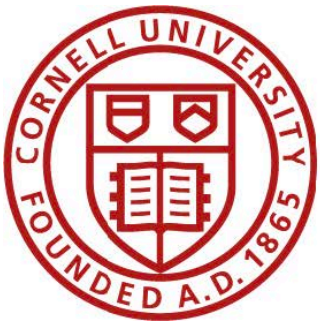

Cornell University

Blackjack Tactics

By

Won Yong Ha, Ryan Lo, Prajnavaro Selamet, and Jiewen Wang

Advised By

Professor Wenqi Yi

Won Yong Ha

Ryan Lo

Prajnavaro Selamet

Jiewen Wang

Professor Wenqi Yi

0 Executive Summary

The purpose of our project is to investigate the effect of using a strategy on the game of Blackjack. We particularly investigate the class of strategies called "card counting". Specially we explored the following card counting strategies: Hi-Lo, Hi-Opt1, Hi-Opt2 and KO. We also added a strategy we call the basic or simple strategy, where the player follows the rules the dealer uses when deciding a course of action. This is to represent the case where the player follows no complicated rules and acts as our base case for strategy comparison. Our objective was to determine which among these strategies is optimal. A Monte Carlo simulation model was created and compared. The comparison took on a "gamblers ruin (Seongjoo Song, 2013)" approach. We modelled 3 distinct types of players: Risk Seeking, Risk Averse and Fun Seeking. We define their conditions as follows:

A. Risk Seeking: Define lose as gambler ends with 0 money and define win as doubling of initial bet (ends with $2000 dollars)
B. Risk Averse: Define lose as gambler ends with half of initial bet size (end with $500) and define win as earning half of initial bet size (end with $1,500)
C. Fun Seeking: Play until gambler goes bankrupt.

We collected the following statistics and computed their means, standard deviation, coefficient of variation and confidence levels:

1. Rounds Until win or Lose
2. Percentage Games Won
3. Amount Won

Between Risk Seeking Player and the Risk Averse player we found surprising results. The Risk Averse player won more times, won more and was riskier (in terms of coefficient of variation) across all strategies. The explanation for these results was that the simulation was showing the "Regression of the Mean" phenomenon often found in stable stochastic processes. Since the house always has an advantage over the player, it more optimal to play as short and as a risky as possible. This limits the chances a player "regress towards a mean". This phenomenon is also observed in other stochastic process such as squash and basketball where if your worse than your opponent, it is better to play a few rounds as possible.

We also recommend that a player should use the Hi-Lo strategy. It showed comparable results to the best strategy KO however is much easier to learn and implement.

And finally, if you just want to have fun and play as long as you can, you should learn the Hi-Opt1 strategy.

1 Introduction

Throughout history there has been a long connection between gambling and the study of probability. Among the earliest of this was treatise of Gerolamo Cardano," The Book on Games of Chance" where its results led to many of the rules we follow in probability (Mlodinow, 2008). One of the earliest formal analysis of Blackjack was done by Roger Baldwin whose results was published Journal of the American Statistical Association in 1956. (Baldwin, 1956). Building on Baldwin's work, Edward O. Thorp published "A Favorable Strategy For Twenty-One" in 1961 which formed the basis for many card counting techniques used today (Thorp, 1961).

The purpose of our project is a result of the natural extension of the past analysis done on Blackjack. Specifically, we wish to answer the question "what is the optimal playing strategy for the game of Blackjack?". And if such a strategy exists how much better is it? Can we make money or beat the house? Such questions are of interest to amateur and professional gamblers who may want to know and evaluate various tools and methodology that may give them the edge to "beat the odds". Casinos may also find the information useful to investigate their potential loses if a player (the Casino's opponent) were to use a strategy. Many of the strategies proposed by these authors were made during a time when Casinos were not aware of efforts of prominent scientist to use their knowledge to challenge the house advantage. Now, modern casino rules have changed this, this has a large effect on the previous results and advantages stated by these papers.

This paper will be investigating the question of interest through several sections. Section 1 is the introduction. Section 2 will discuss the history of black and its rules and terminology. This will give the reader a context of Blackjack and how its rules and chances affect how a player can win. Section 3 summarizes the four most used counting card strategies which will be used as test cases. Section 4 will present the methodology and tools used to conduct the analysis. Section 5 presents the results and finally Section 6 presents our conclusion.

2 Blackjack

History

There is no exact clear indication of where the history of Blackjack game originated. But one of the most notable timeline of Blackjack dates back to the Romans, where they use wooden cards with different numbering system than the one we use now to achieve the same goal which is to get a specific determine value (Wintle, 2010) . After much deeper research, historian led to a plausible conclusion that Blackjack

game actually derived and influenced from the European countries. One of these European influence can be found from the Italian, specifically from the game of “thirty one” where it is said the concept of game play is the same. Other historians argue that it was derived from a French game “Vingt En Un”. Another historian source also pointed out that it was a combination of multiple casinos game in Europe such as Roulette, hazard, trente-et-quarante, faro, baccarat and etc (Snyder, 2006) . However, the game “Thirty One” and “Vingt En Un” provided significant contribution to the base game of Blackjack this these game will be explored more in details.

The game by the fellow Italian, “thirty one”, was a essentially the same as Blackjack however the objective is to sum the cards value until it reaches thirty one instead of 21 as in Blackjack. Another notable difference is that the game required to draw a minimum of 3 cards from the deck instead of two in found in Blackjack (Ofton, n.d.) . This makes sense due to the cumulative value of the card with only two card cannot and will not exceed the 31-boundary limit in order to win the game. The French game “Vingt En Un”, which direct translation in English translate to ”twenty one” was first appeared in in the early to late 1700s. Later on in the early 20th century the name evolved to “Blackjack”. The name Blackjack comes from this game that coincidentally involved a combination of Ace and Jack of spades (Snyder, 2006).

Back in the 90s the game was still called “21” and remained unpopular in the United States. Not until the State of Nevada first chose to make gambling legal that the game “21” took off. As the game popularity grew in casinos in Nevada so as the name plate, so much so that it affected how people are calling them, hence the name “Blackjack” (Wintle, 2010).

Rules

The gameplay rules of blackjack is simple, the objective goal is to get the value of 21 on your card in hand or as close to 21 as possible with some possible arrangement hierarchy of different cards. Originally the game Blackjack is played with one single deck that consisting of 52 cards. Due to modern day business influence on making money by the dealer, also known as casinos, the game played typically use four to eight decks combined which is then reshuffled. This reduces the chance of players winning and counting cards so they say. We will dig deeper in our analysis on this part. If the deck have depleted two-thirds of the way, the deck is need to be reshuffled.

A normal Blackjack table consist of a single dealer which represent the casino’s house and up to 6 players with a minimum of 2 players to start play. This allow the flexibility for the dealer to effectively make themselves more money or potentially lose more money. At the very first state, player needs to place a bet which is kept unknown to other players. The player will start off with the dealer drawing two cards, one

card is kept unknown by facing down and the other is shown by facing it up. Some of these rules might have been modified by other casinos to reduce the chance of cheating by only allowing players to have all cards drawn face up.

Every card has a value, card with rank 2 to 10 has their own numeric point value of their own, face cards (jack, queens, and kings) are counted as 10 points, and ace card can be count as 1 point of 11 points depending on the situation to which is more favorable to the hands of player or dealer. The hand containing ace and valued as 11 that do not go over 21 is called soft. Other value are referred as hard. The objective is to get the total points of 21 without exceeding, if exceed we called this bust. When two cards totaling points of 21 with one ace and any face card this is called a Blackjack. This is the highest rated points in the game beating other possible combination of card totaling the same amount.

The game continues with the player drawing cards and making the decision. The next step is to check whether the dealer's hand contains blackjack combination. If the dealer does have Blackjack, the gameplay automatically ends and dealer wins, unless the player have blackjack combination as well. In this case the game is then forfeited and no money is exchanged. In general if the player has Blackjack, the player immediately wins 1.5 times the original bet and the game ends. If no Blackjack if present, the player can continue playing with a hit. A hit is requesting an additional card to be added onto his hand hence the added points on the player's card possession. If a player hit and exceed the 21 points limit, the player is bust and the player is immediately loses the bet to the dealer. Another option is that player decide to stand, meaning ending his turn in hope the next player or the dealer does not have card points higher than the player. To make matter more interesting, if a player have exactly two card, the player can wish to have a double down, where his bet is double and opting to draw additional one more card from the deck. Another option that the player have is to split hand. This entails separating the player's two cards into two hands. The player then must still have the same bet however, this increases the player's chance of totaling of 21.

After all the player have their turn, it is now time for the dealer to make a move. Typically the dealer begins by flipping over and revealing his card. The dealer perform hits it until his deck card reach a value of sixteen. The dealer usually stand on all total card points of 17 or greater. In the case of ace, which can take values 1 or 11, 11 resulting in either a 17 or a ,7 a game can choose to follow "Soft 17" rule or "Hard 17" rule. In a "Soft 17" the hand value is considered a 17 and the dealer stays. In a "Hard 17" the hand value is considered a 7 and the dealer must hit again following the same rules as before. We assume a "Soft 17 rule" as this is the most common rule used in casinos.

This technique is very common to prevent the dealer from going bust and losing money to the house, although there is still a slight probability that the dealer go bust.

Now that we have all the ground rules layout, the final step to do is for the settlement. Dealer has to receive any player gone bust and also pay any player 1.5 times of its initial bet if the player have Blackjack combination. If for some reason the dealer bust, the dealer need to pay any player left in the game. Also if the dealer did not bust, dealer have to pay to any player who has a higher hand total than his own card points. Finally, a push happens if no money is exchanged if a player's hand points total is the same as the dealer's. After all bets are settled and accounted for, the game ends and the next round begins (Dunross, 2016).

3 Blackjack Strategies

Hi-Lo Blackjack Card Counting System

All blackjack card counting strategies are on the basis that some cards are more advantageous to the player (and thus disadvantageous to the house) than other cards. By keeping a count of the cards being dealt during a hand, a player can adjust his bets accordingly to increase the expected payoff for each hand. By extending this logic to the series of hands a player faces, he can increase the expectation for the duration he is at the table.

One of the most basic and popular card counting methods is the Hi-Lo strategy which was first introduced in 1963 by Harvey Dubner

Following is a brief explanation of how to use the Hi-Lo strategy.

Step 1: Assign a point value to each card.

Low Cards: Two, Three, Four, Five and Six cards are assigned a value of plus (+) 1. Low cards are helpful to the dealer who must take a hit if the total value of his hand is under 17. They are less likely to bust if the deck has more low cards in it.

High Cards: Ten, Jack, Queen, King, and Ace are assigned a value of minus (-) 1. When the deck contains a large number of high cards, the dealer is more likely to bust if he must make a hit. It also gives the players a higher chance of getting dealt 17 or higher, including blackjack.

Neutral Cards: Seven, Eight and Nine are valued at zero (0). It allows the numbers of high cards and low cards to be equal so the final count of an entire shoe equals zero, thus keeps the balance of the system.

Step 2: Start with a "Running Count" of zero at the start of the shoe. The running count is the process of adding and subtracting the values of each card as they are dealt. It is maintained between rounds (but is

reset if the shoe is reshuffled). A positive count means that a higher number of small cards have been played, and that the deck possesses higher cards than lower cards.

Step 3: Divide the running count by the number of decks remaining, to get what is known as the "True Count". The running count is used 99% of the time, but is not the final value to gauge a player's wager. The player needs to convert the running count into the true count. The true count thus gives the relative proportion of high value cards within the deck.

Step 4: The greater the true count, the more you should bet. This is where card counting becomes more art than science. The true count determines the size of your bet. The way you get your advantage with card counting is by betting more when the count is positive and betting the minimum when it is negative. You determine the size of your bets by the true count. Each bet is a unit and it is determined by the size of your base bet. The amount of units you bet from minimum to maximum is known as the spread. When you play a double deck game you can get the advantage by spreading your bets from one unit to six. In a six or eight deck game you will have to spread from one to 12 units. The chart below shows you bet based on the true count.

Spread Based on Count		
True Count	2 Decks	6 Decks
0 or less	1 unit	1 unit
+1	2 units	2 units
+2	3 units	4 units
+3	4 units	8 units
+4	5 units	10 units
+5	6 units	12 units

Step 5: The true count is not only used in the wager. In some cases, the true count might affect how an individual should play his hand. The table shown below is called Illustrious 18. It shows a series of scenarios during which a player should pay against Basic Strategy's recommendation.

Illustrious 18			
Play	True Count	Above	Below

Insurance	+3	Take	No
16 Vs. 10	+0	Split/Stay	Hit
15 Vs. 10	+4	Stay	Hit
10,10 Vs. 5	+5	Split	Stay
10,10 Vs. 6	+4	Split	Stay
10 Vs. 10	+4	Double	Hit
12 Vs. 3	+2	Split/Stay	Hit
12 Vs. 2	+3	Split/Stay	Hit
11 Vs. A	+1	Double	Hit
9 Vs. 2	+1	Double	Hit
10 Vs. A	+4	Double	Hit
9 Vs. 7	+3	Double	Hit
16 Vs. 9	+5	Split/Stay	Hit
13 Vs. 2	-1	Stay	Hit
12 Vs. 4	+0	Split/Stay	Hit
12 Vs. 5	-2	Split/Stay	Hit
12 Vs. 6	-1	Split/Stay	Hit
13 Vs. 3	-2	Stay	Hit

The player should stand/double/split if the True Count equals or exceeds the Index Number, otherwise hit. The player should take insurance if the True Count is +3 or greater (Vaidyanathan, 2014).

Hi Opt I Blackjack Card Counting System

Introduction of High Opt I Card Counting System

Hi-Opt I is relatively simple 1st level balanced strategy developed by Lance Humble and Carl Cooper. This strategy was built on the Einstein count developed by Charles Einstein in the late 60's. The Hi-Opt I (Hi Opt 1) blackjack card counting system, a version of Hi-Opt (highly optimum) counting systems, is derived from Hi-Lo card counting system. This counting strategy is similar to Hi-Lo but it is optimized by adding a few more rules and changing some card value. The optimization make the count more accurate. The Hi-Opt I system is also often referred to as the Einstein Count.

If you are wanting to become a serious blackjack player, using the Hi-Opt I card counting system is definitely an excellent choice. Compared to previous card counting systems, this system is more mathematically advanced and can provide a slightly larger edge for the player. So if you are highly skilled

and lucky, you can expect the Hi-Opt I system to increase your probability to make a long-term profit from blackjack. you could potentially make serious cash. Some blackjack players claims that this small extra percentage of player advantage isn't enough to recognize using a more complicated system, but some players disagree and feel that the Hi-Opt I strategy is a far superior counting method.

Hi-Opt I

The Hi-Opt I system is based on the general card counting principles. The system is used to develop a running count which helps the player to determine the appropriate bet size. It is not complicate and only simple math is used. As we can see from the table below, this system is also based on adding and subtracting the number 1. There are separate groups of card values. All 2's in the deck carry a value of 0 in this system and don't change the count when they are played. The 16 other small cards (3-6) are counted as plus 1 as they are played. The 12 middle cards (7-9) are considered neutral cards and do not change the count as they are played. The 16 big cards (10-King), excluding Aces, are counted as minus 1 as they are played. There are a same number of +1 and -1 cards. This is beneficial because it simplifies the process of counting cards in live play. When using the Hi-Opt I system these cards offset or cancel one another out. This can make it easier to maintain the running count of the Hi-Opt I. The Hi-Opt I is a balanced count. This means that the count begins at 0 and should also end at 0 when all of the cards have been dealt from the shoe. The system is used to develop a running count which helps the player to determine the appropriate bet size. For example, When the running count is high (+2, +3, +4), bets are increased. When the running count drops into a negative zone (-1, -2, -3), the bet size is reduced.

2	3	4	5	6	7	8	9	10	J	Q	K	A
0	+1	+1	+1	+1	0	0	-0	-1	-1	-1	-1	0

Keep Track of Aces

Besides the value of 2’s, another difference between Hi-Lo and Hi-Opt I is the value assigned to the aces. There are separate rule variations for tracking Aces in the Hi-Opt I counting system. This system doesn't keep track of Aces in the card count but there are still changes to basic strategy that need to be made depending on how many aces have been played (hi-opt-i, n.d.).

Hi Optimization 2

The Hi Optimization 2 (Hi-Opt II) card counting system is an extension to the Hi Optimization 1 strategy. Typically, this Hi-Opt II strategy is used by more advance Blackjack players to increase their chances of winning by counting cards. On a side note, players are not recommended to use this method if they have not master the Hi-Opt I method system.

The basic of this Hi-Opt II strategy is basically the same as the Hi-Opt I, the major difference is that certain cards have a value of 2 instead of only 0 and 1s. The Table 1 below will show what I meant by cards with value 2 in more detail.

Card Face	Count Index
2	+1
3	+1
4	+2
5	+2
6	+1
7	+1
8	0
9	0
10	-2
J	-2
Q	-2
K	-2
A	0

Table 1: Hi-Optimization 2 Card Count Index Value

As the table above shown, the face card of 2, 3, 6, 7 have an index value of +1, the face card of 4 and 5 have index value of +2, card with face value of 8, 9, Ace have an index value of 0 and lastly card with face value of 10, J, Q, K have a count index value of -2.

The objective strategy for Hi-Opt II is to have overall count index value balance or in other words the count index value must sum up to 0 ideally. From the current research, all of my sources pointed to this method is proven to be extremely complicated with Blackjack players, only advance players that have master Hi-

Opt can be reassure that Hi-Opt II method will be advantageous to those individuals (Snyder, 1984). The main persistent issue with this method is to keep track of which card value are you running on and which of the index value that you are counting in the player's mind. Not to confused these two with one and the other, one value is for the actual count to win 21 Blackjack and the other is for making decision on the count index.

To see whether this method make sense let's begin a simple gameplay. Pretend that you were given 2 cards from the deck, the first card in hand has a face value of 3 and the second card has a face value of 7. At this point your count index is +2, since the objective is to balance the count index to a 0, ideally we would draw another card in favor of that -2 count index. Let say that you decided to draw another card from the deck and it has a face value of Q. Q has an index count card of -2, so therefore we would have a sum count index value of 0, which is perfectly balance. The actual value of these 3 drawn cards are 2 + 7 + 10 = 19. At this point, naturally, we would choose to stand, since the actual card value that it holds is close to our objective which is 21 Blackjack and also this is the best possible outcome at the current stage. If we further push the envelope and in hope to get the 21 value, we would most likely to go bust. Therefore, in theory this Hi-Opt II method work as it should however in reality we cannot have guaranteed that we could get the card index value that matches with our preference since the card drawn by the dealer is always unpredictable / random (Snyder A. , 1984).

KO

By now you must have realized that Blackjack is a game based on probability. Frequent Blackjack players sooner or later must have come up with a strategy on their own in order to increase their chances of winning. These different variations of strategy sometimes are not easy to proof using mathematical notation. One of the most basic and widely used blackjack strategy is to count cards. This strategy is also known as the "Knocked Out" strategy or commonly known as KO Blackjack.

A typical strategy to play Blackjack is by counting cards, know when to add or hold the current state of the cards in hand is the golden rule to win the game theoretically. To explain this further we need to use the balance card counting system where if you add all of the card count index we would end up to 0. Essentially KO Blackjack method is to assigned a count index number to each specific cards, as shown in the Table 2 below.

Card Face	Count Index

2	+1
3	+1
4	+1
5	+1
6	+1
7	+1
8	0
9	0
10	-1
J	-1
Q	-1
K	-1
A	-1

TABLE 2: KO Blackjack Card Index Value

The objective of this KO method is to have the count index value of 0, where in this case the card in hand index value is balance or equivalent to 0. KO Blackjack method is built upon the Hi-Lo count strategy, so assigning an index value is no unfamiliar territory. However, Blackjack players often finding arduous to convert the actual face value to the counting index value. This conversion most of the time screws up the player counting strategy, which make sense. This situation is similar in effect as a person trying to multitask at the same moment continuously till the game ends. This method is explained in greater depth in Knock-Out Blackjack by Olaf Vancura and Ken Fuchs. (Vancura, 2005)

To test this method let's take an example of a gameplay, where you have 2 cards in hand. The first card has a face value of 2 and the second card has a face value of 10, this card could be 10, J, Q, K. Your count index in this case is 0 or balance. If you decide to add another card, which is completely acceptable, it should be a card that has a count index value of 0 to stay balance. Let say that you decided to take the third card and fortunately you got a face value of card of 8. Your total face value card is 20 which is still let than 21. This time you did not go bust however, if the first card you have is a 4, 5, 6 or 7, you would go bust. Therefore, this technique is not 100 percent proven to beat the system, moreover this technique is only a guide to caters and increases your chance of scoring that 21 Blackjack.

The gameplay test will differ slightly if we were only playing solely with 2 cards. If the first card, we got has a face value of 7 and the next card we received is an Ace. This combination is also a balance

however, counting the actual value of the cards we should realized that it has an actual value of 18, which is close to our 21 target. Should we decide to continue drawing an additional card it would likely to increase the chance of going bust by a huge margin, therefore the best case in this balance situation is to remain at a stand (Snyder A. , 1984).

4 Methodology

To investigate the question "what is the optimal playing strategy for the game of Blackjack?", the Monte Carlo Method or a Monte Carlo simulation. The Monte Carlo Method was chosen because many of the characteristics of the game Blackjack make it very tractable to implement as a simulation. The rules are simple and fixed. There are two entities: the player and dealer. This makes them easy to code as the actions available are limited. It also makes it easy to run repeated simulation with easy with modern computers with each simulation acting as game. And because it is a gambling game, a natural statistic of interest are payoffs and risks. The methodology was implemented in Java Programming language.

I. Functions

   To simulate a basic game, it was necessary to create several functions that exhibit the rules and action that a Blackjack game can perform. These include: 1. shuffling of the deck 2. drawing from a deck 3. hit 4. Stay 5. make a bet and 6. hit

II. Classes

   The game also has several classes that represent the common entities in a game. These include:1. The deck with their suits 2.the player and finally 3.the dealer.

III. Monte Carlo Method Process and Scenario Assumptions

   a. The simulation has certain rules that most casino's follow and adjust according to the number of simulations.
      i. Play with starting point
      ii. No double or split
      iii. Fixed Betting
         1. Expected winning: 100
         2. Expected losing: 10
      iv. Minimum card counting to consider as winning: 10
   b. The simulation developed with Java programming language and consist of 9 files.
      i. Makefile
         1. Compiling and running the java files with specific commands
      ii. Card,java

1. Represent each card information and count

iii. Deck.java
1. Represent all cards on the one deck
2. Available to draw a card

iv. Dealer.java
1. Represent a dealer
2. Play with dealer rule

v. Player.java
1. Represent a player
2. Play with player rule decided by counting method

vi. Counting.java
1. Represent a counting procedure
2. Available to change the method

vii. OneDeckGame.java
1. Represent one pack of deck game

viii. OneGame.java
1. Represent one game

ix. Game.java (Main)
1. Represent the game controller

IV. Output of Interests

To answer the question of "what is the optimal playing strategy for the game of Blackjack" we run a simulation or repeated experiment for each of the 4 strategies plus that of the simple strategy which essentially mimics the moves that dealer must follows.

As each strategy is used we are interested in knowing: How often did the gambler lose and how often did he win? And how many hands did we play to win or lose. Essentially, we took on a "gamblers ruin" approach under 3 scenarios

a. Scenario 1: Define lose as gambler ends with 0 money and define win as doubling of initial bet (ends with $2000 dollars)
b. Scenario 2: Define lose as gambler ends with half of initial bet size (end with $500) and define win as earning half of initial bet size (end with $1,500)
c. Scenario 3: Play until gambler goes bankrupt.

These scenarios are meant to mimic the profile of a typical gambler. Scenario 1, represents our risk seeking gambler, scenario 2 represents our risk averse gambler and scenario 3 represents our gambler who just wants to play the game as long as possible (just wants to have fun).

As each sample is generated we compute the following points estimates:

1. Mean
2. Standard Deviation
3. Coefficient of Variation
4. 95% confidence intervals

5 Results

Scenario 1 (Risk Seeking)

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Rounds Until win or Lose				
Average				
99.48	95.53	94.53	105.00	97.62
Standard Deviation				
77.50	76.47	75.30	86.79	82.83
Coefficient of Variation				
1.28	1.25	1.26	1.21	1.18
Confidence Interval Lower				
(52.43)	(54.35)	(53.05)	(65.12)	(64.73)
Confidence Interval Upper				
251.39	245.40	242.12	275.12	259.96

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Percentage Games Won				
Average				
19.9%	18.7%	18.4%	21.7%	20.5%
Standard Deviation				
39.9%	39.0%	38.8%	41.2%	40.4%
Coefficient of Variation				
0.50	0.48	0.47	0.53	0.51
Confidence Interval Lower				
-58.39%	-57.76%	-57.58%	-59.13%	-58.67%
Confidence Interval Upper				
98.19%	95.16%	94.38%	102.53%	99.67%

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Average Amount Won				
Average				
413.15	387.93	382.25	449.50	425.08
Standard Deviation				
829.40	809.37	805.49	854.37	837.61
Coefficient of Variation				
0.50	0.48	0.47	0.53	0.51
Confidence Interval Lower				
(1,212.48)	(1,198.44)	(1,196.51)	(1,225.07)	(1,216.65)
Confidence Interval Upper				
2,038.78	1,974.29	1,961.01	2,124.07	2,066.80

Scenario 2(Risk Averse)

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Rounds Until win or Lose				
Average				
33.04	32.07	31.77	33.30	34.10
Standard Deviation				
26.90	25.09	24.61	25.57	25.90
Coefficient of Variation				
1.23	1.28	1.29	1.30	1.32
Confidence Interval Lower				
(19.69)	(17.11)	(16.46)	(16.83)	(16.66)
Confidence Interval Upper				
85.78	81.25	80.00	83.42	84.85

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Percentage Games Won				
Average				
32.9%	34.9%	32.0%	31.4%	36.0%
Standard Deviation				
47.0%	47.7%	46.7%	46.4%	48.0%
Coefficient of Variation				
0.70	0.73	0.69	0.68	0.75
Confidence Interval Lower				
-59.24%	-58.57%	-59.48%	-59.61%	-58.13%
Confidence Interval Upper				
125.04%	128.37%	123.48%	122.41%	130.13%

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Average Amount Won				
Average				
800.05	823.10	790.45	784.55	836.65
Standard Deviation				
544.44	551.15	540.91	536.34	554.98
Coefficient of Variation				
1.47	1.49	1.46	1.46	1.51
Confidence Interval Lower				
(267.06)	(257.15)	(269.74)	(266.68)	(251.11)
Confidence Interval Upper				
1,867.16	1,903.35	1,850.64	1,835.78	1,924.41

Scenario 3 (Fun Seeking)

Simple	Hi-Lo	Hi-Opt1	Hi-Opt2	KO
Rounds Until win or Lose				
Average				
172.44	783.08	891.61	767.49	345.73
Standard Deviation				
211.38	1,031.53	1,060.77	963.47	482.41
Coefficient of Variation				
0.82	0.76	0.84	0.80	0.72
Confidence Interval Lower				
(241.87)	(1,238.71)	(1,187.49)	(1,120.90)	(599.80)
Confidence Interval Upper				
586.75	2,804.87	2,970.71	2,655.89	1,291.26

6 Conclusion

After running our simulations, we found some surprising results. We did a side by side comparison of the two type of players [1], the risk seeking and risk averse and produced the following table:

Rounds Until win or Lose		
Gambler Type:	**Risk Seeking**	**Risk Averse**
Average	98.43	32.86
Standard Deviation	79.78	25.61
Coefficient of Variation	1.24	1.28

Average Amount Won		
Gambler Type:	**Risk Seeking**	**Risk Averse**
Average	411.58	806.96
Standard Deviation	827.25	545.57
Coefficient of Variation	0.50	1.48

Percentage Games Won		
Gambler Type:	**Risk Seeking**	**Risk Averse**
Average	19.8%	33.4%
Standard Deviation	39.9%	47.2%
Coefficient of Variation	0.50	0.71

In terms of percentage of games won we were expecting that the average would not be far from each other since the tails of what is considered a loss and win were equidistant from each other however the Risk Averse player significantly won more won more games (19.8% vs 33.4%). Another surprising result was that the Risk Averse Player also on average won more money despite the lower satisfaction with winning and desire to lose as little as possible. The Risk Averse Player won almost twice that of our risk loving (807 vs 412). Another surprising result, was that across all strategies, the Risk Averse, despite being risk averse was riskier than the Risk Seeking gambler. The coefficients of variations were all higher in terms of performance (wins and amounts won). This sort of counterintuitive results is similar to observed results by other researchers who analyze sports. One such example is the study of Squash by Brodie (Mark Broadie, 1993) . Essentially, our result is caused by a "regression to the mean" behavior often found in stable stochastic processes. In squash or in shooting basketball, the worse player is always better off playing a shorter game. The worse player takes advantage of his or her variability and prematurely cuts offs revealing

[1] The fun-loving player always ends bankrupt so comparison would be meaningless

his or her true mean. We see the same phenomenon among our two Blackjack players. Remember you can never beat the house. This has been studied by other researchers (Vaidyanathan, 2014). And Casinos have always change the rule to counter such tactics (Barker, 2015). The Risk Seeking Player with the higher state space, plays longer (98 vs 33) and thus will regress to the mean (where the house almost always wins). If you look at our fun-loving player, you never reach a point where you win forever. You always end bankrupt with a probability of 1[2]. Our Risk Averse Player has a higher variability and shorter state space and thus can somewhat delay his or her "regression to the mean".

In terms of strategy we can limit ourselves to selecting the best strategy of our Risk Averse Player since performance wise they are better than the Risk Seeking Player. If we refer to the table we can see that top strategies based on %winning of the game and amount won is KO and Hi-Lo

Metric	Hi-Lo	KO
Average Amount Won	823.10	836.65
Percentage Games Won	34.9%	36.0%

There is a slight edge to KO over the Hi-Lo however given the complexity of doing a KO strategy the player is better of doing the Hi-Lo. Less complex and close pay-off.

Now, if you are player who enjoys the casino and just wants to play we can look at Scenario 3 table. Hi-Opt1 has average round of 892. So, a player who wants to enjoy the casino should use that strategy.

[2] We played around with larger simulation runs, we never found a point where the simulation ran forever

References

Baldwin, R. C. (1956). The Optimum strategy in Blackjack. *Journal of the American Statistical Association*, 429-439.

Barker, J. (2015, March 29). Card counters, casinos battle to tilt the odds. *The Baltimore Sun*, pp. http://www.baltimoresun.com/business/bs-bz-counting-cards-20150328-story.html.

Dunross, I. (2016). *Blackjack: Everything You Need To Know About Blackjack From Beginner To Expert.* CreateSpace Independent Publishing Platform .

hi-opt-i. (n.d.). Retrieved from www.blackjacktactics.com: https://www.blackjacktactics.com/blackjack/strategy/card-counting/hi-opt-i/

Mark Broadie, D. J. (1993). An Application of Markov Chain Analysis to the Game of Squash*. 1023–1035.

Mlodinow, L. (2008). *The Drunkard's Walk: How Randomness Rules Our Lives.* Pantheon Books.

Ofton, L. (. (n.d.). *The History of Blackjack*. Retrieved from blackjackapprenticeship: https://www.blackjackapprenticeship.com/resources/history-of-blackjack/

Seongjoo Song, J. S. (2013). A Note on the History of the Gambler's Ruin Problem. *Communications for Statistical Applications and Methods*, 157–168.

Snyder, A. (1984, 10 03). *Blackjackforumonline*. Retrieved 05 10, 2017, from Arnold Snyder's Blackjack Forum: http://www.blackjackforumonline.com/content/sdcnt.htm

Snyder, A. (2006). *history-of-blackjack.htm*. Retrieved from www.blackjackforumonline.com: http://www.blackjackforumonline.com/content/history-of-blackjack.htm

Thorp, E. (1961). A favorable strategy for twenty-one. *Proceedings of the National Academy of Sciences*, 111-112.

Vaidyanathan, A. (2014). Monte Carlo Comparison of Strategies of Blackjack. *Undergraduate Thesis Deparment of Mathematics*, 24-25. Retrieved from wizardofodds.com: https://wizardofodds.com/games/blackjack/card-counting/high-low/

Vancura, O. (2005). *Knock-Out Blackjack: The Easiest Card Counting System Ever Devised.* Las Vegas, NV, USA: Huntington Press.

Wintle, A. (2010, February 11). *History of Blackjack.* . Retrieved from www.wopc.co.uk: http://www.wopc.co.uk/history/blackjack/blackjack

Java Code

```
//----------------------------------------
//
// Card.java
//
//----------------------------------------

import java.util.ArrayList;
import java.util.Arrays;

public class Card {

  int cardSuit, cardNumber;

  /*
  cardSuit
    0 : Spade
    1 : Heart
    2 : Diamond
    3 : Club
  cardNumber
    1 : Ace
    2 ~ 10 : Number
    11 : Jack
    12 : Queen
    13 : King
  */

  public Card(int _cardSuit, int _cardNumber) {
    cardSuit = _cardSuit;
    cardNumber = _cardNumber;
  }

  public int getCardSuit() {
    return cardSuit;
  }

  public int getCardNumber() {
    return cardNumber;
  }

  public ArrayList<Integer> getCount() {
    if(cardNumber == 1)
      return new ArrayList<>(Arrays.asList(1, 11));
    else if(cardNumber > 10)
      return new ArrayList<>(Arrays.asList(10));
    else
      return new ArrayList<>(Arrays.asList(cardNumber));
  }

  public String toString() {

    String cardSuitStr = "", cardNumberStr = "";

    switch(cardSuit) {
      case 0:
        cardSuitStr = "Spade";
        break;
      case 1:
        cardSuitStr = "Heart";
        break;
      case 2:
        cardSuitStr = "Diamond";
        break;
      case 3:
```

```
JCC = javac
JVM = java
.SUFFIXES: .java .class
.java.class: ; $(JC) $(JFLAGS) $*.java

JFLAGS = -g

default: classes

MAIN = Game

CLASSES = \
        Game.java \
        OneGame.java \
        OneDeckGame.java \
        Card.java \
        Deck.java \
        Counting.java \
        Dealer.java \
        Player.java

Game.class: Game.java
        $(JCC) $(JFLAGS) Game.java

OneGame.class: OneGame.java
        $(JCC) $(JFLAGS) OneGame.java

OneDeckGame.class: OneDeckGame.java
        $(JCC) $(JFLAGS) OneDeckGame.java

Card.class: Card.java
        $(JCC) $(JFLAGS) Card.java

Deck.class: Deck.java
        $(JCC) $(JFLAGS) Deck.java

Counting.class: Counting.java
        $(JCC) $(JFLAGS) Counting.java

Dealer.class: Dealer.java
        $(JCC) $(JFLAGS) Dealer.java

Player.class: Player.java
        $(JCC) $(JFLAGS) Player.java

classes: $(CLASSES:.java=.class)

test: Card.class OneGame.class OneDeckGame.java Deck.class Counting.class Dealer.class
Player.class classes $(MAIN).class
        $(JVM) $(MAIN)

clean:
        $(RM) *.class
```

```
        cardSuitStr = "Club";
    }

    switch(cardNumber) {
      case 11:
        cardNumberStr = "Jack";
        break;
      case 12:
        cardNumberStr = "Queen";
        break;
      case 13:
        cardNumberStr = "King";
        break;
      default:
        cardNumberStr = Integer.toString(cardNumber);
    }

    return cardSuitStr + "-" + cardNumberStr;
  }
}
```

```
//----------------------------------------
//
// Deck.java
//
//----------------------------------------

import java.util.LinkedList;
import java.util.ArrayList;
import java.util.List;
import java.util.Collections;

public class Deck {

  LinkedList<Card> cardList = new LinkedList<Card>();

  public Deck(int deckNumber) {
    while (deckNumber > 0) {
      for(int i = 0; i < 4; i++) {
        for(int j = 1; j < 14; j++) {
          Card card = new Card(i, j);
          cardList.add(card);
        }
      }
      deckNumber--;
    }
  }

  public void shuffleCard() {
    Collections.shuffle(cardList);
  }

  public Card popCard() {
    return cardList.pop();
  }

  public int getDeckSize() {
    return cardList.size();
  }

  public String toString() {

    String str = "";
    for(int i = 0; i < cardList.size(); i++) {
      str += (cardList.get(i).toString() + " ");
    }
    return str;
  }
}
```

```
//----------------------------------------
//
// Dealer.java
//
//----------------------------------------

import java.util.ArrayList;
import java.util.Arrays;

public class Dealer {

  ArrayList<Card> handCards;
  boolean isHit;
  boolean isBust;

  public Dealer () {
    handCards = new ArrayList<>();
    isHit = true;
    isBust = false;
  }

  public boolean getIsHit() {
    return isHit;
  }

  public void setIsHit(boolean _isHit) {
    isHit = _isHit;
  }

  public boolean getIsBust() {
    return isBust;
  }

  public ArrayList<Card> getHandCards() {
    return handCards;
  }

  public int dealing(Card card) {
    handCards.add(card);

    ArrayList<Integer> counts = getHand();

    if(isAllHigher(counts, 21)) {
      isBust = true;
      isHit = false;
    } else {
      if (isAnyBetween(counts, 17, 22)) {
        isHit = false;
      } else {
        if (counts.contains(17)) {
          if(counts.size() != 2) {
            isHit = false;
          }
        }
      }
    }
    return getClosestLess(counts, 22);
  }

  public static boolean isAllHigher(ArrayList<Integer> array, int num) {
    for (int i = 0; i < array.size(); i++) {
      if(array.get(i) <= num)
        return false;
    }
```

```
  return true;
}

public static boolean isAnyBetween(ArrayList<Integer> array, int num1, int num2) {
  for (int i = 0; i < array.size(); i++) {
    int num = array.get(i);
    if(num > num1 && num < num2)
      return true;
  }
  return false;
}

public static int getClosestLess(ArrayList<Integer> array, int num) {
  int result = 0;

  for (int i = 0; i < array.size(); i++) {
    int currNum = array.get(i);
    if(currNum > result && currNum < num)
      result = currNum;
  }
  return result;
}

public ArrayList<Integer> getHand() {

  ArrayList<Integer> result = new ArrayList<>(Arrays.asList());;

  for (int i = 0; i < handCards.size(); i++) {
    Card card = handCards.get(i);
    ArrayList<Integer> cardCount = card.getCount();
    if (result.size() == 0)
      result = (ArrayList<Integer>) cardCount.clone();
    else {
      ArrayList<Integer> preResult = (ArrayList<Integer>) result.clone();
      for (int x = 0; x < cardCount.size(); x++) {
        for (int y = 0; y < preResult.size(); y++) {
          if (x == 0) {
            result.set(y, preResult.get(y) + cardCount.get(0));
          } else {
            result.add(preResult.get(y) + cardCount.get(x));
          }
        }
      }
    }
  }
  return result;
}

public void clearHand() {
  handCards = new ArrayList<>();
  isHit = true;
  isBust = false;
}

public String toString() {
  String text = "Dealer: ";
  text += "Hand Cards: ";
  for (int i = 0; i < handCards.size(); i++) {
    text += (handCards.get(i).toString() + " ");
  }
  text += ("|| Hand: " + getHand());
  if(isBust)
    return text + "\n" + "Dealer Bust";
  return text;
```

```
    }
}
```

```
//----------------------------------------
//
// Player.java
//
//----------------------------------------

import java.util.ArrayList;
import java.util.Arrays;

public class Player {

  ArrayList<Card> handCards;
  float money;
  boolean isHit;
  boolean isBust;
  boolean is21;

  public Player(float startMoney) {
    handCards = new ArrayList<>();
    money = startMoney;
    isHit = true;
    isBust = false;
    is21 = false;
  }

  public float getMoney() {
    return money;
  }

  public void betMoney(float _money) {
    money -= _money;
  }

  public void addMoney(float _money) {
    money += _money;
  }

  public boolean getIsHit() {
    return isHit;
  }

  public void setIsHit(boolean _isHit) {
    isHit = _isHit;
  }

  public boolean getIsBust() {
    return isBust;
  }

  public boolean getIs21() {
    return is21;
  }

  public ArrayList<Card> getHandCards() {
    return handCards;
  }

  public int getHighestCount() {
    return getClosestLess(getHand(), 22);
  }

  public int playing(Card card) {
    handCards.add(card);
```

```
  ArrayList<Integer> counts = getHand();

  if(isAllHigher(counts, 21)) {
    isBust = true;
    isHit = false;
  } else {
    if (isAnyBetween(counts, 17, 22)) {
      isHit = false;
    } else {
      if (counts.contains(17)) {
        if(counts.size() != 2) {
          isHit = false;
        }
      }
    }
  }

  //-----------------------------------------------------
  // Check 21 at once
  if(counts.contains(21) && handCards.size() == 2) {
    is21 = true;
  }
  //-----------------------------------------------------
  return getClosestLess(counts, 22);
}

public static boolean isAllHigher(ArrayList<Integer> array, int num) {
  for (int i = 0; i < array.size(); i++) {
    if(array.get(i) <= num)
      return false;
  }
  return true;
}

public static boolean isAnyBetween(ArrayList<Integer> array, int num1, int num2) {
  for (int i = 0; i < array.size(); i++) {
    int num = array.get(i);
    if(num > num1 && num < num2)
      return true;
  }
  return false;
}

public static int getClosestLess(ArrayList<Integer> array, int num) {
  int result = 0;

  for (int i = 0; i < array.size(); i++) {
    int currNum = array.get(i);
    if(currNum > result && currNum < num)
      result = currNum;
  }
  return result;
}

public ArrayList<Integer> getHand() {

  ArrayList<Integer> result = new ArrayList<>(Arrays.asList());;

  for (int i = 0; i < handCards.size(); i++) {
    Card card = handCards.get(i);
    ArrayList<Integer> cardCount = card.getCount();
    if (result.size() == 0)
      result = (ArrayList<Integer>) cardCount.clone();
    else {
      ArrayList<Integer> preResult = (ArrayList<Integer>) result.clone();
```

```
        for (int x = 0; x < cardCount.size(); x++) {
          for (int y = 0; y < preResult.size(); y++) {
            if (x == 0) {
              result.set(y, preResult.get(y) + cardCount.get(0));
            } else {
              result.add(preResult.get(y) + cardCount.get(x));
            }
          }
        }
      }
    }
    return result;
  }

  public void clearHand() {
    handCards = new ArrayList<>();
    isHit = true;
    isBust = false;
    is21 = false;
  }

  public String toString() {
    String text = "Player: ";
    text += "Hand Cards: ";
    for (int i = 0; i < handCards.size(); i++) {
      text += (handCards.get(i).toString() + " ");
    }
    text += (" || Hand: " + getHand());
    if(isBust)
      return text + "\n" + "Player Bust";
    return text;
  }
}
```

```
//----------------------------------------
//
// Counting.java
//
//----------------------------------------

public class Counting {

  int count;
  boolean isCounting = false;
  String method;

  public Counting(int _count, String _method) {
    count = _count;
    if(_method.equals("none"))
      isCounting = false;
    else
      isCounting = true;
    method = _method;
  }

  public int getCount() {
    return count;
  }

  public void setMethod(String newMethod) {
    method = newMethod;
  }

  public String getMethod() {
    return method;
  }

  public boolean getIsCounting() {
    return isCounting;
  }

  public void countCard(Card card) {
    if(isCounting) {
      if (method.equals("HiLo")) {
        HiLo(card);
      } else if (method.equals("HiOpt1")) {
        HiOpt1(card);
      } else if (method.equals("HiOpt2")) {
        HiOpt2(card);
      } else if (method.equals("KO")) {
        KO(card);
      }
    }
  }

  public String toString() {
    return "Method : " + method + "\tCount: " + count;
  }

  //-----------------------------------------------------------------
  //Methods

  //HiLo
  public void HiLo(Card card) {
    int cardNumber = card.cardNumber;

    if(cardNumber > 1 && cardNumber < 7)
      count++;
```

```
    else if(cardNumber > 9 || cardNumber == 1)
      count--;
    if(count < 0)
      count = 0;
  }

  public void HiOpt1(Card card) {
    int cardNumber = card.cardNumber;

    if(cardNumber >= 3 && cardNumber <= 6)
      count++;
    else if(cardNumber > 9)
      count--;
    if(count < 0)
      count = 0;
  }

  public void HiOpt2(Card card) {
    int cardNumber = card.cardNumber;

    if(cardNumber >= 2 && cardNumber <= 6) {
      count++;
      if (cardNumber >= 4 && cardNumber <= 5)
        count++;
    }
    else if(cardNumber > 9)
      count -= 2;
    if(count < 0)
      count = 0;
  }

  public void KO(Card card) {
    int cardNumber = card.cardNumber;

    if(cardNumber > 1 && cardNumber < 8)
      count++;
    else if(cardNumber > 9 || cardNumber == 1)
      count--;
    if(count < 0)
      count = 0;
  }

  //---------------------------------------------------------------
}
```

```
//----------------------------------------
//
// OneDeckGame.java
//
//----------------------------------------

import java.util.ArrayList;
import java.util.Arrays;

public class OneDeckGame {

  String countingMethod ="";
  float money = 0;
  float maxMoney = 0;
  int minCount = 0;
  boolean isDouble = false;
  boolean isSplit = false;
  boolean isFixedBetting = false;
  float bettingFixedWinMoney = 0;
  float bettingFixedLoseMoney = 0;
  int bettingWinRate = 0;
  int bettingLoseRate = 0;
  float totalWinMoney = 0;
  float totalLoseMoney = 0;
  int totalWinWithWinBetting = 0;
  int totalLoseWithLoseBetting = 0;

  public OneDeckGame(String _countingMethod, float _money, float _maxMoney, int _minCount,
boolean _isDouble, boolean _isSplit, boolean _isFixedBetting, float _bettingFixedWinMoney,
float _bettingFixedLoseMoney,int _bettingWinRate, int _bettingLoseRate) {
    //-----------------------------------------------------
    // Parameters
    //
    // Order
    // 0      Counting Method
    // 1      Starting Money
    // 2      Maximum Money Player Reach
    // 3      Minimum Count for Start Betting
    // 4      Rule Possible for Double
    // 5      Rule Possible for Split
    // 6      Fixed Betting
    // 7      Fixed Win Betting (Non-Counting Game -> WinBetting)
    // 8      Fixed Lose Betting
    // 9      Win Betting Rate
    // 10     Lose Betting Rate
    // 11     Total Money Player Win
    // 12     Total Money Player Lose
    //-----------------------------------------------------

    countingMethod = _countingMethod;
    money = _money;
    maxMoney = _maxMoney;
    minCount = _minCount;
    isDouble = _isDouble;
    isSplit = _isSplit;
    isFixedBetting = _isFixedBetting;
    bettingFixedWinMoney = _bettingFixedWinMoney;
    bettingFixedLoseMoney = _bettingFixedLoseMoney;
    bettingWinRate = _bettingWinRate;
    bettingLoseRate = _bettingLoseRate;
  }

  public float[] PlayOneDeckGame() {
```

```
    //-----------------------------------------------------
    // Return values
    //
    // Order
    // 0      Number Of Total Games
    // 1      Number Of Player Wins
    // 2      Number Of Dealer Wins
    // 3      Number Of Pushes
    // 4      Number Of Player Wins by High Count
    // 5      Number Of Player Wins by 21
    // 6      Number Of Player Wins by Dealer Bust
    // 7      Number Of Dealer Wins by High Count
    // 8      Number Of Dealer Wins by Player Bust
    // 9      Remained Money
    // 10     Maximum Money Player Reach
    // 11     Total Money Player Win
    // 12     Total Money Player Lose
    //-----------------------------------------------------

    Deck deck = new Deck(6);
    Dealer dealer = new Dealer();
    Player player = new Player(money);
    int dealerCount = 0, playerCount = 0;
    Counting counting = new Counting(0, countingMethod);

    int numTotalGame = 0, numPlayerWin = 0, numDealerWin = 0, numPush = 0;
    int numPlayerWinWithHighCount = 0, numPlayerWinWith21 = 0, numPlayerWinWithDealerBust =
0;
    int numDealerWinWithHighCount = 0, numDealerWinWithPlayerBust = 0;

    float bettingMoney = 0;
    float bettingWinMoney = 0;
    float bettingLoseMoney = 0;

    if(isFixedBetting) {
      bettingWinMoney = bettingFixedWinMoney;
      bettingLoseMoney = bettingFixedLoseMoney;
    } else {
      bettingWinMoney = bettingWinMoney = money / bettingWinRate;
      bettingLoseMoney = money / bettingWinRate / bettingLoseRate;
    }

    boolean isPlaying = true;

    //Shuffling
    deck.shuffleCard();

    System.out.println("\n");

    while(deck.getDeckSize() > 14) {

      //-----------------------------------------------------
      //Playing Status
      if(!isPlaying && counting.getIsCounting()) {
        if(counting.getCount() >= minCount) {
          isPlaying = true;
        }
      }
      //-----------------------------------------------------

      //-----------------------------------------------------
      //Betting Money

      //Counting
      if(isPlaying) {
```

```
  if(counting.getCount() >= minCount) {
    bettingMoney = bettingWinMoney;
  } else {
    bettingMoney = bettingLoseMoney;
  }

  if(player.getMoney() < bettingMoney) {
    bettingMoney = player.getMoney();
  }

} else {
  bettingMoney = 0;
}

//Non-Counting
if(!counting.getIsCounting()) {
  bettingMoney = bettingWinMoney;
  if(player.getMoney() < bettingMoney) {
    bettingMoney = player.getMoney();
  }
}

player.betMoney(bettingMoney);

System.out.println("Player Money: " + player.getMoney());
System.out.println("Betting Money: " + bettingMoney);

//-------------------------------------------------------

//-------------------------------------------------------
//Initialize the Game
System.out.println("Initialize the Game");
if(dealer.getIsHit()) {
  Card card = deck.popCard();
  counting.countCard(card);
  dealerCount = dealer.dealing(card);
  System.out.println(dealer.toString() + " ");
}
System.out.println(counting.toString() + " ");

Card hiddenCard = deck.popCard();

if(player.getIsHit()) {
  Card card1 = deck.popCard();
  Card card2 = deck.popCard();
  counting.countCard(card1);
  counting.countCard(card2);
  playerCount = player.playing(card1);
  playerCount = player.playing(card2);
  System.out.println(player.toString() + " ");
}
//-------------------------------------------------------

if(player.getIs21()) {
  numPlayerWin++;
  numPlayerWinWith21++;
  float winMoney = bettingMoney + (float)(bettingMoney * 1.5);

  player.addMoney(winMoney);
  if (bettingMoney == bettingWinMoney) {
    totalWinWithWinBetting++;
  }
  totalWinMoney += winMoney;

} else {
```

```
        while(player.getIsHit()) {
          //------------------------------
          //Counting exception
          if(counting.getCount() >= minCount && counting.getIsCounting()) {
            if (player.getHighestCount() > 14) {
              player.setIsHit(false);
              break;
            }
          }
          //------------------------------
          Card card = deck.popCard();
          counting.countCard(card);
          playerCount = player.playing(card);
        }
      }

      //-----------------------------------------------------
      // Double Method
      if (isDouble) {
        ArrayList<Integer> dealerHand = dealer.getHand();
        if(!(dealerHand.size() == 2 || dealerHand.get(0) == 10) && counting.getCount() >=
minCount) {
          if(player.getMoney() >= bettingMoney) {
            bettingMoney += bettingMoney;
            player.betMoney(bettingMoney);
          }
        }
      }
      //-----------------------------------------------------

      //Flip the hidden dealer card
      counting.countCard(hiddenCard);
      dealerCount = dealer.dealing(hiddenCard);

      if(!player.getIsBust() && !player.getIs21()) {
        while(dealer.getIsHit()) {
          Card card = deck.popCard();
          counting.countCard(card);
          dealerCount = dealer.dealing(card);
        }
      }
      dealer.setIsHit(false);

      System.out.println("Player: " + playerCount + " ");
      System.out.println("Dealer: " + dealerCount + " ");

      if(!player.getIs21()) {
        if(playerCount > dealerCount) {
          if(dealerCount == 0) {
            numPlayerWinWithDealerBust++;
          } else {
            numPlayerWinWithHighCount++;
          }
          totalWinMoney += bettingMoney;
          numPlayerWin++;
          if (bettingMoney == bettingWinMoney) {
            totalWinWithWinBetting++;
          }
          player.addMoney(bettingMoney + bettingMoney);
        } else if(playerCount < dealerCount) {
          if(playerCount == 0) {
            numDealerWinWithPlayerBust++;
          } else {
            numDealerWinWithHighCount++;
          }
```

```
          totalLoseMoney += bettingMoney;
          if (bettingMoney == bettingLoseMoney) {
            totalLoseWithLoseBetting++;
          }
          numDealerWin++;
        } else {
          numPush++;
          player.addMoney(bettingMoney);
        }
      }

      if(!dealer.getIsHit() && !player.getIsHit()) {
        System.out.println(dealer.toString());
        System.out.println(player.toString());
        System.out.println("Finish the game");

        dealer.clearHand();
        player.clearHand();
      }

      System.out.println("Player Money: " + player.getMoney());

      if(player.getMoney() > maxMoney) {
        maxMoney = player.getMoney();
      }

      numTotalGame++;
      if(player.getMoney() < 1) {
        break;
      }

      if(player.getMoney() < 500 || player.getMoney() > 1500) {
        break;
      }

      System.out.print("\n");
    }
    System.out.println("Game Finished");
    return new float[]{numTotalGame, numPlayerWin, numDealerWin, numPush,
numPlayerWinWithHighCount, numPlayerWinWith21, numPlayerWinWithDealerBust,
numDealerWinWithHighCount, numDealerWinWithPlayerBust, player.getMoney(), maxMoney,
totalWinMoney, totalLoseMoney, totalWinWithWinBetting, totalLoseWithLoseBetting};
  }
}
```

```
//----------------------------------------
//
// OneGame.java
//
//----------------------------------------

public class OneGame {

  String countMethod = "";
  float startingMoney = 0;
  int minCount = 0;
  int numGame = 0;
  boolean isDouble = false;
  boolean isSplit = false;
  boolean isFixedBetting = false;
  float bettingFixedWinMoney = 0;
  float bettingFixedLoseMoney = 0;
  int bettingWinRate = 0;
  int bettingLoseRate = 0;
  float totalWinMoney = 0;
  float totalLoseMoney = 0;
  int totalWinWithWinBetting = 0;
  int totalLoseWithLoseBetting = 0;

  public OneGame (String _countMethod, float _startingMoney, int _minCount, int _numGame,
boolean _isDouble, boolean _isSplit, boolean _isFixedBetting, float _bettingFixedWinMoney,
float _bettingFixedLoseMoney, int _bettingWinRate, int _bettingLoseRate) {
    //------------------------------------------------
    // Parameters
    //
    // Order
    // 0      Counting Method
    // 1      Starting Money
    // 2      Minimum Count for Start Betting
    // 3      Number of Deck to Play
    // 4      Rule Possible for Double
    // 5      Rule Possible for Split
    // 6      Fixed Betting
    // 7      Fixed Win Betting (Non-Counting Game -> WinBetting)
    // 8      Fixed Lose Betting
    // 9      Win Betting Rate
    // 10     Lose Betting Rate
    //------------------------------------------------
    countMethod = _countMethod;
    startingMoney = _startingMoney;
    minCount = _minCount;
    numGame = _numGame;
    isDouble = _isDouble;
    isSplit = _isSplit;
    isFixedBetting = _isFixedBetting;
    bettingFixedWinMoney = _bettingFixedWinMoney;
    bettingFixedLoseMoney = _bettingFixedLoseMoney;
    bettingWinRate = _bettingWinRate;
    bettingLoseRate = _bettingLoseRate;
  }

  public float[] PlayOneGame() {

    //------------------------------------------------
    // Return values
    //
    // Order
    // 0      Number of Total Games
    // 1      Number of Player Wins
```

```
    // 2      Number of Dealer Wins
    // 3      Number of Pushes
    // 4      Number of Player Wins by High Card
    // 5      Number of Player Wins by 21
    // 6      Number of Plyaer Wins by Dear Bust
    // 7      Number of Dealer Wins by Hish Card
    // 8      Number of Dealer Wins by Player Bust
    // 9      Remained Money
    // 10     Maximum Money Player Reach
    // 11     Total Money Player Win
    // 12     Total Money Player Lose
    // 13     Total Win with Winning Betting
    // 14     Total Lose with Losing Betting
    //-----------------------------------------------------

    int numTotalGame = 0;
    int numPlayerWin = 0;
    int numDealerWin = 0;
    int numPush = 0;

    int numPlayerWinWithHighCount = 0;
    int numPlayerWinWith21 = 0;
    int numPlayerWinWithDealerBust = 0;
    int numDealerWinWithHighCount = 0;
    int numDealerWinWithPlayerBust = 0;

    float money = startingMoney;
    float maxMoney = money;

    float[] result;

    for(int i = 0; i < numGame; i++) {
      OneDeckGame oneDeckGame = new OneDeckGame(countMethod, money, maxMoney, minCount,
isDouble, isSplit, isFixedBetting, bettingFixedWinMoney, bettingFixedLoseMoney,
bettingWinRate, bettingLoseRate);
      result = oneDeckGame.PlayOneDeckGame();
      numTotalGame += result[0];
      numPlayerWin += result[1];
      numDealerWin += result[2];
      numPush += result[3];
      numPlayerWinWithHighCount += result[4];
      numPlayerWinWith21 += result[5];
      numPlayerWinWithDealerBust += result[6];
      numDealerWinWithHighCount += result[7];
      numDealerWinWithPlayerBust += result[8];
      money = result[9];
      maxMoney = result[10];
      totalWinMoney += result[11];
      totalLoseMoney += result[12];
      totalWinWithWinBetting += result[13];
      totalLoseWithLoseBetting += result[14];

      if (money < 1) {
        break;
      }

      if (money < 500 || money > 1500) {
        break;
      }
    }

    return new float[]{numTotalGame, numPlayerWin, numDealerWin, numPush,
numPlayerWinWithHighCount, numPlayerWinWith21, numPlayerWinWithDealerBust,
numDealerWinWithHighCount, numDealerWinWithPlayerBust, money, maxMoney, totalWinMoney,
totalLoseMoney, totalWinWithWinBetting, totalLoseWithLoseBetting};
```

```
    }
}
```

```
//----------------------------------------
//
// Game.java
//
//----------------------------------------

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Game {
    public static void main(String args[]) throws FileNotFoundException {

    //-----------------------------------------------------
    // Non-static variables
    int numberOfStat = 1000;
    int numberOfOneGameDeck = 100;
    float startingMoney = 1000;
    // countMethod = none | HiLo | HiOpt1 | HiOpt2 | KO
    String countMethod = "none";
    boolean isDouble = false;
    boolean isSplit = false;
    boolean isFixedBetting = true;
    float bettingFixedWinMoney = 100;
    float bettingFixedLostMoney = 10;
    int bettingWinRate = 0;
    int bettingLoseRate = 0;
    //-----------------------------------------------------

    float result[][] = new float[numberOfStat][15];
    float finalResult[][] = new float[15][5];
    float totalNumberGame = 0;
    float totalPlayerWinning = 0;
    float totalDealerWinning = 0;
    float totalPush = 0;
    float totalMaxMoney = 0;

    PrintWriter printWriter = new PrintWriter(new File("result.csv"));
    StringBuilder stringBuilder = new StringBuilder();

    stringBuilder.append("Total Game");
    stringBuilder.append(',');
    stringBuilder.append("Player Win");
    stringBuilder.append(',');
    stringBuilder.append("Dealer Win");
    stringBuilder.append(',');
    stringBuilder.append("Push");
    stringBuilder.append(',');
    stringBuilder.append("Player Winning Type");
    stringBuilder.append(',');
    stringBuilder.append("High Card");
    stringBuilder.append(',');
    stringBuilder.append("Dealer Bust");
    stringBuilder.append(',');
    stringBuilder.append("21");
    stringBuilder.append(',');
    stringBuilder.append("Dealer Winning Type");
    stringBuilder.append(',');
    stringBuilder.append("High Card");
    stringBuilder.append(',');
    stringBuilder.append("Player Bust");
    stringBuilder.append(',');
    stringBuilder.append("Money");
    stringBuilder.append(',');
```

```
    stringBuilder.append("Max Money");
    stringBuilder.append(',');
    stringBuilder.append("Toal Win Money");
    stringBuilder.append(',');
    stringBuilder.append("Total Lose Money");
    stringBuilder.append(',');
    stringBuilder.append("Total Win with Winning Betting");
    stringBuilder.append(',');
    stringBuilder.append("Total Lose with Losing Betting");
    stringBuilder.append("\n");

    //-----------------------------------------------------
    // Final Result
    //
    // Order
    // 0      Average Total Game
    // 1      Average Player Winning Rate
    // 2      Average Dealer Winning Rate
    // 3      Average Push
    // 4      Average Max Money
    //
    //-----------------------------------------------------

    for(int num = 10; num < 11; num++) {
        stringBuilder.append("Minimum count " + num);
        stringBuilder.append("\n");
      for(int i = 0; i < numberOfStat; i++) {
        OneGame oneGame = new OneGame(countMethod, startingMoney, num, numberOfOneGameDeck,
isDouble, isSplit, isFixedBetting, bettingFixedWinMoney, bettingFixedLostMoney,
bettingWinRate, bettingLoseRate);
        result[i] = oneGame.PlayOneGame();
        totalNumberGame += result[i][0];
        totalPlayerWinning += (result[i][1] / result[i][0]);
        totalDealerWinning += (result[i][2] / result[i][0]);
        totalPush += (result[i][3] / result[i][0]);
        totalMaxMoney += result[i][10];
      }

      System.out.println("Result " + num);
      System.out.println("Total Game\tPlayer Win\tDealer Win\tPush\tPlayer Winning
Type:\tHigh Card:\t Dealer Bust:\t 21:\tDealer Winning Type:\t\t\tHigh Card:\tPlayer
Bust:\tMoney:\tMax Money:");

      for(int i = 0; i < numberOfStat; i++) {
        System.out.println(Integer.toString((int)result[i][0]) + "\t\t" +
Integer.toString((int)result[i][1]) + "\t\t" + Integer.toString((int)result[i][2]) + "\t\t" +
Integer.toString((int)result[i][3]) + "\t\t\t\t" + Integer.toString((int)result[i][4]) +
"\t\t" + Integer.toString((int)result[i][5]) + "\t\t" + Integer.toString((int)result[i][6]) +
"\t\t\t\t" + Integer.toString((int)result[i][7]) + "\t\t" + Integer.toString((int)result[i]
[8]) + "\t\t" + String.format("%.2f", result[i][9]) + "\t" + String.format("%.2f", result[i]
[10]));
        stringBuilder.append(Integer.toString((int)result[i][0]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][1]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][2]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][3]));
        stringBuilder.append(',');
        stringBuilder.append("");
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][4]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][5]));
        stringBuilder.append(',');
```

```
        stringBuilder.append(Integer.toString((int)result[i][6]));
        stringBuilder.append(',');
        stringBuilder.append("");
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][7]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][8]));
        stringBuilder.append(',');
        stringBuilder.append(String.format("%.2f", result[i][9]));
        stringBuilder.append(',');
        stringBuilder.append(String.format("%.2f", result[i][10]));
        stringBuilder.append(',');
        stringBuilder.append(String.format("%.2f", result[i][11]));
        stringBuilder.append(',');
        stringBuilder.append(String.format("%.2f", result[i][12]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][13]));
        stringBuilder.append(',');
        stringBuilder.append(Integer.toString((int)result[i][14]));
        stringBuilder.append("\n");
    }

      finalResult[num][0] = (float)totalNumberGame / numberOfStat;
      finalResult[num][1] = (float)totalPlayerWinning / numberOfStat;
      finalResult[num][2] = (float)totalDealerWinning / numberOfStat;
      finalResult[num][3] = (float)totalPush / numberOfStat;
      finalResult[num][4] = (float)totalMaxMoney / numberOfStat;

      totalNumberGame = 0;
      totalPlayerWinning = 0;
      totalDealerWinning = 0;
      totalPush = 0;
      totalMaxMoney = 0;
    }
    printWriter.write(stringBuilder.toString());
    printWriter.close();
    System.out.println("Final Result");
    System.out.println("Min Count\tAvg Game\tAvg Player Win\tAvg Dealer Win\tAvg Push\tAvg
Max Money");

    for(int i = 0; i < 15; i++) {
      System.out.println(i + "\t\t" + String.format("%.8f", finalResult[i][0]) + "\t" +
String.format("%.8f", finalResult[i][1]) + "\t" + String.format("%.8f", finalResult[i][2]) +
"\t" + String.format("%.8f", finalResult[i][3]) + "\t" + String.format("%.8f", finalResult[i]
[4]));
    }
  }
}
```