 1;

assert lex\_order\_12(9,10,11) == 220;

assert lex\_order\_13(0,1,2,3,4) == 1;

assert lex\_order\_13(8,9,10,11,12) == 1277 + 10;

{

final Hand h = new Hand(new Card(0), new Card(1), new Card(14), new Card(2), new Card(15));

final HandRank hr = new HandRank(h);

assert hr.major\_rank == two\_pairs;

assert hr.minor\_rank == 133;

}

{

final Hand h = new Hand(new Card(3), new Card(1), new Card(14), new Card(2), new Card(15));

final HandRank hr = new HandRank(h);

assert hr.major\_rank == two\_pairs;

assert hr.minor\_rank == 134;

}

{

final Hand h = new Hand(new Card(0), new Card(13), new Card(2), new Card(15), new Card(1));

final HandRank hr = new HandRank(h);

assert hr.major\_rank == two\_pairs;

assert hr.minor\_rank == 12;

}

{

final Hand h = new Hand(new Card(0), new Card(1), new Card(14), new Card(2), new Card(3));

final HandRank hr = new HandRank(h);

assert hr.major\_rank == one\_pair;

assert hr.minor\_rank == 221;

}

// Running through all possible hands, and detemining the ranks:

int[] statistics\_ranks = new int[num\_hand\_ranks+1];

int[] statistics\_major\_ranks = new int[num\_major\_hand\_ranks+1];

for (int c1 = 0; c1 < Card.num\_cards - 4; ++c1) {

final Card C1 = new Card(c1);

for (int c2 = c1+1; c2 < Card.num\_cards - 3; ++c2) {

final Card C2 = new Card(c2);

for (int c3 = c2+1; c3 < Card.num\_cards - 2; ++c3) {

final Card C3 = new Card(c3);

for (int c4 = c3+1; c4 < Card.num\_cards - 1; ++c4) {

final Card C4 = new Card(c4);

for (int c5 = c4+1; c5 < Card.num\_cards; ++c5) {

final Card C5 = new Card(c5);

final Hand h = new Hand(C1,C2,C3,C4,C5);

final HandRank hr = new HandRank(h);

++statistics\_ranks[hr.rank];

++statistics\_major\_ranks[hr.major\_rank];

}

}

}

}

}

statistics\_major\_ranks[0] = 0;

for (int i = 0; i <= num\_major\_hand\_ranks; ++i)

assert statistics\_major\_ranks[i] == size\_major\_ranks[i];

for (int rank = 1; rank <= num\_hand\_ranks; ++rank) {

final HandRank hr = new HandRank(rank);

assert statistics\_ranks[rank] == size\_ranks[hr.major\_rank];

}

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Hand {

// Provides poker hands in a standardised form, sorted by descending

// ranks.

// Three constructors are given, for reading five cards, an array of cards,

// or for reading from standard input.

// If the hand does not consist of five different cards, then an exception

// is thrown.

// Access to cards by get(i), where 1 <= i <= 5.

// Dependencies: In.java.

public static final int hand\_size = 5;

public static final int num\_hands = 2598960; // = binom(52,5)

// This constructor takes over ownership of c1,...,c5:

public Hand(final Card c1, final Card c2, final Card c3, final Card c4, final Card c5) {

cards = new Card[hand\_size];

cards[0] = c1;

cards[1] = c2;

cards[2] = c3;

cards[3] = c4;

cards[4] = c5;

prepare\_hand();

}

// This constructor takes over ownership of the five elements of h:

public Hand(final Card[] h) {

assert h != null;

assert h.length == hand\_size;

cards = new Card[hand\_size];

for (int i = 0; i < hand\_size; ++i)

cards[i] = h[i];

prepare\_hand();

}

// Reading from an input stream:

public Hand(final In in) {

cards = new Card[hand\_size];

for (int i = 0; i < hand\_size; ++i) {

final CardRank rank = new CardRank(in.readString());

if (! in.readString().equals("of"))

throw new RuntimeException("After the card-rank a string different from \"of\" has been found.");

final Suit suit = new Suit(in.readString());

cards[i] = new Card(rank, suit);

}

prepare\_hand();

}

public Card get(final int i) {

assert i >= 1;

assert i <= hand\_size;

return cards[i-1];

}

public String toString() {

String result = "";

for (int i = 0; i < hand\_size; ++i) {

final Card c = cards[i];

result += c.rank.toString();

result += " of ";

result += c.suit.toString();

result += "; ";

}

return result;

}

public boolean equals(final Hand h) {

for (int i = 0; i < hand\_size; ++i)

if (! h.cards[i].equals(cards[i]))

return false;

return true;

}

private final Card[] cards;

// Sorting by selection sort:

private static void sort\_by\_ranks(final Card[] h) {

for (int i = 0; i < h.length-1; ++i) {

int index\_min = i;

for (int j = i+1; j < h.length; ++j)

if (h[j].rank.index < h[index\_min].rank.index)

index\_min = j;

if (index\_min != i) {

final Card temp = h[i];

h[i] = h[index\_min];

h[index\_min] = temp;

}

}

}

/\* Remarks: If we wanted to use a sorting algorithm from the Java library,

then we needed some means to "tell" that algorithm the sorting criterions;

by what we learned in the module, yet we cannot provide such means.

By the above private method we can provide a specialised method,

tailored for our needs.

\*/

public static void check\_all\_different(final Card[] h) {

for (int i = 0; i < hand\_size - 1; ++i)

if (h[i].equals(h[i+1]))

throw new RuntimeException("Two identical cards were found in a hand.");

}

private void prepare\_hand() {

sort\_by\_ranks(cards);

check\_all\_different(cards);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class ExchangeRequest {

// For an object e of type ExchangeRequest, by e.number\_cards we obtain

// the number of cards to be exchanged, while by get\_index(i) for

// 1 <= i <= number\_cards we get the index of the i-th card to be exchanged.

/\*

For example assume that we want to exchange cards 2 and 5:

int[] ea = new int[2];

ea[0] = 2;

ea[1] = 5;

final ExchangeRequest e = new ExchangeRequest(ea);

assert e.number\_cards == 2;

assert e.get\_index(1) == 2;

assert e.get\_index(2) == 5;

\*/

public static final int max\_exchanges = 2;

public static final int total\_num\_requests = 1 + 5 + 10;

public final int number\_cards;

public ExchangeRequest() {

number\_cards = 0;

exchange\_indices = null;

}

public ExchangeRequest(final int e1) {

assert e1 >= 1;

assert e1 <= Hand.hand\_size;

number\_cards = 1;

exchange\_indices = new int[1];

exchange\_indices[0] = e1;

}

public ExchangeRequest(final int e1, final int e2) {

assert e1 >= 1;

assert e1 <= Hand.hand\_size;

assert e2 >= 1;

assert e2 <= Hand.hand\_size;

assert e1 < e2;

number\_cards = 2;

exchange\_indices = new int[2];

exchange\_indices[0] = e1;

exchange\_indices[1] = e2;

}

public ExchangeRequest(int[] exchange\_indices\_) {

assert exchange\_indices\_ != null;

assert(exchange\_indices\_.length <= max\_exchanges);

for (int i = 0; i < exchange\_indices\_.length; ++i) {

assert(exchange\_indices\_[i] >= 1);

assert(exchange\_indices\_[i] <= Hand.hand\_size);

}

if (exchange\_indices\_.length == max\_exchanges)

assert exchange\_indices\_[0] < exchange\_indices\_[1];

exchange\_indices = exchange\_indices\_;

number\_cards = exchange\_indices.length;

}

public int get\_index(final int i) {

assert i >= 1;

assert i <= number\_cards;

return exchange\_indices[i-1];

}

public String toString() {

if (number\_cards == 0)

return "()";

if (number\_cards == 1)

return "(" + exchange\_indices[0] + ")";

return "(" + exchange\_indices[0] + "," + exchange\_indices[1] + ")";

}

public boolean equals(final ExchangeRequest E) {

if (E.number\_cards != number\_cards) return false;

if (number\_cards == 0) return true;

if (number\_cards == 1) return E.exchange\_indices[0] == exchange\_indices[0];

return E.exchange\_indices[0] == exchange\_indices[0] && E.exchange\_indices[1] == exchange\_indices[1];

}

public static ExchangeRequest[] all\_requests() {

final ExchangeRequest[] E = new ExchangeRequest[total\_num\_requests];

E[0] = new ExchangeRequest();

for (int i = 1; i <= Hand.hand\_size; ++i)

E[i] = new ExchangeRequest(i);

int next\_index = Hand.hand\_size+1;

for (int i = 1; i <= Hand.hand\_size-1; ++i)

for (int j =i+1; j <= Hand.hand\_size; ++j)

E[next\_index++] = new ExchangeRequest(i,j);

return E;

}

private final int[] exchange\_indices;

public static void main(String[] args) {

final ExchangeRequest[] E = all\_requests();

assert E.length == total\_num\_requests;

for (int i = 0; i < total\_num\_requests; ++i)

System.out.println(E[i]);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Evaluation {

public static final int num\_other\_cards = Card.num\_cards - Hand.hand\_size;

public Evaluation(final Hand h) {

assert h != null;

hand = h;

other\_cards = new Card[num\_other\_cards];

int[] deck = new int[Card.num\_cards];

for (int i = 0; i < Card.num\_cards; ++i)

deck[i] = i;

for (int i = 1; i <= Hand.hand\_size; ++i)

deck[h.get(i).index()] = -1;

int next\_index = 0;

for (int i = 0; i < Card.num\_cards; ++i)

if (deck[i] != -1)

other\_cards[next\_index++] = new Card(deck[i]);

}

// For an exchange-request e, compute the array of possible outcomes

// as hand-ranks, together with their probabilities, sorted by

// descending hand-ranks:

public EvaluationResult evaluate(final ExchangeRequest e) {

if (e.number\_cards == 0) {

EvaluatedOutcome eo = new EvaluatedOutcome(new HandRank(hand), 1.0);

EvaluatedOutcome[] result = new EvaluatedOutcome[1];

result[0] = eo;

return new EvaluationResult(result);

}

int[] count\_outcomes = new int[HandRank.num\_hand\_ranks+1];

int num\_possibilities;

// run through all possibilities, and enter the result into count\_outcomes:

final Card[] new\_hand = new Card[5];

for (int i = 0; i < Hand.hand\_size; ++i)

new\_hand[i] = hand.get(i+1);

if (e.number\_cards == 1) {

num\_possibilities = num\_other\_cards;

final int exchange\_index = e.get\_index(1)-1;

for (int i = 0; i < num\_other\_cards; ++i) {

new\_hand[exchange\_index] = other\_cards[i];

++count\_outcomes[new HandRank((new Hand(new\_hand))).rank];

}

}

else {

num\_possibilities = num\_other\_cards \* (num\_other\_cards-1);

final int exchange\_index\_1 = e.get\_index(1)-1;

final int exchange\_index\_2 = e.get\_index(2)-1;

for (int i = 0; i < num\_other\_cards; ++i) {

new\_hand[exchange\_index\_1] = other\_cards[i];

for (int j = i+1; j < num\_other\_cards; ++j) {

new\_hand[exchange\_index\_2] = other\_cards[j];

++count\_outcomes[new HandRank((new Hand(new\_hand))).rank];

}

}

}

int count\_different\_ranks = 0;

// determine the number of different ranks entered into count\_outcomes:

for (int i = 1; i <= HandRank.num\_hand\_ranks; ++i)

if (count\_outcomes[i] != 0) ++count\_different\_ranks;

// transfer the results into an array of evaluated outcomes:

EvaluatedOutcome[] result = new EvaluatedOutcome[count\_different\_ranks];

int next\_index = 0;

for (int i = 1; i <= HandRank.num\_hand\_ranks; ++i)

if (count\_outcomes[i] != 0)

result[next\_index++] = new EvaluatedOutcome(new HandRank(i), (double) count\_outcomes[i] / num\_possibilities);

return new EvaluationResult(result);

}

private final Hand hand;

private final Card[] other\_cards;

public static void main(String[] args) {

final Bank b = new Bank(1);

final Hand h = b.orig\_hand(1);

System.out.println(h);

final Evaluation E = new Evaluation(h);

{

System.out.println("\nNo exchange:");

final ExchangeRequest e = new ExchangeRequest();

final EvaluationResult R = E.evaluate(e);

assert R.length == 1;

System.out.println(R);

}

{

System.out.println("\nExchange last card:");

final int[] ea = new int[1];

ea[0] = 5;

final ExchangeRequest e = new ExchangeRequest(ea);

final EvaluationResult R = E.evaluate(e);

System.out.println(R);

}

}

}

// Oliver Kullmann, 4.12.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class EvaluationResult {

public EvaluationResult(final EvaluatedOutcome[] E) {

assert E != null;

results = E;

length = results.length;

}

public EvaluatedOutcome get(final int index) {

assert index >= 1;

assert index <= length;

return results[index-1];

}

public String toString() {

String res = "";

for (int i = 0; i < length; ++i)

res += "\n" + (i+1) + ": " + results[i];

return res;

}

private final EvaluatedOutcome[] results;

public final int length;

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class EvaluatedOutcome {

// For a given hand and an exchange request, an "evaluated outcome"

// is one of the possible hand-ranks which can arise after exchange,

// together with the probability that this hand-rank will be obtained.

public EvaluatedOutcome(final HandRank hr, final double p) {

assert p >= 0;

assert p <= 1;

hand\_rank = hr;

prob = p;

}

public final HandRank hand\_rank;

public final double prob;

public String toString() {

return hand\_rank + "\nProbability: " + prob;

}

public boolean equals(final EvaluatedOutcome e) {

return e.hand\_rank.equals(hand\_rank) && e.prob == prob;

}

public static void main(String[] args) {

System.out.println(new EvaluatedOutcome(new HandRank(1), 0.9));

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class CardRank {

// Wrapper class around card ranks as integer indices.

// Note that a smaller index means a higher card.

public static final int num\_ranks = 13;

public static final int ace = 0;

public static final int king = 1;

public static final int queen = 2;

public static final int jack = 3;

public static final int ten = 4;

public static final int nine = 5;

public static final int eight = 6;

public static final int seven = 7;

public static final int six = 8;

public static final int five = 9;

public static final int four = 10;

public static final int three = 11;

public static final int two = 12;

public final int index;

public CardRank(final int i) {

assert(i >= 0);

assert(i < num\_ranks);

index = i;

}

public CardRank(final String str) {

index = determine\_index(str);

}

public String toString() {

return rank\_names[index];

}

public boolean equals(final CardRank s) {

return s.index == index;

}

private static final String[] rank\_names = {"Ace", "King", "Queen", "Jack", "10", "9", "8", "7", "6", "5", "4", "3", "2"};

private static int determine\_index(final String str) {

for (int i = 0; i < num\_ranks; ++i)

if (str.equals(rank\_names[i]))

return i;

throw new RuntimeException("Does not represent a card rank: " + str);

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Card {

// A "card" as a pair of (card-)rank and suit.

public static final int num\_cards = Suit.num\_suites \* CardRank.num\_ranks;

public final CardRank rank;

public final Suit suit;

public Card(final CardRank r, final Suit s) {

rank = r;

suit = s;

}

public Card(final int i) {

assert i >= 0;

assert i < num\_cards;

rank = new CardRank(i % CardRank.num\_ranks);

suit = new Suit(i / CardRank.num\_ranks);

}

public int index() {

return suit.index \* CardRank.num\_ranks + rank.index;

}

public String toString() {

return rank.toString() + " of " + suit.toString();

}

public boolean equals(final Card c) {

return c.rank.equals(rank) && c.suit.equals(suit);

}

// Demonstration of functionality:

public static void main(String[] args) {

System.out.println(new Card(0));

System.out.println(new Card(51));

final Card c1 = new Card(new CardRank("Jack"), new Suit("Spades"));

System.out.println(c1);

// Demonstration that here references are harmless, since CardRank, Suit

// and also Card behave similar to value-types (they are non-mutable):

CardRank cr2 = new CardRank("5");

Suit s2 = new Suit("Hearts");

final Card c2 = new Card(cr2, s2);

System.out.println(c2);

cr2 = new CardRank("King");

s2 = new Suit("Clubs");

System.out.println(c2); // still the same, since the old cr2- and s2-objects didn't change

System.out.println(new Card(cr2,s2));

}

}

// Oliver Kullmann, 26.11.2009 (Swansea)

/\* Copyright 2009 Oliver Kullmann \*/

class Bank {

// By Bank(n) for 0 <= n <= 7 we create n hands, which can be obtained

// by orig\_hand(p). Then by perform\_exchanges(E) for an array E of

// exchange requests we exchange cards in the hands, where the new hands

// are available then by new\_hand(p). Such exchange requests can be

// used arbitrarily often (always using the original deck of cards and the

// original hands).

public Bank(final int n) {

assert n >= 0;

assert n <= max\_number\_players;

number\_players = n;

deck = new int[Card.num\_cards];

for (int i = 0; i < Card.num\_cards; ++i)

deck[i] = i;

shuffle\_deck();

orig\_hands = new Hand[number\_players];

set\_orig\_hands();

}

public static final int max\_number\_players = 7; // 7\*(5+2) = 49

public final int number\_players;

// Returns the hand of a player before exchange:

public Hand orig\_hand(final int player) {

assert player >= 1;

assert player <= number\_players;

return orig\_hands[player-1];

}

// Returns the hand of a player after exchange:

public Hand new\_hand(final int player) {

assert player >= 1;

assert player <= number\_players;

assert new\_hands != null;

return new\_hands[player-1];

}

// Given an array E of exchange-requests (one for each player),

// set new\_hands accordingly; this function can be applied

// arbitrarily often:

public void perform\_exchanges(final ExchangeRequest[] E) {

assert E.length == number\_players;

new\_hands = new Hand[number\_players];

int next\_card\_index = number\_players \* Hand.hand\_size;

for (int i = 0; i < number\_players; ++i) {

Card[] h = new Card[5];

for (int j = 0; j < Hand.hand\_size; ++j)

h[j] = orig\_hands[i].get(j+1);

final ExchangeRequest e = E[i];

for (int j = 1; j <= e.number\_cards; ++j) {

h[e.get\_index(j)-1] = new Card(deck[next\_card\_index]);

++next\_card\_index;

}

new\_hands[i] = new Hand(h);

}

}

private final int[] deck; // here the card \*indices\* are used

private final Hand[] orig\_hands;

private Hand[] new\_hands;

// Random shuffling of the deck of cards:

private void shuffle\_deck() {

for (int i = 0; i < Card.num\_cards; ++i) {

final int rand = i + (int) (Math.random() \* (Card.num\_cards-i));

final int t = deck[rand];

deck[rand] = deck[i];

deck[i] = t;

}

}

// Set orig\_hands:

private void set\_orig\_hands() {

Card[] h = new Card[5];

int current\_card\_index = 0;

for (int i = 0; i < number\_players; ++i) {

for (int c = 0; c < Hand.hand\_size; ++c, ++current\_card\_index)

h[c] = new Card(deck[current\_card\_index]);

orig\_hands[i] = new Hand(h);

}

}

// Tests:

public static void main(String[] args) {

final int num\_players = 7;

Bank B = new Bank(num\_players);

for (int i = 0; i < Card.num\_cards; ++i)

System.out.println((i+1) + ": " + new Card(B.deck[i]));

assert B.number\_players == num\_players;

assert B.deck.length == 52;

assert B.orig\_hands.length == num\_players;

assert B.new\_hands == null;

for (int i = 1; i <= num\_players; ++i)

System.out.println(i + ": " + B.orig\_hand(i));

ExchangeRequest[] E = new ExchangeRequest[num\_players];

for (int i = 0; i < num\_players; ++i) {

int[] e = new int[2];

e[0] = 1; e[1] = 3;

E[i] = new ExchangeRequest(e);

}

B.perform\_exchanges(E);

for (int i = 1; i <= num\_players; ++i)

System.out.println(i + ": " + B.new\_hand(i));

for (int i = 0; i < num\_players; ++i) {

int[] e = new int[0];

E[i] = new ExchangeRequest(e);

}

B.perform\_exchanges(E);

for (int i = 1; i <= num\_players; ++i)

System.out.println(i + ": " + B.new\_hand(i));

}

}

## Tables of probability for Texas Hold ‘em poker:

The following tables were all taken from [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011.

Frequency of 5-card poker hands**:**

## Picture3 for dissertation.png

Frequency of 7-card poker hands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hand | Frequency | Probability | Cumulative | Odds |
| Straight flush | 41,584 | 0.0311% | 0.0311% | 3,216 : 1 |
| (Royal flush) | 4,324 | 0.0032% | --- | 30,939 : 1 |
| Four of a kind | 224,848 | 0.168% | 0.199% | 594 : 1 |
| Full house | 3,473,184 | 2.60% | 2.80% | 37.5 : 1 |
| [Flush](http://en.wikipedia.org/wiki/Rank_of_hands_(poker)#Flush) | 4,047,644 | 3.03% | 5.82% | 32.1 : 1 |
| Straight | 6,180,020 | 4.62% | 10.4% | 20.6 : 1 |
| Three of a kind | 6,461,620 | 4.83% | 15.3% | 19.7 : 1 |
| Two pair | 31,433,400 | 23.5% | 38.8% | 3.26 : 1 |
| One pair | 58,627,800 | 43.8% | 82.6% | 1.28 : 1 |
| No pair | 23,294,460 | 17.4% | 100% | 4.74 : 1 |
| Total | 133,784,560 | 100% | 100% | 0 : 1 |

Frequency of 5-card lowball poker hands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hand | Distinct hands | Frequency | Probability | Cumulative | Odds |
| 5-high | 1 | 1,024 | 0.0394% | 0.0394% | 2,537.05 : 1 |
| 6-high | 5 | 5,120 | 0.197% | 0.236% | 506.61 : 1 |
| 7-high | 15 | 15,360 | 0.591% | 0.827% | 168.20 : 1 |
| 8-high | 35 | 35,840 | 1.38% | 2.21% | 71.52 : 1 |
| 9-high | 70 | 71,680 | 2.76% | 4.96% | 35.26 : 1 |
| 10-high | 126 | 129,024 | 4.96% | 9.93% | 19.14 : 1 |
| Jack-high | 210 | 215,040 | 8.27% | 18.2% | 11.09 : 1 |
| Queen-high | 330 | 337,920 | 13.0% | 31.2% | 6.69 : 1 |
| King-high | 495 | 506,880 | 19.5% | 50.7% | 4.13 : 1 |
| Total | 1,287 | 1,317,888 | 50.7% | 50.7% | 0.97 : 1 |

Frequency of 7-card lowball poker hands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hand | Frequency | Probability | Cumulative | Odds |
| 5-high | 781,824 | 0.584% | 0.584% | 170.12 : 1 |
| 6-high | 3,151,360 | 2.36% | 2.94% | 41.45 : 1 |
| 7-high | 7,426,560 | 5.55% | 8.49% | 17.01 : 1 |
| 8-high | 13,171,200 | 9.85% | 18.3% | 9.16 : 1 |
| 9-high | 19,174,400 | 14.3% | 32.7% | 5.98 : 1 |
| 10-high | 23,675,904 | 17.7% | 50.4% | 4.65 : 1 |
| Jack-high | 24,837,120 | 18.6% | 68.9% | 4.39 : 1 |
| Queen-high | 21,457,920 | 16.0% | 85.0% | 5.23 : 1 |
| King-high | 13,939,200 | 10.4% | 95.4% | 8.60 : 1 |
| Total | 127,615,488 | 95.4% | 95.4% | 0.05 : 1 |

Starting hands

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hand shape | Number of hands | Suit combinations for each hand | Combinations | Dealt specific hand | | Dealt any hand | |
| Probability | Odds | Probability | Odds |
| Pocket pair | 13 |  | 13 × 6 = 78 |  | 220 : 1 |  | 16 : 1 |
| Suited cards | 78 |  | 78 × 4 = 312 |  | 331 : 1 |  | 3.25 : 1 |
| Unsuited cards non paired | 78 |  | 78 × 12 = 936 |  | 110 : 1 |  | 0.417 : 1 |

Here are the probabilities and odds of being dealt various other types of starting hands.

|  |  |  |
| --- | --- | --- |
| Hand | Probability | Odds |
| AKs (or any specific suited cards) | 0.00302 | 331 : 1 |
| AA (or any specific pair) | 0.00452 | 220 : 1 |
| AKs, KQs, QJs, or JTs (suited cards) | 0.0121 | 81.9 : 1 |
| AK (or any specific non-pair incl. suited) | 0.0121 | 81.9 : 1 |
| AA, KK, or QQ | 0.0136 | 72.7 : 1 |
| AA, KK, QQ or JJ | 0.0181 | 54.3 : 1 |
| Suited cards, jack or better | 0.0181 | 54.3 : 1 |
| AA, KK, QQ, JJ, or TT | 0.0226 | 43.2 : 1 |
| Suited cards, 10 or better | 0.0302 | 32.2 : 1 |
| Suited connectors | 0.0392 | 24.5 : 1 |
| Connected cards, 10 or better | 0.0483 | 19.7 : 1 |
| Any 2 cards with rank at least queen | 0.0498 | 19.1 : 1 |
| Any 2 cards with rank at least jack | 0.0905 | 10.1 : 1 |
| Any 2 cards with rank at least 10 | 0.143 | 5.98 : 1 |
| Connected cards (cards of consecutive rank) | 0.157 | 5.38 : 1 |
| Any 2 cards with rank at least 9 | 0.208 | 3.81 : 1 |
| Not connected nor suited, at least one 2-9 | 0.534 | 0.873 : 1 |

Head-to-head starting hand matchups

|  |  |  |
| --- | --- | --- |
| Favorite-to-underdog matchup | Probability | Odds for |
| Pair vs. 2 undercards | 0.83 | 4.9 : 1 |
| Pair vs. lower pair | 0.82 | 4.5 : 1 |
| Pair vs. 1 overcard, 1 undercard | 0.71 | 2.5 : 1 |
| 2 overcards vs. 2 undercards | 0.63 | 1.7 : 1 |
| Pair vs. 2 overcards | 0.55 | 1.2 : 1 |

The following table shows the number of hand combinations for up to nine opponents.

|  |  |
| --- | --- |
| Opponents | Number of possible hand combinations |
| 1 | 1,225 |
| 2 | 690,900 |
| 3 | 238,360,500 |
| 4 | 56,372,258,250 |
| 5 | ≈9.7073 × 1012 (more than 9 trillion) |
| 6 | ≈1.2620 × 1015 (more than 1 quadrillion) |
| 7 | ≈1.2674 × 1017 (more than 126 quadrillion) |
| 8 | ≈9.9804 × 1018 (almost 10 quintillion) |
| 9 | ≈6.2211 × 1020 (more than 622 quintillion) |

The following table shows the probability that before the flop another player has a larger pocket pair when there are one to nine other players in the hand.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Probability of facing a larger pair when holding | Against 1 | Against 2 | Against 3 | Against 4 | Against 5 | Against 6 | Against 7 | Against 8 | Against 9 |
| KK | 0.0049 | 0.0098 | 0.0147 | 0.0196 | 0.0244 | 0.0293 | 0.0342 | 0.0391 | 0.0439 |
| QQ | 0.0098 | 0.0195 | 0.0292 | 0.0388 | 0.0484 | 0.0579 | 0.0673 | 0.0766 | 0.0859 |
| JJ | 0.0147 | 0.0292 | 0.0436 | 0.0577 | 0.0717 | 0.0856 | 0.0992 | 0.1127 | 0.1259 |
| TT | 0.0196 | 0.0389 | 0.0578 | 0.0764 | 0.0946 | 0.1124 | 0.1299 | 0.1470 | 0.1637 |
| 99 | 0.0245 | 0.0484 | 0.0718 | 0.0946 | 0.1168 | 0.1384 | 0.1593 | 0.1795 | 0.1990 |
| 88 | 0.0294 | 0.0580 | 0.0857 | 0.1125 | 0.1384 | 0.1634 | 0.1873 | 0.2101 | 0.2318 |
| 77 | 0.0343 | 0.0674 | 0.0994 | 0.1301 | 0.1595 | 0.1874 | 0.2138 | 0.2387 | 0.2619 |
| 66 | 0.0392 | 0.0769 | 0.1130 | 0.1473 | 0.1799 | 0.2104 | 0.2389 | 0.2651 | 0.2890 |
| 55 | 0.0441 | 0.0862 | 0.1263 | 0.1642 | 0.1996 | 0.2324 | 0.2623 | 0.2892 | 0.3129 |
| 44 | 0.0490 | 0.0956 | 0.1395 | 0.1806 | 0.2186 | 0.2532 | 0.2841 | 0.3109 | 0.3334 |
| 33 | 0.0539 | 0.1048 | 0.1526 | 0.1967 | 0.2370 | 0.2729 | 0.3040 | 0.3300 | 0.3503 |
| 22 | 0.0588 | 0.1141 | 0.1654 | 0.2124 | 0.2546 | 0.2914 | 0.3222 | 0.3464 | 0.3633 |

The following table gives the probability that a hand is facing two or more larger pairs before the flop. From the previous equations, the probability Pm is computed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Probability of facing multiple larger pairs when holding | Against 2 | Against 3 | Against 4 | Against 5 | Against 6 | Against 7 | Against 8 | Against 9 |
| KK | < 0.00001 | 0.00001 | 0.00003 | 0.00004 | 0.00007 | 0.00009 | 0.00012 | 0.00016 |
| QQ | 0.00006 | 0.00018 | 0.00037 | 0.00061 | 0.00091 | 0.00128 | 0.00171 | 0.00220 |
| JJ | 0.00017 | 0.00051 | 0.00102 | 0.00171 | 0.00257 | 0.00360 | 0.00482 | 0.00621 |
| TT | 0.00033 | 0.00099 | 0.00200 | 0.00335 | 0.00504 | 0.00709 | 0.00950 | 0.01226 |
| 99 | 0.00054 | 0.00164 | 0.00330 | 0.00553 | 0.00836 | 0.01177 | 0.01580 | 0.02045 |
| 88 | 0.00081 | 0.00244 | 0.00493 | 0.00828 | 0.01253 | 0.01769 | 0.02378 | 0.03084 |
| 77 | 0.00112 | 0.00341 | 0.00689 | 0.01160 | 0.01758 | 0.02487 | 0.03351 | 0.04353 |
| 66 | 0.00149 | 0.00454 | 0.00918 | 0.01550 | 0.02353 | 0.03335 | 0.04503 | 0.05861 |
| 55 | 0.00191 | 0.00583 | 0.01182 | 0.01998 | 0.03040 | 0.04318 | 0.05840 | 0.07619 |
| 44 | 0.00239 | 0.00728 | 0.01480 | 0.02506 | 0.03821 | 0.05438 | 0.07371 | 0.09635 |
| 33 | 0.00291 | 0.00890 | 0.01812 | 0.03075 | 0.04698 | 0.06699 | 0.09099 | 0.11919 |
| 22 | 0.00349 | 0.01068 | 0.02180 | 0.03706 | 0.05673 | 0.08107 | 0.11034 | 0.14484 |

The following table shows the probability that before the flop another player has an ace with a larger kicker in the hand.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Probability of facing an ace with larger kicker when holding | Against 1 | Against 2 | Against 3 | Against 4 | Against 5 | Against 6 | Against 7 | Against 8 | Against 9 |
| AK | 0.00245 | 0.00489 | 0.00733 | 0.00976 | 0.01219 | 0.01460 | 0.01702 | 0.01942 | 0.02183 |
| AQ | 0.01224 | 0.02434 | 0.03629 | 0.04809 | 0.05974 | 0.07126 | 0.08263 | 0.09386 | 0.10496 |
| AJ | 0.02204 | 0.04360 | 0.06468 | 0.08529 | 0.10545 | 0.12517 | 0.14445 | 0.16331 | 0.18175 |
| AT | 0.03184 | 0.06266 | 0.09250 | 0.12139 | 0.14937 | 0.17645 | 0.20267 | 0.22805 | 0.25263 |
| A9 | 0.04163 | 0.08153 | 0.11977 | 0.15642 | 0.19154 | 0.22520 | 0.25745 | 0.28837 | 0.31799 |
| A8 | 0.05143 | 0.10021 | 0.14649 | 0.19038 | 0.23202 | 0.27152 | 0.30898 | 0.34452 | 0.37823 |
| A7 | 0.06122 | 0.11870 | 0.17266 | 0.22331 | 0.27086 | 0.31550 | 0.35741 | 0.39675 | 0.43369 |
| A6 | 0.07102 | 0.13700 | 0.19829 | 0.25523 | 0.30812 | 0.35726 | 0.40291 | 0.44531 | 0.48471 |
| A5 | 0.08082 | 0.15510 | 0.22338 | 0.28615 | 0.34384 | 0.39687 | 0.44561 | 0.49041 | 0.53160 |
| A4 | 0.09061 | 0.17301 | 0.24795 | 0.31609 | 0.37806 | 0.43442 | 0.48567 | 0.53227 | 0.57465 |
| A3 | 0.10041 | 0.19073 | 0.27199 | 0.34509 | 0.41085 | 0.47000 | 0.52322 | 0.57109 | 0.61416 |
| A2 | 0.11020 | 0.20826 | 0.29552 | 0.37315 | 0.44223 | 0.50370 | 0.55840 | 0.60706 | 0.65037 |

The following are some general probabilities about what can occur on the board. These assume a "random" starting hand for the player.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Board consisting of | Making on flop | | Making by turn | | Making by river | |
| Prob. | Odds | Prob. | Odds | Prob. | Odds |
| Three or more of same suit | 0.05177 | 18.3 : 1 | 0.13522 | 6.40 : 1 | 0.23589 | 3.24 : 1 |
| Four or more of same suit |  |  | 0.01056 | 93.7 : 1 | 0.03394 | 28.5 : 1 |
| Rainbow flop (all different suits) | 0.39765 | 1.51 : 1 | 0.10550 | 8.48 : 1 |  |  |
| Three cards of consecutive rank (but not four consecutive) | 0.03475 | 27.8 : 1 | 0.11820 | 7.46 : 1 | 0.25068 | 2.99 : 1 |
| Four cards to a straight (but not five) |  |  | 0.03877 | 24.8 : 1 | 0.18991 | 4.27 : 1 |
| Three or more cards of consecutive rank and same suit | 0.00217 | 459 : 1 | 0.00869 | 114 : 1 | 0.02172 | 45.0 : 1 |
| Three of a kind (but not a full house or four of a kind) | 0.00235 | 424 : 1 | 0.00935 | 106 : 1 | 0.02128 | 46 : 1 |
| A pair (but not two pair or three or four of a kind) | 0.16941 | 4.90 : 1 | 0.30417 | 2.29 : 1 | 0.42450 | 1.36 : 1 |
| Two pair (but not a full house) |  |  | 0.01037 | 95.4 : 1 | 0.04716 | 20.2 : 1 |

The following table gives the probability that no overcards will come on the flop, turn and river, for each of the pocket pairs from 3 to K.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Holding pocket pair | No overcard on flop | | No overcard by turn | | No overcard by river | |
| Prob. | Odds | Prob. | Odds | Prob. | Odds |
| KK | 0.7745 | 0.29 : 1 | 0.7086 | 0.41 : 1 | 0.6470 | 0.55 : 1 |
| QQ | 0.5857 | 0.71 : 1 | 0.4860 | 1.06 : 1 | 0.4015 | 1.49 : 1 |
| JJ | 0.4304 | 1.32 : 1 | 0.3205 | 2.12 : 1 | 0.2369 | 3.22 : 1 |
| TT | 0.3053 | 2.28 : 1 | 0.2014 | 3.97 : 1 | 0.1313 | 6.61 : 1 |
| 99 | 0.2071 | 3.83 : 1 | 0.1190 | 7.40 : 1 | 0.0673 | 13.87 : 1 |
| 88 | 0.1327 | 6.54 : 1 | 0.0649 | 14.40 : 1 | 0.0310 | 31.21 : 1 |
| 77 | 0.0786 | 11.73 : 1 | 0.0318 | 30.48 : 1 | 0.0124 | 79.46 : 1 |
| 66 | 0.0416 | 23.02 : 1 | 0.0133 | 74.26 : 1 | 0.0040 | 246.29 : 1 |
| 55 | 0.0186 | 52.85 : 1 | 0.0043 | 229.07 : 1 | 0.0009 | 1,057.32 : 1 |
| 44 | 0.0061 | 162.33 : 1 | 0.0009 | 1,095.67 : 1 | 0.0001 | 8,406.78 : 1 |
| 33 | 0.0010 | 979.00 : 1 | 0.0001 | 15,352.33 : 1 | 0.0000 | 353,125.67 : 1 |

For reference, the probability and odds for some of the more common numbers of outs are given here.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Example drawing to | Outs | Make on turn | | Make on river | | Make on turn or river | |
| Prob. | Odds | Prob. | Odds | Prob. | Odds |
| Inside straight flush; Four of a kind | 1 | 0.0213 | 46.0 : 1 | 0.0217 | 45.0 : 1 | 0.0426 | 22.5 : 1 |
| Open-ended straight flush; Three of a kind | 2 | 0.0426 | 22.5 : 1 | 0.0435 | 22.0 : 1 | 0.0842 | 10.9 : 1 |
| High pair | 3 | 0.0638 | 14.7 : 1 | 0.0652 | 14.3 : 1 | 0.1249 | 7.01 : 1 |
| Inside straight; Full house | 4 | 0.0851 | 10.7 : 1 | 0.0870 | 10.5 : 1 | 0.1647 | 5.07 : 1 |
| Three of a kind or two pair | 5 | 0.1064 | 8.40 : 1 | 0.1087 | 8.20 : 1 | 0.2035 | 3.91 : 1 |
| Either pair | 6 | 0.1277 | 6.83 : 1 | 0.1304 | 6.67 : 1 | 0.2414 | 3.14 : 1 |
| Full house or four of a kind; (see note) Inside straight or high pair | 7 | 0.1489 | 5.71 : 1 | 0.1522 | 5.57 : 1 | 0.2784 | 2.59 : 1 |
| Open-ended straight | 8 | 0.1702 | 4.88 : 1 | 0.1739 | 4.75 : 1 | 0.3145 | 2.18 : 1 |
| Flush | 9 | 0.1915 | 4.22 : 1 | 0.1957 | 4.11 : 1 | 0.3497 | 1.86 : 1 |
| Inside straight or pair | 10 | 0.2128 | 3.70 : 1 | 0.2174 | 3.60 : 1 | 0.3839 | 1.60 : 1 |
| Open-ended straight or high pair | 11 | 0.2340 | 3.27 : 1 | 0.2391 | 3.18 : 1 | 0.4172 | 1.40 : 1 |
| Inside straight or flush; Flush or high pair | 12 | 0.2553 | 2.92 : 1 | 0.2609 | 2.83 : 1 | 0.4496 | 1.22 : 1 |
|  | 13 | 0.2766 | 2.62 : 1 | 0.2826 | 2.54 : 1 | 0.4810 | 1.08 : 1 |
| Open-ended straight or pair | 14 | 0.2979 | 2.36 : 1 | 0.3043 | 2.29 : 1 | 0.5116 | 0.955 : 1 |
| Open-ended straight or flush; Flush or pair; Inside straight, flush or high pair | 15 | 0.3191 | 2.13 : 1 | 0.3261 | 2.07 : 1 | 0.5412 | 0.848 : 1 |
|  | 16 | 0.3404 | 1.94 : 1 | 0.3478 | 1.88 : 1 | 0.5698 | 0.755 : 1 |
|  | 17 | 0.3617 | 1.76 : 1 | 0.3696 | 1.71 : 1 | 0.5976 | 0.673 : 1 |
| Inside straight or flush or pair; Open-ended straight, flush or high pair | 18 | 0.3830 | 1.61 : 1 | 0.3913 | 1.56 : 1 | 0.6244 | 0.601 : 1 |
|  | 19 | 0.4043 | 1.47 : 1 | 0.4130 | 1.42 : 1 | 0.6503 | 0.538 : 1 |
|  | 20 | 0.4255 | 1.35 : 1 | 0.4348 | 1.30 : 1 | 0.6753 | 0.481 : 1 |
| Open-ended straight, flush or pair | 21 | 0.4468 | 1.24 : 1 | 0.4565 | 1.19 : 1 | 0.6994 | 0.430 : 1 |

The following shows the approximations and their absolute and relative errors for both methods of approximation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Outs | Make on turn or river | | | | | | | Make on river | | | | | | |
| Actual | (x × 4)% | | | (x × 3 + 9)% | | | Actual | (x × 2)% | | | (x × 2 + (x × 2) ÷ 10)% | | |
| Est. | Error | % Error | Est. | Error | % Error | Est. | Error | % Error | Est. | Error | % Error |
| 1 | 4.2553% | 4% | −0.26% | 6.00% | 4% | −0.26% | 6.00% | 2.1739% | 2% | −0.17% | 8.00% | 2% | −0.17% | 8.00% |
| 2 | 8.4181% | 8% | −0.42% | 4.97% | 8% | −0.42% | 4.97% | 4.3478% | 4% | −0.35% | 8.00% | 4% | −0.35% | 8.00% |
| 3 | 12.4884% | 12% | −0.49% | 3.91% | 12% | −0.49% | 3.91% | 6.5217% | 6% | −0.52% | 8.00% | 7% | +0.48% | 7.33% |
| 4 | 16.4662% | 16% | −0.47% | 2.83% | 16% | −0.47% | 2.83% | 8.6957% | 8% | −0.70% | 8.00% | 9% | +0.30% | 3.50% |
| 5 | 20.3515% | 20% | −0.35% | 1.73% | 20% | −0.35% | 1.73% | 10.8696% | 10% | −0.87% | 8.00% | 11% | +0.13% | 1.20% |
| 6 | 24.1443% | 24% | −0.14% | 0.60% | 24% | −0.14% | 0.60% | 13.0435% | 12% | −1.04% | 8.00% | 13% | −0.04% | 0.33% |
| 7 | 27.8446% | 28% | +0.16% | 0.56% | 28% | +0.16% | 0.56% | 15.2174% | 14% | −1.22% | 8.00% | 15% | −0.22% | 1.43% |
| 8 | 31.4524% | 32% | +0.55% | 1.74% | 32% | +0.55% | 1.74% | 17.3913% | 16% | −1.39% | 8.00% | 18% | +0.61% | 3.50% |
| 9 | 34.9676% | 36% | +1.03% | 2.95% | 36% | +1.03% | 2.95% | 19.5652% | 18% | −1.57% | 8.00% | 20% | +0.43% | 2.22% |
| 10 | 38.3904% | 40% | +1.61% | 4.19% | 39% | +0.61% | 1.59% | 21.7391% | 20% | −1.74% | 8.00% | 22% | +0.26% | 1.20% |
| 11 | 41.7206% | 44% | +2.28% | 5.46% | 42% | +0.28% | 0.67% | 23.9130% | 22% | −1.91% | 8.00% | 24% | +0.09% | 0.36% |
| 12 | 44.9584% | 48% | +3.04% | 6.77% | 45% | +0.04% | 0.09% | 26.0870% | 24% | −2.09% | 8.00% | 26% | −0.09% | 0.33% |
| 13 | 48.1036% | 52% | +3.90% | 8.10% | 48% | −0.10% | 0.22% | 28.2609% | 26% | −2.26% | 8.00% | 29% | +0.74% | 2.62% |
| 14 | 51.1563% | 56% | +4.84% | 9.47% | 51% | −0.16% | 0.31% | 30.4348% | 28% | −2.43% | 8.00% | 31% | +0.57% | 1.86% |
| 15 | 54.1166% | 60% | +5.88% | 10.87% | 54% | −0.12% | 0.22% | 32.6087% | 30% | −2.61% | 8.00% | 33% | +0.39% | 1.20% |
| 16 | 56.9843% | 64% | +7.02% | 12.31% | 57% | +0.02% | 0.03% | 34.7826% | 32% | −2.78% | 8.00% | 35% | +0.22% | 0.62% |
| 17 | 59.7595% | 68% | +8.24% | 13.79% | 60% | +0.24% | 0.40% | 36.9565% | 34% | −2.96% | 8.00% | 37% | +0.04% | 0.12% |
| 18 | 62.4422% | 72% | +9.56% | 15.31% | 63% | +0.56% | 0.89% | 39.1304% | 36% | −3.13% | 8.00% | 40% | +0.87% | 2.22% |
| 19 | 65.0324% | 76% | +10.97% | 16.86% | 66% | +0.97% | 1.49% | 41.3043% | 38% | −3.30% | 8.00% | 42% | +0.70% | 1.68% |
| 20 | 67.5301% | 80% | +12.47% | 18.47% | 69% | +1.47% | 2.18% | 43.4783% | 40% | −3.48% | 8.00% | 44% | +0.52% | 1.20% |
| 21 | 69.9352% | 84% | +14.06% | 20.11% | 72% | +2.06% | 2.95% | 45.6522% | 42% | −3.65% | 8.00% | 46% | +0.35% | 0.76% |
| 22 | 72.2479% | 88% | +15.75% | 21.80% | 75% | +2.75% | 3.81% | 47.8261% | 44% | −3.83% | 8.00% | 48% | +0.17% | 0.36% |
| 23 | 74.4681% | 92% | +17.53% | 23.54% | 78% | +3.53% | 4.74% | 50.0000% | 46% | −4.00% | 8.00% | 51% | +1.00% | 2.00% |

The following table shows the probability and odds of making a runner-runner from a common set of outs and the equivalent normal outs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Likely drawing to | Common outs | Probability | Odds | Equivalent outs |
| Four of a kind (with pair) Inside-only straight flush | 2 | 0.00093 | 1,080 : 1 | 0.02 |
| Three of a kind (with no pair) | 3 | 0.00278 | 359 : 1 | 0.07 |
|  | 4 | 0.00556 | 179 : 1 | 0.13 |
|  | 5 | 0.00925 | 107 : 1 | 0.22 |
| Two pair or three of a kind (with no pair) | 6 | 0.01388 | 71.1 : 1 | 0.33 |
|  | 7 | 0.01943 | 50.5 : 1 | 0.46 |
|  | 8 | 0.02590 | 37.6 : 1 | 0.61 |
|  | 9 | 0.03330 | 29.0 : 1 | 0.78 |
| Flush | 10 | 0.04163 | 23.0 : 1 | 0.98 |

The following table shows the probability and odds of making a runner-runner from a disjoint set of outs for common situations and the equivalent normal outs.

|  |  |  |  |
| --- | --- | --- | --- |
| Drawing to | Probability | Odds | Equivalent outs |
| Outside straight | 0.04440 | 21.5 : 1 | 1.04 |
| Inside+outside straight | 0.02960 | 32.8 : 1 | 0.70 |
| Inside-only straight | 0.01480 | 66.6 : 1 | 0.35 |
| Outside straight flush | 0.00278 | 359 : 1 | 0.07 |
| Inside+outside straight flush | 0.00185 | 540 : 1 | 0.04 |

The following table gives the compound probability and odds of making a runner-runner for common situations and the equivalent normal outs.

|  |  |  |  |
| --- | --- | --- | --- |
| Drawing to | Probability | Odds | Equivalent outs |
| Flush, outside straight or straight flush | 0.08326 | 11.0 : 1 | 1.98 |
| Flush, inside+outside straight or straight flush | 0.06938 | 13.4 : 1 | 1.65 |
| Flush, inside-only straight or straight flush | 0.05550 | 17.0 : 1 | 1.30 |

1. [www.**pkr**.com](http://www.pkr.com) accessed 19th April 2011. [↑](#footnote-ref-1)
2. Journal of algorithms. San Diego [etc.] Academic Press. Volume 1 -41.

   Brassard, G. and Bratley, P. (1988) Algorithmics: theory and practice.

   Fethi A. Rabhi, F.A. and Lapalme, G. (1999) Algorithms: a functional programming approach.

   Paull**,** M.C. (1988) Algorithm design: a recursion transformation framework**.**

   Barclay, K.A. and Gordon, B.J. (1994) C++: problem solving and programming**.**

   Hahn, B.D. (1993)C++: a practical introduction**.**

   Ettinger, J. (1994) Programming in C++**.**

   Cadenhead, R. (2010) Sams Teach Yourself Java in 24 Hours.

   Hortsmann, C. (2008) Big Java: Third Edition.

   Sedgewick, R and Wayne, K. (2008) Introduction to programming in Java: an interdisciplinary approach. [↑](#footnote-ref-2)
3. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. [↑](#footnote-ref-3)
4. [http://en.wikipedia.org/wiki/Poker accessed 19th April 2011](http://en.wikipedia.org/wiki/Poker%20accessed%2019th%20April%202011). Green, J.H. *An Exposure of the Arts and Misteries of Gambling* (G. B. Zieber, Philadelphia, 1843), [↑](#footnote-ref-4)
5. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. (when it was first mentioned in print in a handbook of games) [↑](#footnote-ref-5)
6. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. Taken from the source: “…The wild card (around 1875), lowball and split-pot poker (around 1900), and community card poker games (around 1925).” [↑](#footnote-ref-6)
7. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. [↑](#footnote-ref-7)
8. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. Taken from the source; “In the later 1970s, the first serious poker strategy books appeared due to the WSOP and the popularity it generated. Books of note on the topic include: [Super/System](http://en.wikipedia.org/wiki/Super/System) by [Doyle Brunson](http://en.wikipedia.org/wiki/Doyle_Brunson) ([ISBN 1-58042-081-8](http://en.wikipedia.org/wiki/Special:BookSources/1580420818)) and *Caro's Book of Poker Tells* by [Mike Caro](http://en.wikipedia.org/wiki/Mike_Caro) ([ISBN 0-89746-100-2](http://en.wikipedia.org/wiki/Special:BookSources/0897461002)), followed later by *The Theory of Poker* by [David Sklansky](http://en.wikipedia.org/wiki/David_Sklansky) ([ISBN 1-880685-00-0](http://en.wikipedia.org/wiki/Special:BookSources/1880685000)).” [↑](#footnote-ref-8)
9. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. [↑](#footnote-ref-9)
10. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. [↑](#footnote-ref-10)
11. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011. [↑](#footnote-ref-11)
12. <http://en.wikipedia.org/wiki/Poker> accessed 19th April 2011.

    <http://www.pokerstars.com/wsop/> accessed 19th April 2011. [↑](#footnote-ref-12)
13. See the poker programs in Java in the appendices. The notes are denoted by /\* or //. [↑](#footnote-ref-13)
14. See the poker programs in Java in the appendices. The notes are denoted by /\* or //.

    <http://www.cs.swan.ac.uk/~csoliver/ProgrammingJava201011/CS-M41_Programs/Courseworks/200910/Two/> accessed 19th April 2011. [↑](#footnote-ref-14)
15. <http://people.math.sfu.ca/~alspach/comp18/> accessed 19th April 2011.

    <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011.

    <http://www.cs.swan.ac.uk/~csoliver/ProgrammingJava201011/CS-M41_Programs/Courseworks/200910/Two/> accessed 19th April 2011. [↑](#footnote-ref-15)
16. <http://en.wikipedia.org/wiki/Poker_calculator> accessed 19th April 2011. [↑](#footnote-ref-16)
17. See the poker programs in Java in the appendices.

    <http://www.cs.swan.ac.uk/~csoliver/ProgrammingJava201011/CS-M41_Programs/Courseworks/200910/Two/> accessed 19th April 2011.

    McDonald, J. (1996) Strategy in poker, business & war. [↑](#footnote-ref-17)
18. <http://en.wikipedia.org/wiki/Poker_calculator> accessed 19th April 2011. [↑](#footnote-ref-18)
19. <http://en.wikipedia.org/wiki/Normalization_(statistics)> accessed 19th April 2011. [↑](#footnote-ref-19)
20. <http://en.wikipedia.org/wiki/Poker_calculator> accessed 19th April 2011.

    Botermans, J. (2008) [translated from the Spanish by Fankbonner, E.L.].The book of games : strategy, tactics & history. [↑](#footnote-ref-20)
21. <http://en.wikipedia.org/wiki/Poker_calculator> accessed 19th April 2011.

    Petriv**,** M. (1996). *Hold'em Odds Book*. [↑](#footnote-ref-21)
22. <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011. [↑](#footnote-ref-22)
23. <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-23)
24. McDonald, J. (1996) Strategy in poker, business & war.

    [http://en.wikipedia.org/wiki/Poker\_probability\_(Texas\_Hold\_’em)](http://en.wikipedia.org/wiki/Poker_probability_(Texas_Hold_'em)) accessed 19th April 2011. [↑](#footnote-ref-24)
25. Petriv, M. (1996). *Hold'em Odds Book*.

    [http://en.wikipedia.org/wiki/Poker\_probability\_(Texas\_Hold\_’em)](http://en.wikipedia.org/wiki/Poker_probability_(Texas_Hold_'em)) accessed 19th April 2011. [↑](#footnote-ref-25)
26. [http://en.wikipedia.org/wiki/Poker\_probability\_(Texas\_Hold\_’em)](http://en.wikipedia.org/wiki/Poker_probability_(Texas_Hold_'em)) accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-26)
27. [http://en.wikipedia.org/wiki/Poker\_probability\_(Texas\_Hold\_’em)](http://en.wikipedia.org/wiki/Poker_probability_(Texas_Hold_'em)) accessed 19th April 2011.

    Petriv**,** M. (1996). *Hold'em Odds Book*. [↑](#footnote-ref-27)
28. <http://en.wikipedia.org/wiki/Decision_tree> accessed 19th April 2011. [↑](#footnote-ref-28)
29. [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-29)
30. [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-30)
31. [http://en.wikipedia.org/wiki/Texas\_Hold\_’Em](http://en.wikipedia.org/wiki/Texas_Hold_'Em) accessed 19th April 2011. [↑](#footnote-ref-31)
32. <http://www.cigital.com/papers/download/developer_gambling.php> accessed 19th April 2011. [↑](#footnote-ref-32)
33. <http://en.wikipedia.org/wiki/Doxygen> accessed 19th April 2011.

    <http://www.stack.nl/~dimitri/doxygen/index.html> accessed 19th April 2011. [↑](#footnote-ref-33)
34. <http://www.stack.nl/~dimitri/doxygen/index.html> accessed 19th April 2011.

    [http://en.wikipedia.org/wiki/Doxygen](http://en.wikipedia.org/wiki/doxygen) accessed 19th April 2011. [↑](#footnote-ref-34)
35. [http://en.wikipedia.org/wiki/Doxygen](http://en.wikipedia.org/wiki/doxygen) accessed 19th April 2011. [↑](#footnote-ref-35)
36. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19thApril 2011. [↑](#footnote-ref-36)
37. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19thApril 2011. [↑](#footnote-ref-37)
38. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19thApril 2011. [↑](#footnote-ref-38)
39. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-39)
40. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19th April 2011. [↑](#footnote-ref-40)
41. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19th April 2011. [↑](#footnote-ref-41)
42. <http://en.wikipedia.org/wiki/Git_(software)> accessed 19th April 2011.

    <https://github.com/> accessed 19th April 2011. [↑](#footnote-ref-42)
43. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-43)
44. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-44)
45. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-45)
46. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-46)
47. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-47)
48. <http://en.wikipedia.org/wiki/Fundamental_theorem_of_poker> accessed 19th April 2011. [↑](#footnote-ref-48)
49. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-49)
50. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-50)
51. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-51)
52. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-52)
53. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-53)
54. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. Quote taken from Source. [↑](#footnote-ref-54)
55. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-55)
56. <http://en.wikipedia.org/wiki/Morton's_theorem> accessed 19th April 2011. [↑](#footnote-ref-56)
57. <http://en.wikipedia.org/wiki/M-ratio> accessed 19th April 2011. [↑](#footnote-ref-57)
58. <http://en.wikipedia.org/wiki/M-ratio> accessed 19th April 2011. [↑](#footnote-ref-58)
59. <http://en.wikipedia.org/wiki/Q-ratio> accessed 19th April 2011. [↑](#footnote-ref-59)
60. <http://mathworld.wolfram.com/Poker.html> accessed 19th April 2011.

    <http://people.math.sfu.ca/~alspach/comp18/> accessed 19th April 2011. [↑](#footnote-ref-60)
61. <http://mathworld.wolfram.com/Poker.html> accessed 19th April 2011. [↑](#footnote-ref-61)
62. Packel, E. (2006) the Mathematics of Games and Gambling, 2nd Ed. [↑](#footnote-ref-62)
63. <http://en.wikipedia.org/wiki/Fold_equity> accessed 19th April 2011. [↑](#footnote-ref-63)
64. <http://en.wikipedia.org/wiki/Fold_equity> accessed 19th April 2011. [↑](#footnote-ref-64)
65. <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011. Quote taken from source. [↑](#footnote-ref-65)
66. <http://en.wikipedia.org/wiki/Poker_probability> accessed 19th April 2011. [↑](#footnote-ref-66)
67. <http://www.dmoz.org/Games/Gambling/Poker/Software_and_Tools/> accessed 19th April 2011.

    <http://www.pagat.com/poker/software.html> accessed 19th April 2011.

    h[ttp://www.pokerlistings.com/poker-software-other](http://www.pokerlistings.com/poker-software-other) accessed 19th April 2011.

    <http://www.pokerlistings.com/poker-software-other> accessed 19th April 2011. [↑](#footnote-ref-67)
68. <http://www.pagat.com/poker/software.html> accessed 19th April 2011. [↑](#footnote-ref-68)
69. <http://www.dmoz.org/Games/Gambling/Poker/Software_and_Tools/> accessed 19th April 2011. [↑](#footnote-ref-69)
70. <http://poker.cs.ualberta.ca/> accessed 19th April 2011. [↑](#footnote-ref-70)
71. <http://poker.cs.ualberta.ca/publications/billings.msc.pdf> accessed 19th April 2011. [↑](#footnote-ref-71)
72. <http://poker.cs.ualberta.ca/publications/IJCAI03.pdf> accessed 19th April 2011.

    <http://poker.cs.ualberta.ca/publications/schauenberg.msc.pdf> accessed 19th April 2011.

    <http://poker.cs.ualberta.ca/publications/NIPS07-cfr.pdf> accessed 19th April 2011.

    <http://poker.cs.ualberta.ca/publications/abourisk.msc.pdf> accessed 19th April 2011. To name but a few available on the site. [↑](#footnote-ref-72)
73. <http://poker.cs.ualberta.ca/acpc_code/project_acpc_server.tar.bz2> accessed 19th April 2011. [↑](#footnote-ref-73)
74. <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup#xpokereval> accessed 19th April 2011. [↑](#footnote-ref-74)
75. <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> accessed 19th April 2011. [↑](#footnote-ref-75)
76. <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> accessed 19th April 2011. [↑](#footnote-ref-76)
77. <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> accessed 19th April 2011.

    <http://archives1.twoplustwo.com/showflat.php?Cat=0&Number=8513906&page=0&fpart=1&vc=1> accessed 19th April 2011. [↑](#footnote-ref-77)
78. <http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup> accessed 19th April 2011. [↑](#footnote-ref-78)
79. <http://www.caspercomsci.com/> accessed 19th April 2011. [↑](#footnote-ref-79)