jackblack: A blackjack game

COOPER LYNN

CANDIDATE NUMBER: 5501

CENTRE NUMBER: 16115

Contents

# Analysis

[Analysis 1](#_Toc103348852)

[Project Problem 3](#_Toc103348853)

[Research 3](#_Toc103348854)

[Rules of basic Blackjack 3](#_Toc103348855)

[5 Card Charlie 4](#_Toc103348856)

[Insurance 4](#_Toc103348857)

[House Advantage 4](#_Toc103348858)

[Player Advantage 4](#_Toc103348859)

[Counting 5](#_Toc103348860)

[Identifying Covered Cards 5](#_Toc103348861)

[Shuffle Tracking 5](#_Toc103348862)

[Different types of Blackjack 6](#_Toc103348863)

[American Blackjack/Classic Blackjack 6](#_Toc103348864)

[European Blackjack 6](#_Toc103348865)

[Vegas Strip Blackjack 6](#_Toc103348866)

[Blackjack Switch 7](#_Toc103348867)

[Blackjack Perfect Pairs 7](#_Toc103348868)

[Spanish 21 7](#_Toc103348869)

[Programming languages 7](#_Toc103348870)

[Python 7](#_Toc103348871)

[C# 8](#_Toc103348872)

[Java 8](#_Toc103348873)

[VB.net 8](#_Toc103348874)

[AI In blackjack 8](#_Toc103348875)

[What is an AI? 8](#_Toc103348876)

[Hardware 9](#_Toc103348877)

[Target Audience 9](#_Toc103348878)

[Online Blackjack 9](#_Toc103348879)

[Objectives: 10](#_Toc103348880)

# Documented Design

[File Organisation 13](#_Toc103348881)

[GUI plan 15](#_Toc103348882)

[Background 15](#_Toc103348883)

[Menu 16](#_Toc103348884)

[General Flowchart 18](#_Toc103348885)

[Action Flowchart 19](#_Toc103348886)

[Dealer Action Flowchart 19](#_Toc103348887)

[AI Action Flowchart 20](#_Toc103348888)

[Non – Counting 20](#_Toc103348889)

[Counting 21](#_Toc103348890)

[Player Actions 22](#_Toc103348891)

[Randomness 22](#_Toc103348892)

[Data Storage 23](#_Toc103348893)

[Deck and Inventories 23](#_Toc103348894)

[Bets 23](#_Toc103348895)

[Class Diagram 24](#_Toc103348896)

[Hierachy Diagram 25](#_Toc103348897)

# Technical Solution

[Python Code 26](#_Toc103348898)

# Testing

[Test Tables 99](#_Toc103348899)

# Evalutation

[How well did I accomplish my objectives? 115](#_Toc103348900)

[Future plans 119](#_Toc103348901)

[Bibliography 119](#_Toc103348902)

[Research 119](#_Toc103348903)

[Images 120](#_Toc103348904)

Analysis

# Project Problem

Blackjack is a game that has been around since the beginning of the 17th century, going by many different names and branching off into many different variants while still having the core rules of blackjack. However, I wanted to create a version of Blackjack against an AI dealer with 2 other AI who are also playing the game which have varied difficulties by using their own separate rules to determine what move they choose is.

I am choosing this as my project because I find it hard to find blackjack games that don’t require transaction of real life currency and also blackjack games without any high stakes. It could also help attract people away from gambling real money and gamble on this game instead meaning there is no penalty to losing in this game.

# Research

## Rules of basic Blackjack

Each player is given 2 cards from either a single, double, 4, 6 or 8 deck of cards after all Joker cards are removed. The cards each have their own value, with all number cards having the same value as their respective number and face cards having a value of 10. However, an ace card can either have the value of 11 or 1 depending on the choice of the player. If the player or dealer is to have both a face card and an ace card that would cause a blackjack, which would be an instant win unless a push occurs, this would result in a pay out of “3 to 2”.

The aim of the game is to reach a score of 21 or closest towards without going over (bust). If the player is to bust, then they would lose their bet they have put down. Also, the house/dealer will be playing and if they score higher (without busting) then the player will lose as well. However, in most varieties of blackjack dealers must play with specific rules, such as they must hit until their total score is over 17, or that the dealer cannot double down or split their deck.

The dealer will be given 2 cards as well, however the second card must be placed face down and can only be checked if the dealer has a face card for his first card. If the dealer has a blackjack all players who don’t have blackjack as well will lose the round.

Two special actions players can take is “split” and “double down”. Split is available directly after the player is given their first two cards and they have the same value (e.g., a three of hearts and a three of clubs). When a player splits, their deck is split into two piles and they double their bet, with the original bet going to one pile and the new bet going to the other pile, they essentially play with two piles of cards, potentially allowing for them to double their earnings. Double down is when a player will double their bet and take only one card. After this they are unable to hit again and must stand until the round has ended.

## 5 Card Charlie

5 card Charlie is a rule that has been removed from most casinos because it favours the player rather than the house. The rule is if the player was to receive 5 cards without going bust, they would automatically win the round.

## Insurance

Some games of blackjack allow for insurance of their bet. This is done when a dealer has an ace card shown face up from the dealing. The players are then allowed to bet if the dealer has a blackjack (a face card/10 card is the card that’s not shown). The bet for insurance is usually half the player’s main bet, allowing the player to break even if the dealer has a blackjack. If the dealer does not have blackjack, then that bet is lost to the dealer.

## House Advantage

The House (dealer) usually has the higher advantage of winning compared to the players, depending on the rules that are followed. For example, the most popular number of decks to be played while playing blackjack is 6 decks, which gives the house an advantage of 0.64%, which is much higher than its 0.17% advantage with only one deck.

Another reason is that the player can bust before the dealer plays his hand, meaning that a loss occurs to the player no matter what the dealer’s hand is. Therefore, the player is at an immediate disadvantage as even if the dealer goes bust, the player has already lost their bet.

However blackjack has the lowest house advantage compared to other games such as poker or roulette.

## Player Advantage

There is multiple things the player can do to try increase their advantage over the dealer. One of the most popular strategies is to “count the cards”.

### Counting

Card counting is used by the player to give the player a house edge of up to -2 %. It is done by counting and remembering what cards have been dealt from the deck and adjusting their bet around that. For example, if the deck has been played and little amount of face and 10 cards have been played, it might be in favour for the player to double down rather than to hit due to the increased chance of them getting a better hand.

There are multiple strategies on how to count cards, however all of them share similarities between each other, such as they all use a scoring system, with high cards lowering score and lower cards increasing the score.

Card counting itself is deemed legal but is supported within casinos, forcing you to either stop card counting or leave the casino, potentially even banning you from the casino. However, using an external device to count the cards is illegal within the United States and United Kingdom

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Card Strategy | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10, J, Q, K | A | Level of count |
| Hi-Lo | +1 | +1 | +1 | +1 | +1 | 0 | 0 | 0 | -1 | -1 | 1 |
| Hi-Opt I | 0 | +1 | +1 | +1 | +1 | 0 | 0 | 0 | -1 | 0 | 1 |
| Hi-Opt II | +1 | +1 | +2 | +2 | +1 | +1 | 0 | 0 | -2 | 0 | 2 |
| KO | +1 | +1 | +1 | +1 | +1 | +1 | 0 | 0 | -1 | -1 | 1 |
| Omega II | +1 | +1 | +2 | +2 | +2 | +1 | 0 | -1 | -2 | 0 | 2 |
| Red 7 | +1 | +1 | +1 | +1 | +1 | 0 or+1 | 0 | 0 | -1 | -1 | 1 |
| Halves | +0.5 | +1 | +1 | +1.5 | +1 | +0.5 | 0 | -0.5 | -1 | -1 | 3 |
| Zen Count | +1 | +1 | +2 | +2 | +2 | +1 | 0 | 0 | -2 | -1 | 2 |
| 10 Count | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | -2 | +1 | 2 |

Taken from - https://en.wikipedia.org/wiki/Card\_counting

### Identifying Covered Cards

An easy way of identifying cards that are face down is by the memorising the marking and wear on the back of cards, allowing for the player to judge their next move. So potentially this could also tie into counting as it can be used to see what card has been given to the dealer even if it is face down or what card the player would receive if they were to hit or double down.

### Shuffle Tracking

A much harder way of being able to increase player advantage is shuffle tracking, which involves looking at groups of cards (referred to as slugs, clumps or packs) and choosing your play depending on when said groups of cards are in play. This is of course a very hard method as you must be able to track the cards going at a very high speed and depending on the dealer it could be even harder; however, it is much harder to detect by the house.

## Different types of Blackjack

### Amscan 255570 Casino Blackjack Felt Tablecover 0.9m x 1.8m - 1 Pc : Amazon.co.uk: Toys & GamesAmerican Blackjack/Classic Blackjack

Figure Classic Blackjack Felt Table cover

This is the most popular version of the game; this is where the dealer will take his hole (hidden) card before the players have their go. If the shown card is an ace or a face/10 card, then the dealer can peek at the hole card to see if the dealer has a blackjack. There are no restrictions for doubling down in this game mode and splitting can be done up to 3 times by any pair of cards that aren’t an ace, which can only be split once.

### European Blackjack

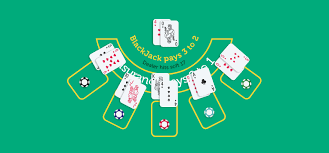
The dealer is unable to get their hole card before the players have their go. There are also more restrictions compared to classic blackjack, such as dealer is unable to look at their hole card until the players have played their hand before being able to check if their hole card allows for a blackjack. Doubling down and splitting is also restricted, such as only being able to double down at values of 9, 10 and 11. Splitting can only be done once and only with pairs of 10s or face cards

Figure European Blackjack table layout

### Vegas Strip Blackjack

This is a very popular game mode for blackjack, being found commonly on online casinos. The rules involve using 4 decks, and has almost the exact same rules as American Blackjack, however the dealer can only peek at his hole card if the card being shown is an ace card

### Blackjack Switch

Blackjack Switch is a version of blackjack allowing for players to place identical bets on two hands and the ability for the player to switch the top two cards between the hands from the start of the round. “Natural Blackjacks” (Pulled straight from deck, face/10 card and an ace) only pay 1:1 in this game mode compared to the usual 3:2. Also if the dealer lands on a 22 then the whole game is a push unless player owns a natural blackjack.

Figure Blackjack Switch Table

### Blackjack Perfect Pairs

Blackjack Perfect Pairs is an optional side bet on the first two cards that are given to the player. There are 3 ways to win the side bet, getting a:

* Mixed Pair = Same number/face value, different suit, different colour
  + Pay-out is 5/1
* Colour Pair = Same number/face value, different suit, same colour
  + Pay-out is 10/1
* Perfect Pair = Same number/face value, same suit, same colour
  + Pay-out is 30/1

### Spanish 21

Spanish 21 is played with 48 card Spanish deck instead of standard deck (standard deck can be used, however must remove all cards with a value of 10 that are not face cards). The dealer must stand on 17 however some variants of Spanish 21 make the dealer hit on a soft 17. Natural Blackjacks cause an automatic win and is paid 3:2 regardless of what the dealer owns. Also, if a player manages to get 21, are paid regardless of if the dealer also gets 21 later in the game.

## Programming languages

Programming Languages are the tools that programmers use to write instructions for a computer to follow. This A level NEA project allows me to use a variation of programming languages.

### Python

This is the language I am most comfortable using due to spending most of my education with this programming language. I have been using this language for multiple years now making it a strong choice in language to choose.

### C#

I am not very experienced with this programming language, however if I was presented with a C# code I would probably still be able to understand it as I have seen a few videos featuring this language in the past. This wouldn’t be a strong option for me to make as my programming language.

### Java

I am very inexperienced with this programming language. This would not be a very suitable choice for me.

### VB.net

I am very inexperienced with this programming language. This would not be a very suitable choice for me.

I am choosing my programming language to be Python. I am choosing this language as it is the programming language I have the most experience with. However other programmers have also made programs for blackjack using different programming languages, such as Java, C# or VB.net, however these programming languages don’t suit my preference.

## AI In blackjack

### What is an AI?

Artificial Intelligence, which is also known as AI, is intelligence which has been simulated via machines. There are multiple different forms of artificial intelligence, but this can be condensed to 4 mains categories:

* Reactive
  + This AI has no memory and only responds to different stimuli
* Limited memory
  + This AI uses memory to help improve its responses to stimuli
* Theory of mind
  + This AI understands the need for other intelligent entities
* Self-aware
  + This AI has an expanded intelligence to be on level with a human and is aware of its existence

I’m going to attempt to use both a reactive and limited memory AI for my blackjack game, as the reactive AI can be used for the basic strategy for blackjack and for the dealer. Whereas limited memory would be useful for AI when card counting has been unlocked for the AI that game. I can use the memory of the AI to determine the size of its bets, and whether it should hit at certain scenarios.

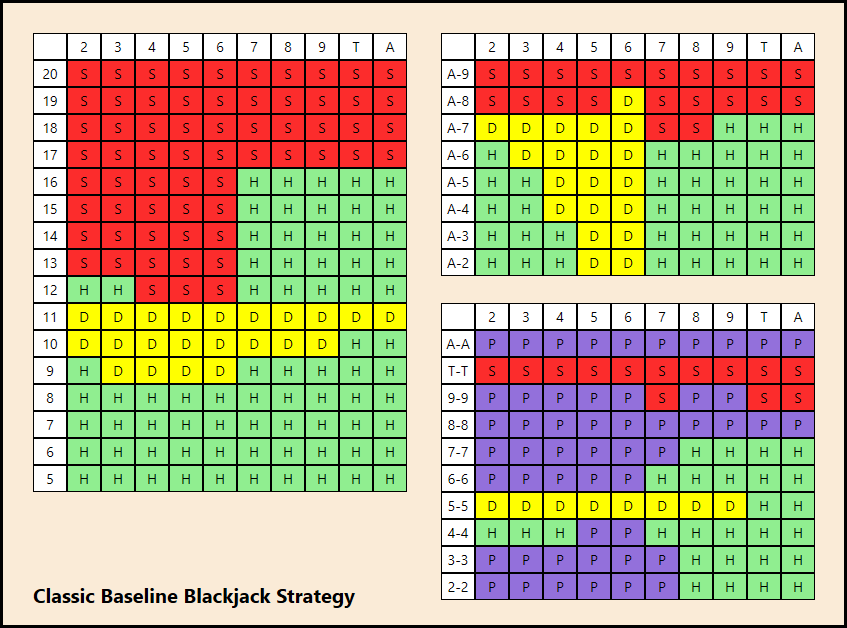


Figure Optimised Blackjack Strategy

I will also need an AI for the dealer, but this will be a very basic reactive AI as this needs to follow a certain set of rules like standard games of blackjack, such as American Blackjack.

## Hardware

To be able to run this software you will need a computer with a monitor of 1800x720 otherwise the canvas and images will be cut short. Also, will not be able to work on mobile devices.

# Target Audience

My target audience is people who like to gamble however are afraid or don’t like the idea of losing money and people who are simply bored and want a game to play to without any stakes and to pass time with a goal to attempt to reach.

# Online Blackjack

Blackjack is found very frequently across many games and online casions, however most of these websites or games requires the need of betting with real money, rather then virtual currency. This means accessibility to gambling has never been easier as it can be done from any device with connection to the internet.

Due to the nature of gambling it is locked behind an age rating of 18 here in the UK, so most websites will block access, due to the requirements by government, unless sufficient document is given proving you are above the age rating.

# Objectives:

1. Main menu
   1. The menu should appear as a canvas
   2. The menu should appear when the program is executed
   3. The menu should have 3 buttons
      1. “Start Game” when clicked should be able to run backgroundTotal function
      2. “Load Game” when clicked should be able to run loadGame function
      3. “Exit Game” when should be able to run exitGame function
2. Blackjack Table
   1. The file “Improved Background for nea.png” should be located from a subfolder
   2. The blackjack table should appear when “Start Game” button has been pressed
   3. Have 2 Buttons appear at top left
      1. Return to menu button
         1. Button should appear as an image
         2. Button should run openNewMenu function when clicked
      2. Play Button
         1. Button should appear as an image
         2. Button should run gameplay function when clicked
3. Load Game
   1. Should be able to load saved variables from .txt files
4. Save Game
   1. Should be able to save variables into .txt files
5. AI Counting
   1. Should have a random chance whether they undergo counting methods
   2. Both AI should have separate counting methods
      1. AI 1 should have Hi-Lo counting method
      2. AI 2 should have Halves counting method
   3. Counting should occur at the start of their go and end of round
6. Deck
   1. Deck of cards from a total of 52 available cards
   2. The total deck should be made from 4 decks of 52 cards
   3. Will be updated whenever a card is removed from the deck
   4. If the deck is lowered to beneath 104 cards, then the deck should be reshuffled with all cards
7. Gameplay
   1. Player should enter how much they want to bet
   2. Both AI should enter a bet for the round
   3. AI 1 should be given choices
      1. Hit
         1. 1 Card is added to the AI 1 hand from deck
      2. Stand
         1. Ai 1 Should stand and their go is over
      3. Double Down
         1. Ai 1 should only double down when on 11 or 10 depending on true count
         2. This should double the bet the AI 1 has placed
         3. Hits card
         4. Ends AI 1’s go
   4. Player should then be given choices
      1. Hit
         1. 1 card is added to the player hand from deck
      2. Stand
         1. Player stands and their go is over
      3. Double down
         1. This should double the bet the player has set
         2. Should hit card
         3. End players go
      4. Split
         1. This should only happen if both values of card are the same
         2. This should create a new bet for the player
         3. This should create a new inventory for the split hand
   5. AI 2 should be given choices
      1. Hit
         1. 1 Card is added to the AI 2 hand from deck
      2. Stand
         1. Ai 2 Should stand and their go is over
      3. Double Down
         1. Ai 2 should only double down when on 11 or 10 depending on true count
         2. This should double the bet the AI 2 has placed
         3. Hit card
         4. AI 2 should end their go
   6. Dealer should be given choices
      1. Hit
         1. If dealer is less than 17 will take one card from deck
      2. Stand
         1. Once dealer is over 17 will stand and end their go
   7. If player goes bust
      1. Player’s hand should end
      2. Player will lose the bet they have placed
   8. If AI goes bust
      1. Ai player will lose their bet
      2. Ai hand will end
   9. If dealer goes bust
      1. Dealer hand should end
      2. Any player/AI should win unless they themselves have bust
   10. If player gets blackjack
       1. The pay-out is 3:2
       2. Hand is skipped
   11. If AI have blackjack
       1. Pay-out is 3:2
       2. Ai Hand is skipped
   12. If Dealer has blackjack
       1. Dealer Hand is skipped
8. Cards
   1. All cards from all inventories should be shown
   2. Dealers hidden card should be covered by an image
   3. A back of card image should appear to simulate the deck
9. Game End
   1. Once the player would reach 50,000 a label will congratulate them

* Able to open .png files
* Access images from subfolders
* Access .txt files from subfolders

Documented Design

# File Organisation

Graphical user interface

Description automatically generated

Figure Overall Folder Structure

I am planning to have my python code to be on the initial folder for the project, with 4 different subfolders, each having their own purpose.

Background folder

This folder will have all images that will be used to establish the background and clear cards.

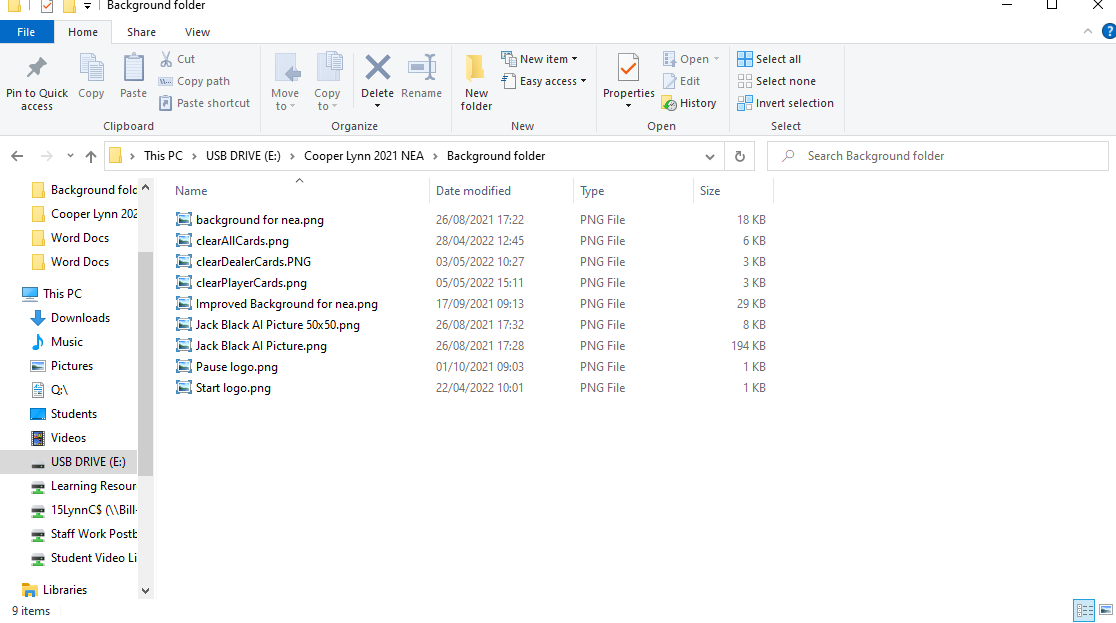


Figure Background folder Structure

playingCards

This folder will have every image of playing cards and the back of a playing card to be used when displaying the contents of inventories.

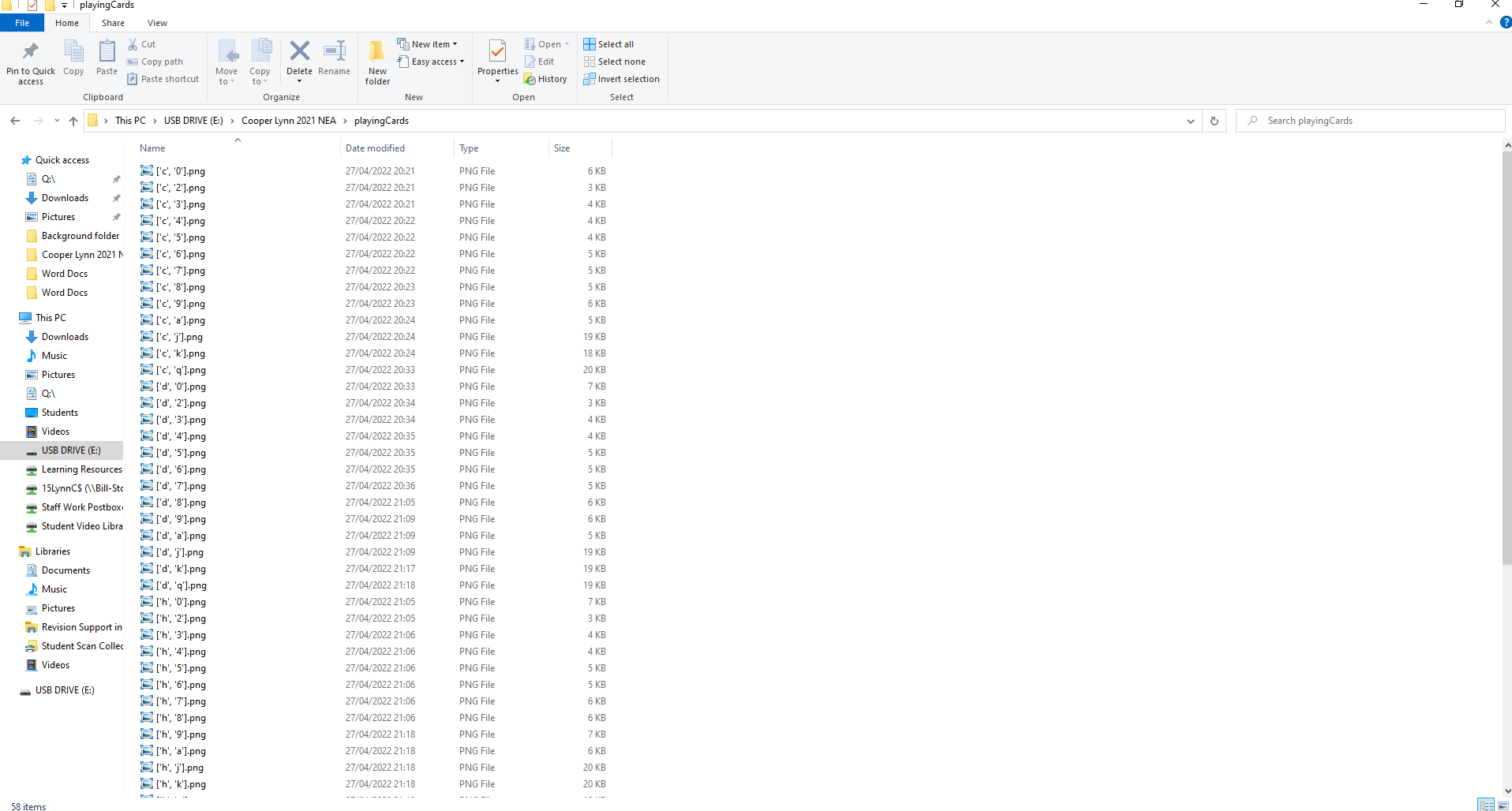


Figure playingCard Folder Structure

Save Folder

This folder will hold the .txt folders that will be used to store variables to be loaded back if the user decides to.

saveFile.txt would store the total cash the player has, trueCount for both AI if they were counting and the inventories for all.

saveDeck.txt would store each card from the deck on a separate line.

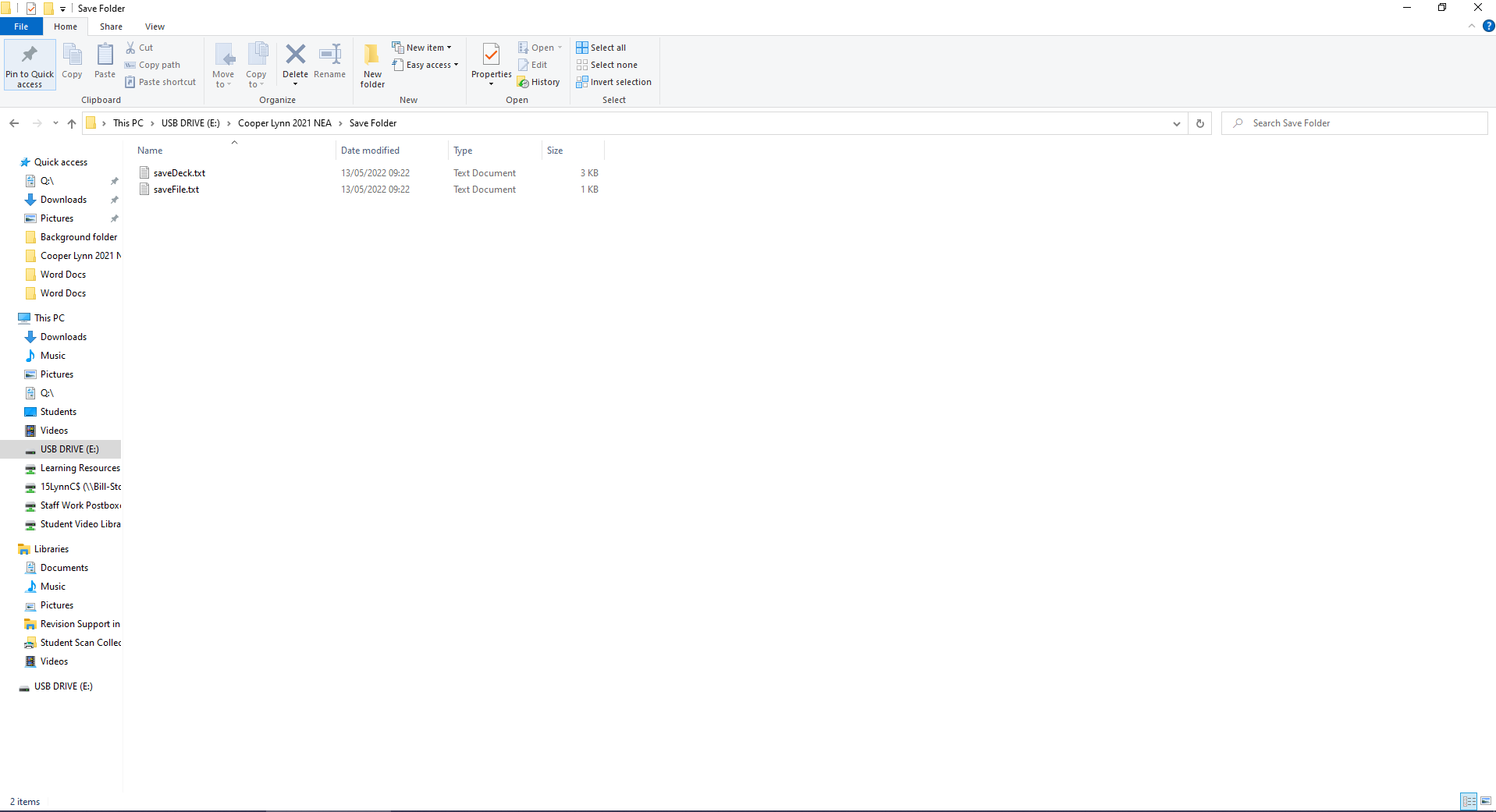


Figure Save Files Folder Structure

# GUI plan

## Background

My first idea on how I could create a GUI for blackjack would be to simulate a table for blackjack, with sections where players can make bets and where their cards and dealer’s cards are laid out.

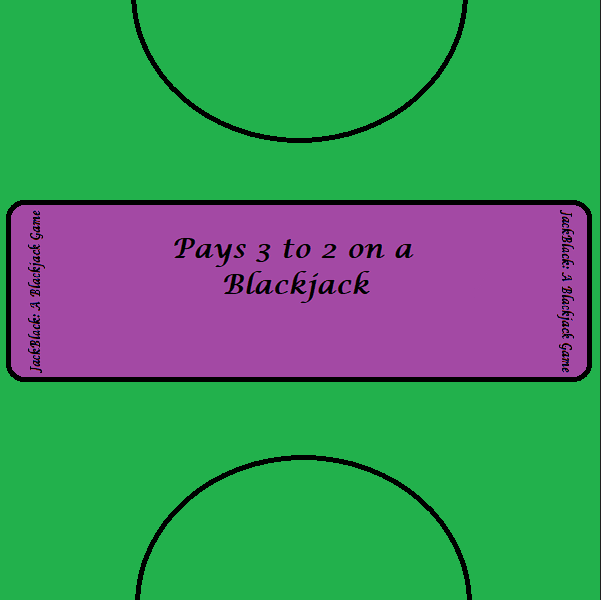


Figure Initial design

However, this design would prevent me from being able to have AI as there would be only enough room for the player and dealer to lay their cards out, so I redesigned the background to be longer and have two extra semicircles for the AIs to be able to lay their cards out.



Figure Final Design

## Menu

For my menu all I need is to start, load or quit the game, so a small window with 3 buttons would work fine for me.

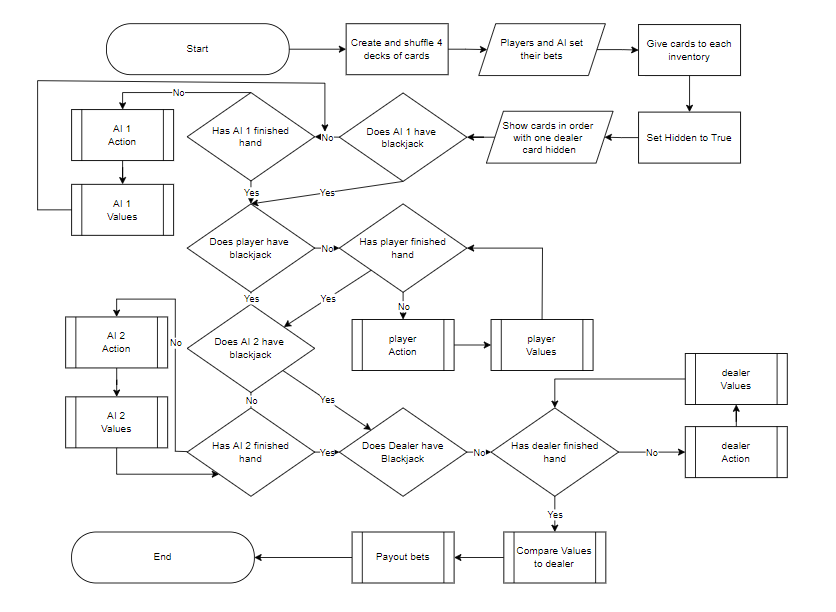
A picture containing rectangle

Description automatically generated

Figure Menu Design

# General Flowchart

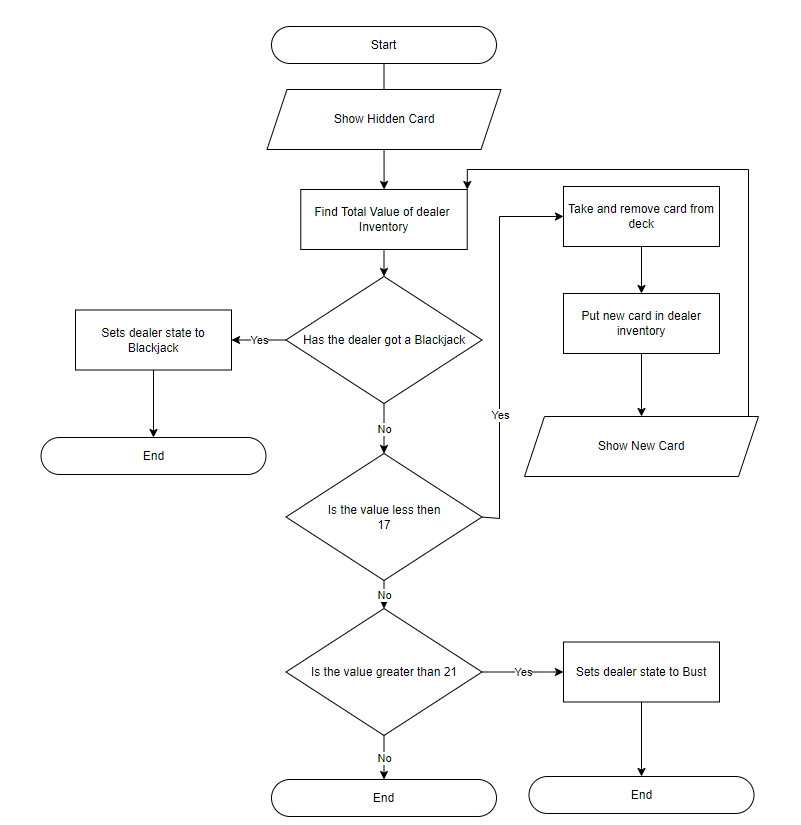
This flowchart will demonstrate the basic activities that would take place in one round of blackjack. The Process modules will be expanded on further into this document, but this is roughly how each round will play out.



# Action Flowchart

## Dealer Action Flowchart

The dealer must play by strict rules, where dealer must stand on 17 or higher, and if they are lower than they must hit. Also, the dealer must hide their first card delt to them until it is their turn to decide.



## AI Action Flowchart

### Non – Counting

This flowchart will show the actions AI would take if counting were enabled for the AI. This will simulate a reactive AI as it will have no memory of previous rounds.

Diagram

Description automatically generated

### Counting

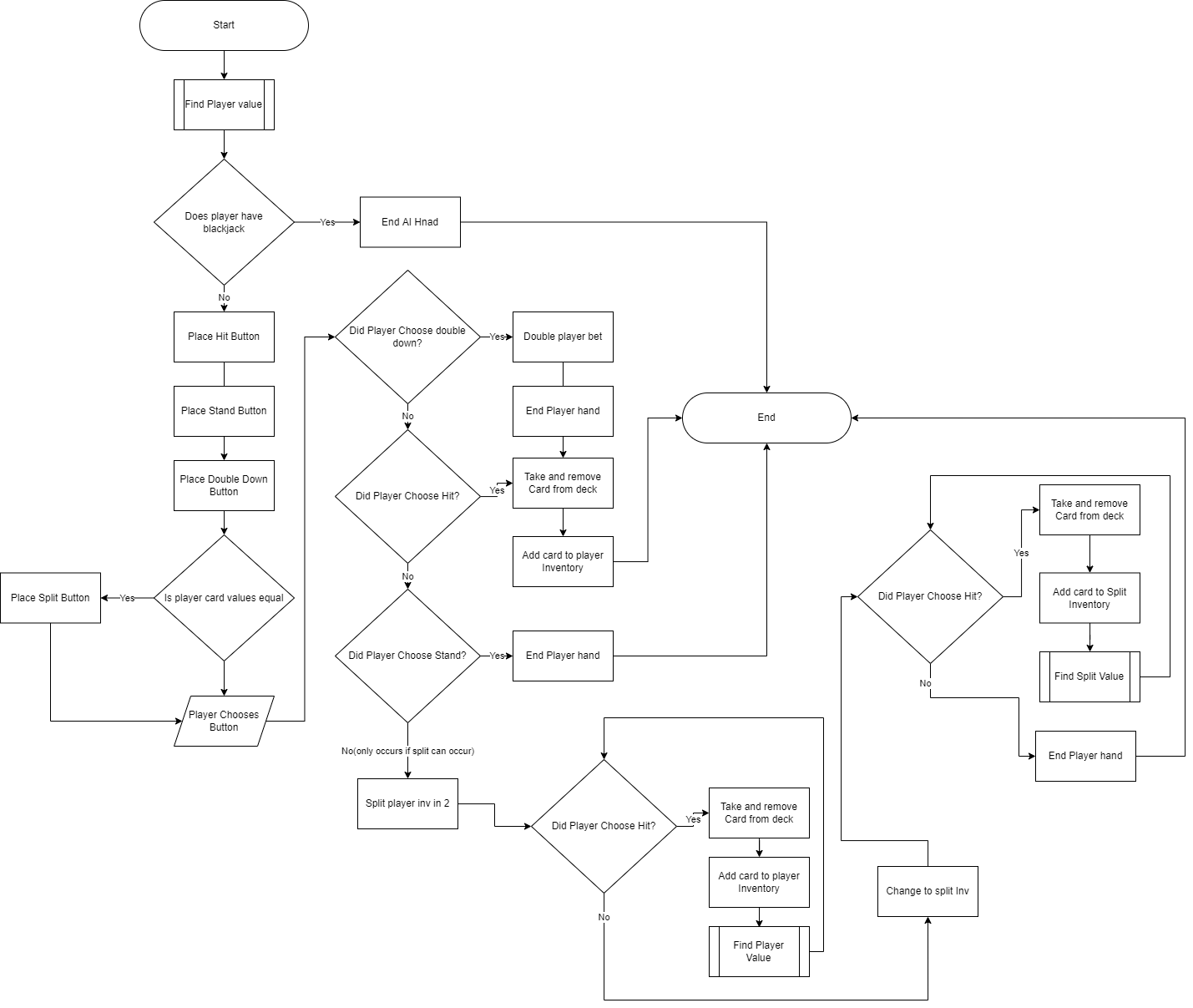
This AI will be more complicated compared to the reactive AI due to its ability to decide whether it is worth hitting, standing or doubling down at certain values due to the knowledge of previous rounds due to the cards being counted. This AI will be the Limited Memory AI.

Diagram, engineering drawing

Description automatically generated

## Player Actions

This is a flowchart that describes the actions a player can take when it their hand is in play. The player has an option to split on this flowchart, allowing for more flexibility and choice within the game.



# Randomness

This game will rely heavily on randomness, due to the need to randomly shuffle cards to prevent any bias to dealer/players. Luckily, as I have chosen python as my language to program with I can import a library called “random”. This will allow me to randomly shuffle an array, which is what I am planning to store my deck of cards in.

“random.shuffle(deckOfCards)” – This would randomly shuffle all entities within this array named deckOfCards

# Data Storage

This program requires data storage to function due to the need of updating inventories and decks of cards. So it is essential that these can be updated frequently and be used globally within the code.

## Deck and Inventories

I am planning for decks and inventories to be stored via separate arrays, meaning “cards” can be removed and appened to different arrays quickly with few lines of code. Also arrays are able to store enough cards for 4 deck blackjack, making it suitable for the game

## Bets

Bets will be stored as integers as the player will only be able to increase their total bet in increments of 10 meaning that the value should never reach a decimal number, however if this somehow manages to change the storage value, I will use a round function which is built into pythons library to make the number suitable to be stored as an integer again.

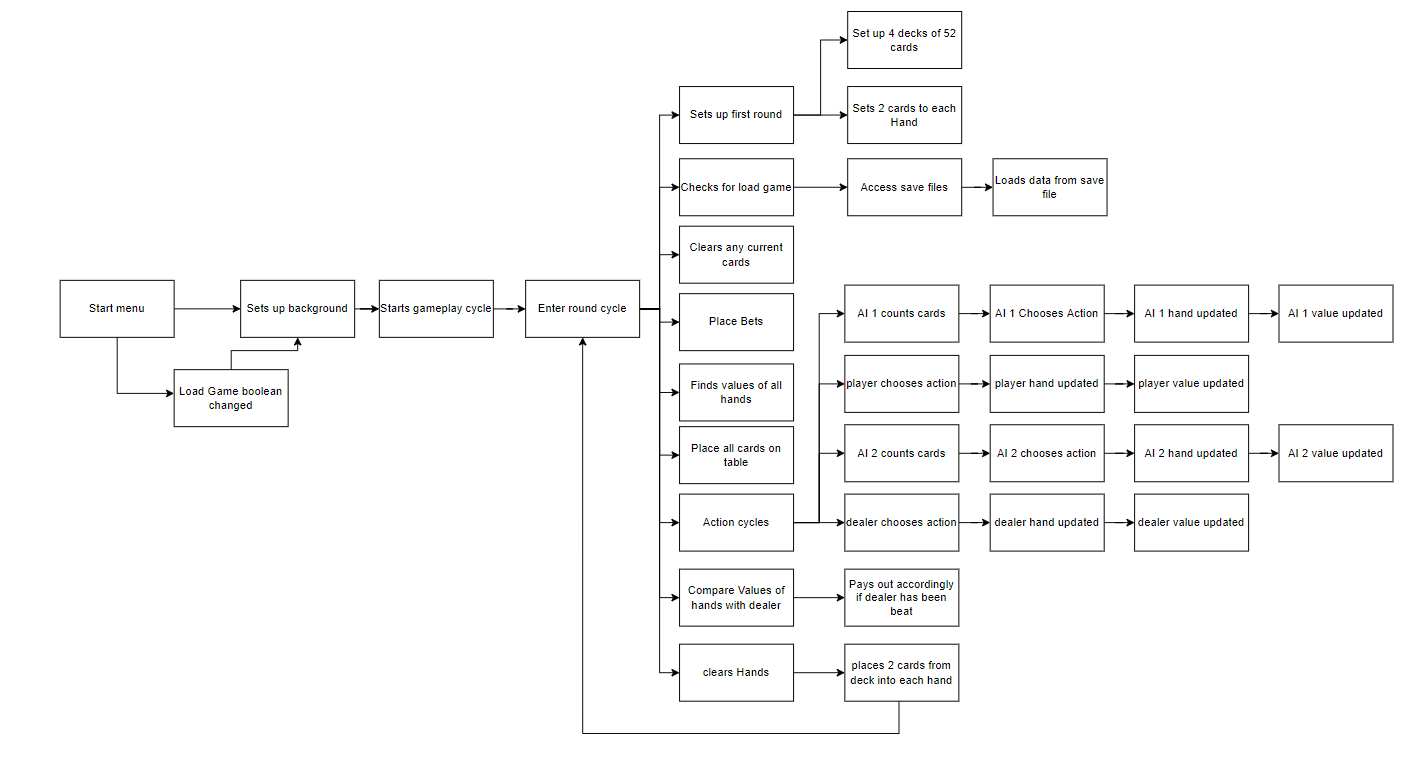
# Class Diagram

Figure Class Diagram

This is a diagram showing the relationships between the classes I am planning on implementing within the program. The Rounds class is going to be my superclass where most of the game processes will occur, but have sectioned certain processes off into different classes where it makes sense. Such as the counting methods will occur within the aiCounting class.

# Hierachy Diagram

This diagram will show how the code will progress through the game, with the the program going from left to right and priority being from top to bottom.



Technical Solution

# Python Code

from tkinter import\* #imports tkinter to be used in GUI

import time #imports time to slow down code, more user friendly

import random #imports random, allowing for true shuffle of deck

import os #imports os incase needed to interact with Operating System

global tk

tk = Tk()

global canvas

class Cards():

def \_\_init\_\_(self, inv1, inv2, inv3, inv4, inv5):

self.inv1 = inv1 #sets inventory for class to player inventory

self.inv2 = inv2 #sets inventory for class to ai1 inventory

self.inv3 = inv3 #sets inventory for class to ai2 inventory

self.inv4 = inv4 #sets inventory for class to dealer inventory

self.inv5 = inv5 #sets inventory for class for the player split inventory

def deckingCards(self):

self.deckOfCards = [] #clears the deck of cards to nothing

for i in range (0,4): #sets the amount of decks to be used to 4

for each in ["h","d","s","c"]: #assigns a house to each card

for value in ['a','2','3','4','5','6','7','8','9','0','j','q','k']: #assigns a value to each card

self.card = [each,value] #combines house and value to create card

self.deckOfCards.append(self.card) #adds card to the deck

random.shuffle(self.deckOfCards) #shuffles the deck so is random order

return self.deckOfCards #returns deck to main code

def showCardsai1(self, inv2):

print("This is showCards method")

x2 = 1250 #sets the x coordinate for the first card in A1 inv

x3 = 45 #sets the x coordinate for the first card in AI2 inv

time.sleep(0.5) #adds a timed break into the code to improve visuals

self.inv2Card1\_img = PhotoImage(file="playingCards/"+str(self.inv2[0])+".png") #sets the image to be inserted from playingCards subfolder to be the first card in A1 inv

backG.create\_image(x2, 575, image=self.inv2Card1\_img, anchor=NW) #places image on the canvas

time.sleep(0.5) #adds a timed break into the code to improve visuals

tk.update()

x2= x2+95 #moves the x coordinates so next image wont be overlapping

self.inv2Card2\_img = PhotoImage(file="playingCards/"+str(self.inv2[1])+".png") #sets the image to be inserted to be the next card in A1 inv

backG.create\_image(x2, 575, image=self.inv2Card2\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv2)>=3: #if the length of inv is over 3 then adds the 3rd card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x2= x2+95 #moves the x coordinates so next image wont be overlapping

self.inv2Card3\_img = PhotoImage(file="playingCards/"+str(self.inv2[2])+".png") #sets the image to be inserted from playingCards subfolder to be the next card in A1 inv

backG.create\_image(x2, 575, image=self.inv2Card3\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv2)>=4: #if the length of inv is over 4 then adds the 4th card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x2= x2+95 #moves the x coordinates so next image wont be overlapping

self.inv2Card4\_img = PhotoImage(file="playingCards/"+str(self.inv2[3])+".png") #sets the image to be inserted from playingCards subfolder to be the next card in A1 inv

backG.create\_image(x2, 575, image=self.inv2Card4\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv2)>=5: #if the length of inv is over 5 then adds the 5th card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x2= x2+95 #moves the x coordinates so next image wont be overlapping

self.inv2Card5\_img = PhotoImage(file="playingCards/"+str(self.inv2[4])+".png") #sets the image to be inserted from playingCards subfolder to be the next card in A1 inv

backG.create\_image(x2, 575, image=self.inv2Card5\_img, anchor=NW) #places image on the canvas

tk.update()

backG.pack()

tk.update()

def showCardsai2(self, inv3):

x3 = 45

self.inv3Card1\_img = PhotoImage(file="playingCards/"+str(self.inv3[0])+".png") #sets the image from playingCards subfolder to be inserted to the first card in AI2 inv

backG.create\_image(x3, 575, image=self.inv3Card1\_img, anchor=NW) #places image on canvas

tk.update()

x3=x3+95 #moves the x coordinates so next image wont be overlapping

time.sleep(0.5) #adds a timed break into the code to improve visuals

self.inv3Card2\_img = PhotoImage(file="playingCards/"+str(self.inv3[1])+".png") #sets the image from playingCards subfolder to be inserted to the next card in AI2 inv

backG.create\_image(x3, 575, image=self.inv3Card2\_img, anchor=NW) #places image on canvas

tk.update()

time.sleep(0.5) #adds a timed break into the code to improve visuals

if len(self.inv3)>=3: #if the length of inv is over 3 then adds the 3th card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x3= x3+95 #moves the x coordinates so next image wont be overlapping

self.inv3Card3\_img = PhotoImage(file="playingCards/"+str(self.inv3[2])+".png") #sets the image from playingCards subfolder to be inserted to the next card in AI2 inv

backG.create\_image(x3, 575, image=self.inv3Card3\_img, anchor=NW) #places image on canvas

tk.update()

if len(self.inv3)>=4: #if the length of inv is over 4 then adds the 4th card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x3= x3+95 #moves the x coordinates so next image wont be overlapping

self.inv3Card4\_img = PhotoImage(file="playingCards/"+str(self.inv3[3])+".png") #sets the image from playingCards subfolder to be inserted to the next card in AI2 inv

backG.create\_image(x3, 575, image=self.inv3Card4\_img, anchor=NW) #places image on canvas

tk.update()

if len(self.inv3)>=5: #if the length of inv is over 5 then adds the 5th card to be shown

time.sleep(0.5) #adds a timed break into the code to improve visuals

x3= x3+95 #moves the x coordinates so next image wont be overlapping

self.inv3Card5\_img = PhotoImage(file="playingCards/"+str(self.inv3[4])+".png") #sets the image from playingCards subfolder to be inserted to the next card in AI2 inv

backG.create\_image(x3, 575, image=self.inv3Card5\_img, anchor=NW) #places image on canvas

tk.update()

def showPCards(self,inv1):

x1 = 645

tk.update()

self.inv1Card1\_img = PhotoImage(file="playingCards/"+str(self.inv1[0])+".png") #sets the image from playingCards subfolder to be inserted to the first card in player inv

backG.create\_image(x1, 575, image = self.inv1Card1\_img, anchor=NW) #places image on canvas

tk.update()

time.sleep(0.5) #adds a timed break into the code to improve visuals

x1=x1+95 #moves the x coordinates so next image wont be overlapping

self.inv1Card2\_img = PhotoImage(file="playingCards/"+str(self.inv1[1])+".png") #sets the image from playingCards subfolder to be inserted to be the next card in player inv

backG.create\_image(x1, 575, image = self.inv1Card2\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=3:

time.sleep(0.5) #adds a timed break into the code to improve visuals

x1= x1+95 #moves the x coordinates so next image wont be overlapping

self.inv1Card3\_img = PhotoImage(file="playingCards/"+str(self.inv1[2])+".png") #sets the image from playingCards subfolder to be inserted to be the next card in player inv

backG.create\_image(x1, 575, image=self.inv1Card3\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=4:

time.sleep(0.5) #adds a timed break into the code to improve visuals

x1= x1+95 #moves the x coordinates so next image wont be overlapping

self.inv1Card4\_img = PhotoImage(file="playingCards/"+str(self.inv1[3])+".png") #sets the image from playingCards subfolder to be inserted to be the next card in player inv

backG.create\_image(x1, 575, image=self.inv1Card4\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=5:

time.sleep(0.5) #adds a timed break into the code to improve visuals

x1= x1+95 #moves the x coordinates so next image wont be overlapping

self.inv1Card5\_img = PhotoImage(file="playingCards/"+str(self.inv1[4])+".png") #sets the image from playingCards subfolder to be inserted to be the next card in player inv

backG.create\_image(x1, 575, image=self.inv1Card5\_img, anchor=NW) #places image on the canvas

tk.update()

def dealerCards(self, inv4, hidden):

x4 = 1030

self.inv4 = inv4

if hidden == True:#if the card needs to be hidden

self.inv4Card1\_img = PhotoImage(file="playingCards/BackOfCard90p.png") #opens subfolder playingCards and opens back of card imahe

backG.create\_image(x4, 65, image=self.inv4Card1\_img, anchor=NW) #places image on the canvas

elif hidden == False:#if the card doesnt need to be hidden

self.inv4Card1\_img = PhotoImage(file="playingCards/"+str(self.inv4[0])+".png") #opens subfolder playing card and opens the image of card needed

backG.create\_image(x4, 65, image=self.inv4Card1\_img, anchor=NW) #places image on the canvas

tk.update()

x4=x4-95 #moves x coordinate so images wont overlap

time.sleep(0.5) #added a timed break to improve on visuals

self.inv4Card2\_img = PhotoImage(file="playingCards/"+str(self.inv4[1])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in dealer inv

backG.create\_image(x4, 65, image=self.inv4Card2\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv4)>=3:#if inv4 length greater then 3

time.sleep(0.5) #adds a timed break into the code to improve visuals

x4= x4-95 #moves the x coordinates so next image wont be overlapping

self.inv4Card3\_img = PhotoImage(file="playingCards/"+str(self.inv4[2])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in dealer inv

backG.create\_image(x4, 65, image=self.inv4Card3\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv4)>=4:#if inv4 length greater then 4

time.sleep(0.5) #adds a timed break into the code to improve visuals

x4= x4-95 #moves the x coordinates so next image wont be overlapping

self.inv4Card4\_img = PhotoImage(file="playingCards/"+str(self.inv4[3])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in dealer inv

backG.create\_image(x4, 65, image=self.inv4Card4\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv4)>=5:#if inv4 length greater then 5

time.sleep(0.5) #adds a timed break into the code to improve visuals

x4= x4-95 #moves the x coordinates so next image wont be overlapping

self.inv4Card5\_img = PhotoImage(file="playingCards/"+str(self.inv4[4])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in dealer inv

backG.create\_image(x4, 65, image=self.inv4Card5\_img, anchor=NW) #places image on the canvas

tk.update()

def showSplitP(self, inv1, inv5):

y1 = 575#sets y coordinate

tk.update()

self.inv1 = inv1

self.inv1Card1\_img = PhotoImage(file="playingCards/"+str(self.inv1[0])+".png") #sets the image from sub folder playingCards to be inserted to be the first card in player inv

backG.create\_image(645, y1, image = self.inv1Card1\_img, anchor=NW) #places image

tk.update()

if len(self.inv1)>=2:#if length of inv1 is greater then 2

y1=y1-55 #moves the y coordinates so next image wont be overlapping

self.inv1Card2\_img = PhotoImage(file="playingCards/"+str(self.inv1[1])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in player inv

backG.create\_image(645, y1, image = self.inv1Card2\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=3:#if length of inv1 is greater then 3

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv1Card3\_img = PhotoImage(file="playingCards/"+str(self.inv1[2])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in player inv

backG.create\_image(645, y1, image=self.inv1Card3\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=4:#if length of inv1 is greater then 4

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv1Card4\_img = PhotoImage(file="playingCards/"+str(self.inv1[3])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in player inv

backG.create\_image(645, y1, image=self.inv1Card4\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv1)>=5:#if length of inv1 is greater then 5

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv1Card5\_img = PhotoImage(file="playingCards/"+str(self.inv1[4])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in player inv

backG.create\_image(645, y1, image=self.inv1Card5\_img, anchor=NW) #places image on the canvas

tk.update()

y1 = 575#sets y coordinate

tk.update()

self.inv5Card1\_img = PhotoImage(file="playingCards/"+str(self.inv5[0])+".png") #opens sub folder playingCards and gets first card from inv image

backG.create\_image(1025, y1, image = self.inv5Card1\_img, anchor=NW) #places image on canvas

tk.update()

if len(self.inv5)>=2:#if length of inv5 is greater then 2

y1=y1-55 #moves the y coordinates so next image wont be overlapping

self.inv5Card2\_img = PhotoImage(file="playingCards/"+str(self.inv5[1])+".png") #sets the image to be inserted to be the next card in inv5

backG.create\_image(1025, y1, image = self.inv5Card2\_img, anchor=NW) #places image on the canva

tk.update()

if len(self.inv5)>=3:#if length of inv5 is greater then 3

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv5Card3\_img = PhotoImage(file="playingCards/"+str(self.inv5[2])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in inv5

backG.create\_image(1025, y1, image=self.inv5Card3\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv5)>=4:#if length of inv5 is greater then 4

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv5Card4\_img = PhotoImage(file="playingCards/"+str(self.inv5[3])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in inv5

backG.create\_image(1025, y1, image=self.inv5Card4\_img, anchor=NW) #places image on the canvas

tk.update()

if len(self.inv5)>=5:#if length of inv5 is greater then 5

y1= y1-55 #moves the y coordinates so next image wont be overlapping

self.inv5Card5\_img = PhotoImage(file="playingCards/"+str(self.inv5[4])+".png") #sets the image from sub folder playingCards to be inserted to be the next card in inv5

backG.create\_image(1025, y1, image=self.inv5Card5\_img, anchor=NW) #places image on the canvas

tk.update()

class aiCounting():#class used by first ai if undergoing counting procedure

def \_\_init\_\_(self, inv1, inv2, inv3, inv4, runTotal\_1, runTotal\_2, deck, trueCount1, trueCount2, playedCardsAI1, playedCardsAI2):

self.score = 0 #sets score used for counting to 0

self.inv1 = inv1 #sets inventory for class to player inventory

self.inv2 = inv2 #sets inventory for class to ai1 inventory

self.inv3 = inv3 #sets inventory for class to ai2 inventory

self.inv4 = inv4 #sets inventory for class to dealer inventory

self.runTotal\_1 = runTotal\_1#sets the runTotal 1 for ai1 counting

self.runTotal\_2 = runTotal\_2#sets the runTotal 2 for ai2 counting

self.deck = deck #sets deck for class to be current deck

self.trueCount1 = trueCount1#sets the true count for ai1 to the current true count

self.trueCount2 = trueCount2#sets the true count for ai2 to the current true count

self.playedCardsAI1 = playedCardsAI1

self.playedCardsAI2 = playedCardsAI2

def counting1(self, c, inv1, inv2, inv3, inv4, deck, trueCount1, playedCards): #counting strategy Hi-Lo

print("hi-lo")

for i in range(0,len(inv1)): #for however many cards are in the players inv

self.temp = self.inv1[i] #puts one card from inv into a temporary attribute

self.value = self.temp[1] #finds the value of card from temporay attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_1 value by 1 for each high value card

self.runTotal\_1 = self.runTotal\_1 -1

print("-1")

elif int(self.value) <=6: #increases the runTotal\_1 value by 1 for each lower value card

self.runTotal\_1 = self.runTotal\_1+1

print("+1")

print("run total 1:", self.runTotal\_1)

for i in range(0,len(inv2)): #for however many cards are in the AI1 inv

self.temp = self.inv2[i] #puts one card from ai1 inv into a temporary attribute

self.value = self.temp[1] #finds the value of the card from the temporary attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_1 value by 1 for each high value card

self.runTotal\_1 = self.runTotal\_1 -1

print("-1")

elif int(self.value) <=6: #increases the runTotal\_1 value by 1 for each lower value card

self.runTotal\_1 = self.runTotal\_1+1

print("+1")

print("run total 1:", self.runTotal\_1)

for i in range(0,len(inv3)): #for however many cards are in the AI2 inv

self.temp = self.inv3[i] #puts one card from ai1 inv into a temporary attribute

self.value = self.temp[1] #finds the value of the card from the temporary attribute

print(self.value)

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_1 value by 1 for each high value card

print("-1")

self.runTotal\_1 = self.runTotal\_1 -1

elif int(self.value) <=6: #increases the runTotal\_1 value by 1 for each lower value card

self.runTotal\_1 = self.runTotal\_1+1

print("+1")

print("run total 1:", self.runTotal\_1)

self.temp = self.inv4[1] #puts shown card from dealer into a temporary attribute

self.value = self.temp[1] #finds the value of the card from the temporary attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_1 value by 1 for each high value card

print("-1")

self.runTotal\_1 = self.runTotal\_1 -1

elif int(self.value) <=6: #increases the runTotal\_1 value by 1 for each lower value card

self.runTotal\_1 = self.runTotal\_1+1

print("+1")

print("run total 1:", self.runTotal\_1)

self.trueCount1 = (self.runTotal\_1/(len(self.deck)/52)) #finds out the trueCount1 which can be used across all rounds to come

print("number of decks remaining: ",len(self.deck)/52)

print("running total: ", self.runTotal\_1)

print("true count: ", self.trueCount1)

return self.trueCount1

def counting2(self, c, inv1, inv2, inv3, inv4, deck, trueCount2): #counting strategy halves

print("halves")

for i in range(0,len(inv1)): #for however many cards are in the players inv

self.temp = self.inv1[i] #puts one card from player inv into a temporary attribute

self.value = self.temp[1] #finds the value of card from temproray attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_2 value by 1 for each high value card

self.runTotal\_2 = self.runTotal\_2-1

print("-1")

elif int(self.value) ==9: #decreases the runTotal\_2 value by 0.5 for each 9 card

self.runTotal\_2 = self.runTotal\_2-0.5

print("-0.5")

elif int(self.value) ==7 or int(self.value) ==2: #increases the runTotal\_2 value by 0.5 for each 2 or 7 card

self.runTotal\_2 = self.runTotal\_2+0.5

print("+0.5")

elif int(self.value) == 3 or int(self.value) == 4 or int(self.value) == 6: #increases the runTotal\_2 value by 1 for each 3, 4 or 6 card

self.runTotal\_2 = self.runTotal\_2+1

print("+1")

elif int(self.value) ==5: #increases the runTotal\_2 valye by 1.5 for each 5 card

self.runTotal\_2 = self.runTotal\_2+1.5

print("+1.5")

print("run total 2:",self.runTotal\_2)

for i in range(0,len(inv2)): #for however many cards are in the AI1 inv

self.temp = self.inv2[i] #puts one card from ai1 inv into a temporary attribute

self.value = self.temp[1] #finds the value of card from temproray attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_2 value by 1 for each high value card

self.runTotal\_2 = self.runTotal\_2-1

print("-1")

elif int(self.value) ==9: #decreases the runTotal\_2 value by 0.5 for each 9 card

self.runTotal\_2 = self.runTotal\_2-0.5

print("-0.5")

elif int(self.value) ==7 or int(self.value) ==2: #increases the runTotal\_2 value by 0.5 for each 2 or 7 card

self.runTotal\_2 = self.runTotal\_2+0.5

print("+0.5")

elif int(self.value) == 3 or int(self.value) == 4 or int(self.value) == 6: #increases the runTotal\_2 value by 1 for each 3, 4 or 6 card

self.runTotal\_2 = self.runTotal\_2+1

print("+1")

elif int(self.value) ==5: #increases the runTotal\_2 valye by 1.5 for each 5 card

self.runTotal\_2 = self.runTotal\_2+1.5

print("+1.5")

print("run total 2:",self.runTotal\_2)

for i in range(0,len(inv3)): #for however many cards are in the AI2 inv

print(self.inv3)

self.temp = self.inv3[i] #puts one card from ai2 inv into a temporary attribute

self.value = self.temp[1] #finds the value of card from temporary attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_2 value by 1 for each high value card

self.runTotal\_2 = self.runTotal\_2-1

print("-1")

elif int(self.value) ==9: #decreases the runTotal\_2 value by 0.5 for each 9 card

self.runTotal\_2 = self.runTotal\_2-0.5

print("-0.5")

elif int(self.value) ==7 or int(self.value) ==2: #increases the runTotal\_2 value by 0.5 for each 2 or 7 card

self.runTotal\_2 = self.runTotal\_2+0.5

print("+0.5")

elif int(self.value) == 3 or int(self.value) == 4 or int(self.value) == 6: #increases the runTotal\_2 value by 1 for each 3, 4 or 6 card

self.runTotal\_2 = self.runTotal\_2+1

print("+1")

elif int(self.value) ==5: #increases the runTotal\_2 value by 1.5 for each 5 card

self.runTotal\_2 = self.runTotal\_2+1.5

print("+1.5")

self.temp = self.inv4[1] #puts shown card from dealer into a temporary attribute

self.value = self.temp[1] #finds the value of card from temporary attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_2 value by 1 for each high value card

self.runTotal\_2 = self.runTotal\_2-1

print("-1")

elif int(self.value) ==9: #decreases the runTotal\_2 value by 0.5 for each 9 card

self.runTotal\_2 = self.runTotal\_2-0.5

print("-0.5")

elif int(self.value) ==7 or int(self.value) ==2: #increases the runTotal\_2 value by 0.5 for each 2 or 7 card

self.runTotal\_2 = self.runTotal\_2+0.5

print("+0.5")

elif int(self.value) == 3 or int(self.value) == 4 or int(self.value) == 6: #increases the runTotal\_2 value by 1 for each 3, 4 or 6 card

self.runTotal\_2 = self.runTotal\_2+1

print("+1")

elif int(self.value) ==5: #increases the runTotal\_2 valye by 1.5 for each 5 card

self.runTotal\_2 = self.runTotal\_2+1.5

print("+1.5")

self.trueCount2= (self.runTotal\_2/(len(self.deck)/52)) #finds trueCount2 which can be used across all rounds to come

print("run total 2:",self.runTotal\_2)

print(self.trueCount2)

return self.trueCount2

def ai1EndgameCount(self, playedCardsAI1, trueCount1, deck):

print("hilo")

print(self.playedCardsAI1)

for i in range(0, len(playedCardsAI1)): #for length in array

self.temp = self.playedCardsAI1[i] #gets card from array

self.value = self.temp[1] #finds value from card

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_1 value by 1 for each high value card

self.runTotal\_1 = self.runTotal\_1 -1

print("-1")

elif int(self.value) <=6: #increases the runTotal\_1 value by 1 for each lower value card

self.runTotal\_1 = self.runTotal\_1+1

print("+1")

print(len(self.deck))

print(self.runTotal\_1)

self.trueCount1 =(self.runTotal\_1/(len(self.deck)/52)) #finds out truecount by dividing total+run count by length of deck remaining

return self.trueCount1

def ai2EndgameCount(self, playedCardsAi2, trueCount2, deck):

self.playedCardsAi2 = playedCardsAi2

print("in ai2 endgame count")

print(self.playedCardsAi2)

for i in range(0, len(self.playedCardsAi2)): #for however many cards are in the AI2 inv

self.temp = self.playedCardsAi2[i] #puts one card from ai2 inv into a temporary attribute

self.value = self.temp[1] #finds the value of card from temporary attribute

print(self.value)

if self.value == 'q' or self.value =='k' or self.value =='j' or self.value =='0' or self.value == 'a': #decreases the runTotal\_2 value by 1 for each high value card

self.runTotal\_2 = self.runTotal\_2-1

print("-1")

elif int(self.value) ==9: #decreases the runTotal\_2 value by 0.5 for each 9 card

self.runTotal\_2 = self.runTotal\_2-0.5

print("-0.5")

elif int(self.value) ==7 or int(self.value) ==2: #increases the runTotal\_2 value by 0.5 for each 2 or 7 card

self.runTotal\_2 = self.runTotal\_2+0.5

print("+0.5")

elif int(self.value) == 3 or int(self.value) == 4 or int(self.value) == 6: #increases the runTotal\_2 value by 1 for each 3, 4 or 6 card

self.runTotal\_2 = self.runTotal\_2+1

print("+1")

elif int(self.value) ==5: #increases the runTotal\_2 value by 1.5 for each 5 card

self.runTotal\_2 = self.runTotal\_2+1.5

print("+1.5")

print(len(self.deck))

print(self.runTotal\_2)

self.trueCount2 =(self.runTotal\_2/(len(self.deck)/52)) #finds out truecount by dividing total+run count by length of deck remaining

print(self.value, " ", self.runTotal\_2, "", len(self.deck))

return self.trueCount2

def actions(self, value, aceCount, trueCount):#ai actions if counting enabled

self.Hand = False#sets hand to false

self.value = value#initialises value to class

self.aceCount = aceCount# initialises aceCount to class

self.trueCount = trueCount#initialises trueCount to class

if self.value == 20:#if the ai has a value of 20

self.state = "Stand"#ai will stand

elif self.value == 19:#if ai has value of 19

self.state = "Stand"#ai will stand

elif self.value == 18:#if ai has value of 18

self.state = "Stand"#ai will stand

elif self.value == 17 and self.aceCount>0:#if ai has 17 and an ace

self.state = "Hit"#ai will hit

elif self.value == 17:#if ai has 17 and no ace

self.state = "Stand"#ai will stand

elif self.value == 16 and (self.trueCount<-1 or self.aceCount>0):#if ai has 16 and high chance of low card or owns an ace card

self.state = "Hit"#ai will hit

elif self.value == 16:#if ai has 16 and no ace or no high chance of low card

self.state = "Stand"#ai will stand

elif self.value == 15 and (self.trueCount<-0.5 or self.aceCount>0):#if ai has 15 and high chance of low card or owns an ace card

self.state = "Hit"#ai will hit

elif self.value == 11:#if ai is at 11

self.state = "Double Down"#ai will double down

elif self.value == 10 and self.trueCount>2:#if ai is at 10 and high chance of getting a high value card

self.state = "Double Down"#ai will double down

else:#if ai has any other value

self.state = "Hit"#ai will hit

return self.state#will return state back to gameCycle method

def getRunTotal\_1(self):

return self.runTotal\_1

def getRunTotal\_2(self):

return self.runTotal\_2

class Rounds():

def \_\_init\_\_(self, inv1, inv2, inv3, inv4, deck): #initialises rounds class

self.score = 0 #sets score used for counting to 0

self.inv1 = inv1 #sets inventory for class to player inventory

self.inv2 = inv2 #sets inventory for class to ai1 inventory

self.inv3 = inv3 #sets inventory for class to ai2 inventory

self.inv4 = inv4 #sets inventory for class to dealer inventory

self.deck = deck #sets deck for class to be current deck

self.countingChance1 = 0 #sets the chance for counting for ai1 to occur to be 0

self.countingChance2 = 0 #sets the chance for counting for ai2 to occur to be 0

self.trueCount1 = 0 #sets the trueCount1 to 0 for start of game

self.trueCount2 = 0 #sets the trueCount2 to 0 for start of game

self.ai1Value = 0 #sets the value of a1 inv to 0

self.pValue = 0 #sets the value of player inv to 0

self.ai2Value = 0 #sets the value of ai2 inv to 0

self.dealerValueH = 0 #sets the value of the dealers hidden value to 0

self.dealerValueS = 0 #sets the value of the dealers shown value to 0

self.dealerValue = 0 #sets the value of the dealer total value to 0

self.aceAi1Count = 0 #sets the count of aces AI1 has to 0

self.acePCount = 0 #sets the count of aces player has to 0

self.aceAi2Count = 0 #sets the count of aces AI2 has to 0

self.aceDealerCount = 0 #sets the count of aces dealer has to 0

self.ai1Bet = 5000 #sets the bet of the ai1 to 5000

self.ai2Bet = 5000 #sets the bet of the ai2 to 5000

self.playerBet = 5000 #sets the bet of the player to 5000

self.dealerHit = False #sets the dealerHit to false

self.splitValue = 0 #sets the value of split inv to 0

def playerHit(self):#if the player has chosen to hit

self.pHit = True#prevents player from getting blackjack

self.pHand = False#keeps player in hand

self.pCurrentHit = True#allows player to hit

def playerStand(self):#player chosen to stant

print("player trying to stand")

self.pHand = True#ends players hand

self.pSplit = False#prevents split from occuring

self.inv1Split = True

self.inv5Split = True

self.pSplit = False

def playerDouble(self):#player doubles down

self.pHit = True#prevents player from getting blackjack

self.pHand = False#keeps player hand going

self.pDDHit = True#allows player to double down

def playerSplit(self):#player chooses to split

self.pSplit = True

def values(self,inv1, inv2, inv3, inv4):

#resets all variables used in values

self.ai1Value = 0

self.ai2Value = 0

self.pValue = 0

self.aceAi1Count = 0

self.aceAi2Count = 0

self.acePCount = 0

self.aceDealerCount = 0

for i in self.inv2:

#determins ai1 card values

#gets the first card in ai1 inventory

elementposition = self.inv2.index(i)

self.ai1Card = self.inv2[elementposition]

self.ai1CardValue = self.ai1Card[1]#finds the value of the first card

if self.ai1CardValue == "k" or self.ai1CardValue == "q" or self.ai1CardValue == "j" or self.ai1CardValue =="0":

self.ai1CardValue = 10 #changes the value of a face or 10 card to a 10

self.ai1Face = True

elif self.ai1CardValue == "a":

self.ai1CardValue = 11 #changes the value of an ace to 11 and changes boolean

self.ai1AceHand = True

self.aceAi1Count = self.aceAi1Count + 1

print(self.ai1Value)

print(self.ai1CardValue)

self.ai1Value = int(self.ai1Value)+int(self.ai1CardValue)#finds the combined total value of cards of the ai1

for i in range(0,self.aceAi1Count):

if self.ai1Value>21: #if the value exceeds 21 and contains an ace, one ace card will revert to the value of 1

print("ace moved to 1 ai1")

self.ai1Value = self.ai1Value-10

self.aceAi1Count = self.aceAi1Count-1

for j in self.inv1:

#determins ai1 card values

#gets the card in player inventory

elementposition = self.inv1.index(j)

self.pCard = self.inv1[elementposition]

self.pCardValue = self.pCard[1]#finds the value of the card

print(self.pCard)

print(self.pCardValue)

if self.pCardValue == "k" or self.pCardValue == "q" or self.pCardValue == "j" or self.pCardValue =="0":

self.pCardValue = 10 #changes the value of a face or 10 card to 10

elif self.pCardValue == "a":

self.pCardValue = 11 #changes the value of an ace to 11 and changes boolean

self.pAceHand = True

self.acePCount = self.acePCount + 1

self.pValue = int(self.pValue)+int(self.pCardValue)#finds the combined total value of cards of the player

for i in range(0,self.acePCount):

if self.pValue>21: #if the value exceeds 21 and contains an ace, one ace card will revert to the value of 1

print("ace moved to 1 player")

self.pValue = self.pValue-10

self.acePCount = self.acePCount-1

for k in self.inv3:#for length of inv3

#determins ai1 card values

#gets the card in ai2 inventory

elementposition = self.inv3.index(k)

self.ai2Card = self.inv3[elementposition]

self.ai2CardValue = self.ai2Card[1]#finds the value of the card

if self.ai2CardValue == "k" or self.ai2CardValue == "q" or self.ai2CardValue == "j" or self.ai2CardValue =="0":

self.ai2CardValue = 10 #changes the value of a face or 10 card to 10

elif self.ai2CardValue == "a":

self.ai2CardValue = 11 #changes the value of an ace to 11 and changes boolean

self.ai2AceHand = True

self.aceAi2Count = self.aceAi2Count + 1

self.ai2Value = int(self.ai2Value)+int(self.ai2CardValue)#finds the combined total value of cards of the ai2

for i in range(0,self.aceAi2Count):

if self.ai2Value>21: #if the value exceeds 21 and contains an ace, one ace card will revert to the value of 1

self.ai2Value = self.ai2Value-10

self.aceAi2Count = self.aceAi2Count-1

global ai1Value

global pValue

global ai2Value

global dealerValue

global splitValue

def splitValues(self, inv5):

self.splitValue = 0

self.aceSplitCount = 0

for i in self.inv5:#for length of inv5

#gets card from split inventory

elementposition = self.inv5.index(i)

self.splitCard = self.inv5[elementposition]

self.splitCardValue = self.splitCard[1]#finds the value of the card

if self.splitCardValue == "k" or self.splitCardValue == "q" or self.splitCardValue == "j" or self.splitCardValue =="0":

self.splitCardValue = 10 #changes the value of a face or 10 card to 10

self.splitFace = True

elif self.splitCardValue == "a":

self.splitCardValue = 11 #changes the value of an ace to 11 and changes boolean

self.splitAceHand = True

self.aceSplitCount = self.aceSplitCount + 1

self.splitValue = int(self.splitValue)+int(self.splitCardValue)#finds the combined total value of cards of the split inv

for i in range(0,self.aceSplitCount):

if self.splitValue>21: #if the value exceeds 21 and contains an ace, one ace card will revert to the value of 1

print("ace moved to 1 ai1")

self.splitValue = self.splitValue-10

self.aceSplitCount = self.aceSplitCount-1

return self.splitValue

def dealerValueCounter(self, inv4):

self.dealerValue = 0

for l in self.inv4:#for how long inv 4 is

#finds value of card

elementposition = self.inv4.index(l)

self.dealerCard = self.inv4[elementposition]

self.dealerCardValue = self.dealerCard[1]

if self.dealerCardValue == "k" or self.dealerCardValue == "q" or self.dealerCardValue == "j" or self.dealerCardValue =="0":

self.dealerCardValue = 10 #changes the value of a face or 10 card to 10

elif self.dealerCardValue == "a":

self.dealerCardValue = 11 #changes the value of an ace to 11 and changes boolean

self.dealerAceHand = True

self.aceDealerCount = self.aceDealerCount + 1#increases ace count for dealer

self.dealerValue = int(self.dealerValue)+int(self.dealerCardValue)#finds the combined total value of cards of the dealer

for i in range(0,self.aceDealerCount):

if self.dealerValue>21:

self.dealerValue = self.dealerValue-10

self.aceDealerCount = self.aceDealerCount-1

#if the value exceeds 21 and contains an ace, one ace card will revert to the value of 1

global dealerValue

def betUp10(self):#if player chooses up 10

if self.playerBet>=10:#will check if player has enough cash

global pCashCounter, pBetCounter

self.playerBet = self.playerBet-10#will remove 10 from total cash

self.playerPlayBet = self.playerPlayBet + 10#will add 10 to current bet

#destroys current labels

pCashCounter.destroy()

pBetCounter.destroy()

self.betChange = True#changes boolean to true

else:

#states that u are broke in console

print("ur broke lol")

def betUp100(self):#if player chooses up 100

if self.playerBet>=100:#check if player has enough cash

global pCashCounter, pBetCounter

self.playerBet = self.playerBet-100#will remove 100 from total cash

self.playerPlayBet = self.playerPlayBet+100#will add 100 to current bet

#will destroy old labels

pCashCounter.destroy()

pBetCounter.destroy()

self.betChange = True#changes boolean to true

def betDown10(self):#if player wants to remove 10 from current bet

global pCashCounter, pBetCounter

if self.playerPlayBet-10>=0:#checks if enough cash is in current bet

self.playerBet = self.playerBet+10#adds 10 to total cash

self.playerPlayBet = self.playerPlayBet-10#remove 10 from current bet

#destroys old labels

pBetCounter.destroy()

pCashCounter.destroy()

self.betChange = True#changes boolean to true

def betDown100(self):#if player chooses down 100

global pCashCounter, pBetCounter

if self.playerPlayBet-100>=0:#checks if player has enough cash in current bet

self.playerBet=self.playerBet+100#adds 100 to total cash

self.playerPlayBet = self.playerPlayBet-100#removes 100 from current bet

pCashCounter.destroy()#destroys old labels

pBetCounter.destroy()

self.betChange = True#changes boolean to true

def confirmBet(self):#if player has confirmed their bet

if self.playerPlayBet >0:#checks to see if player has acutally bet

self.playerBetChosen = True#ends the betting process

elif self.playerBet<0:#if player hasnt actually bet

actuallyBetUPoor = Label(tk, text="You do not have enough cash for this bet\nIf you have run out of cash\npress the play button again to restart")#label made to say they dont have enough to bet that

#(cont from prev line) and if the user has ran out of money they will be told to restart their game

actuallyBetUPoor.place(x=850, y=400)#places label

tk.update()

time.sleep(3)#time delay added for visual improvement

actuallyBetUPoor.destroy()#destroys label

else:

actuallyBetUPoor = Label(tk, text="You need to actually bet some money")#label made to say they cant bet nothing

actuallyBetUPoor.place(x=850, y=400)#places label

tk.update()

time.sleep(3)#adds delay for visual improvement

actuallyBetUPoor.destroy()#destroys label

def playerSplitAction(self, inv1):#this function splits inv1 into two different invs

clearPlayerCards = PhotoImage(file="Background folder/clearPlayerCards.png")

backG.create\_image(897, 630, image=clearPlayerCards)

tk.update()

time.sleep(1)

self.temp = self.inv1[1]#takes 2nd card from inv1

self.inv5.append(self.temp)#adds card to inv5

self.inv1.pop(1)#removes card from player inv

cards = Cards(self.inv1, self.inv2, self.inv3, self.inv4, self.inv5)#initializes card class due to new inv

cards.showSplitP(self.inv1, self.inv5)#show new inv

self.pSplit = True

def saveGame(self):

f = open("Save Folder/saveFile.txt", "w")#opens subfolder Save folder and opens a txt file in write mode

f.write(str(self.playerBet)+"\n"+str(self.trueCount1)+"\n"+str(self.trueCount2)+"\n")#writes integer values as strings on each line in txt file

for i in range(0, len(self.inv1)):#for the length of inv1

f.write(str(self.inv1[i])+"\n")#adds card to next line in txt file

for i in range(0, len(self.inv2)):#for length of inv2

f.write(str(self.inv2[i])+"\n")#adds card to next line in txt file

for i in range(0, len(self.inv3)):#for length of inv3

f.write(str(self.inv3[i])+"\n")#adds card to next line in txt file

for i in range(0, len(self.inv4)):#for length of inv4

f.write(str(self.inv4[i])+"\n")#adds card to next line in txt file

f.close()#closes saveFile.txt

f = open("Save Folder/saveDeck.txt", "w")#opens subfolder Save folder and opens saveDeck txt file in write mode

for i in range(0, len(self.deck)):#for length of deck

f.write(str(self.deck[i])+"\n")#adds card to the next line in txt file

f.close()#closes saveDeck file

def loadGame(self):

#resets arrays to allow for loading of variables

self.inv1 = []

self.inv2 = []

self.inv3 = []

self.inv4 = []

self.deck = []

f = open("Save Folder/saveFile.txt", "r")#accesses sub folder Save Folder to open txt saveFile in read mode

#reads each line from file and assigns line to a variable

lineFromFile = f.readline().rstrip()#reads line

self.playerBet = int(lineFromFile)#variable set to contents of line

lineFromFile = f.readline().rstrip()#reads line

self.trueCount1 = float(lineFromFile)#variable set to contents of line

lineFromFile = f.readline().rstrip()#reads line

self.trueCount2 = float(lineFromFile)#variable set to contents of line

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv1.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv1.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv2.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv2.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv3.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv3.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv4.append(invAdder)#adds card to inv

lineFromFile = f.readline().rstrip()#reads line

invAdder = [lineFromFile[2], lineFromFile[7]]#only takes two characters from line (house, value)

self.inv4.append(invAdder)#adds card to inv

f.close()#closes file

f = open("Save Folder/saveDeck.txt", "r")#accesses sub folder Save folder to open txt saveDeck in read mode

for line in f:#for every line in file will read

try:#will try to execute code

deckAdder = [line[2], line[7]]#only takes two characters from line (house, value)

self.deck.append(deckAdder)#adds card to deck

except:#if code couldnt be executed

pass#does nothing so doesnt affect code

f.close

f.close

def gameCycle(self, countingChance1, countingChance2, inv1, inv2, inv3, inv4, deck):

global loadBoolean

global pHitButton

global pStandButton

self.roundStart(countingChance1, countingChance2, inv1, inv2, inv3, inv4, deck)#intializes the start of the round

if loadBoolean == True: #if the loadBoolean = True then the game will load data from seperate files

self.loadGame()

loadBoolean = False #returns boolean to default value to stop false positives

self.values(self.inv1, self.inv2, self.inv3, self.inv4) #finds te values of image

self.runTotal\_1 = 0

self.runTotal\_2 = 0

while self.playerBet<50000: # creates a while loop to help end game at 50000 score

self.saveGame() #saves game variables to two different files

print(self.deck)

print(len(self.deck))

#used to test split

self.inv1 = []

self.inv1.append(['c', 'k'])

#self.inv1.append(['c', 'a'])

self.inv1.append(['c', 'k'])

#resets all variables and arrays

self.splitBet = 0

self.inv5 = []

self.playedCardsAI1 = []

self.playedCardsAI2 = []

self.values(self.inv1, self.inv2, self.inv3, self.inv4)

self.ai1Hit = False

self.ai2Hit = False

self.ai1Blackjack = False

self.ai2Blackjack = False

cards = Cards(self.inv1, self.inv2, self.inv3, self.inv4, self.inv5)#initializes Card class incase of change from load

self.handClear = PhotoImage(file="Background folder/clearAllCards.png")#accesses Background folder sub folder to assign an image

backG.create\_image(0,444,image=self.handClear,anchor=NW)#places image over to hide player and AI previous cards

self.dealerClear = PhotoImage(file = "Background folder/clearDealerCards.PNG")#accesses Background folder sub folder to assign an image

backG.create\_image(601, 60, image=self.dealerClear,anchor=NW)#places image over dealer previous carsd

counting = aiCounting(self.inv1, self.inv2, self.inv3, self.inv4,self.runTotal\_1,self.runTotal\_2, self.deck, self.trueCount1, self.trueCount2, self.playedCardsAI1, self.playedCardsAI2) # initializes aiCounting class

#resets variables

self.Hidden = True

self.playerBetChosen = False

self.playerPlayBet = 0

#player bet

global pCashCounter, pBetCounter

pBetButtonUp10 = Button(tk, text="Increase bet by 10", command = self.betUp10)#creates button to increase bet by total of 10

pBetButtonUp10.place(x=710, y=480)#places button on canvas

pBetButtonUp100 = Button(tk, text="Increase bet by 100", command = self.betUp100)#creates button to increase bet by total of 100

pBetButtonUp100.place(x=830, y=480)#places button on canvas

pBetButtonDown10 = Button(tk, text="Decrease bet by 10", command = self.betDown10)#creates button to decrease bet by total of 10

pBetButtonDown10.place(x=950, y=480)#places button on canvas

pBetButtonDown100 = Button(tk, text="Decrease bet by 100", command = self.betDown100)#creates button to decrease bet by total of 100

pBetButtonDown100.place(x=1080, y=480)#places button on canvas

pBetFinish = Button(tk, text="Confirm Bet", command = self.confirmBet)#creates button to confirm bet

pBetFinish.place(x=845, y=400)#places button on canvas

tk.update()

self.betChange = True #allows bet Change labels to appear

while self.playerBetChosen == False:

if self.betChange == True:

pCashCounter = Label(tk, text="Total Cash Available:"+str(self.playerBet))#sets a label for players total cash

pCashCounter.place(x=730, y=450)#places label

pBetCounter = Label(tk, text="Total Bet for this round:"+str(self.playerPlayBet))#sets a label for players current bet

pBetCounter.place(x=890, y=450)#places label

tk.update()

self.betChange = False #exits if statement

tk.update()

#destroys buttons so game can continue

pBetButtonUp10.destroy()

pBetButtonUp100.destroy()

pBetButtonDown10.destroy()

pBetButtonDown100.destroy()

pBetFinish.destroy()

pBetCounter.destroy()

pCashCounter.destroy()

#finds AIs bet dependant of the AIs true count

if self.trueCount1>=0:

self.modifier = (100+(10\*self.trueCount1))/100

self.ai1PlayBet = abs(round((self.ai1Bet\*0.2) \* self.modifier))#finds bet and rounds to nearest whole positive number

elif self.trueCount1<0:

self.modifier = (100-(10\*self.trueCount1))/100

self.ai1PlayBet = abs(round((self.ai1Bet\*0.2) / (-1\*self.modifier)))#finds bet and rounds to nearest whole positive number

self.ai1Bet = self.ai1Bet - self.ai1PlayBet

if self.trueCount2>=0:

self.modifier = (100+(10\*self.trueCount2))/100

self.ai2PlayBet = abs(round((self.ai2Bet\*0.2) \* self.modifier))#finds bet and rounds to nearest whole positive number

elif self.trueCount2<0:

self.modifier = (100+(10\*self.trueCount2))/100

self.ai2PlayBet = abs(round((self.ai2Bet\*0.2) \* self.modifier))#finds bet and rounds to nearest whole positive number

self.ai2Bet = self.ai2Bet - self.ai2PlayBet

#creates labels to show current Bets

pBetLabel = Label(tk, text="Player Current Bet: " + str(self.playerPlayBet))

ai1BetLabel = Label(tk, text="AI 1 Current Bet: " + str(self.ai1PlayBet))

ai2BetLabel = Label(tk, text="AI 2 Current Bet: " + str(self.ai2PlayBet))

#places labels

pBetLabel.place(x=450, y=10)

ai1BetLabel.place(x=450, y=30)

ai2BetLabel.place(x=450, y=50)

tk.update()

time.sleep(0.5)#time gap to improve visuals

cards.showCardsai1(self.inv2)#displays inventory of inv2

time.sleep(0.75)#time gap to improve visuals

cards.showPCards(self.inv1)#displays inventory of player

time.sleep(0.75)#time gap to improve visuals

cards.showCardsai2(self.inv3)#displays inventory of inv3

time.sleep(0.75)#time gap to improve visuals

#shows dealers Cards with first one being hidden

self.Hidden = True

cards.dealerCards(self.inv4, self.Hidden)

self.dealerHit == False

if self.dealerValue == 21 and self.dealerHit == False:#finds out if dealer has blackjack off start

print("dealer Blackjack")

self.dealerState = "Blackjack" #sets the state of the dealer to Blackjack

#prevents other members from having their go

self.pHand = True

self.ai2Hand = True

self.ai1Hand = True

#ai1 decision

self.ai1Hand = False

self.ai1State = ""

if self.countingChance1 != 1: #if the ai is not undergoing counting

self.DD = False #sets double down to false

while self.ai1Value<17 and self.DD == False and len(self.inv2)<5 and self.ai1Hand == False:

if self.ai1Value == 11 and self.ai1Hit == False: #if the ai1 is 11 and hasnt hit before then will double down

self.HitCard=self.deck[0] #card set from top deck

self.deck.pop(0)#card from top of deck removed

self.inv2.append(self.HitCard)#added card to inv2

self.ai1PlayBet = self.ai1PlayBet\*2#doubles the current ai1 bet

ai1BetLabel.destroy()#destroys old ai1 bet label

#updates ai1 bet label

ai1BetLabel = Label(tk, text="AI 1 Current Bet: " + str(self.ai1PlayBet))

ai1BetLabel.place(x=450, y=50)

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds value of inventories

if int(self.ai1Value) <17:#ai will hit if under17

#hit card

self.HitCard=self.deck[0]#card set from top deck

self.deck.pop(0)#card removed from top of deck

self.inv2.append(self.HitCard)#card added to inv2

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds value of inventories

self.ai1Hit = True #means the AI cant double down

if self.ai1Value == 21 and self.ai1Hit == False: #checks for blackjack

self.ai1State = "Blackjack" #sets ai1 state to Blackjack

if len(self.inv2) == 5 and self.ai1Value<=21: #checks for 5 card charlie

self.ai1State = "5 Card Charlie" #sets ai1 state to 5 card charlier

elif int(self.ai1Value)>21: #checks to see if ai1 is bust

self.ai1State = "Bust" #sets ai1 state to bust

if self.ai1Value == 21 and self.ai1Hit == False: #checks to see if ai1 has a blackjack

print("ai1 Blackjack")

self.ai1State = "Blackjack"#sets the state to Blackjack

self.ai1Hand = True#ends ai1 hand

if self.countingChance1 ==1:#ichecks to see if ai is going to be counting

print("ai1 counting choices")

self.trueCount1 = counting.counting1(self.countingChance1, self.inv1, self.inv2, self.inv3, self.inv4, self.deck, self.trueCount1, self.playedCards)#finds the true count before ai1 actions

print()

print(self.trueCount1)

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds value of inventory

print("before ai1 while loop")

#resets variables

self.ai1Counter = 0

self.ai1Hit = False

while self.ai1Hand == False:

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds values of inventories

print(self.ai1Value)

print(self.ai1State)

self.ai1State = counting.actions(self.ai1Value, self.aceAi1Count, self.trueCount1)#ai decides what action it will do

if self.ai1State == "Hit":

self.HitCard=self.deck[0]#takes card from top of deck

self.deck.pop(0)#remove card from deck

self.inv2.append(self.HitCard)#adds card to inventory

self.playedCardsAI1.append(self.HitCard)

if self.ai1State == "Double Down":

self.HitCard=self.deck[0]#takes card from deck

self.deck.pop(0)#removes it from array

self.inv2.append(self.HitCard)#adds card to inv2

self.playedCardsAI1.append(self.HitCard)

self.ai1PlayBet = self.ai1PlayBet\*2#dubles ai1 bet

ai1BetLabel.destroy()#destroys old bet label

ai1BetLabel = Label(tk, text="AI 1 Current Bet: " + str(self.ai1PlayBet))#creates label for new bet

ai1BetLabel.place(x=450, y=30)#places label

self.ai1State = "Stand"#ends ai1 hand

if len(self.inv2) == 5 and self.ai1Value<=21:

self.ai1State = "5 Card Charlie"#if length of inv is 5 and value less then or equal to 21 then 5 card charlie awarded

self.ai1Hand = True#ends ai1 hand

if self.ai1State == "Stand" or self.ai1Value<=21:#if ai1 stands or busts ends hand

self.ai1Hand = True#ends ai1 hand

cards.showCardsai1(self.inv2)#updates gui to show new ai1 cards

tk.update()

self.ai1Hand = True

#resets all player variables

self.pSplit = False

self.pHand = False

self.HitCard = False

self.pState = ""

self.splitState = ""

#runs functions to make sure they still exist and work

self.playerHit()

self.playerDouble()

pHitButton = Button(tk, text="Hit", command = self.playerHit)#creates button allowing for player to hit

pHitButton.place(x = 810, y=480)#places button

pStandButton = Button(tk, text="Stand", command = self.playerStand)#creates button allowing for player to stand

pStandButton.place(x=840, y=480)#places button

pDoubleButton = Button(tk, text="Double Down", command = self.playerDouble)#creates a button allowing for player to double down

pDoubleButton.place(x=884, y=480)#places button

pSplitButton = Button(tk, text="Split", command = self.playerSplit)#creates a button allowing for player to split if two cards have same value

#finds the values of both cards

self.card1 = self.inv1[0]

self.card2 = self.inv1[1]

self.value1 = self.card1[1]

self.value2 = self.card2[1]

print(self.value1, self.value2)

if self.value1 == self.value2: #compares values of both cards to see if they are the same

pSplitButton.place(x=974, y=480)#places split button if two cards are the same

cards.showPCards(self.inv1)#updates player cards

backG.pack()

tk.update()

self.pCurrentHit = False#reset variable

if self.pValue == 21 and self.HitCard == False:#checks to see if player has a blackjack

self.pState = "Blackjack"#sets pState to Blackjack

self.pHand = True#ends players hand

time.sleep(1)#delays code by 1 second to improve visuals

while self.pHand == False:#creates a loop for player to choose action

if len(self.inv1) == 5 and self.pValue<=21:#if the length of inv1 is 5 and value less than or equal to 21 5 card charlie awarded

self.pState = "5 Card Charlie"#sets state to 5 card charlie

self.pHand = True#ends player hand

#resets variables

self.pHand = False

self.pCurrentHit = False

self.pDDHit = False

tk.update()

if self.pCurrentHit == True:#player chooses hit

print("player is trying to hit")

pDoubleButton.destroy()#destroys double down button

self.HitCard = self.deck[0]#takes one card from top of deck

self.deck.pop(0)#deletes card from top of deck

self.inv1.append(self.HitCard)#adds card to inv1

self.playedCardsAI1.append(self.HitCard)#adds card to playedCardsAI1 for counting

cards.showPCards(self.inv1)#updates gui to show new player card

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds value of new hand

if self.pValue>21:#checks to see if player has gone bust

self.pState = "Bust"#sets state to bust

self.pHand = True#ends players hand

print("Bust")

self.pCurrentHit = False#resets variable

if self.value1 == self.value2:#if hit is chosen when could split it will destroy split button

pSplitButton.destroy()

tk.update()

if self.pDDHit == True:#if player chooses double down

self.HitCard = self.deck[0]#takes card from top of deck

self.deck.pop(0)#removes card from deck

self.inv1.append(self.HitCard)#adds card to inv1

self.playedCardsAI1.append(self.HitCard)#adds card to array for ai1 to count

cards.showPCards(self.inv1)#updates gui to show new player card

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds the new value

#updates the bet to be double

self.playerBet = self.playerBet - self.playerPlayBet

self.playerPlayBet = self.playerPlayBet\*2

pBetLabel.destroy()#destroys old player bet labe;

pBetLabel = Label(tk, text="Player Current Bet: " + str(self.playerPlayBet))#updates bet label

pBetLabel.place(x=450, y=10)#places new label

if self.pValue>21:#checks to see if player goes bust

self.pState = "Bust"#sets state to bust

self.pHand = True#ends players go

print("Bust")

self.pHand = True#ends players go

if self.value1 == self.value2:#if double down when could of split destroys split button

pSplitButton.destroy()

if self.pSplit == True:#if split has been chosen

print(self.inv1)

#destroys split and double down button

pSplitButton.destroy()

pDoubleButton.destroy()

self.playerSplitAction(self.inv1)#sets up split actions

print("after playerSplitAction")

#creates a new bet for the new inv and removes from total player cash

self.splitBet = self.playerPlayBet

self.playerBet = self.playerBet - self.splitBet

splitBetLabel = Label(tk, text="Split Hand Current Bet: " + str(self.splitBet))#creates new bet label

splitBetLabel.place(x = 450, y=70)#places new label

self.pHand = True

self.pSplit = True

clearPlayerCards = PhotoImage(file="Background folder/clearPlayerCards.png")#opens sub folder Background folder to access image

backG.create\_image(897, 630, image=clearPlayerCards)#places image over old player hand due to inv changes

tk.update()

#reste

self.inv1Split = False

self.inv5Split = False

while self.pSplit == True:

tk.update()

cards.showSplitP(self.inv1, self.inv5)#shows new cards

while self.inv1Split == False:#hand 1

tk.update()

if len(self.inv1) == 5 and self.pValue<=21:#checks for 5 card charlie

self.pState = "5 Card Charlie"#sets state to 5 card charlie

if self.pCurrentHit ==True:#when player chooses to hit on first inv

print("player is trying to hit from split")

self.HitCard = self.deck[0]#takes card from top of deck

self.deck.pop(0)#removes card from deck

self.inv1.append(self.HitCard)#adds card to inv1

self.playedCardsAI1.append(self.HitCard)#adds card to array for ai1 counting

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds new value

if self.pValue>21:#checks to see if player is bust

self.pState = "Bust"#sets state to bust

self.pHand = True

print("Bust")

self.inv1Split = True#ends inv1 hand

self.pCurrentHit = False

cards.showSplitP(self.inv1, self.inv5)#shows new cards

self.inv5Split = False

while self.inv5Split == False:#hand 2

tk.update()

if len(self.inv5) == 5 and self.splitValue<=21:#checks for 5 card charlie

self.splitState = "5 Card Charlie"#sets state to 5 card charlie

if self.pCurrentHit ==True:#player chooses to hit on second inv

print("player is trying to hit from split")

self.HitCard = self.deck[0]#takes card from top of deck

self.deck.pop(0)#removes card from top of deck

self.inv5.append(self.HitCard)#adds card to inv5

self.playedCardsAI1.append(self.HitCard)#adds card to array for ai1 counting

self.splitValue = self.splitValues(self.inv5)#finds the value of the new inv

print(self.splitValue)

if self.splitValue>21:#checks to see if inv is bust

self.splitState = "Bust"#sets state to bust

self.inv5Split = True

self.pSplit = False

print("Bust")

self.inv5Split = True#ends inv5 hand

self.pCurrentHit = False

cards.showSplitP(self.inv1, self.inv5)#shows new cards

if self.splitValue>0:

self.pSplit = True

print("out of loops")

if self.value1 == self.value2:#if split button has been placed

try:

pSplitButton.destroy()#destroy split button

except:

pass

#destroys player action buttons

pHitButton.destroy()

pDoubleButton.destroy()

pStandButton.destroy()

tk.update()

#resests ai2 state

self.ai2State = ""

if self.countingChance2 != 2:#checks if ai2 isnt counting

self.DD = False#reset variable

while self.ai2Value<17 and len(self.inv3)<5 and self.DD == False:#checks to see if ai2 hand is over

if self.ai2Value == 11:#double down

#doubles ai2 bet

self.ai2PlayBet = self.ai2PlayBet\*2

ai2BetLabel.destroy()#destroys old bet

ai2BetLabel = Label(tk, text="AI 2 Current Bet: " + str(self.ai2PlayBet))#updates new bet label

ai2BetLabel.place(x=450, y=50)#places new bet label

self.DD = True

if int(self.ai2Value) <=17:#ai will hit

print("ai2 Hitting at:", self.ai2Value)

#hit card

self.HitCard=self.deck[0]#card taken from top of deck

print(self.deck[0])

self.deck.pop(0)#card removed from top of deck

self.inv3.append(self.HitCard)#card added to inv

self.playedCardsAI1.append(self.HitCard)#adds card to playedCardsAI1 for counting

print(self.inv3)

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds new values

print()

print()

print(self.inv3, " ", self.ai2Value)

self.ai2Hit = True#prevents ai2 from getting blackjack state after hitting

if self.ai2Value == 21 and self.ai2Hit == False:#checks for blackjack

self.ai2State = "Blackjack"#sets state to blackjack

if len(self.inv3) == 5 and self.ai2Value<=21:#checks for 5 card charlie

self.ai2State = "5 Card Charlie"#sets state to 5 card charlie

elif int(self.ai2Value)>21:#checks to see if ai has bust

self.ai2State = "Bust"#sets state to bust

self.ai2Hand = False

self.ai2Counter = 0

self.ai2State = ""

if self.ai2Value == 21 and self.ai2Hit == False:#checks for blackjack

print("ai2 Blackjack")

self.ai2State = "Blackjack"

self.ai2Hand = True

if self.countingChance2 ==2:#checks if ai2 will be counting

self.trueCount2 = counting.counting2(self.countingChance2, self.inv1, self.inv2, self.inv3, self.inv4, self.deck, self.trueCount2)#counts current cards using halves method

while self.ai2Hand == False:#while ai2 hand is going

self.values(self.inv1, self.inv2, self.inv3, self.inv4)#finds values of cards

self.ai2State = counting.actions(self.ai2Value, self.aceAi2Count, self.trueCount2)#ai2 chooses action

if self.ai2State == "Hit":#if ai2 hits

self.HitCard=self.deck[0]#card taken from top of deck

self.deck.pop(0)#card removed from top of deck

self.inv3.append(self.HitCard)#card added to inv3

self.playedCardsAI1.append(self.HitCard)#adds card to array for ai1 counting

if self.ai2State == "Double Down":#double down

self.HitCard=self.deck[0]#card taken from top of deck

self.deck.pop(0)#card removed from top of deck

self.inv3.append(self.HitCard)#adds card to ai2 inv

self.playedCardsAI1.append(self.HitCard)#adds card to array for ai1 counting

#doubles ai2 bet

self.ai2PlayBet = self.ai2PlayBet\*2

ai2BetLabel.destroy()#deletes old bet

ai2BetLabel = Label(tk, text="AI 2 Current Bet: " + str(self.ai2PlayBet))#updates new bet label

ai2BetLabel.place(x=450, y=50)#places label

self.ai2State = "Stand"#ends ai2 hand

if self.ai2State == "Stand" or self.ai2Value<=21:#if ai2 hand should end

self.ai2Hand = True#ends ai2 hand

if len(self.inv3) == 5 and self.ai2Value<=21:#checks ai2 for 5 card charlie

self.ai2State = "5 Card Charlie"#sets state to 5 card charlie

cards.showCardsai2(self.inv3)#updates gui to show new cards

tk.update()

#resets variables

self.dealerState = ""

time.sleep(2)

self.Hidden = False

self.playedCardsAi2 = []

cards.dealerCards(self.inv4, self.Hidden)

print("poopoo")

self.dealerHit = False

self.dealerHand = False

self.dealerCounter = 1

self.dHand = False

while self.dealerValue<17 and self.dHand == False:#dealer must hit if less then 17 or hand hasnt finished

if int(self.dealerValue)<17:

#hit card

self.HitCard=self.deck[0]#card taken from top of deck

self.deck.pop(0)#card removed from top of deck

self.inv4.append(self.HitCard)#card added to inv4

self.playedCardsAi2.append(self.HitCard)#card added to array for ai1 counting

self.dealerValueCounter(self.inv4)#finds the value for the dealer

self.dealerHit = True#prevents blackjack

cards.dealerCards(self.inv4, self.Hidden)#updates gui to show new dealer cards

tk.update()

if len(self.inv4) == 5:#prevents dealer from having more then 5 cards

self.dHand = True#ends dealers hand

if self.dealerValue == 21 and self.dealerHit == False:#checks for blackjack

self.dealerHand = True#ends dealer hand

self.dealerState = "Blackjack"#sets state to blackjack

elif int(self.dealerValue)>21:#checks for bust

self.dealerHand = True#ends dealers hand

self.dealerState = "Bust"#sets state to bust

if self.dealerValue>16:#if value greater then 17

self.dealerHand = True#ends dealer hand

cards.dealerCards(self.inv4, self.Hidden)#updates gui to show new dealer cards

#ai1

if self.ai1State == "5 Card Charlie":#checks to see if ai1 has a 5 card charlie

self.ai1Bet = self.ai1Bet+2\*(self.ai1PlayBet)#updates bet as won

ai1StateLabel = Label(tk, text="5 Card Charlie")#changes label to 5 card charlie

#5 CARD CHARLIE

elif self.ai1State == "Blackjack" and self.dealerState == "Blackjack":#checks to see if a blackjack push has occured

self.ai1Bet = self.ai1Bet+self.ai1PlayBet#bet goes back to original

#PUSH

ai1StateLabel = Label(tk, text="Push")#changes label to push

elif self.ai1State == "Blackjack":#checks to see if ai1 has a blackjack

self.ai1BlackjackBet = self.ai1PlayBet + int(round((self.ai1PlayBet/2)))#pays out bet 3/2

self.ai1Bet = self.ai1PlayBet+ self.ai1Bet+self.ai1BlackjackBet

ai1StateLabel = Label(tk, text="Blackjack")#changes label to blackjack

#BLACKJACK

elif self.dealerState == "Blackjack":#checks to see if dealer has a blackjack

ai1StateLabel = Label(tk, text="Dealer Blackjack")#updates label to dealer blackjack

#DEALER WON

elif self.dealerValue>21 and self.ai1Value<=21:#checks to see if dealer bust and ai1 hasnt

#DEALER BUST

self.ai1Bet = self.ai1Bet+2\*(self.ai1PlayBet)#ai wins round and gets double the bet back

ai1StateLabel = Label(tk, text="Dealer Bust")#changes label to dealer Bust

elif self.ai1Value<self.dealerValue and self.dealerValue<22:#checks to see if ai1 value is less then dealer value

#DEALER HIGHER

ai1StateLabel = Label(tk, text="Dealer Higher")#set label to dealer high

elif self.ai1Value == self.dealerValue and (self.ai1Value<=21 or self.dealerValue<=21):#if the dealer and ai have same value while being less then or equal to 21

self.ai1Bet = self.ai1Bet+self.ai1PlayBet#bet goes back to original

ai1StateLabel = Label(tk, text="Push")#changes label to push

#PUSH

elif self.ai1Value>self.dealerValue and self.ai1Value<=21:#if the ai is higher then the dealer while being less then or equal 21

self.ai1Bet = self.ai1Bet+ 2\*(self.ai1PlayBet)#ai wins round and gets double bet back

ai1StateLabel = Label(tk, text="Higher than Dealer")#changes label to higher then dealer

#BEAT DEALER

elif self.ai1Value>21 or self.ai1Value == "Bust":#if the ai has gone bust

ai1StateLabel = Label(tk, text="Bust")#changes label to bust

#BUST

self.ai1Value = 0

#player

if self.pState == "5 Card Charlie":#checks to see if player has 5 card charlie

self.playerBet = self.playerBet+(2\*self.playerPlayBet)#changes bet to won

pStateLabel = Label(tk, text="5 Card Charlie")#sets label to 5 card charlie

#5 CARD CHARLIE

elif self.pState == "Blackjack" and self.dealerState == "Blackjack":#checks if blackjack push occurs

self.playerBet = self.playerBet+self.playerPlayBet#bet goes back to original

pStateLabel = Label(tk, text="Push")#label set to push

#PUSH

elif self.pState == "Blackjack":#checks if player has blackjack

self.playerBlackjackBet = self.playerPlayBet + (self.playerPlayBet/2)#bet payed 3/2

self.playerBet = int(self.playerBet) +int(self.playerPlayBet)+int(self.playerBlackjackBet)

pStateLabel = Label(tk, text="Blackjack")#set label to blackjack

#PLAYER BLACKJACK

elif self.dealerState == "Blackjack":#if dealer has blackjack

pStateLabel = Label(tk, text="Dealer Blackjack")#sets label to dealer blackjack

self.playerBet = self.playerBet#bet stays the same

#DEALER BLACKJACK

elif self.dealerValue>21 and self.pValue<=21:#checks to see if dealer has bust while player value less then 22

#DEALER BUST

self.playerBet = self.playerBet+2\*(self.playerPlayBet)#bet changed to won

pStateLabel = Label(tk, text="Dealer Bust")#set label to dealer bust

elif self.pValue<self.dealerValue and self.dealerValue<22:#if dealer is higher then player value and less then 22

#DEALER HIGHER

pStateLabel = Label(tk, text="Dealer Higher")#set label to dealer higher

self.playerBet = self.playerBet#bet stays the same

elif self.pValue == self.dealerValue and (self.pValue<=21 or self.dealerValue<=21):#checks to see if push

self.playerBet = self.playerBet+self.playerPlayBet#bet set back to original

pStateLabel = Label(tk, text="Push")#label set to push

#PUSH

elif self.pValue>self.dealerValue and self.pValue<=21:#checks to see if player value higher then dealer while less then 22

self.playerBet = self.playerBet+ 2\*(self.playerPlayBet)#bet changed to won

pStateLabel = Label(tk, text="Higher than Dealer")#label set to higher than dealer

#BEAT DEALER

elif self.pValue>21 or self.pValue == "Bust":#checks if player has bust

pStateLabel = Label(tk, text="Bust")#changes label to bust

self.playerBet = self.playerBet#bet stays the same

#BUST

self.pValue = 0

int(self.playerBet)

#ai2

if self.ai2State == "5 Card Charlie":#checks to see if ai2 has a 5 card charlie

self.ai2Bet = self.ai2Bet+2\*(self.ai2PlayBet)#updates bet as won

ai2StateLabel = Label(tk, text="5 Card Charlie")#changes label to 5 card charlie

elif self.ai2State == "Blackjack" and self.dealerState == "Blackjack":#checks to see if a blackjack push has occured

self.ai2Bet = self.ai2Bet+self.ai2PlayBet#bet goes back to original

ai2StateLabel = Label(tk, text="Push")#changes label to push

#PUSH

elif self.ai2State == "Blackjack":#checks to see if ai2 has a blackjack

self.ai2BlackjackBet = self.ai2PlayBet + (self.ai2PlayBet/2)#pays out bet 3/2

self.ai2Bet = self.ai2PlayBet+self.ai2Bet+self.ai2BlackjackBet

ai2StateLabel = Label(tk, text="Blackjack")#changes label to blackjack

#AI2 BLACKJACK

elif self.dealerState == "Blackjack":#checks to see if dealer has a blackjack

ai2StateLabel = Label(tk, text="Dealer Blackjack")#updates label to dealer blackjack

#DEALER BLACKJACK

elif self.dealerValue>21 and self.ai2Value<=21:#checks to see if dealer bust and ai2 hasnt

#DEALER BUST

self.ai2Bet = self.ai2Bet+2\*(self.ai2PlayBet)#ai wins round and gets double the bet back

ai2StateLabel = Label(tk, text="Dealer Bust")#changes label to dealer Bust

elif self.ai2Value<self.dealerValue and self.dealerValue<22:#checks to see if ai2 value is less then dealer value

#DEALER HIGHER

ai2StateLabel = Label(tk, text="Dealer Higher")#set label to dealer high

elif self.ai2Value == self.dealerValue and (self.ai2Value<=21 or self.dealerValue<=21):#if the dealer and ai have same value while being less then or equal to 21

self.ai2Bet = self.ai2Bet+self.ai2PlayBet#bet goes back to original

ai2StateLabel = Label(tk, text="Push")#changes label to push

#PUSH

elif self.ai2Value>self.dealerValue and self.ai2Value<=21:#if the ai is higher then the dealer while being less then or equal 21

self.ai2Bet = self.ai2Bet+ 2\*(self.ai2PlayBet)#ai wins round and gets double bet back

ai2StateLabel = Label(tk, text="Higher than Dealer")#changes label to higher then dealer

#BEAT DEALER

elif self.ai2Value>21 or self.ai2Value == "Bust":#if the ai has gone bust

ai2StateLabel = Label(tk, text="Bust")#changes label to bust

#BUST

if self.pSplit == True:#if player has chosen to split

if self.splitState == "5 Card Charlie":#checks for 5 card charlie

self.playerBet = self.playerBet+2\*(self.playerPlayBet)#changes bet to won

pStateLabel = Label(tk, text="5 Card Charlie")#sets label to 5 card charlie

elif self.dealerState == "Blackjack":#checks if dealer has blackjack

splitStateLabel = Label(tk, text="Dealer Blackjack")#changes label to dealer blackjack

self.playerBet = self.playerBet#bet stays the same

#DEALER BLACKJACK

elif self.dealerValue>21 and self.splitValue<=21:#checks to see if dealer has bust while split value less then 22

#DEALER BUST

self.playerBet = self.playerBet+2\*(self.splitBet)#changes bet to won

splitStateLabel = Label(tk, text="Dealer Bust")#sets label to dealer bust

elif self.splitValue<self.dealerValue and self.dealerValue<22:#checks to see if dealer value higher

#DEALER HIGHER

splitStateLabel = Label(tk, text="Dealer Higher")#sets label to dealer higher

self.playerBet = self.playerBet#bet stays the same

elif self.splitValue == self.dealerValue and (self.splitValue<=21 or self.dealerValue<=21):#checks to see if a push has occured underneath 22

self.playerBet = self.playerBet+self.splitBet#bet returns to normal

splitStateLabel = Label(tk, text="Push")#label set to push

#PUSH

elif self.splitValue>self.dealerValue and self.splitValue<=21:#checks if split value is less then 22 but higher then dealer

self.playerBet = self.playerBet+ 2\*(self.splitBet)#changes bet to won

splitStateLabel = Label(tk, text="Higher than Dealer")#sets label as higher then dealer

#BEAT DEALER

elif self.splitValue>21 or self.splitValue == "Bust":#checks if bust

splitStateLabel = Label(tk, text="Bust")#changes label to bust

self.playerBet = self.playerBet#bet stays the same

self.ai2Value = 0

self.x1 = 850#sets x coord for player incase of split

if self.pSplit==True:#checks if player has split

self.x1 = self.x1-90#moves x coord for player label back 90 pixels

splitStateLabel.place(x=1050, y=400)#places split label

ai1StateLabel.place(x=1250, y=400)#places ai1 label

pStateLabel.place(x=self.x1,y=400)#places player label

ai2StateLabel.place(x=250, y=400)#places ai2 label

tk.update()

time.sleep(5)#adds a timer to improve visuals

#removes labels from being seen

ai1StateLabel.lower()

pStateLabel.lower()

ai2StateLabel.lower()

if self.pSplit==True:#if player has split

splitStateLabel.lower()#moves label so cannot be seen

tk.update()

#ai do endgame counting for what cards have been dealt after initial count

if self.countingChance1 ==1:

print(self.trueCount1)

self.trueCount1 = counting.ai1EndgameCount(self.playedCardsAI1, self.trueCount1, self.deck)

self.runTotal\_1 = counting.getRunTotal\_1()

print(self.trueCount1)

if self.countingChance2 ==2:

print(self.trueCount2)

self.trueCount2 = counting.ai2EndgameCount(self.playedCardsAi2, self.trueCount2, self.deck)

self.runTotal\_2 = counting.getRunTotal\_2()

print(self.trueCount2)

print(len(self.deck))

time.sleep(1)#adds timer to improve visuals

inv = Inventory()

if self.splitValue>0:#if player split

splitBetLabel.destroy()#destroys split label

pBetLabel.destroy()#destroys player label

ai1BetLabel.destroy()#destroys ai1 label

ai2BetLabel.destroy()#destorys ai2 label

tk.update()

print(len(self.deck))

if len(self.deck)<104:#checks to see if length of deck is less then half of original lenght

self.deck = cards.deckingCards()#reshuffles deck with all cards

self.trueCount1 = 0#resets trueCount1

self.trueCount2 = 0#resets trueCount2

self.deck = inv.startInv(self.deck)#sets up inventories for new round

#gets inventories from inventory class

self.inv1 = inv.getInv1()#inv1

self.inv2 = inv.getInv2()#inv2

self.inv3 = inv.getInv3()#inv3

self.inv4 = inv.getInv4()#inv4

#finds values of new inventories

self.values(self.inv1, self.inv2, self.inv3, self.inv4)

self.dealerValueCounter(self.inv4)

gameLabel = Label(tk, text="You have reached over 50,000!\nWell done!\nPress the play button to play again")#if player reaches 50000 cash, this label will be set

gameLabel.place(x=850, y=350)#label will be placed and game will need to be played again to progress

tk.mainloop()

def roundStart(self, countingChance1, countingChance2, inv1, inv2, inv3, inv4, deck):

self.countingChance1 = random.randint(1,1)#decides if AIs will do counting methods for the rest of the game

print(self.countingChance1)

self.countingChance2 = random.randint(2,2)

print(self.countingChance2)

#resets variables and arrays

self.trueCount1 = 0

self.trueCount2 = 0

self.playedCards = []

self.inv5 = []

self.handClear = PhotoImage(file="Background folder/clearAllCards.png")#accesses Background folder sub folder to assign an image

backG.create\_image(0,444,image=self.handClear,anchor=NW) #places image to cover any cards from player or A1

self.dealerClear = PhotoImage(file = "Background folder/clearDealerCards.PNG")#accesses Background folder sub folder to assign an image

backG.create\_image(601, 60, image=self.dealerClear,anchor=NW) #places image to cover any dealer cards

cards = Cards(inv1, inv2, inv3, inv4, self.inv5) #initializes Cards Class

#creates 4 shuffled deck of cards added together

self.deck = cards.deckingCards()

self.deckFull = self.deck #sets deckFull variable to same contents as deck(full length of deck)

inv = Inventory() #initializes inventory class

self.deck = inv.startInv(cards.deckOfCards) #adds 2 cards to each inv

self.inv1 = inv.getInv1()#gets inv1 from inv class

self.inv2 = inv.getInv2()#gets inv2 from inv class

self.inv3 = inv.getInv3()#gets inv3 from inv class

self.inv4 = inv.getInv4()#gets inv4 from inv class

self.values(self.inv1, self.inv2, self.inv3, self.inv4) # finds the value of each inventory

#dealer card Values

self.dealerCardH = self.inv4[0] # finds the hidden card out of dealer inv

self.dealerCardS = self.inv4[1] # finds the shown card out of dealer inv

self.dealerValueH = self.dealerCardH[1]#finds the value of the hidden card

self.dealerAceHand = False #sets booleans to false so no false positives occur

self.dealerdoubleAce = False

if self.dealerValueH == "k" or self.dealerValueH == "q" or self.dealerValueH == "j" or self.dealerValueH =="0": #if dealer gets a face or 10 card, value is set to 10

self.dealerValueH = 10

elif self.dealerValueH == "a":# sets the value of shown card to 11 and dealerAceHand to True if ace is the hidden dealer card

self.dealerValueH = 11

self.dealerAceHand = True

self.dealerValueS = self.dealerCardS[1]

if self.dealerValueS == "k" or self.dealerValueS == "q" or self.dealerValueS == "j" or self.dealerValueS =="0": #if dealer gets a face or 10 card, value is set to 10

self.dealerValueS = 10

elif self.dealerValueS == "a":# sets the value of shown card to 11 and dealerAceHand to True if ace is the shown dealer card

self.dealerValueS = 11

self.dealerAceHand = True

elif self.dealerValueS == 11 and self.dealerValueH == 11:#sets dealerdoubleAce to true and dealerAceHand to false if dealer inv contains two aces

self.dealerAceHand = False

self.dealerdoubleAce = True

self.dealerValue = int(self.dealerValueS)+int(self.dealerValueH) #finds the total value of the dealers card

if self.dealerValue>21 and self.dealerAceHand == True: #if the dealer has an ace and is somehow bust it will remove 10 so isnt bust off start

self.dealerValue = self.dealerValue-10

elif self.dealerdoubleAce == True: # if the dealer has two aces it will remove 10 off the total value, so not bust instantly

self.dealerValue = self.dealerValue-10

#used to return all inventorys and values of inventorys if needed

def getRInv1(self):

return self.inv1

def getRInv2(self):

return self.inv2

def getRInv3(self):

return self.inv3

def getRInv4(self):

return self.inv4

def getRValue1(self):

return self.pValue

def getRValue2(self):

return self.ai1Value

def getRValue3(self):

return self.ai2Value

def getRValueH4(self):

return self.dealerValueH

def getRValueT4(self):

return self.dealerValue

class Inventory():

def \_\_init\_\_(self):# initializes class by defining all inventorys as empty

self.inv1 = []

self.inv2 = []

self.inv3 = []

self.inv4 = []

#creates inventory of all players by adding card to each inventory twice and removes each card from deck

def startInv(self, deckOfCards):

self.n = deckOfCards

for i in range (0,2): #2 cards for each inventory

self.inv1.append(self.n[0]) #adds card to inv1

self.n.pop(0)#removes card from deck

self.inv2.append(self.n[0]) #adds card to inv2

self.n.pop(0)#removes card from deck

self.inv3.append(self.n[0]) #adds card to inv3

self.n.pop(0)#removes card from deck

self.inv4.append(self.n[0]) #adds card to inv4

self.n.pop(0)#removes card from deck

return deckOfCards

#returns all inventorys to the main part of program

def getInv1(self):

return self.inv1

def getInv2(self):

return self.inv2

def getInv3(self):

return self.inv3

def getInv4(self):

return self.inv4

def exitGame():

#closes the game

exit()

def backgroundTotal():

#creating the background of the GUI

global background

global JackBlack

global PausePicture

global backG

global pauseButton

global tk

b1.destroy()#destroys start game button

b2.destroy()#destroys load game button

b3.destroy()#destroys exti game button

backG = Canvas(tk, width=1800, height=720)#sets the new canvas dimensions

backG.pack()

tk.title("JackBlack: A Blackjack Game")

background = PhotoImage(file='Background folder/Improved Background for nea.png')#opens Background folder to access the background table for the game

backG.create\_image(0,0, image=background, anchor=NW) #places the background table on the canvas

#Adding in a picture of Jack Black (The AI opponent)

JackBlack = PhotoImage(file='Background folder/Jack Black AI Picture 50x50.png')#opens Background folder to access image for the dealer

backG.create\_image(875,0, image=JackBlack, anchor=NW) #places image above dealer hand to simulate the dealer

#adds pause button to the top right of the game

pausePicture = PhotoImage(file='Background folder/Pause logo.png') #accesses Background folder sub folder to access image for button

pauseButton = Button(tk, image=pausePicture, command=openNewMenu, anchor ='w', borderwidth=0)#creates a button to open a new pause menu, button will appear as an image

pauseButtonWindow = backG.create\_window(5,5, anchor ='nw', window=pauseButton) #places pause button on canvas

startPicture = PhotoImage(file='Background folder/Start logo.png') #opens Background folder subfolder to access image for button

startButton = Button(tk, image=startPicture, command=gameplay, anchor ='w',borderwidth = 0) #creates a button to carry out gameplay function and appear as an image

startButton.place(x=80, y=5) #places start button on canvas

deckPicture = PhotoImage(file='playingCards/BackOfCard150x211.png') #accesses back of card image from playingCards sub folder

backG.create\_image(1350, 15, image = deckPicture, anchor=NW) #places image to simulate stack of cards

tk.mainloop()

def gameplay():

handClear = PhotoImage(file="Background folder/clearAllCards.png") #accesses image from Background folder sub folder

backG.create\_image(0,444,image=handClear,anchor=NW) #places image ontop of player and AI hands so cards dissappear

dealerClear = PhotoImage(file = "Background folder/clearDealerCards.PNG")#accesses image from Background folder sub folder

backG.create\_image(601, 60, image=dealerClear,anchor=NW) #places image ontop of dealer hand so cards dissappear

playerInv = [] #sets all inventorys to empty so inventorys dont carry over from other games

aiInv1 = []

aiInv2 = []

dealerInv = []

deckCards = []

gameRound = Rounds(playerInv, aiInv1, aiInv2, dealerInv, deckCards) #initializes Rounds Class

gameRound.gameCycle(0, 0, playerInv, aiInv1, aiInv2, dealerInv, deckCards) #Runs gameCycle method within Rounds Class

def loadGame():

global loadBoolean #sets variable to be global so can be accessed within classes

loadBoolean = True #sets boolean to true so game will be loaded from file once started

backgroundTotal() #automatically runs the background loading part of game

def openNewMenu():

#this def creates a new menu when the pause button is clicked

global nb1 #sets variables to be global so can be accesed in different functions

global nb2

global nb3

newMenu = Toplevel(tk)

newMenu.title("Menu") #changes name of window to Menu

nb1 = Button(newMenu, text="Start New Game", command=gameplay).grid()#creates a button that will start the game again

nb3 = Button(newMenu, text="Exit Game", command=exitGame).grid() #creates a button which will quit the game

def menu():

#creates a menu to either start, load or quit game

global canvas #sets variables to be global so can be accesed in different functions

global b1

global b2

global b3

canvas = Canvas(tk, width=300, height=400) #creates a canvas for gui to appear

tk.title("Menu") #changes name of window to Menu

b1 = Button(tk, text="Start Game", command=backgroundTotal) #creates a button that will start the game

b1.pack()

b2 = Button(tk, text="Load Game", command=loadGame) #creates a button which will allow for a previous save game to be loaded

b2.pack()

b3 = Button(tk, text="Exit Game", command=exitGame) #creates a button which will quit the game

b3.pack()

loadBoolean = False #sets the load function

menu() #runs the menu function

Testing

# Test Tables

|  |  |
| --- | --- |
| Menu  Testing to see if the menu will load | |
| Expected – A small canvas with 3 buttons:   * “Start Game” * “Load Game” * “Exit Game” | Outcome – |
| Notes: -Menu is extremely small but still shows all 3 Buttons | Worked:  🗸 |
| Background  Testing to see if the background and images will load once game has been started | |
| Expected – “Improved Background for nea.png” should be background, along with images and buttons to start and load another menu | Outcome - |
| Notes: - All items have loaded correctly and to the correct size. | Worked:  🗸 |
| New Menu  Testing to see if the new menu would load | |
| Expected – A new menu will load in its own separate window, not interfering or blocking/halting the code. | Outcome - |
| Notes: - Save game has been removed due to the game not starting yet so nothing is able to be saved. | Worked:  🗸 |
| gameplay  This is the function that runs after the play button has been clicked, it should initialise decks and ready the first round, allowing for player to decide their bet | |
| Expected – A range of buttons and labels to be shown once the player has clicked the play button at top left of screen | Outcome - |
| Notes: Worked as expected, allowing for player to now choose their bet for their first round | Worked:  🗸 |
| Increase Bet by 10  The player needs to be able to bet in this game so it is essential that the player can increase the bet and see how much they have and how much they are betting | |
| Expected – The total cash to be decreased by 10 and bet to increase by 10, being updated on the GUI | Outcome - |
| Notes: - Hard to see on table but the total cash amount has been decreased by 10 and the bet has been increased by 10 | Worked:  🗸 |
| Increase Bet by 100  The player needs to be able to bet in this game so it is essential that the player can increase the bet and see how much they have and how much they are betting | |
| Expected - The total cash to be decreased by 100 and bet to increase by 100, being updated on the GUI | Outcome - |
| Notes: - The total cash amount has been decreased by 100 and the bet has been increased | Worked:  🗸 |
| Decrease Bet by 10  The player needs to be able to change their bet in case the player can’t or doesn’t want to bet as much so you must be able to decrease the bet and update the total cash they own | |
| Expected – The total cash should be increased by 10 while the bet should decrease by 10 | Outcome - |
| Notes: - The total cash gets increased by 10 and the bet decreases by 10 | Worked:  🗸 |
| Decrease Bet by 100  The player needs to be able to change their bet in case the player can’t or doesn’t want to bet as much so you must be able to decrease the bet and update the total cash they own | |
| Expected – The total cash should be Increased by 100 while the bet should decrease by 100 | Outcome – |
| Notes: - The expected outcome worked however it didn’t change the bet label but instead placed a new label over the top of the old one, meaning if the new label was a shorter length than the old label, the old label would still be visible. | Worked:  ~  It met expectations but not to the extent I need |
| Decrease Bet by 100 v2  The player needs to be able to change their bet in case the player can’t or doesn’t want to bet as much so you must be able to decrease the bet and update the total cash they own | |
| Expected – The total cash should be Increased by 100 while the bet should decrease by 100 without previous label from being visible | Outcome - |
| Notes: - I changed the function so when the bet is increased or decreased, it will destroy the old label before placing the new label, meaning the old label is no longer visible | Worked:  🗸 |
| AI bets  AI should bet their own bets after the player chooses their own bet | |
| Expected: - Player, AI 1 and AI 2 bets would be placed on separate labels to be able to be seen by the player on GUI, the AI should only bet 1000 due to it being the start of the game | Outcome - |
| Notes: - This used to be placed at the top right of the GUI however when I was programming on a smaller monitor with a lower resolution then the program needed the bet labels were cut from vision, so I moved them to the left side of the GUI next to the play button which also looks more pleasant for the user. | Worked:  🗸 |
| Betting Limit – Out of Cash  If the player is to bet when their total cash is 0, this should stop them from doing so as a bet is needed to play and win the game | |
| Expected: - The total cash should stay as 0 and should not decrease, whereas the current bet should not increase in number | Outcome - |
| Notes: - “ur broke lol” is only a console prompt to allow me to test this feature, it is printed whenever the bet 10 button is pressed to confirm this is working, where the total cash isn’t decreasing etc. | Worked:  🗸 |
| Betting Limit – No Bet  The player needs to bet in order to increase/decrease their progress to the goal of 50000, so a prompt must tell the player to bet | |
| Expected: - If the player tries to bet with 0 bet, then a prompt will appear to tell the user to place a bet. | Outcome - |
| Notes: - This appears directly over the confirm bet button and freezes the GUI for 3 seconds, giving the user enough time to read the prompt | Worked:  🗸 |
| Deck  An array of 4 decks of 52 cards needs to be made which can then be used to deal out cards into inventories and calculate true counts | |
| Expected – An array of cards should be printed in the console in a shuffled random order, there should be 200 cards as 8 cards have already been given to inventories | Outcome - |
| Notes: - The total length of an array of 4 decks of 52 cards should be 208 however 8 cards have already been dealt to the inventories from the start of round   * This also means that 8 cards have already successfully been sent into the inventories, allowing them to be shown on the table | Worked:  🗸 |
| Cards on table  The cards must be able to be seen on the GUI otherwise the player won’t be able to choose an action due to not knowing their own and other inventories | |
| Expected: - 2 Cards should be shown per each inventory with the dealers first card being hidden by the back of its own card | Outcome: |
| Note – Usually you will not see the table like this as the AI1 (right side hand) would have already played, possible showing 1-3 more cards then expected, however it could still appear like this if the AI 1 was to have a high value hand | Worked:  🗸 |
| AI actions  Before the player can choose their own actions, the AI on the right must be able to choose their own actions before standing at a suitable value (depending on true count) or going bust | |
| Expected – The AI1 should hit a minimum of 1 time to prove it will hit and stand at a suitable value | Outcome: |
| Notes: The AI initially started off with the King and Three before hitting and receiving a 6, putting the value of AI1 inventory to 19, which is set to a value where the AI should stand  AI 2 will also work the same way so doesn’t need to be tested, only difference being is the counting method(if either AI will be counting at all) | Worked:  🗸 |
| Player Actions  To be able to play the game a player must be able to interact with the game, which is where the list of buttons a player can interact with should appear | |
| Expectations: - A list of 4 buttons should appear:   * Hit * Stand * Double down * Split   Split should only appear if the player owns two cards with identical values | Outcome: |
| Notes: I had to directly edit the code to make the players inventory have two cards which have identical values, as otherwise it could take a long time to test this naturally due to the randomness involved. | Worked:  🗸 |
| Player Actions – Split  When a player is given an option to split, their bet must double and be put into two different hands, as split effectively is playing as two different hands. | |
| Expectation – Once split button is pressed, the inventory should separate showing the two cards on separate sides of player’s section   * Another bet label will appear for the split inventory | Outcome: |
| Notes: The Hit and stand button remain while the split and double down buttons disappear, I didn’t want the player to have too high of an advantage when it comes to splitting due to the planned addition of 5 card Charlie into the game | Worked:  🗸 |
| Player Action – Hit  If the player is not satisfactory with the value of their inventory a hit action should occur, where a card is taken and removed from the deck and placed into the inventory of the player | |
| Expected: A new card to appear in the players hand on the GUI | Outcome: |
| Notes:  This is still from the same split hand from earlier, however the hit principle works the exact same whether the player has chosen to split or not, except that when not in split cards are added horizontally rather than vertically | Worked:  🗸 |
| Player Action – Stand  Once the player has reached a suitable inventory value they may choose to stand, ending their go of their hand (if player has split, they will go to their next split hand until they bust or stand on that one) | |
| Expected: - The player to have hit at least once and stood, showing that they have decided to end their go | Outcome: |
| Notes:  The player has hit once and decided to stand, ending their go and deleting all buttons which would control the player’s actions. | Worked:  🗸 |
| Player Action – Double Down  If a player is feeling lucky, they might choose to double down, which means doubling their current bet they have placed, before receiving one final card and then standing immediately afterwards. This cannot be done after the player has already hit or split in this game. | |
| Expectation: - One card will be shown next to the two cards the player has been given already   * The players bet of 100 will increase to 200 and updates the label for bet | Outcome: |
| Notes:  This shows that the player had an original value of 12, doubled down, received an 8-card making their inventory a total of 20, and then their hand automatically ending due to the double down | Worked:  🗸 |
| Dealer Shows Card  Once all AIs and player have ended their hands, the dealer is allowed to play their hands, and to start this off they must reveal their hole card. | |
| Expectation:   * Dealer to have two cards showing both their values | Outcome: |
| Notes:  This occurs immediately after the end of AI2s hand | Worked:  🗸 |
| Dealer must stand on 17  Dealer must play by a specific set of rules, where they must stand if they have a higher value than 17, and if not, they must hit | |
| Expectation: - The dealer must have a hand that shows a value of over 17 in which they either stand or bust. | Outcome: |
| Notes: - This shows that the dealer had an outcome of 20, which is greater then 17, allowing the dealer to stand, ending the all the of the round’s hands | Worked:  🗸 |
| Compare Values  At the end of every round, the values of each inventory must be compared to the dealers to see whether they have won the round. | |
| Expected: - A set of 3 labels (4 if a split) to showing the result of the round | Outcome: |
| Notes: The first image is a normal hand, whereas the second image is a test hand for splitting the hand, however both images show labels for the states they have with the dealer | Worked:  🗸 |
| Bet Pay-out  For the player to win they will need to be paid out in bet, otherwise there would be no way to reach 50,000. However, they must also be able to lose bets if they go bust or under the dealer. | |
| Evaluation: - This will show a pay out in when a player splits after betting 100, in one hand the player will go bust and the other hand the player will beat the dealer. | Outcome: |
| Notes:  This shows the overall cash gain is 0 as 100 was lost in the bust, another 100 was put down as the split hand bet, then 100 was won from beating the dealer, leaving the player back at 5000 total cash | Worked:  🗸 |
| Blackjack  A blackjack is when a face/10 card and an ace card has been dealt naturally off the deck from the start of the round. | |
| Expectation – When the player bets 1000 to get a blackjack, their hand automatically will end and their bet will be paid out 3:2 (1500 gain), unless the dealer also has a blackjack. | Outcome: |
| Notes:  The game recognised as a blackjack and the hand ended immediately, however the payout of the blackjack is irregular, due to it missing 1000, what has happened is I have forgot to add the original player bet back to the total and only added the 3:2 pay out | Worked:  x |
| Blackjack v2  A blackjack is when a face/10 card and an ace card has been dealt naturally off the deck from the start of the round. | |
| Expectation – When the player bets 1000 to get a blackjack, their hand automatically will end and their bet will be paid out 3:2 (1500 gain), unless the dealer also has a blackjack. | Outcome: |
| Notes:  The payout has been fixed, however the total cash is no longer an integer and now is a float, so to fix this I just need to make the total pay-out an integer rather then a float | Worked:  ~ |
| Blackjack v3  A blackjack is when a face/10 card and an ace card has been dealt naturally off the deck from the start of the round. | |
| Expectation – When the player bets 1000 to get a blackjack, their hand automatically will end and their bet will be paid out 3:2 (1500 gain), unless the dealer also has a blackjack. | Outcome: |
| Notes:  Total cash is now in an integer and bet is being paid out properly. | Worked:  🗸 |
| In game Menu  This is where the player can restart the game again (even though there is a restart game button next to the menu button) or exit the game formally. | |
| Expected outcome – Once the menu has been clicked, 3 buttons should appear:   * Start New Game * Exit Game | Outcome: |
| Notes:   * Start New Game has same function as the button next to it, but still wanted to have this menu in the game * This was going to have a save game feature however it would have been better to save the game once ever start of the round rather than manually save it | Worked:  🗸 |
| Save Game  Once every round starts, the game should save the current cash of player, true counts for both ai and inventories in one file and in the other file should be the entire deck. | |
| Expected:  Should save the outcome of two rounds of blackjack, where the player shouldn’t have 5000 score due to fluctuating bet | Outcome: |
| Notes:  This shows the bets from the current game and save files are the exact same, showing the game has just been saved | Worked:  🗸 |
| Load the Game  If the player wanted to pick up where they left off from the previous session, then they will be able to after pressing the load function from the menu from start-up | |
| Expected:  The game will be opened via load game button. The total cash and decks should be the same as within both save files | Outcome: |
| Notes: Another game was played to make the scores different to the previous test  The decks from the console and file share the same order, proving they are both the same  The total cash from the file and game are also the same, proving the load worked | Worked:  🗸 |
| AI1 Counting  AI 1 will use the Hi-Lo Counting method to try and predict whether they will receive high or low value cards. | |
| Expected: - AI 1 will correctly identify the true count for its first full round | Outcome: |
| Notes: Hi-Lo counts anything less then 7 a low value card (+1) and anything greater than 10 a high value card (-10)  This isn’t correct as I reset the running count to 0 each round. | Worked:  x |
| AI1 Counting v2  AI 1 will use the Hi-Lo Counting method to try and predict whether they will receive high or low value cards. | |
| Expected: - AI 1 will correctly identify the true count for its first full round | Outcome: |
| Notes: -The true count from AI is said to be 0.7959, the actual true count is 0.7959 | Worked: 🗸 |
| AI2 Counting  AI2 will use the Halves Counting method to try and predict whether they will receive high or low value cards | |
| Expected: - AI 2 will correctly identify the true count for its first full round | Outcome: |
| Notes: This is using the same hand as previous test  Ai true count = 0  Actual true count = 0 | Worked: 🗸 |

Evaluation

# How well did I accomplish my objectives?

1. Main menu
   1. The menu should appear as a canvas
   2. The menu should appear when the program is executed
   3. The menu should have 3 buttons
      1. “Start Game” when clicked should be able to run backgroundTotal function
      2. “Load Game” when clicked should be able to run loadGame function
      3. “Exit Game” when should be able to run exitGame function

I feel like I completed this objective by including a menu which loads up when I start the program, and allows me to click all 3 buttons that run their respective functions

1. Blackjack Table
   1. The file “Improved Background for nea.png” should be located from a subfolder
   2. The blackjack table should appear when “Start Game” button has been pressed
   3. Have 2 Buttons appear at top left
      1. Return to menu button
         1. Button should appear as an image
         2. Button should run openNewMenu function when clicked
      2. Play Button
         1. Button should appear as an image
         2. Button should run gameplay function when clicked

This objective has been completed to a high enough standard, as when the start game or load game button has been clicked from the menu, the image for the background loads, along with both buttons.

1. Load Game
   1. Should be able to load saved variables from .txt files

I feel like this objective has been complete, as you are able to load a previous game which is stored on two save files. However I could have expanded it to be better, by being able to load more variables and also adapting the code to be able to load a round in progress rather then the start of the round

1. Save Game
   1. Should be able to save variables into .txt files

I feel like this objective has been complete, as at the start of every round, all essential variables are saved to two save files to be loaded up at a later date. However I could have expanded it to be better, by having the option to save a game during a round manually, rather than only having an autosave.

1. AI Counting
   1. Should have a random chance whether they undergo counting methods
   2. Both AI should have separate counting methods
      1. AI 1 should have Hi-Lo counting method
      2. AI 2 should have Halves counting method
   3. Counting should occur at the start of their go and end of round

I feel like I have completed this objective as both AI produce a correct true count and running count, however I could improve this by giving the AI more options in which methods they can use, simulating variety in playstyle

1. Deck
   1. Deck of cards from a total of 52 available cards
   2. The total deck should be made from 4 decks of 52 cards
   3. Will be updated whenever a card is removed from the deck
   4. If the deck is lowered to beneath 104 cards, then the deck should be reshuffled with all cards

This objective has been completed and was essential for the rest of the program and gameplay to function.

1. Gameplay
   1. Player should enter how much they want to bet
   2. Both AI should enter a bet for the round

I think these objectives have been completed effectively, as once the player has chosen their bet, labels appear at the top of the GUI displaying how much has been bet, and updating depending on actions of both AI and player

* 1. AI 1 should be given choices
     1. Hit
        1. 1 Card is added to the AI 1 hand from deck
     2. Stand
        1. Ai 1 Should stand and their go is over
     3. Double Down
        1. Ai 1 should only double down when on 11 or 10 depending on true count
        2. This should double the bet the AI 1 has placed
        3. Hits card
        4. Ends AI 1’s go

These objectives have been completed due to the AI having done all 3 actions within the game, meaning they all work correctly.

* 1. Player should then be given choices
     1. Hit
        1. 1 card is added to the player hand from deck
     2. Stand
        1. Player stands and their go is over
     3. Double down
        1. This should double the bet the player has set
        2. Should hit card
        3. End players go
     4. Split
        1. This should only happen if both values of card are the same
        2. This should create a new bet for the player
        3. This should create a new inventory for the split hand

I met these objectives as when the player’s hand has started they are prompted with 3 (possibly 4) buttons to Hit, Stand, Double Down, and if they own two initial cards with same values, Split. However the Split section could be expanded in the future, by allowing for recursive splitting and doubling down while in split hand.

* 1. AI 2 should be given choices
     1. Hit
        1. 1 Card is added to the AI 2 hand from deck
     2. Stand
        1. Ai 2 Should stand and their go is over
     3. Double Down
        1. Ai 2 should only double down when on 11 or 10 depending on true count
        2. This should double the bet the AI 2 has placed
        3. Hit card
        4. AI 2 should end their go

These objectives have been completed due to the AI having done all 3 actions within the game, meaning they all work correctly.

* 1. Dealer should be given choices
     1. Hit
        1. If dealer is less than 17 will take one card from deck
     2. Stand
        1. Once dealer is over 17 will stand and end their go

I have met these objectives due to the dealer being able to hit a card and standing before going bust, shown in the testing stage.

* 1. If player goes bust
     1. Player’s hand should end
     2. Player will lose the bet they have placed
  2. If AI goes bust
     1. Ai player will lose their bet
     2. Ai hand will end
  3. If dealer goes bust
     1. Dealer hand should end
     2. Any player/AI should win unless they themselves have bust

These objectives have been completed as whenever a bust occurs, the bust hand would be skipped and later in the game it will show a label for the outcome of the hand, showing the hand has bust

* 1. If player gets blackjack
     1. The pay-out is 3:2
     2. Hand is skipped
  2. If AI have blackjack
     1. Pay-out is 3:2
     2. Ai Hand is skipped
  3. If Dealer has blackjack
     1. Dealer Hand is skipped

I have met these objectives as at the end of the round the player can see a return of their bet plus a clear payout of 3:2 for their bet, and if the AI or Dealer has a blackjack the player will only see their hand is skipped.

1. Cards
   1. All cards from all inventories should be shown
   2. Dealers hidden card should be covered by an image
   3. A back of card image should appear to simulate the deck

These objectives have been met as whenever a card is added to an inventory via hitting or from the start of the round, the cards are updated per hand. Also the image for deck appears from the start of the game.

1. Game End
   1. Once the player would reach 50,000 a label will congratulate them

This objective has been met as when the player reaches 50,000 a label will appear congratulating them on their win before prompting the user to restart the game if they want to play again.

In the future I also want to feature a label for when the player hits 0 total cash and prompt the user to restart the game due to effectively losing the game.

* Able to open .png files

This objective has been met as all images which are visable have been png images. I want to expand on this further in the futer to be able to load up .jpeg and .gif images and possibly even animations

* Access images from subfolders

I have met this objective as all images have been stored in subfolders: Background folder and playingCards

* Access .txt files from subfolders

I have met this objective as both save files which can be loaded are stored within the subfolder: Save Folder

# Future plans

My main future plans for this project is to implent sound and animation into the game. This will improve the user experience of the game making it more enjoying to play.

Appendix

# Bibliography

## Research

Reynolds, T., 2020. Blackjack insurance: what it is, how it works, when to take it. (2020). *The Telegraph*. [online] 14 Feb. Available at: https://www.telegraph.co.uk/betting/casino-guides/blackjack/insurance/#:~:text=What%20is%20insurance%20in.

Wikipedia. (2020). *Blackjack*. [online] Available at: https://en.wikipedia.org/wiki/Blackjack.

‌Bicycle Playing Cards. (n.d.). *Blackjack – Card Game Rules*. [online] Available at: https://bicyclecards.com/how-to-play/blackjack/.

‌Maverick, J.B. (2019). *Why Does the House Always Win? A Look at Casino Profitability*. [online] Investopedia. Available at: https://www.investopedia.com/articles/personal-finance/110415/why-does-house-always-win-look-casino-profitability.asp.

‌

Quora. (n.d.). *How does a Casino get its house advantage in blackjack?* [online] Available at: https://www.quora.com/How-does-a-Casino-get-its-house-advantage-in-blackjack

‌Tunica. (2017). *Why Does the House Have an Edge in Blackjack?* [online] Available at: https://tunicatravel.com/blog/2017/05/why-does-the-house-have-an-edge-in-blackjack/.

‌Wikipedia. (2020). *Card counting*. [online] Available at: https://en.wikipedia.org/wiki/Card\_counting.

‌ Reynolds, T., The five most popular versions of blackjack. (2020). *The Telegraph*. [online] 14 Feb. Available at: https://www.telegraph.co.uk/betting/casino-guides/blackjack/types/.

Wikipedia. (2022). *Spanish 21*. [online] Available at: https://en.wikipedia.org/wiki/Spanish\_21.

Reynolds, T., European Blackjack: how to play and where to play it in the UK. (2020). *The Telegraph*. [online] 14 Feb. Available at: https://www.telegraph.co.uk/betting/casino-guides/blackjack/european-blackjack/.

‌

Reynolds, T., Blackjack Switch: how to play and where to play it in the UK. (2020). The Telegraph. [online] 14 Feb. Available at: https://www.telegraph.co.uk/betting/casino-guides/blackjack/blackjack-switch/

Wikipedia. (2022). *Blackjack Switch*. [online] Available at: https://en.wikipedia.org/wiki/Blackjack\_Switch.

Newton, H., www.bestcasinosites.net. (2022). *Vegas Strip Blackjack Review 2022 Rules & Strategy Card*. [online] 1 Feb. Available at: https://www.bestcasinosites.net/blackjack/vegas-strip-blackjack/.

‌Joshi, N. (n.d.). *7 Types Of Artificial Intelligence*. [online] Forbes. Available at: https://www.forbes.com/sites/cognitiveworld/2019/06/19/7-types-of-artificial-intelligence/?sh=765fdb7f233e ].

## 

## Images

Figure 1 –

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.co.uk%2Famscan-Casino-Blackjack-Table-Covers%2Fdp%2FB0008GHX3I&psig=AOvVaw2z3hkUwKRlYwnMqQeuAhpN&ust=1651051719399000&source=images&cd=vfe&ved=0CAoQjhxqFwoTCMjJ3Yu1sfcCFQAAAAAdAAAAABAK>

Figure 2 - <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.casino.org%2Fblackjack%2Feuropean%2F&psig=AOvVaw3WkMxoI4zFGqzD3ScBr7ZS&ust=1651051996033000&source=images&cd=vfe&ved=0CAoQjhxqFwoTCIC31o-2sfcCFQAAAAAdAAAAABAP>

Figure 3 - <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.bestonlinecasino.com%2Fgames%2Fblackjack%2Fswitch%2F&psig=AOvVaw0vjktZlib-VoI97RBzCIA_&ust=1651051406454000&source=images&cd=vfe&ved=0CAkQjRxqFwoTCIj3jsO1sfcCFQAAAAAdAAAAABAE>