CASINO EUCHRE: FROM THE MIDWEST TO LAS VEGAS

A Thesis submitted for Albion College Honors

Claire E. Mitchell

April 1, 2021

Albion College

Contents

1 Acknowledgements

This research project was not a singular effort, far from it. Without the help of many individuals I would not have been able to complete it.

First, thank you to my advisor and the originator of the idea of casino euchre, Dr. Mark Bollman, without whom I would never been able to complete this project. Thank you to my committee for taking part in this process and reading my drafts.

Next, I would like to recognize Dr. Vanessa McCaffrey, Renee Kreger and the FURSCA committee for giving me the opportunity to do my research through FURSCA. I would also like to thank everyone who played hands of Casino Euchre with me.

Thank you to my brother, a software engineer, who was extremely helpful in both teaching me computer programming and yelling at me for doing it wrong. Without him I would have likely cried more out of frustration and less from being yelled at.

Finally, I would like to thank Niko, my American Eskimo dog friend. He was essential in helping me debug my code and keeping me sane during the grueling coding process. For reference he can be seen in Image 1.

Image 1

2 Introductions

If baseball is America's favorite pastime, then gambling could be argued as America's guilty pleasure. There are around 500 commercial casinos in the United States [1]. Each casino is packed with games, but what does it take to make a casino game successful? Can these guides be applied to other games to make them fit for the casino? Euchre is argued to be Michigan's favorite card game. What if casinos and euchre could be merged? Could the guides to success for casino games be applied to euchre to make it a crowd drawing casino game? The idea of euchre as a feasible casino game will be investigated over the course of this paper.

2.1 Euchre

Euchre is a card game commonly played in the Midwest. It is now most widely known in Michigan. The exact origin of euchre is debated, but has roots in Germany and is theorized to be a derivation of juckerspiel brought to America by German immigrants [2]. Euchre is played using a 24-card deck that spans from 9 to Ace using all suits. The game is typically played with two two-player teams. Each player is dealt five cards. A trump card is then declared by the players and the players play tricks, or individual rounds, totaling to five. The player to the left of the dealer plays a card and then the other players try and beat that card by laying down one of their own of higher ranking. Players must follow suit if possible, failing to do this is called reneging. The rankings are based on the declared trump. An example of the rankings with the trump being hearts can be seen in Figure 1.

Figure 1

The person with the highest valued card wins the trick. The points awarded to the team depends on a number of factors listed below. A player can also choose to play without a partner if they believe they can win without them. If a partnership chooses the trump and the other partnership wins more tricks that is called being euchred. The points are as follows:

Partnership choosing trump wins 3 or 4 tricks – 1 point

Partnership choosing trump wins 5 tricks – 2 points

Lone hand wins 3 or 4 tricks – 1 point

Lone hand wins 5 tricks – 4 points

Partnership or lone hand is euchred, opponents score 2 points

The dealer rotates to the left after each round, with each round containing five tricks. The first team to 10 points wins. Many variations of the game exist, but the core concepts remain the same throughout.

2.2 Casino Games

Americans spent upwards of $44 billion dollars in 2018 at casinos [3]. This is significantly more than is spent on movie theatre tickets. Gambling is one of America's favorite pastimes. There are currently around 1,200 Nevada approved casino games [4]. Some of these are regularly played in casinos, like Texas Hold'em and Keno whereas others have been retired or are seldom played and rarely if ever heard of. There are even more casino games that never made it to the floor. These games failed for any number of reasons, from idea to execution.

There are a few common components to successful casino games, which are largely discussed in *The Essentials of Casino Game Design* [5]. For the game to even be considered for a casino game there has to be a house advantage, the percentage of total initial wagers the casino expects to earn over the long run if all players used perfect player strategy against the game [6]. The casino needs to be able to make money off of the game. The game also needs to be relatively balanced; if the house advantage is too high then no one would play because they wouldn't feel like they had a chance to win, but if it's too low it's not profitable for the casino. There are exceptions to this rule, like Keno, a game notorious for its very high house advantage of 34-36% [7]. Blackjack is another exception known for having a low house advantage of under 1% [6]. Although there are exceptions, games that are not well established have better odds of becoming successful in a casino if they stay within the house advantage range of 2-6% [6]. Lastly, the game needs to be fast and easy to learn. Players need to be able to follow the game and pick up the rules. This will allow for a wide variety of players to be able to play, creating maximum profit for the casino. There are also smaller factors that are important when creating a casino game, such as name and logo.

Names that are often popular aren't too complex, having too many descriptors can drive a player away. Numbers in the name are acceptable, like Three Card Poker.

The names are also short and to the point, like Blackjack. Never overdo a name and never fall in love with one as it might change [5]. The importance of the name is often understated, but if the player doesn't understand the name or the have an idea of the game from the name then they are never going to sit down to play. For these reasons I initially chose to name the adapted version of euchre 3-Card Euchre, because the player and the dealer both had three cards. It had a number in it which often is successful, and it was clear and to the point. It was later changed to Casino Euchre, when a fourth card was added to the players hand.

2.3 Where I Started

Dr. Mark Bollman had the original idea for this project. He thought it would be interesting to turn euchre into a casino game, as it has never been done before. I heard him discuss this at a talk he gave at Albion in 2018 and told him I was interested in working with him on the project. When I joined the project he already had a rough outline of the adapted rules. These rules were the basis of Casino Euchre and can be found on the following page, as written by Dr. Bollman.

Three-Card Euchre, Version 2.0

January 29, 2019

1. One to six players or teams of players may play against a single dealer.

2. The game is played with a standard 24-card euchre deck consisting of the 9s through aces in all 4 suits.

3. As in standard euchre, one suit will be designated as the trump suit (see item 8). Cards of the trump suit outrank any card of a non-trump suit.

4. The ranks of the cards shall be as follows:

   - **Within the trump suit:**
     —The jack of the trump suit—the *right bower*—shall rank highest.
     —The jack of the opposite suit—the *left bower*—in the same color shall rank next.
     —The remaining cards of the trump suit shall rank Ace, King, Queen, 10, 9 in descending order.
   - **In the non-trump suits:** The cards shall rank in the usual order: Ace, King, Queen, Jack, 10, 9.

5. Play begins with each player making an ante bet.

6. Each player is dealt 3 cards face down. Players may look at their cards, but may not share information about their cards with each other.

7. The dealer receives 3 cards dealt face down to the table, and does not look at them.

8. The next card in the deck is turned up and its suit becomes the trump suit.

9. Players may either fold their cards without showing them, forfeiting the Ante bet, or make a Play bet of twice their ante. This 3-unit total wager is distributed as an equal bet on each card. Players may also double down by making a Play bet of 5 times their original wager, with the total 6-unit wager distributed as 2 units per card.

10. Working from right to left, the dealer turns over the first card from the table, and players choose a card from their hands to play against it. Players must follow suit if possible.

11. The higher card, as defined in item 4, wins the trick. Winning wagers are paid even money.

12. Step 10 is repeated for the second and third cards, again moving from right to left.

My role was to take over the project and complete it, with the guidance of Dr. Bollman. This research project can be broken down into five stages. The first was to do research on casino game design and gambling mathematics. Dr. Bollman recommended a set of books to read and draw inspiration from. The books will be discussed in Section 3. The second was to do gameplay and rule adaptation. The the third was to execute probability calculations and determine a basic player strategy. The fourth was to create a computer program that played casino euchre at the desired rules and finalize player strategy. The fifth and final stage was to create art for the game as if it were going into a casino.

3 Research

3.1 Essentials of Casino Game Design: From the Cocktail Napkin to the Casino Floor

This book was one of the most helpful books I read throughout the research project. It provided me with all of the elements of a successful casino game discussed in Section 2.2. It guided me in choosing a name, logo, and changing the rules as needed. It held very little mathematical information, but a plethora of information on what goes into making a casino game. [5]

3.2 Contemporary Casino Table Game Design

In terms of an intersection between mathematics and game design, this book is it. It discussed much more mathematics than *Essentials of Casino Game Design* while still maintaining a lot of information on casino game design. This book was helpful in general mathematical overview, and understanding the role of mathematics in the casino game design process. [6]

3.3 Basic Gambling Mathematics: The Numbers Behind the Neon

To quote Dr. Bollman, "This is the best book about gambling mathematics." *Basic Gambling: The Numbers Behind the Neon* written by Dr. Mark Bollman is a comprehensive look into the math that goes behind every casino game. It begins by discussing the fundamentals of probability and statistics and moves on to more advanced probability and the application of statistics in many popular casino games, including; keno, blackjack, roulette, and many more. This book was extremely helpful in The Math stage, as a mathematical guide. [7]

4 Playing

The playing stage was all about the playing of the game using the adapted rules Dr. Bollman had established. A lot of information was gathered by just playing the game over and over again. I played over 300 hands, recording the dealer's hand, the players' hand, and the number of wins and loses. This provided a general feel for House advantage and gave some insight into player strategy. Along with the hands I played independently with willing participants, Dr. Bollman and I spent a few hours in the beginning playing the initial adapted version of euchre with each other, Dr. Bollman acted as the dealer and I as the player. A couple important observations came from the repetitive playing.

The first observation was that hands that are not considered good in traditional euchre were good in the casino version. This is not necessarily a bad thing as people who know traditional euchre well might fall victim to this difference, thus creating casino revenue. It may also spark a competitive spirit in the player to master both versions. For example in traditional euchre three non trump aces would be a bad hand whereas that is a good hand in casino euchre.

The second observation was a problem in the adapted rules. Through playing the game at the initial rules I, the player, was feeling very discouraged. I was losing almost every hand, I started to feel excited when I broke even. It got to the point that I didn't want to play because I kept losing. If I, the researcher, did not want to play because I was losing, any real casino attendee certainly would have quit long before I had begun to feel discouraged. If a game doesn't make the player feel as though they have a chance of winning the game is dead before it even hits the floor. This feeling meant the house advantage was too high, thus it was critical that we lessened the house advantage and gave more to the player.

Essentials of Casino Game Design had discussed different variations of already existing casino games. One of the adaptations took a popular casino game and added a card to the hand. I remember this when brainstorming ways to lessen the house advantage. I decided on adding a card to the players hand, giving them four and allowing them to discard one of their cards prior to playing. This would allow the player a few benefits. The first benefit is that it increases the chances of players having better hand, since they have more cards to choose from and the option to get rid of their worst card. Another benefit is that it provides the player with some sense of control, this control also makes them more interested in the game. Previously the player received their hand and played it without much thought; since the player always follows the dealer and is required to follow suit they often only have one or two choices for a card to play eliminating a lot of strategy that exists in traditional euchre. Allowing them to discard a card lets the player in on the game and feel more in control. This discrepancy between the number of cards in the players hand and the dealers can also work towards giving the player a false feeling of security; they have more cards so they feel they have an advantage. The trade off for adding a card to the player's hand is that less players can play against the dealer. The maximum would be five, but then every card would be in play and card counting would become

more of a problem, thus four is recommended. The four card rule was permanently added to the rules of casino euchre after playing with the new rule for hundreds of hands. Volunteer players were asked whether they preferred three card or four and everyone preferred the four card hand.

5 The Math

5.1 Probability Overview

This section will be covering an overview of basic probability and some applications. All of these concepts can be found in *Basic Gambling: The Numbers Behind the Neon* and/or *Contemporary Casino Table Game Design*. Some definitions:

- *Sample Space*: set of all possible outcomes of an experiment, S.
- *Event*: a subset of the sample space, $A \subset S$. A possible outcome of the experiment.
- Probability of an event A is represented as $P(A)$. P is a function defined on the power set of S with its range in the interval [0,1], $P : \mathcal{P}(S) \to [0, 1]$.
- *Intersection*: The intersection of sets A and B is the set of all elements in both A and B. Represented as

$$A \cap B := \{x \in S \ : \ x \in A \wedge x \in B\}^{\circ}.$$

- *Union*: The union of sets A and B is the set of all elements in A or B, or both. Represented as

$$A \cup B := \{x \in S \ : \ x \in A \vee x \in B\}^{\circ}.$$

The first major concepts of gambling mathematics are counting principles. These

are the basis for most of gambling mathematics. It is important to be able to count every event when performing calculations. These calculations can be used to find the probability of a player winning a specific hand, house advantage, and much more.

- *Factorial*: Product of all positive integers up to and including n. Thus,

$$n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot (n-1) \cdot n. \tag{1}$$

For example "5 factorial" would be:

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120. \tag{2}$$

Permutations: A selection of r items from a set of n so that order matters. In simpler terms permutation counts the number of ways a group of items can be selected when order matters. The number of ways to chose r items from a set of n can be expressed by the equation:

$$nPr = \frac{n!}{(n-r)!} \tag{3}$$

Example:

Five people finish a race, how many ways can the five people come in First, Second, and Third place?

For this problem John coming in first, Susie in second, and Liam in third is different from Susie in first then Liam then John. This means order matters and permutations need to be used. The total number of elements in the set is n, which in this case is 5, the number of people in the race. The number of elements being chosen is r, which in this example is 3, the number of winners.

These variables substituted into the equation results in:

$$nPr = \frac{5!}{(5-3)!} = \frac{5!}{2!} = \frac{120}{2} = 60 \tag{4}$$

Thus the number of ways five people could come in first, second, or third in race is 60.

- *Combinations*: The number of ways r items can be chosen from a set of n items when order does not matter. A combination can be expressed by the following equations:

$$nCr = \frac{n!}{(n-r)!r!} = \frac{nPr}{r!} = \begin{pmatrix} n \\ r \end{pmatrix} \tag{5}$$

Example:

How many ways can a five person team be selected from 30 people chosen at random?

A team of Susie, John, Liam, Candace, and Hannah is the same as a team of John, Susie, Hannah, Liam, and Candace. Thus order does not matter and combination can be used to calculate.

For this problem, the total number of items, n, is 30 and the number of items being chosen, r, is 5. This expressed verbally by saying "30 choose 5". These can be substituted into the combination formula to result in:

$$\begin{pmatrix} 30 \\ 5 \end{pmatrix} = \frac{30!}{(30-5)!5!} = \frac{30!}{25!5!} = 142,506 \tag{6}$$

This means there are 142,506 ways to choose five people from a group of 30.

- *Multiplication Principle*: If A and B are independent events, then:

$$P(A \cap B) = P(A) \cdot P(B) \tag{7}$$

- *Addition Principle*: If A and B are mutually exclusive events ($A \cap B = \varnothing$), then:

$$P(A \cup B) = P(A) + P(B) \tag{8}$$

5.2 Calculations

When I started, I first calculated probability of a certain player hand winning against the dealer. This worked well for extreme hands, like if the player had the highest or lowest ranked cards. The more in the middle hands, good but not great, were much harder to calculate. This was due to the fact that it was nearly impossible to account for every situation or event. It theoretically could have been done, but would have taken far too long to accomplish in the time we had available. It became clear fairly early on that the computer program would have to be the main focus in order to devise a clear player strategy.

The first calculations were done by hand. An example of the type of calculations that were done in an attempt to devise player strategy can be seen below.

Example: *What is the probability the dealer has at least one trump if you have one?*

- The probability the dealer has AT LEAST one trump is the same as one minus the probability the dealer has zero trumps.

- There are 7 trumps in a deck. The player has one in their hand and one is

displayed as the trump, and thus there are 5 trumps still available. The player also has 2 other cards in their hand, thus there are 15 non-trump cards available.

- For the dealer to have no trumps they would have 0 of the 5 available trumps thus there are five choose zero ways for the dealer to have no trumps or, equivalently, one way. If the dealer has no trumps then they have three non-trumps. This means of the fifteen remaining non-trump cards the dealer has three of them. There are 15 choose 3 ways for that to happen, or 455 ways.

- The total number of ways the dealer could have 3 cards out of the total remaining 20 is 20 choose 3, or 1,140 ways. Thus the probability the dealer has no trumps is $\frac{455}{1140}$. This can be subtracted from one to get the probability the dealer has at least one trump. This is shown in the following equation:

$$1 - \frac{C(5,0)C(15,3)}{C(20,3)} = 1 - \frac{455}{1,140} = .601 \tag{9}$$

- The probability the dealer has at least one trump given that the player has one is about 60%. This information and similar calculations can be used to determine if the player's hand has a good chance of competing with an unknown dealer's hand. If you have low trump you may not want to play given it is likely the dealer will also have a trump.

6 Coding

The coding was simultaneously the most frustrating phase of this project, as well as my favorite. I started the program early on in the research and slowly built on it as the time went on. Overall, I wrote around 1,000 lines of code. The point of the program was to make calculations easier and more accurate, as it eliminates much of the human error. The complex hand calculations were too much for me to do alone, so

it was agreed that a computer program would be built that played the game and could assess hands and other gameplay observations. This was the most frustrating phase due to my lack of computer science knowledge. Prior to my research I had Computer Science I, which prepared me for the basics, but not all of the programming I had in front of me. Overall the experience I gained from working through this code creation greatly improved my computer science skills. There were many hiccups along the way, but through lots of research and tears I worked through them. Some necessary concepts and definitions for computer programming can be seen below:

- *Class*: an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods). [8] In layperson terms it is like a mini program that can be used in a larger program.

- *Objects*: described data type and all the functions that go together with that data type.[9] An object in computer science is similar to that of one in real life. A car is an object and the data that makes up a specific car could be; make, model, year, color, etc.

- *Method*: functions done to objects.[9] In the car example this could be something as simple as displaying the car's model.

The first step in executing the computer program was to create the fundamentals for any card game: the cards and the deck. I found lots of code on creating cards from open source codes and so I used parts of those and adapted them to the euchre deck. These were the classes I started early in the research process. I didn't have much of a plan for how the code was going to play the game, but I knew I would need a deck of cards.

The game was first programmed using three cards and without strategic advice. The goal of this was to get a basic card game playable and work from there. Every

card game can be broken down into a few key areas: dealing, shuffling, players' hands, and gameplay. The code was not written in a linear way from one class to another, it was written cyclically and gradually. Each class was edited each time a new element was added. All of the code, written in java, can be found in Appendix A.

6.1 Card / Deck

The Card class is the shortest class and changed the least, its goal was to create the cards that are used in euchre and present them via words. The cards were broken down into two aspects, rank and suit. Four suits were made, heart, diamond, spade, and club. Six ranks were created, 9-A. This created 24 cards that could be used. I found many classes online for cards, so I utilized aspects of those and adapted it to the euchre deck [10]. The card being displayed in words is a huge feature of this class. The *Override* method coverts the numerical information for each card into a sentence that states the card. Without the *Override* method a Jack of Spades would be displayed as Card(1,2) with 1 indicating spades and 2 indicating Jack. This is confusing for obvious reasons. It would require me to remember the number correlation to each suit and rank. The Override method converts Card(1,2) to "Jack of spades", which is far more intuitive.

The Deck class utilized the Card class to create a deck of cards and perform tasks. I again found the basis for creating a deck online and utilized that with some adaptations [10]. The deck class can make a deck, shuffle, and deal cards to the player and the dealer. The program does not deal in the most efficient way, but it is the way that made sense to me when I made the program initially, and it works.

The deck class works on an array system, a deck is just a list of cards and that is what the basis of an array is, a list of items. In this class cards are created and placed into the deck array via two nested for loops. The inner adds all of the ranks and the outer adds all of the suits. This array can then be manipulated in other methods to

achieve desired results.

A critical part of every card game is shuffling. An un-shuffled deck makes for a very boring and biased game. Essentially, to shuffle the deck the *shuffle* method takes a random card in the deck and moves it to a random location within the deck array. This is repeated an arbitrary 75 times, thus shuffling the deck.

The *deal* method is another important method in the deck class. I chose to deal by giving the dealer three cards, dealing the trump and then dealing the players four cards. This is done by assigning each element individually, so player's first card, player's second card, etc. The variables of the player's cards, dealer's cards, and trump are all assigned cards from the deck starting from the front of the deck and moving down in the aforementioned order. The *deal* method then prints for the player their cards and the trump, keeping the dealer's cards a mystery.

6.2 Main

The main code contains all of the fun. It calls from the deck class and has many methods. This code started off as allowing the player to play a card against the dealer. The code did not decide who won and it would let the player play a card any number of times, which is obviously not allowed in the real game. The code was adapted and expanded to scan the player's hand and suggest if they should fold or play, allowing the player to choose to fold or play. This was done by adding methods. If the player chooses to play they can then choose a card to play against the dealer without reusing cards. The program will also tell the player if they have a card that they have to play in order to follow suit. It also then determines who wins and gives or takes chips accordingly. The fourth card discard was an added part of the code at the very end. This provided the player with four cards and the ability to discard one. This rule was decided early in the research, but was not included in the program until the end. I did not include it earlier on because I wanted to be confident in the code

basics before I figured out how I was going to program the discarding of the fourth card. The cards were divided into three categories for the sake of the code; high, medium, and low. The high cards were Aces and Kings, the medium were Queens and Jacks, and low were nines and tens. This just made for ease of coding, instead of having the program scan and account for six elements it only needed to do three.

An example of a single game played with the program can be seen in Figure 2.

Figure 2

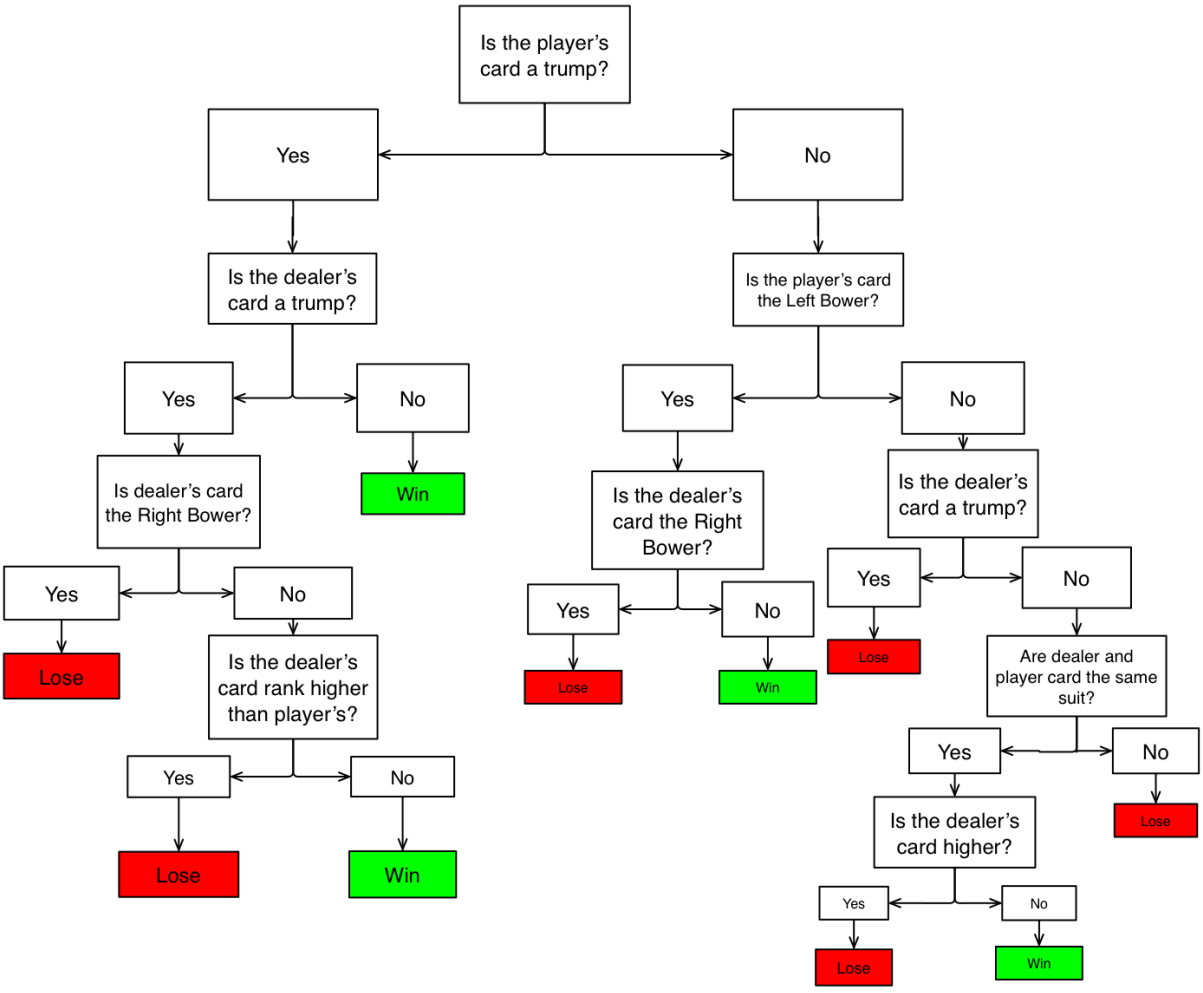
```
Your cards are: King of hearts, King of spades, Queen of diamonds,
The trump card is the: Jack of spades

You have 100 chips
Play this hand. You have two or more trumps.
Would you like to fold? y/n
n
Choose a card to remove (1,2,3,4)
3
Your cards are now: King of hearts, King of spades, 10 of spades
Would you like to place in 3 chips or 6?
3
The dealer's card is the: Ace of hearts

Your cards are: King of hearts, King of spades, 10 of spades
The trump card is the: Jack of spades
You have to play: King of hearts (1)
What card would you like to play against the dealer? 1,2,or 3:
1
You played the: King of hearts

You lose that trick.

The dealer's card is the: 9 of hearts

Your cards are: King of hearts, King of spades, 10 of spades
The trump card is the: Jack of spades
What card would you like to play against the dealer? 1,2,or 3:
3
You played the: 10 of spades

You win that trick.
The dealer's card is the: Jack of hearts

Your cards are: King of hearts, King of spades, 10 of spades
The trump card is the: Jack of spades
What card would you like to play against the dealer? 1,2,or 3:
2
You played the: King of spades

You win that trick.

Win: 2 Lose: 1
You now have 101 chips

Would you like to keep playing? y/n
n

 ----jGRASP: operation complete.
```

The most important methods of the main class are the *contains*, *assess*, *foldSuggest*, and *determine* methods. The *contains* and *assess* methods perform similar tasks. They both investigate the players hand. *Assess* does it in the very beginning of the code to establish what types of cards the player has, i.e high, low, medium, trumps. The *contains* checks to see if the player has the same suit as the card the dealer played and tells the player to play that card.

foldSuggest uses the *assess* method to determine if the player has good enough cards to play. This basically comes down to a trump and at least one high ranked card or two or more Aces. How this was decided will be discussed further in section 6.3. This is where the player strategy comes into play. *foldSuggest* uses ideal player strategy to suggest playing or folding for the player and the player can then decide for themselves whether or not to follow the advice. The *foldSuggest* method goes through many cases to come to its conclusion. For example: How many high cards does the player have? How many medium? Low? How many trumps? Are those trumps high, medium, or low? If there are x number of low cards in the player's hand suggest folding. It goes on to look at many combinations of high, lows, and trumps to decide if folding or playing is the best strategy.

The *determine* method is the longest and most involved method. It compares the dealer's card and the player's card to determine who won. This seems relatively simple, but there are many cases that needed to be examined, including left and right bowers and other trumps. This is done through a series of if else statements and do/for loops. I wrote this method by first making a flow chart to compare the player's and dealer's card. The program follows the flow chard through if/else statements. This flow chart can be seen on the following page in Figure 3.

Figure 3



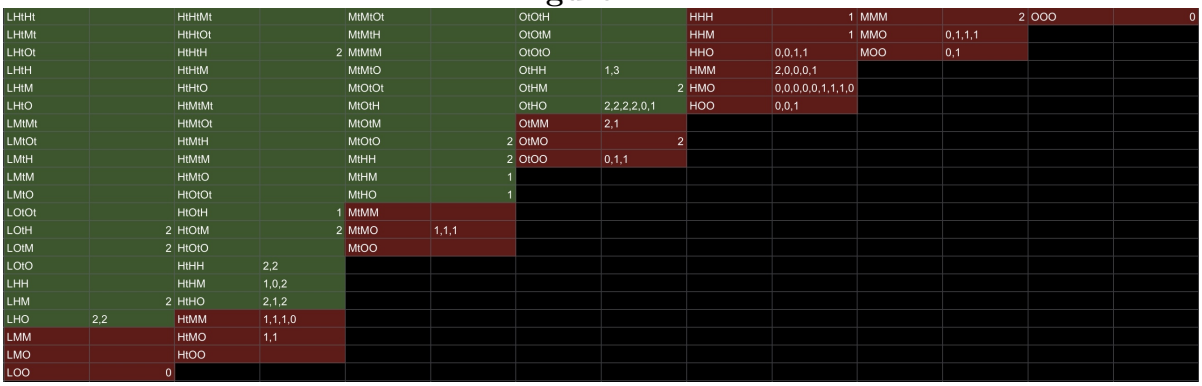
6.3 Player Strategy

The entire goal of the computer program was to help determine player strategy, which hands were good and which were bad. For this reason the *foldSuggest* method was always evolving to fit the current theory of player strategy. As I learned new information about player strategy I changed the code to reflect the new information. Blackjack is a casino game with a very well defined player strategy, if you have less than a certain hand you fold and if you have more you play. There are many charts online that can tell the player what to do given certain dealer hands and other factors.

This is not perfect, as other players cards are not accounted for in the strategy, but it is a pretty solid basis. Casino Euchre didn't seem to have this cut and dry line, but I wanted to find one for a couple reasons. It would make programming the *foldSuggest* method a lot easier, I wouldn't have to account for as many cases. For example it wouldn't have to check for 1 medium trump, 1 low, and 2 medium cards and make a decision, it could just check to see if there was 1 trump and one high card. Also I thought it would make Casino Euchre more successful. Blackjack is one of the most successful casino games in existence and the cut and dry player strategy gives players a sense of confidence in their ability to win, encouraging them to play more.

To determine a strict cutoff point I made a spreadsheet containing all of the combinations of high, medium, and low cards with trumps included. I then played hands using the program and recorded the number of wins (0,1,2,3). One win is required for the player to be equally as well off playing or folding, but two is required for the player to earn more money than they had started with. This is because the ante is one chip and one win and two loses would result in the loss of only one chip which is the same as putting in the ante and folding. The data was collected and a line was drawn on what hands to play and which to fold. The data can be seen below. The red in the graphic is when to fold and the green is when to play. The cutoff was decided by me as a imperfect breaking point. There might be a few more subtleties in the cutoff but based on my data I thought it was a good place to start. This line says to play when the hand has a trump and at least one high card and fold otherwise.

Figure 4



The data collected wasn't as exhaustive as I was hoping, since many hands were not present. I also did not feel confident about dismissing certain hands as I had believed them to be good previously. I thought this required more investigation. To investigate the hands I felt unsure about I had to create another code that allowed me to do many tests on specific hands quickly. This code can be seen in Appendix A in the Test Player Hand section. It is a rough code that is programmed to work, not to be efficient. Through this code I program a player hand and the program plays it against approximately 54 dealer hands and records the wins and losses. It tests 54 hands because the dealer gets 3 cards and each hand it tests rotates out one card replacing it with one of the other available cards resulting in 54 hands. The first hand I wanted to investigate was the player hand of three non trump Aces. This had always played as a good hand in physical hands of the game. When investigated it won 160/270 hands or close to 60%, and while this is not a huge success it is worth playing as the player has a better chance of winning than losing.

The three high cards had only appeared once in the data set I collected and when it did it only won once. Since this made me wonder how the other HHH hands would perform, I next investigated the case of three non trump kings. This hand did not do well and lost more than it won. The extremes of the HHH hands were on different ends of the success spectrum. I wondered about the hand AAK all non trump. After testing this was determined to be a good hand as well, winning more times than

losing on 12 of 15 trials. 15 was relatively arbitrary, I wanted to test the hand many times to make sure the results were consistent and at 15 trials I felt satisfied. AKK was logically the next hand to be tested; when tested, it lost more times than won. This showed that at least two aces or higher needed to be present for the hand to be viable. This is seen as once the hand went from AAK to AKK there was a shift from winning to losing. This created a clear albeit interesting player strategy.

Trump and any high card or at least two Aces is the cutoff for a good hand. The trump and high card cut off was obtained from Figure 4, and the two Aces cutoff was obtained through the further testing mention in the previous paragraph. There are still exceptions to this in terms of the trumps. If a player has the right bower they should automatically play because they are guaranteed to win at least one hand, even though technically they might only have a trump and low cards. There may be more nuances with the left bower that were possibly not accounted for. It only has one card that can beat it, which makes it very good but not infallible. All of these cases are reflected in the *foldSuggest* method seen in Appendix A in the Game with Strategy section.

7 Game Design Art

7.1 Logo Design

A logo is the first thing a person sees of a product and what it says to them will dictate if they consume the product. Every great product has a recognizable logo, and casino games are just that, a product. I wanted to design a logo that could be used on the felt and for general advertisement.

To start drawing the logo I first came up with a concept. I decided on a general layout, color scheme, and plan. These were decided and inspired by an extensive Google image search of casino game logos. I found many casino game logos to be

kitschy and childish, they were sparkly, had a lot of food imagery and bright colors. I wanted to create a more classic timeless logo, with clean lines, bold words, simple imagery, and strategic use of soft and muted colors. I believe this would draw in a more sophisticated crowd, while still appearing youthful and fun to draw in the casual casino attendees. I liked many of the color schemes that successful casino games had they all had a few colors in common; black, gold, and red. The colors covey power, luck, and fortune; all desired themes for a gambling game. Using these colors the player is drawn into the game without even knowing.

In the design I wanted to convey one of the most known and unique aspects of euchre, the right bower. I decided to make the cards present in the logo all of the Jacks, implying the four options for the right bower. This will bring a sense of familiarity to the casino version of this beloved game. I drew elements from different royalty free stock image cards, tracing the parts I wanted and free handing the rest. I combined other image elements along with the stock images to draw the cards in the logo. I wanted the cards to be minimalistic and not distracting while also being recognizable. I wanted the only color to be in the suits so they would stand out more.

I had originally named the game 3-Card Euchre, but it was pointed out to me that that was confusing due to the fact that the player gets four cards initially and four are present in the logo. Due to this I changed the name to Casino Euchre. When renaming the game I considered all of the tips from *Essentials of Casino Game Design*.

I started this when I had very little experience with digital drawing, which added to the amount of time it took me to complete. The logo itself took around four hours to draw and complete. I tried many different variations in lettering, shading, background, and colors. I finally decided to settle on the image seen below.

7.2 Felt Design

Felts are an important part of the casino games. It is the board on which people play the game. I created the felt with the idea of a blackjack table. I wanted clearly marked spaces for the cards and the chips. The players locations are the three cards with the spaces for chips above with up to four players, and the dealer sits above the logo with their cards in the three middle card locations. I also wanted the felt to contain the logo and the general ranking system for players to reference. The final felt design can be seen on the following page.

TRUMP
DISCARD
ANTE
RIGHT BOWER
ANTE
LEFT BOWER
CASINO EUCHRE
A K Q 10 9 TRUMP
ANTE
A K Q J 10 9 NON TRUMP
ANTE

7.3 Rack Card

Rack cards are a form of advertisement that are put in pamphlet racks. It is supposed to convey the game in a small space. I wanted the rules to fit so that people could read and learn the rules of Casino Euchre before even setting foot on the casino floor, thus increasing player confidence and casino earnings. Player confidence and casino revenue may seem in conflict, but they actually tie in together. Every casino game is made so the casino will make money in the long run; if the player is confident in the rules and thinks they can win they will play more resulting in more revenue for the casino. I included the logo for the game on the rack card. I thought this was important as it would make it clear what game the rack card was for and even more importantly make casino attendees recognize the game on the casino floor if they had picked up a rack card. The rankings are included on the rack card for two reasons. The first being that the rankings are the foundation for the game and the trickiest part to understand. The second reason is for those who know regular euchre. The rankings are the fundamental element for the original and adapted versions, thus hopefully providing comfort for those who know the original version, thus drawing them in to play. The rack card is in the color scheme of the casino euchre logo in order to keep things cohesive and to really sell the brand. See the following page for the rack card design.

R U L E S

Start

Player places an ante in to start.

The dealer deals four cards to the player and three to themself.

The player may decide to fold or place in two more chips for a total of three or five chips for a total of six.

The player must then choose one card to discard

Play

The dealer flips their first card. The player must choose a card to play against the dealer, following suit if possible. The Left Bower counts as a trump suit.

The card with the higher ranking wins.

The dealer wins if player can't follow suit or trump.

Repeat for each of the three cards.

R A N K I N G S

J Trump — Right Bower

J Off-Suit Trump — Left Bower

A K Q 10 9 — Trump

A K Q J 10 9 — Non-Trump

8 Next Steps and Conclusion

8.1 Next Steps

While I achieved many of my goals in my research, there are a few avenues I would like to pursue further. I think Casino Euchre would do amazing if adapted as an online casino game. Reneging would be impossible if the game was virtual, as the game could force the player to follow suit. This solves a major problem of Casino Euchre. If given the opportunity I would want to learn how to code gaming interfaces to convert this into either a game for the app store or an online interface in a casino. I played around with a user interface, but this would have required a lot more time and research that I did not have. Instead I focused my energy on devising the optimal player strategy.

I would also like to see an exact house advantage calculated. A feeling of the house advantage can be assessed through just playing the game on repeat. For example the fourth card was added because the player felt as though they were always losing, thus the house advantage was too high. With the adapted rules, the house advantage felt right, it felt like the player could win but the house won in the long run. This was seen through numerous trials, but is still based off of feeling. If I knew the exact advantage I would be able to tell if the rules needed to be adapted further.

One area of the rules I didn't have enough time to dive into is the option to "double up" the bets. I am not sure if this is a good idea for the player in any cases and if it should even be allowed. I would love to investigate how this option actually affects the house advantage and the player's earnings.

Lastly, I would have liked to expand and clean up my code. There are many areas I would have liked to tweak given the time. The player's cards do not get removed from their hand after they play it so it still appears to be a playing option even though the program won't let the player repeat a card. I also would have liked to expand

the player matching suit section. The program suggests to the player if they have a card of the same suit that they have to play it. At this point it does not force them to and will suggest a card that has already been played.

8.2 Conclusion

Overall, it is feasible to turn traditional euchre into a possibly successful casino game. The biggest problem with casino euchre seems to be the inability to monitor reneging, the following of suit. This could easily be fixed by making it a digital interface. After a years worth of research, coding and planning, I come back to my central question: Can Euchre be adapted into a casino game? And if so, would I be able to make it have a playable interface? Through my research of successful casino gaming, I see that Casino Euchre has the main components necessary for a casino game, and, in my opinion, has a chance to make it on the floor. Casino Euchre is easy to learn, has a catchy name, clean simple art, and is based upon a game that a lot of people are already familiar with. Therefore, it is clear to me that I accomplished my goal, and Casino Euchre has the potential to sweep the casino floor.

References

[1] P. b. S. Lock, J. 18, *Casino count US 2019*, **2020**. https://www.statista.com/statistics/187972/number-of-us-commercial-casinos-since-2005/.

[2] D. Parlett, *EUCHRE*. https://www.parlettgames.uk/histocs/euchre.html.

[3] Lock, *US commercial casino gaming consumer spending 2019*, **2020**. https://www.statista.com/statistics/271743/us-consumer-spending-on-commercial-casino-gaming/.

[4] N. Government, *Nevada Gaming Commision Approved Gambling Games Effective March 1, 2021*, **2021**. https://gaming.nv.gov/Modules/ShowDocument.aspx?documentid=7097.

[5] D. Lubin, *The Essentials of Casino Game Design: From the Cocktail Napkin to the Casino Floor*, Huntington Press, **2016**.

[6] E. Jacobson, *Contemporary casino table game design: a practical guide to casino table game design, development and selection for casino management and game developers*, Blue Point Books, **2010**.

[7] M. Bollman, *Basic Gambling Mathematics The Numbers Behind the Neon*, CRC Press, **2014**.

[8] K. B. Bruce, *Foundations of object-oriented languages: types and semantics*, MIT, **2002**.

[9] P. Sengupta, B. B. Chaudhuri, *Object-oriented programming: fundamentals and applications*, Prentice-Hall of India, **1998**.

[10] D. Flemstrom, *Card.java*, **2010**. https://gist.github.com/dflemstr/414869#file-card-java.

9 Appendix A

9.1 Card

```
public class Card
{
  private int rank, suit;
  private String[] suits = { "hearts", "spades", "diamonds", "clubs" };
  private String[] ranks = { "9", "10", "Jack", "Queen", "King", "Ace"};

  public String rankAsString( int __rank )
  {
      return ranks[__rank];
  }

  public Card(int suit, int rank)
  {
      this.rank=rank;
      this.suit=suit;
  }

  public @Override String toString()
  {
      return ranks[rank] + " of " + suits[suit];
  }

  public int getRank()
  {
      return rank;
```

```
    }

    public int getSuit()
    {
        return suit;
    }
}
```

9.2 Deck

```
import java.util.Random;
import java.util.ArrayList;

public class Deck
{
    private ArrayList<Card> deck;
    int index_1, index_2;
    Random generator = new Random();
    Card temp;
    static Card player1;
    static Card dealer1;
    static Card player2;
    static Card dealer2;
    static Card player3;
    static Card dealer3;
    static Card player4;
    static Card trump;
    public Deck(){
```

```
deck = new ArrayList<Card>();
    for (int a=0; a<4; a++)
    {
        for (int b=0; b<=5; b++)
        {
            deck.add( new Card(a,b) );
        }
    }
}
int size;
public void shuffle()
{
    for (int i=0; i<75; i++)
    {
        index_1 = generator.nextInt( deck.size() - 1 );
        index_2 = generator.nextInt( deck.size() - 1 );
        temp = deck.get( index_2 );
        deck.set( index_2 , deck.get( index_1 ) );
        deck.set( index_1, temp );
    }
}

public void displayDeck()
{
    for (Card card : deck)
        System.out.println(card);
}
```

```
public ArrayList<Card> getDeck()
{
    return deck;
}

public void deal()
{
    player1 = deck.get(4);
    player2 = deck.get(5);
    player3 = deck.get(6);
    player4 = deck.get(7);
    dealer1 = deck.get(0);
    dealer2 = deck.get(1);
    dealer3 = deck.get(2);
    trump = deck.get(3);
    System.out.println("Your cards are: "+player1+", "+player2+", "+player3+
    ", "+player4);
    System.out.println("The trump card is the: "+trump);
    System.out.println(" ");
}

public static Card getDealer1()
{
    return dealer1;
}

public Card getdealer2()
{
```

```
        return dealer2;
    }

    public Card getdealer3()
    {
        return dealer3;
    }

    public static int getSuitTrump()
    {
        return trump.getSuit();
    }

  public Card remove(int x)
    {
        return deck.remove(x);
    }
}
```

9.3 Game with Strategy

```
/**********************************
ThreeCardEuchre2.java
Claire Mitchell
Plays 3-card Euchre with strategy
**********************************/
import java.util.Scanner;

public class ThreeCardEuchre2
```

```
{
   String fold;
   String ans;
   String anss;
   int out;
   int trumpcount;
   boolean R= false;
   boolean L= false;
   int highcount;
   int medcount;
   int lowcount;
   int ace;
   int play;
   int gain;
   int x;
   static int playerchips = 100;
   int bet;
   int card1;
   int card2;
   int card3;
   int win;
   int lose;
   Scanner stdIn= new     Scanner(System.in);
   static Deck euchre= new Deck();
   static ThreeCardEuchre2 tce= new ThreeCardEuchre2();

  public static void main(String[] args)
```

```
{
    tce.starting();
}

public void starting()
{
    win = 0;
    lose = 0;
    R = false;
    L = false;
    highcount=0;
    medcount=0;
    lowcount=0;
    trumpcount=0;
    euchre.shuffle();
    euchre.deal();
    tce.start();
    tce.run();
    tce.finall();
}

public void run()// runs the middle portion of the game including all card
selections
{
    for(int i= 1; i<=3; i++)
    {
        dealer(i);
```

```
            contains(i);
            card(i);
            determine(i);
        }
}

public void start() // gives chip stats and calls bet
{
 System.out.println("You have " +playerchips+" chips");
 bet();
}

public void bet()//Initiates betting includes(assess,fold)
{
    assess();
    fold();
    if(fold.equalsIgnoreCase("y"))
            playerchips-=1;
    else
    {
            do
            {
                System.out.println("Would you like to place in 3 chips or 6?");
                bet= stdIn.nextInt();
                    if (bet==3)
                        gain=1;
                    else if (bet==6)
```

```
                    gain=2;
            }while(bet!=3 && bet!=6);
        }
}

public void assess() //Assesses the hand, making note of the types of cards
{
    for(int i=1; i<=4;i++)//Runs through player cards
    {
        if(playerCard(i).getRank() == 2 && playerCard(i).getSuit() ==
        euchre.getSuitTrump()) //Player has R Bower
            R=true;
        if((playerCard(i).getRank() == 2 && tce.check1(i) == true)) //Player
        has L Bower
        {
            L=true;
         }
         if(playerCard(i).getSuit() == euchre.getSuitTrump()
        && !(playerCard(i).getRank() == 2 && tce.check2(i) == true)
        && !(playerCard(i).getRank() == 2 &&
        playerCard(i).getSuit() == euchre.getSuitTrump()) )//Player has a
        trump
        {
            trumpcount++;
         }
        if(playerCard(i).getRank()>=0 && playerCard(i).getRank()<=1) //Player
        has 9 or 10
```

```
                {
                        lowcount++;
                }
                if((playerCard(i).getRank()>1 && playerCard(i).getRank()<=3) &&
                !(playerCard(i).getRank() == 2 && tce.check2(i) == true) &&
                !(playerCard(i).getRank() == 2 && playerCard(i).getSuit() ==
                euchre.getSuitTrump()))//Player has J or Q
                {
                        medcount++;
                }
                if(playerCard(i).getRank()¿3 && playerCard(i).getRank()<=5)//Player
                has K or A
                {
                        if( playerCard(i).getRank()==5) ace++;
                                highcount++;
                }
        }
  }

  public void fold() //Dictates a fold, includes(assess and foldSuggest)
  {
foldSuggest();
        do
        {
                System.out.println("Would you like to fold? y/n");
                fold= stdIn.next();
        }
```

```
        while((!fold.equalsIgnoreCase("y"))&&(!fold.equalsIgnoreCase("n")));
        if (fold.equalsIgnoreCase("y"))
        {
                System.out.println("Would you like to keep playing? y/n");
                ans= stdIn.next();
                        if (ans.equalsIgnoreCase("y"))
                        {
                                playerchips-=1;
                               System.out.println("-------------------------------------------------------------------");
                                tce.starting();
                        }
                        else
                                System.exit(0);
        }
    discard();
    }

    public void foldSuggest() //Suggests whether or not to fold based on hand
    {
        if(euchre.getDeck().get(3).getRank() == 2 && L == true)
                System.out.println("Play. You have highest card.");
         if(R==true)
        {
                if(L == true)
                        System.out.println("Double Down you have left and right bower.");
                else
```

```
            System.out.println("Play this hand.  You have the right bower.");
 }
else if(ace== 3)
        System.out.println("Play this hand you have three aces");
else if(playerCard(1).getSuit() == playerCard(2).getSuit() &&
playerCard(1).getSuit() == playerCard(3).getSuit() && playerCard(1).getSuit()
!= euchre.getSuitTrump())
       System.out.println("Fold this.  All the Same non Trump Suit");
else if (lowcount > 2)
        System.out.println("Fold this this is bad.");
else if(trumpcount==0 && L == false && R == false)
       System.out.println("Fold this hand.  You have no trumps.");
else if(trumpcount>=2)
       System.out.println("Play this hand.  You have two or more trumps.");
else if(L == true)
{
        if(trumpcount > 0)
              System.out.println("Play.");
        else if(highcount >= 1)
              System.out.println("Play.");
        else
               System.out.println("Fold this hand.");
}
else if (trumpcount == 1 && R == false && L == false)
        if(highcount >= 1)
               System.out.println("Play this hand.");
       else
```

```
                System.out.println("Fold this hand");
}

public void discard() //Removes a card of the players choice
{
     do
    {
          System.out.println("Choose a card to remove (1,2,3,4)");
          out = stdIn.nextInt(); euchre.remove(out+3);
     }
    while( out!=1 && out!=2 && out!=3 && out!=4);
System.out.println("Your cards are now: " + playerCard(1) +", "+playerCard(2)
+", "+playerCard(3));
}

public void dealer(int x) //Prints Dealers xth card
{
     System.out.println("The dealer's card is the: " + dealerCard(x));
     System.out.println(" ");
     System.out.println("Your cards are: "+playerCard(1)
    +", "+playerCard(2)+", "+playerCard(3));
     System.out.println("The trump card is the: " + euchre.getDeck().get(3));
}

public void card(int i) //Players card pick against the dealers jth card
{
    if(i==1)
     {
```

```
        do
        {
            System.out.println("What card would you like to play against
            the dealer? 1,2,or 3: ");
            card1= stdIn.nextInt();
        }while(card1!=1 && card1!=2 && card1!=3 );
 x=card1;
 System.out.println("You played the: " + playerCard(card1));
 System.out.println(" ");
 }
 if(i==2)
{
      do
      {
          System.out.println("What card would you like to play against
          the dealer? 1,2,or 3: ");
          card2= stdIn.nextInt();
      }while(card2!=1 && card2!=2 && card2!=3 || card2 == card1);
x=card2;
System.out.println("You played the: " + playerCard(card2));
System.out.println(" ");
 }
if(i==3)
 {
      do
      {
          System.out.println("What card would you like to play against
```

```
                the dealer? 1,2,or 3: ");
                 card3= stdIn.nextInt();
            }while(card3!=1 && card3!=2 && card3!=3 || card3 == card1 ||
          card2==card3);
    x=card3;
    System.out.println("You played the: " + playerCard(card3));
    System.out.println(" ");
    }
}

public void contains(int j)//checks to see if player has matching suits to dealer's
 card
{
     for(int i=1; i<=3; i++)//runs through player cards
     {
          if(playerCard(i).getSuit() == dealerCard(j).getSuit())
          {
               play=i;
               if( play==card1 || play == card2) // if the card hasn't already
               been played
               {
               }
                else
                System.out.println("You have to play: " +playerCard(i) +
               " ("+play+")");
          }
    }
```

```
}

public boolean check1(int i) //Checks to see if player suit is the same color as
trump suit
{
    if((playerCard(i).getSuit()==(euchre.getSuitTrump()+2)))
            return true;
    else if((playerCard(i).getSuit()==(euchre.getSuitTrump()-2)))
            return true;
     else
            return false;
}

public boolean check2(int i) //Checks to see if dealer card i is same color as
trump
{
    if((dealerCard(i).getSuit()==(euchre.getSuitTrump()+2)))
           return true;
    else if((dealerCard(i).getSuit()==(euchre.getSuitTrump()-2)))
            return true;
    else
           return false;
}

public void determine(int j)// j is dealers card number
{
     if(playerCard(x).getSuit()==euchre.getSuitTrump()) //Players card is a
    trump
```

```
{
        if(playerCard(x).getRank()==2) //R Bower
         {
         System.out.println("You win that trick");
         win++;
         System.out.println(" ");
         playerchips+=gain;
         }
        else if (tce.check2(j) == true && dealerCard(j).getRank()==2)
        //Dealer has L Bower and player doesn't have R Bower
         {
                System.out.println("You lose that trick.");
                lose++;
                System.out.println(" ");
                playerchips-=gain;
        }
        else if (dealerCard(j).getSuit()==euchre.getSuitTrump()) //Dealers card
        is a trump
        {
               if(dealerCard(j).getRank()==2) //Dealers card is a RBower
               {
                      System.out.println("You lose that trick.");
                      lose++;
                      System.out.println(" ");
                      playerchips-=gain;
               }
                else
```

```
        {
            if(dealerCard(j).getRank() > playerCard(x).getRank())
            //Dealers rank is higher than player
            {
                System.out.println("You lose that trick.");
                lose++;
                System.out.println(" ");
                playerchips-=gain;
            }
            else
            {
                System.out.println("You win that trick.");
                win++;
                System.out.println(" "); playerchips+=gain;
            }
        }
    }
    else //You have trump and the dealer did not
    {
        if(dealerCard(j).getRank()==2 && tce.check2(j) == true)
        //Dealer has L Bower
        {
            System.out.println("You lose that trick.");
            lose++;
            System.out.println(" ");
            playerchips -= gain;
        }
```

```
                else
                {
                    System.out.println("You win that trick.");
                    win++;
                    System.out.println(" ");
                    playerchips += gain;
                }
            }
        }
        else //Player doesn't have trump
        {
            if((playerCard(x).getRank()==2 && tce.check1(x)==true))
            //Player has L Bower
            {
                if(dealerCard(j).getRank()==2 && dealerCard(j).getSuit()==
                 euchre.getSuitTrump()) //Dealer has R bower
                {
                    System.out.println("You lose that trick.");
                    lose++;
                    System.out.println(" ");
                    playerchips-=gain;
                }
                else
                {
                     System.out.println("You win that trick.");
                     win++;
                     System.out.println(" ");
```

```
                playerchips+=gain;
        }
}
else // Player doesn't have L Bower
{
        if(dealerCard(j).getSuit()==euchre.getSuitTrump()) //Dealer has
        a trump Player doesn't
        {
                System.out.println("You lose that trick.");
                lose++;
                System.out.println(" ");
                playerchips-=gain;
        }
        else
        {
                if(dealerCard(j).getSuit()==playerCard(x).getSuit()) // Dealer
                and Player have same suit
                {
                        if(dealerCard(j).getRank()==2 && tce.check2(j)==true)
                        // Dealer has L Bower
                        {
                                System.out.println("You lose that trick."); lose++;
                                System.out.println(" ");
                                playerchips-=gain;
                        }
                        else if (dealerCard(j).getRank()>playerCard(x).getRank())
                        //Dealers Rank is higher
```

```
                    {
                        System.out.println("You lose that trick.");
                        lose++;
                        System.out.println(" ");
                        playerchips-=gain;
                    }
                    else
                    {
                        System.out.println("You win that trick.");
                        win++;
                        System.out.println(" ");
                        playerchips+=gain;
                    }
                }
                else
                {
                    System.out.println("You lose that trick.");
                    lose++;
                    System.out.println(" ");
                    playerchips-=gain;
                }
            }
        }
    }

public void finall()
```

```
  {
System.out.println("Win: " + win + " Lose: " + lose);
System.out.println("You now have "+playerchips+ " chips");
      do
      {
            System.out.println(" ");
            System.out.println("Would you like to keep playing? y/n");
            anss= stdIn.next();
      }while(!(anss.equalsIgnoreCase("y")||anss.equalsIgnoreCase("n")));
      if (anss.equalsIgnoreCase("y"))
      {
            playerchips-=1;
            System.out.println("-----------------------------------------------");
            tce.starting();
      }
      else
            System.exit(0);
  }

  public Card playerCard(int p)
  {
      return euchre.getDeck().get(p+3);
  }

  public Card dealerCard(int d)
  {
      return euchre.getDeck().get(d-1);
```

```
  }
}
```

9.4 Test Player Hand

```
/**********************************
TestHands.java
Claire Mitchell
tests specific hands against dealer
**********************************/
public class TestHands
{
        static Deck tester= new Deck();
        static Card player1;
        static Card player2;
        static Card player3;
        static Card trump;
        static int win = 0;
        static int lose = 0;
        static int match;
        static int cancel1;
        static int cancel2;
        static Card dealer1;
        static Card dealer2;
        static Card dealer3;

  public static void main(String[] args)
  {
```

```
        win=0;
        lose=0;
        playerHand();
        dealer();
    }

    public static void playerHand() //Shows player hand in this case AKK
    {
        player1 = tester.getDeck().get(5);
        player2 = tester.getDeck().get(10);
        player3 = tester.getDeck().get(16);
        trump = tester.getDeck().get(18);
        System.out.println("Player hand is: " + player1 + ", " + player2 + ", " +
        player3);
        System.out.println("The trump is: " + trump);
    }

    public static void reDeck() // removes player and trump cards from deck
    {
        tester.remove(5);
        tester.remove(9);
        tester.remove(14);
        tester.remove(15);
        tester.displayDeck();
        tester.shuffle();
    }

    public static void auto1()
```

```
{
    if(dealer1.getSuit() == trump.getSuit())
    {
        if(player1.getSuit() == trump.getSuit())
        {
            if(player1.getRank()> dealer1.getRank())
                win++;
            else
                lose++;
        }
        if(player2.getSuit() == trump.getSuit())
        {
            if(player2.getRank()> dealer1.getRank())
                win++;
            else
                lose++;
        }
        if(player3.getSuit() == trump.getSuit())
        {
            if(player1.getRank()> dealer1.getRank())
                win++;
            else
                lose++;
        }
    }
    else
        lose++;
```

```
        match=0;
if(player1.getSuit() == dealer1.getSuit())
        match = 1;
else if(player2.getSuit() == dealer1.getSuit())
        match = 2;
else if(player3.getSuit() == dealer1.getSuit())
        match = 3;
else if(match==0)
{
        cancel1 = (int) (Math.random()*2 +1);
}
if(match != 0)
{
        if(match == 1)
        {
                if(player1.getRank()>dealer1.getRank() )
                        win++;
                else
                        lose++;
                cancel1 = 1;
        }
        if(match == 2)
        {
                if(player2.getRank()>dealer1.getRank())
                        win++;
                else
                        lose++;
```

```
                cancel1 = 2;
            }
            if(match == 3)
            {
                if(player3.getRank()>dealer1.getRank())
                    win++;
                else
                    lose++;
                cancel1 = 3;
            }
        }
    }

    public static void auto2()
    {
        if(dealer2.getSuit() == trump.getSuit())
        {
            if(player1.getSuit() == trump.getSuit())
            {
                if(player1.getRank()> dealer2.getRank())
                    win++;
                else
                    lose++;
            }
            if(player2.getSuit() == trump.getSuit())
            {
                if(player2.getRank()> dealer2.getRank())
```

```
                win++;
            else
                lose++;
        }
        if(player3.getSuit() == trump.getSuit())
        {
            if(player1.getRank> dealer2.getRank())
                win++;
            else
                lose++;
        }
}
else
        lose++;
match=0;
if(player1.getSuit() == dealer2.getSuit())
        match = 1;
else if(player2.getSuit() == dealer2.getSuit())
        match = 2;
else if(player3.getSuit() == dealer2.getSuit())
        match = 3;
if(match==0)
{
        if(cancel1==1)
            cancel2 = 2;
        if(cancel1==2)
            cancel2=3;
```

```
        if(cancel1==3)
                cancel2=1;
}
if (match != 0)
{
        if(match == 1 && cancel1!= 1)
        {
                if(player1.getRank()>dealer2.getRank())
                        win++;
                else
                        lose++;
                cancel2 = 1;
        }
        else if(match == 2 && cancel1!= 2)
        {
                if(player2.getRank()>dealer2.getRank())
                        win++;
                else
                        lose++;
                cancel2 = 2;
        }
        else if(match == 3 && cancel1!= 3)
        {
                if(player3.getRank()>dealer2.getRank())
                        win++;
                else
                        lose++;
```

```
                cancel2 = 3;
            }
                else
                    lose++;
        }
}

public static void auto3()
{
    if(dealer3.getSuit() == trump.getSuit())
    {
        if(player1.getSuit() == trump.getSuit())
        {
            if(player1.getRank()> dealer3.getRank())
                win++;
            else
                lose++;
        }
        if(player2.getSuit() == trump.getSuit())
        {
            if(player2.getRank()> dealer3.getRank())
                win++;
            else
                lose++;
        }
        if(player3.getSuit() == trump.getSuit())
        {
```

```
            if(player1.getRank()> dealer3.getRank())
                win++;
            else
                lose++;
        }
}
else
        lose++;
match = 0;
if(player1.getSuit() == dealer3.getSuit())
        match = 1;
else if(player2.getSuit() == dealer3.getSuit())
        match = 2;
else if(player3.getSuit() == dealer3.getSuit())
        match = 3;
if(match != 0)
{
        if(match == 1 && cancel1!= 1 && cancel2!=2)
        {
            if(player1.getRank()>dealer3.getRank())
                win++;
            else
                lose++;
        }
        else if(match == 2 && cancel1!= 2 && cancel2!=2)
        {
            if(player2.getRank()>dealer3.getRank())
```

```
                win++;
            else
                lose++;
        }
        else if(match == 3 && cancel1!= 3 && cancel2!=3)
        {
            if(player3.getRank()>dealer3.getRank())
                win++;
            else
                lose++;
        }
        else
            lose++;
    }
}

public static void dealer()
{
    reDeck();
    for( int i= 0; i<=17; i++)
    {
        dealer1 =
        tester.getDeck().get(i);
        dealer2 =
        tester.getDeck().get(i+1);
        dealer3 =
        tester.getDeck().get(i+2);
```

```
            System.out.println("Dealer hand is: " + dealer1 + ", " + dealer2
            + ", " + dealer3);
            auto1();
            auto2();
            auto3();
            System.out.println("Wins: " + win+ " Losses: " + lose);
        }
        System.out.println("Wins: " + win+ " Losses: " + lose);
    }
}
```