Blackjack Simulation

# 

[**Overview**](#_ejv8ad9ckpbe) **2**

[**Assumptions:**](#_r7qwp3gvi7de) **2**

[**Global.py**](#_fr9sotcyu1vk) **3**

[**Card.py**](#_6wu11xdusmdm) **3**

[**Dealer.py**](#_zfsdlbkrzh8g) **3**

[Refresh()](#_d8ec2bvdrs8g) 3

[collectChip()](#_r8wabrs1v8cc) 4

[checkLengthCard()](#_g0hl348nrq0x) 4

[hit()](#_743nvvyqn3ft) 4

[deal()](#_f21chpfd4ziu) 4

[play()](#_s74sqb8prl30) 4

[**Player.py**](#_z1uk8uxem1ql) **4**

[hit()](#_9gaubx768sin) 5

[play()](#_v2lfvc8farsk) 5

[maketheBet()](#_vrun16rtnaqe) 5

[playNormal()](#_bi6y286mtdxr) 5

[playWithOdds()](#_2lr3wuvi5o0p) 5

[playRandom()](#_ru7f7f2bkn5u) 6

[**Visualize.py**](#_of0pr37fkcb) **6**

# 

# 

# 

# Overview

This program is used to model a real blackjack card game. Below are the classes that make up of this program and a brief description of each class:

* Global.py: This class is used to hold the constant variables that will affect the simulation such as number of players, number of decks
* Visualize.py: This is the driver class of the program. This class contains the plot that show the statistics of the program
* Player.py: This class represent a real player with different ways to play and attributes such as number of chips,...
* Card.py: This class represent a deck of the game. It will also allow basic operation like draw of shuffle
* Dealer.py: This class represent a dealer of the game. This class allow the basic operation like dealing cards to player

Here is how all of them interacts: The dealer will use the card class to represent the deck to distribute the card to the Player. Each Player will have in common a dealer in order to request a card. The Visualize class will contain both the Dealer and Player and will use the constants in the Global class to create the plots

GitHub Link: https://github.com/RilThunder/CSS-458-In-Class

# 

# Testing

For the simulation, our testing was incorporated along the way in an agile way in which we test each method by printing to the console and make sure they work before we move on. For example, we tested the function draw in the main method by creating a prototype class and call the draw method to make sure we received back a card.

Thus ,we do not have a formal JUnit testing. Another factor that affect this is because our simulation involves random probability as well so there is no determined way to test

# Assumptions:

Here are the assumptions that our simulation will make:

* The Dealer will hit until he/she reaches 17 or more
* There are 2 ways to play in the Dealer class. Which is called soft 17 and hard 17. Basically, hard 17 means that if there is an ace in the dealer class, it will behave as an 11. Soft 17 means that the ace can be treated as a 1 so that the dealer can hit more
* There are 3 ways to play in the Player class. The Player can play normal, play with odds or play Random. Play Normal is like playing in a predefined way like the dealer. Hitting until he/she reaches 17. Playing Odds is like using probability to determine the best course of action. Playing random is like flipping a coin (random walk)
* The Player class does not have a way to play split. In this simulation, we are more focused on using 3 different ways to play and compare their statistics
* Ace will be represented as 11. When drawn upon, the player/dealer will have a chance to convert it back to 1

# Global.py

The Global.py contains the constant for the simulation. Inside the class, the player can change some of the attributes to see how the simulation will change such as letting the Dealer use the soft 17 rules. Or changing the probability the player will double, hit. Changing how many games the player can play and how many rounds for each game. Changing the number of decks, the minimum bet, maximum bet and starting chips.

# Card.py

This represent the total cards that the Dealer will use. Note that a deck contains 52 cards. This class will depend on the number of deck it will use. So it will have a total of 52 \* number of deck cards.

Upon initialization, it will add integer values to represent a deck. Note that we will not have club,diamond,... but all of them will be represented as integer. So in 1 deck, we will have 4 number 1, 4 number 2,...And depending on the number of deck, we will continue to add more cards

The class also have a shuffle method that utilized the numpy package and a draw method that draw the first card from the card.

# Dealer.py

This represent a real dealer in a Blackjack game. The dealer will have a list of Player that the dealer will handle.The dealer also have booleans variable that represent if the dealer stand or busted. And a variable of Card class to represent the cards

Here are the methods in the Dealer class

## Refresh()

This is like clear method. It will reset whatever cards the dealer is having at the moment and the status of the dealer . This method is called at the end of each round

It will also reset each player as well

## collectChip()

This method is called at the end of each round as well. It will check whether the player wins or lose in order to increase/decrease the number of chip in the player. It will also increase/decrease player’s confidence level depending on winning/losing as well (Confidence level affects how many chips that player bets at each round)

## checkLengthCard()

This method is used to make sure that the player have enough cards for himself + the players. Basically, it get a new Card variable with the old number of decks

## hit()

Get another card and add to the dealer’s hand by using the deal method

## deal()

This method is used to distribute the cards to the dealer/player at the beginning of each round (2 cards for each). It is also used to distribute new cards to the dealer/player whenever they require it

## play()

This method represent the sequence of actions the dealer will play. The dealer will continue to draw cards until he/she reaches greater than or equal to 17. The method also change the status of the Dealer whether the dealer busted or not as well, Note that depending on the boolean variable in the Global.py class, the dealer will play either soft 17 or hard 17

# Player.py

This class represent a real player in the blackjack game. It is similar to the dealer where it will have the number of cards in the hand, the status whether the player stand or bust. In addition, the player will have the total chips the player is having plus the amount of chip the player bet for each round. A confidence level variable that affect how the player will bet and a choice to play: play normal, play odds or play random

## hit()

Requesting the dealer to give the player one more card

double()

This method is called when some condition is satisfied that represent a double in the blackjack game. It will double the amount of bet, getting 1 more card (hit) and stop .

## play()

Play the round until either the player stand or busted. This play method depend on the way the player will play and call different methods to accommodate that

## maketheBet()

Making the bet for this round by using the confidence level. The higher the confidence, the more the player will bet. And vice versa. Whenever the bet get below or above a certain limit denoted in the Global class, reset the bet to that limit

## playNormal()

This is similar to the play method in the Dealer class where the player will draw cards until they reach 17 or greater.Then the player will stand. The only difference is that the player will call double() when the cards in in the range 9 and 11 since those are the best places to double. In addition, if the total card reach over 21, then the player will bust. However, if the player has an 11 in their hand then the 11 will be changed to 1, preventing the player to bust

## playWithOdds()

Play with odds is playing in an optimal way. Basically, the player will count how many cards left the player need to have in order to reach 21. We are assuming that the player have memory of what cards have been played => hence knowing the remaining cards. From that the player will be able to calculate how many cards that satisfy the requirements. Hence calculating the probability that the card will be drawn. Only if the probability satisfy a certain condition, then the player will continue to play and hit. Else the player will stop

## playRandom()

This play method is like random walk. In which we generate a random probability of either hitting or staying and compare that to the probability in the Global class. If that probability is greater than, then we will hit/stand

Note that all of these play style all enforce checking if the player bust or not. Or if the player decided to stand

# Visualize.py

This is the main driver of the program. It will create multiple plots to show the statistics of using different way to play or the number of win. In addition, it also allows to write to a text file the entire process of how the players are doing as well. It also allows the detail process of 1 round in 1 game for more details

Some plots that are available are: Pie charts that show the win/loss ratio when using different play methods, line plots of the number of chips after each game, number of wins versus number of rounds

In the main driver, the simulation will happen n number of times with m number of rounds as specified in the Global class. At the same time, it will print to the console the results and writing to a separate text file. Another method called playRound() allows detail information of what cards each player has initially, the card the dealer has and the result of 1 single round in 1 single game. The configurations can be changed in Global class