# BlackJack

Robert Dale

ICT 362 Fall 2018

# Table of Contents

# The Game

This project attempts to implement the popular card game known as BlackJack in a way that can be visually represented through a GUI interface. For anyone unfamiliar with BlackJack, the rules are very simple. The dealer draws from a deck of cards and disperses them around the table. Cards are assigned a point value from 1-11 points and the goal is to get to (or as close to) 21 as possible. The dealer keeps one of his cards face down and only reveals it when every player has either stayed or busted at the table. As a player, a sample hand might go something like this:

1. You have a King (10 points) and a 2 of Clubs (2 points) adding up to a total of 12 points.
2. The dealer has revealed his card and he has a 7 of Hearts (7 points).
3. It's now your turn to decide if you want to Hit (Have the dealer deal you another card) or Stay at your current total.
4. You still don't know what the other card the dealer has but you have pretty safe odds hitting in this scenario and not busting (going over 21).
5. You decide to hit and the dealer gives you a 7 of Spades and your total jumps to 19 points and you decide to Stay.
6. The dealer flips his second card over and reveals it's a 3 of Hearts and he now has to decide to stay or hit. He needs to beat or match your 19, so he'll probably hit and draw another card.
7. The dealer draws an Ace of Diamonds (Worth either 1 or 11 points) and now has a total of 21. You've lost the hand.

The core concept of BlackJack is simple to understand and to grasp. The only curveball is that an Ace can count as either an 11 or a 1 depending on what is most beneficial to you at the time. This allows for some variability in the outcomes. Cards like the Jack, Queen, and King are all worth 10 point values.

I broke down the problem into a few separate areas that would need to be addressed in this program in order to implement the project successfully. If we think in terms of objects and what we have in a BlackJack game we can identify the areas that would need to be separated out from the main body of code. There is a player, there is a dealer, there is a deck (six to eight decks if you're playing with Las Vegas paradigms), and each player has a hand with their current cards. There are other things that can be considered objects though, such as an Artificial Player in the game and the logic behind whether they would hit or stay. This is tied in with the dealer as well so we can attach a base AI to him as well. Now that I've identified how the code needs to be separated, I can start building out java classes to handle the complicated logic in small bitesize pieces.

The Main class is simple and sets up the frames and the buttons that I need for the game. I want a text box to display certain key pieces of information but also wanted a way to visually represent the cards that the player is getting when he is dealt a hand. For that I just did a simple GridLayout and added the panels to the frame. I borrowed function methods from a previous assignment where I fed in button parameters to a creation method and created an actionlistener for each button. On the creation of my ActionListeners I instantiated my Player and Dealer classes to handle how the buttons would react when they are clicked.

The buttons themselves do the following:

- Deal – Resets the hand of the dealer and the player and deals out cards to each player.

- Hit – Checks the hand of the player and assigns a card if the current slot is not null. At the end, I check to see if the player has bust or not.
- Stay – Continues the logic as the dealer whether to stay or to hit based on the players' current total.
- Quit – Exits the game

The Dealer class creates a new dealer, who creates a new deck by merging together six decks and then shuffling the cards together for a master deck. I deal cards through the deal method that just checks if the top card in the deck (the deckposition) is the last card in the pile and if it is reshuffles the deck and starts drawing from the top of the deck again.

The Deck class instantiates suits and ranks and attaches them into each deck which is a string array. I was able to find a list of cards as png from a deck company and was able to implement them into the game as 9H for the 9 of Hearts and etc. Using this, I was able to draw the cards simply by retrieving the card string and associating that to the .png file. The shuffledeck method takes in the number of decks to be shuffled and mixes them up using Math.random and some temp variables.

The Hand class is just a hand of six hard-coded string variables. There's a better way to do this and this is one of the areas that I would focus on if given extra time but for now it works since the player is highly unlikely to draw more than 6 low point value cards. But it can happen and it would probably break the game, so the Deal button resets the hand and re-deals cards.

## The Class

I found the class to be pretty enjoyable and the assignments to be interesting in the sense that there was so much freedom to implement things the way you felt would work the best. I'm graduating with a bachelor's in CS this semester and the style of the course was completely different than what I'm used to since the CS curriculum focuses so heavily on the foundations of CS in general and not much on practical applications for coding. While some CS majors would see this course as an easy A, I took the opportunity to make the projects a little harder on myself where I could and really tried to apply what I learned there into the assignments here. All of this was made possible by the fact that the class is very free-form and just provides some simple instructions on what needs to be accomplished. Personal opinion aside, I feel that some kids may not benefit from this style as much as I did and so their opinion could be completely different than my own but no less valid. Those students may benefit more from a class with more structure in it but I felt that this class was awesome in that it's also similar to how coding in the real world works most of the time. Requirements and Specifications may not be clear cut and there is not a right or even a good answer to the problems a developer may face.

## Further Implementation and Readings

In the event that someone wanted to pick this project up and continue working on it, you can download the source code from my GitHub account located here (https://github.com/rdale28/blackjack). Parts that still need implementation would be to add the images as the cards are dealt. I wasn't able to fully implement this feature and the program would overwrite the cards in the players current hand and display other cards which wasn't helpful for the UI in general. I opted to leave the code in the hierarchy but removed any trace of image printing from the game. In retrospect, if I had worked only on the game aspect and then moved to implement the artificial player functionality I may have been able to finish. But instead, I have no functioning AI other than rudimentary dealer AI and I don't have the completed BlackJack game with image visuals. Part of the Machine Learning would implement card counting as a strategy to beat the dealer. Tracking metrics over thousands of hands should show whether or not the AI has learned to count the cards effectively.

The next stage would be to implement current strategies to counter this used by casinos to see if the algorithm was capable of learning any particular way to beat the odds.

Further reading about how to count cards can be found here (https://www.888casino.com/blog/card-counting-trainer). I would recommend reading the strategy behind card counting if you were planning on continuing this project. Another recommended reading would be to look at statistical analysis and the odds of busting your own hand or the dealer's by following the pseudocode here (https://www.analyticsvidhya.com/blog/2017/04/flawless-winning-strategy-casino-blackjack-data-science/) and attempting to implement the analysis into a decision structure that the AI can use to win more often than not. Another possible implementation would be to add money into the betting pool. The AI might not win more than 50% but betting larger and consistently winning larger bets might indicate that the algorithm learns when to bet large or to bet small based on the odds.

## Conclusion

I wasn't able to implement the pseudo machine learning algorithms that I originally wanted to implement in the game but that is something I will continue working with and doing outside of the class. I am interested to see if I can make a robot learn how to count cards by repeated iterations and if I can store meaningful results from the AI. That being said, this should be a mostly functioning BlackJack game that someone can play to kill time or whatever but still might have some bugs and definitely has some areas that I could have implemented better. I was able to draw the cards but wasn't able to attach the draw onto a button so when you dealt out cards it would show you your card. I was able to turn the project into the barebones of a BlackJack game that can be improved upon going forward and is something I would like continue in my spare time to see if I can get it fully functional.

## Resources

**Card Counting Trainer**
Henry Ph.D - https://www.888casino.com/blog/card-counting-trainer

**Creating a Flawless Winning Strategy in a Casino (blackjack) Using Data Science**
Tavish Srivastava - https://www.analyticsvidhya.com/blog/2017/04/flawless-winning- strategy-casino-blackjack-data-science/

**BlackJackGUI.java**
http://faculty.washington.edu/moishe/javademos/blackjack/BlackjackGUI.java