BHASVIC

Cards Collective

Comp Sci project

Adam Semmakie
Xx/xx/25

Contents

Analysis

This project will be a digital recreation of Blackjack and Poker that can be played in singleplayer or local multiplayer mode. It will remove the need for a physical deck while also allowing players to choose to integrate their own physical cards with digital tracking of currency and game statistics.

I have always enjoyed card games such as uno and snap. Card games such as these allowed me to bond with my family members despite our huge age gaps. I never played poker or blackjack until I was older but I found it much more fun. I think it's a shame I didn't get to play these games when I was younger due to the stigma surrounding them, the lack of equipment (chips) and my family members not completely understanding how to play.

The idea for this project came from realising that most poker or blackjack games focus on gambling or competitive/online play which excludes families and casual groups who want a more relaxed and offline experience. There is a clear gap for a family friendly version of poker and blackjack that maintains the strategic logic of real poker and blackjack without the involvement of real money and high competitiveness.

My project aims to reproduce the table-top experience of card game's: shuffling, dealing and betting cycles but in a clean and accessible interface for all ages.

Similar Games:

Prominence Poker

Source:(https://store.steampowered.com/app/384180/Prominence_Poker//)

- **Platforms:** PlayStation, Xbox and Windows(Steam)
- **Gamemodes:** Poker (No blackjack)
- **Cost:** Free to play
- **Play modes:** Online and single player (No local multiplayer)
- **View point**: 3rd person

Things I would like to move forward/adapt into my project	**Things I don't want to use in my project**
I would like to adapt the store to use chips on different table styles/colourways	My game will be in 1st person as 3rd person gameplay will ruin the realism of blackjack/poker that I am aiming for as my game will most likely be played by a group of people. My game is replicating the table gameplay of blackjack/poker and not the whole room as players are already in a room together.
I would like to integrate the feature that the amount of currency a player has is displayed in number and also size on the table e.g barely any money = a couple chips on the table, lots of money = piles of chips on the table	I wont be integrating online play as there are already loads of games like this that use it and it will bring too much competition while my game is supposed to be relaxed. Instead I will be having local multiplayer and a singleplayer
I will be moving forward the (Texas Holdem) 2 card poker as it is easier than other variations of poker e.g 3 card poker, 7 card poker, and it is the most popular and known variation of poker which my target audience will be more familiar with.	Player customization is unnecessary as my game will be a first person view of the table so no players will be seen such as clothing hats tattoos piercings etc

Blackjack Championship

Sources:(https://store.steampowered.com/app/1178160/Blackjack_Championship/)
(https://www.bbstud.io/blackjack/faq)

- **Platforms:** Windows(Steam), MacOS(Steam), Android (Google play) iOS(App store), Apple TV(App store) and Amazon fire TV stick
- **Gamemodes:** Blackjack (No poker)
- **Cost:** Free to play
- **Play modes:** Online Multiplayer (No local multiplayer or Singleplayer)
- **View point**: 1st person top down

Things I would like to move forward/adapt into my project	**Things I don't want to use in my project**
I like the layout of the table but I will have the cards look more like they are placed on the table than floating above it. I will also have the table in a similar position but I wil have multiple different types for my user to choose from.	I wont be adding profile pictures or country flags in my game as it is unnecessary when other players are next to eachother (local multiplayer) and can be easily identified by their username aswell.
I also like the layout of the buttons: Double, Split, Stand, and Hit and how they are mouse clicks instead of key presses Along with the menu button tucked away in the corner I will be integrating that layout into my gameplay	I wont be integrating the jackpot as that strays from the games actual gameplay of blackjack and poker
I will be adapting slightly but integrating the position of the players around the table because this will match the realistic feel my project is aiming for.	I wont be adding display of the total value of the cards as that takes away from the realism of the game and causes unnecessary clutter as cards should be calculated mentally just as in real life

Poker club

Sources:(https://www.pokerclubgame.com/)

(https://store.steampowered.com/app/1174460/Poker_Club/)

- **Platforms:** PlayStation, Xbox and Windows(Steam)
- **Gamemodes:** Poker (No blackjack)
- **Cost:** £15.99
- **Play modes:** Online and single player (No local multiplayer)
- **View point**: 3rd person

Things I would like to move forward/adapt into my project	**Things I don't want to use in my project**
I would like to adapt the physical movement of the chips when betting as this is a nice touch which makes the game bit more realistic	I won't be integrating hands/arms into the game as I feel this is abit over the top and ruins the simplistic gameplay I am aiming for
I would like to integrate and adapt the sneak peek of the cards shown below only for poker not blackjack. This is because in poker other players cannot see your cards so if a player wants to check their cards all other players must look away and then the player can take a peek at their card values	I also won't be integrating shadows as it is unnecessary and could clutter the gameplay.

Stakeholders:

End User:

My primary end user will be teens and families that enjoy strategy/logic games and want a non-gambling poker/blackjack experience. They'll use local multiplayer at home for short group sessions and beginners will rely on the "Rules" screen and "Poker card value hints" to learn. This aligns with survey feedback from 16-18 year olds doing computer related college courses who preferred blackjack and non-money play.

My secondary end users will be casual players without the necessary equipment to play poker or blackjack (cards, chips, etc) who want a quick digital setup. They'll use the menu to pick blackjack or poker, set players/rounds and track their progress via the "french fries" currency.

My solution is appropriate as it removes the financial risk while preserving card-game strategy. Local multiplayer is better for family/peer groups better than online as the complexity of online is unnecessary for this audience.

Survey:

The test sample is CS social (16-18yrs old Males and Females doing computer related college courses) 15 people participated in the survey

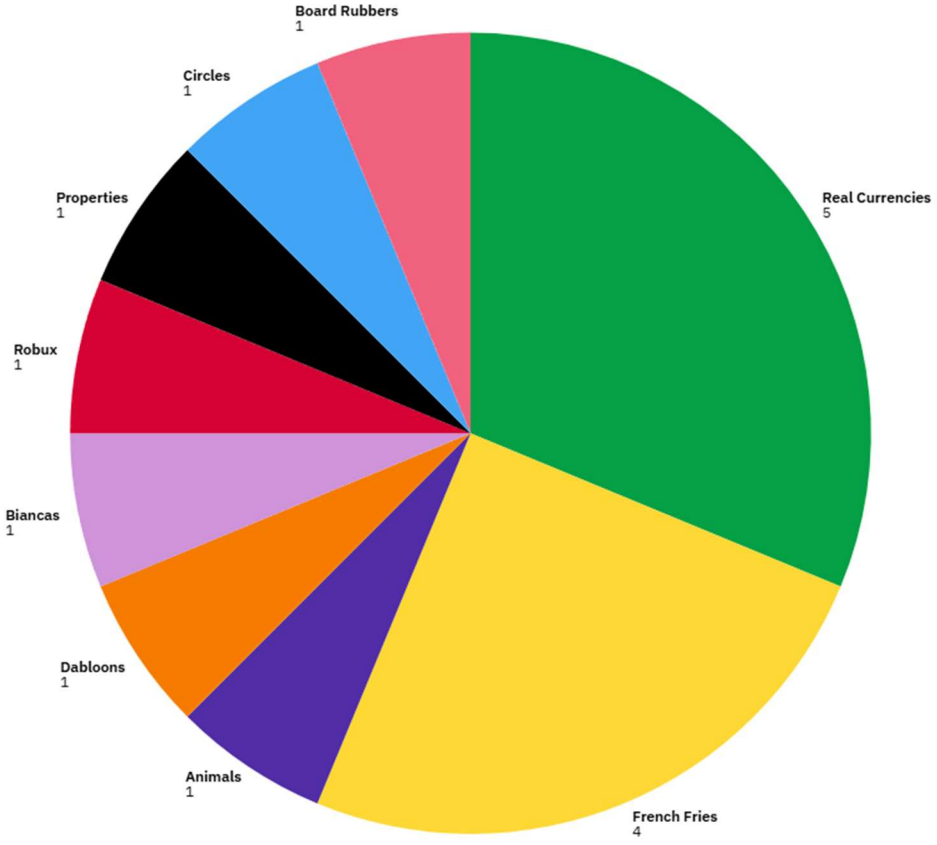
https://forms.office.com/e/uGdr5EQygB

In-Game Features:

Currency (chips/fries):

I would say this is an essential part of the game as poker and blackjack revolve around the use of a currency to play properly. I am using French fries as currency chips as this was highly requested in my stakeholder survey and it is a play on word for actual chips. Also this helps my game stray away from the serious gambling aspect of blackjack and poker by using a different currency from chips or real currencies.

The currency will be displayed in and out of game in the corner or displayed on the table



Physical card integration

This is not an essential part of the game but a unique and useful feature that allows players to use their own deck of cards to play Cards Collective which will keep count of their currency, game history, and can help manage the game for new players compared to playing with the deck of cards without Cards Collective. I haven't seen this in other games but I think it's a good idea if executed correctly and if I have enough time.

Menu

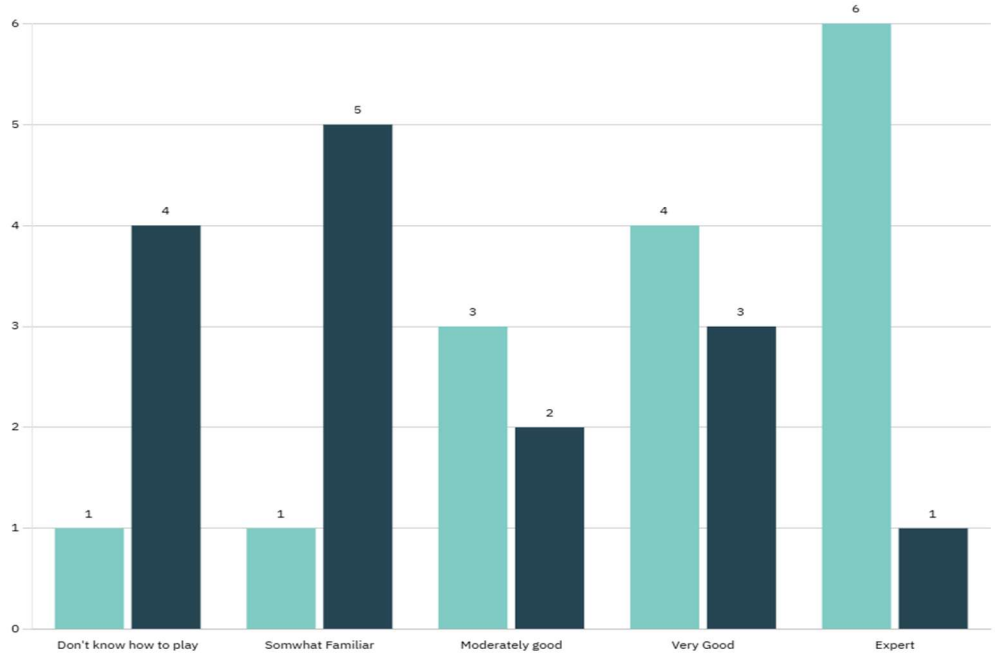
This is an essential feature of the game as players will need a welcome screen presenting options such as:

- Poker
- Blackjack
- Leaderboard
- Shop
- Rules
- Exit

Players will be able to pick which option and be directed to it when clicked.

Rules/How to play

The rules are going to be an essential feature for maintaining new players as my survey showed the majority of my focus group either did not know how to play poker or were somewhat familiar with it (60%).



Poker card value hints

This is an easy and a valued feature for the gameplay in poker as even experienced poker players may need a reminder on the rankings of different card combinations e.g 3 pair (3 of the same value card) is higher than 2 pair (2 of the same card value)

Poker card peek

The sneak peek feature in the poker game mode in my project is essential for gameplay especially in local multiplayer as other players musn't be able to see what cards you have but you must be able to see and make decisions based upon them. To sneak peek at your cards all of the other players participating in poker should look away from the screen so only you can see what cards you have been delt and you hold down the sneak peek button. I am taking inspiration from the game Poker Club

Local multiplayer

This is an essential feature that must be added as groups of friends and family always play card games when they want to indulge in a game and involve everyone some creating a local multiplayer is a necessary feature which will be utilised by lots of players.

Save progress locally (W/L ratio, currency balance, game history, etc)

This is not essential to the gameplay of Cards Collective but I would like to add it as I think it is a necessary feature which will keep players coming back to the game as they can continue with their previous currency balance and resume their game.

Leaderboard

This is not essential to the gameplay of the game however I think it will be a nice feature and bring some competitiveness for the users that enjoy that while allowing casual players just to ignore it or view it and track their progress if they are curious.

Limitations:

Online play:

In the survey many people said they would be playing blackjack instead of poker so adding online play is not essential or necessary to the game as blackjack is mainly a single player game against the dealer and poker will be played in local multiplayer ruling out online play features.

Slot machine

A slot machine was requested as a feature in my stakeholder survey however I think it will ruin the target audience of my game as Cards Collective aims to be a card game without the casino feel so that is why I am not adding it as a feature.

Real currency cash out

This feature is not essential nor necessary. I wont be adding this feature as it aims towards a different target audience and will ruin the casual and fun feel to Cards Collective.

Different playing card and table options

This is not essential and unnecessary and it could ruin the simplistic feel of my project however I may add it as different playing card cosmetics and tables allow players to spend their currency on items which some will work towards helping to maintain some players.

Requirements:

Important	**Essential**
- Leaderboard - Poker card value reminders	- Menu - Blackjack - Poker - Poker Card peek - Currency - GUI - Rules - Poker card value reminders - Card algorithms
Low Priority	**Optional (easy but valued)**
- Currency Shop - Save files (load progress and carry on where started) - Password protected user accounts	- Card integration

Success Criteria:

ID	Feature	Explanation	Justification	Priority
1	**Cards**			**Essential**
1.1	Card integration	Instead of using the games internal card system for the players each players physical cards will be inputted and the game will use those values when determining the winner/losers. POKER(Community cards will be generated by the computer and not be inputted from the users physical cards)		**Desirable**

		BLACKJACK(Every time a card would be delt to a player (e.g Hit) the system will ask what card the player has been delt from the real deck.) *Still experimental in blackjack*		
1.2	Shuffle cards	Shuffles/randomises 52 card deck each round ready to be delt ensuring fairness		**Essential**
1.3	Deal cards	Deals a specified number of cards from shuffled cards deck out to a specified player or dealer to play with/against for blackjack and poker gamemodes. (dequeing these cards from the shuffled deck)		**Essential**
2	**Blackjack**			**Essential**
2.1.1	Hit	Deal player another card		**Essential**
2.1.2	Stand	End players turn without a card		**Essential**
2.2.1	Player turn cycles	Runs turns for each player then finally the dealer		**Essential**
2.2.2	Round result	Shows each players outcome (Win or Draw or Loss/Bust)		**Essential**
2.3	Wager input	Before each round starts players need to place a wager subtracting from their currency		**Essential**
3	**Currency**			**Essential**
3.1	Balance tracking	Keeps track of the amount of currency each player has		**Essential**
3.2	Starting amount	New/fresh players start with a set amount of currency (to be determined)		**Essential**
3.3	Currency updates	Applies win, loss or draw to each players balance immediately after a round is complete		**Essential**
4	**Menu**			**Essential**
4.1	Card integration option	If option is picked card integration will be used		**Desirable**

4.2.1	Blackjack option	Shows screen for blackjack when inputted by the user		**Essential**
4.2.2	Blackjack rules option	Presents rules for blackjack and the blackjack card values		**Essential**
4.3	Shop option	Shows screen for Shop and available items to purchase when inputted by the user		**Desirable**
4.4	Exit option	Closes program safely when inputted		**Essential**
4.5.1	Poker option	Shows screen for Poker when inputted by the user		**Essential**
4.5.2	Poker rules option	Presents rules for poker and the poker card rankings		**Essential**
4.6	No. Rounds input	Set number of rounds for the game when either poker or blackjack are inputted		**Essential**
4.7	Leaderboard option	Shows screen for leaderboard when prompted by user		**Desirable**
4.8	Setting number of players	Prompts user to set number of players and enter usernames of players		**Essential**
4.8	Players/rounds input	Prompts user to set number of players and rounds with simple validation when either poker or blackjack are inputted		**Essential**
5	**Poker**			**Essential**
5.1.1	Card peek	Lets the active player temporarily reveal only their cards on screen while other players look away so they know what they have been delt		**Essential**
5.1.2	Community cards (5)	Display a community card after each player turn cycle is complete		**Essential**
5.1.3	Poker value ranking hints display	Displays a quick reference for the poker rankings of cards mid game		**Essential**
5.2.1	Player turn cycles	Sequentially cycles through each player allowing them to decide to perform “player actions” if not folded or all in. Until all 5 community cards have been displayed and final bet has been made. Which is where the round will end		**Essential**

5.2.2	Turn indicator	Indicates which players turn it is (active player) and indicate folded players clearly		**Essential**
5.3.1	Player actions	Execute player choice (fold,check,raise, all in)		**Essential**
5.3.1.1	Fold	Fold (e.g give up) removes player from the turn cycle for the rest of the round and counts as a loss losing out on any winnings		**Essential**
5.3.1.2	Check	Check keeps you in the round without losing any currency and finishes your turn as long as the rest of the players check for that turn cycle		**Essential**
	Call	Call (i.e call bluff) subtracts the amount of the current bet from players currency and finishes turn		**Essential**
5.3.1.3	Raise	Raise increases the amount of currency of the bet of the active players choice resetting turn cycle for all players that are not out.		**Essential**
5.3.1.4	All in	All in increases the bets amount to ALL the active players currency, resetting the turn cycle for all players that are not out. Players that are all in for that round are no longer in the turn cycle but are still in the game and entitled to any currency increase if they win.		**Essential**
5.3.2	Antes	Captures each players mandatory initial bet at the start of a round		**Essential**
5.3.3	Winner calculations	Determine winning hand(s) and handles folded and loss players appropriately		**Essential**
5.3.4	Central pot	Displays the total currency on the table to be claimed (total of the rounds bets)		**Essential**
6	**GUI**			**Essential**
6.1	Input handling	Accepts keyboard and mouse inputs/interactions with GUI		**Essential**

		and handles them appropriately		
6.1	Card visualisation	Every card (52) has its own card image e.g queen of hearts, ace of spades, 4 of diamonds, etc each card being in the correct precise position of the screen (player1 cards being infront of dealer or player2)		**Essential**
6.2	Blackjack GUI	Displays table backdrop, options, players, relevant cards, buttons etc		**Essential**
6.4	Menu GUI	Displays table backdrop, options, etc		**Essential**
6.5	Poker GUI	Displays table backdrop, options, players, relevant cards, buttons etc		**Essential**
6.5.1	Visual representation of currency	Displays fries on the table as portions that scale with the amount (Handful, Stack, Bag, Bucket)		**Desirable**
7	**Save/Load**			**Desirable**
7.1	Leaderboard	Ranks top 10 players and displays their game data (Currency, W/D/L, Total games played etc)		**Desirable**
7.1.1	Save username and currency	Saves (to an external file) a players username and how much currency they have accumulated		**Desirable**
7.1.2	Load username and currency	Loads (from an external file) an existing/previous players username and currency		**Desirable**
7.1.3	Save Wins/Draws/Losses	Saves (to an external file) a players total wins, draws and losses		**Desirable**
7.1.4	Load Wins/Draws/Losses	Loads (from an external file) an existing/previous players total wins,draws and losses		**Desirable**
7.2	Player accounts			**Desirable**
7.2.1	Player creation	Creates a new player profile with a unique username (inputted from user) and a password (inputted from user),		**Desirable**

		which will be saved (to external file)		
7.2.2	Player login	Using previously made credentials a returning player can continue playing where they left off maintaining the same amount of currency from the last time they played and contributing to their statistics.		**Desirable**
7.2.3	Player creation	Saves a new password assigned to new username which is inputted by the user, which will be saved (to external file)		**Desirable**
7.3	Automatic save	Automatically append external file every time a player is created and when a game is finished		**Desirable**
7.4	Load(ing data) error handling	If data is corrupted or missing program should start cleanly without freezing or crashing(If external file is missing a new one will be created)		**Desirable**

Hardware requirements:

Platform: Desktop or Laptop

Chosen for local multiplayer on one screen. Mobile and console have been excluded to reduce complexity and precise controls

Input devices: Keyboard and mouse

They are the primary input for menus, betting and controlling turns. They also produce precise inputs and are one of the most common forms of input such that my end user will most likely be familiar with.

Storage: 1GB free space

This ensures there is space for the installation of Python, PyGame library, images and save files

Processor: Dual-core 2.0GHz or more

This is required for rendering PyGame and processing python card logic without delay/lag

(Primary) Memory: 4GB RAM minimum

This is to ensure that the PyGame GUI performance is smooth and not choppy

Python requires atleast 2GB RAM according to “geeksforgeeks.org” (source) so 4GB is a reasonable minimum for my game

Software Requirements:

Operating System: MacOS, Linux or Windows 10+

These operating systems support python and pygame, allowing cross-platform compatibility for end users.

Software:

Python 3.12+ Chosen for its simplicity and readability

PyGame library Chosen to display graphics, handle inputs and wide range of features

The majority of the code is in python and PyGame so they must be installed to play the game. They are both free and open-source software making my solution easily accessible for my end user.

Computational Methods:

My solution is computationally suitable because card game mechanics can be precisely expressed through algorithms and data structures. The following computational methods are applied:

Abstraction

Abstraction is used to simply complex real-world card handling into efficient digital processes.

- In real life, shuffling cards involves physical randomness and human verification. My program replaces this with a sorting algorithm (using python's built-in random module) that reorders the deck list each time ShuffleCards() is called
- Only necessary details (card value and suit) are stored. Visual and physical details such as bending or texture are ignored.
- This simplification reduces code complexity, ensures fairness and makes the program efficient for repeated use

Example:

ShuffleCards() function randomises the 52-card array to create a new deck before each round. This removes unnecessary real-world complexity but achieves the same outcome (an unpredictable deck order)

Decomposition

Cards Collective is decomposed into modular sub systems, each performing a specific role that can be developed and tested independently.

- **Menu system**: handles navigation between blackjack, poker, leaderboard and rules screens
- **Card system**: manages deck creation, shuffle, deal and card value handling
- **Blackjack system**: manages order and sequence of rounds/turns, applies game rules(e.g over 21 = loss), W/D/L calculation, card totals
- **Poker system**: manages order and sequence of rounds/turns, manages hands, betting logic and inputs, W/D/L calculation, sneak peek feature
- **Currency system**: manages number of chips for each player, applies changes to chip values (subtract, add, etc) and updates values

This modular structure allows each component to be developed, tested and debugged independently. It improves maintainability, reduces complexity and allows shared systems (such as Card system and currency system) to be reused across both Blackjack and Poker modes.

Algorithmic Thinking

My project uses multiple algorithms to ensure functionality such as:

- **Shuffle algorithm:** Randomises the deck
- **Deal cards algorithm:** allocates cards to specific players and dealer using loops and data structures
- **Currency update algorithm:** Modifies player balances when bets are placed or rounds are finished
- **Turn logic algorithm:** Cycles between players in local multiplayer and updates GUI indicators accordingly

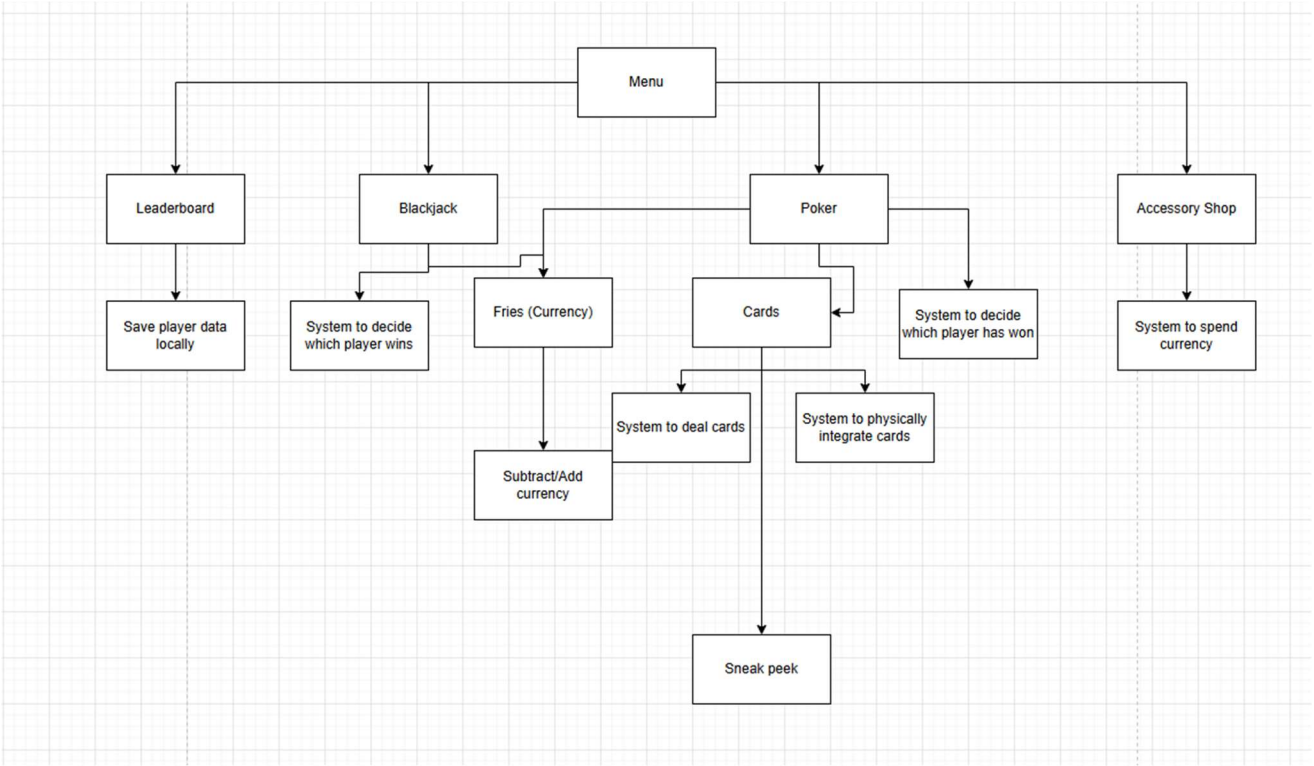
These algorithms are predictable and repeatable which makes it suitable to manage and test.

Justification

My solution is well suited to a computational approach because card games rely on clear logical rules, randomisation and turn based decisions that can be accurately represented by algorithms and data structures. Computers can handle these processes consistently and efficiently which ensures fairness, accuracy, scalability and repeatability in every game.

Design

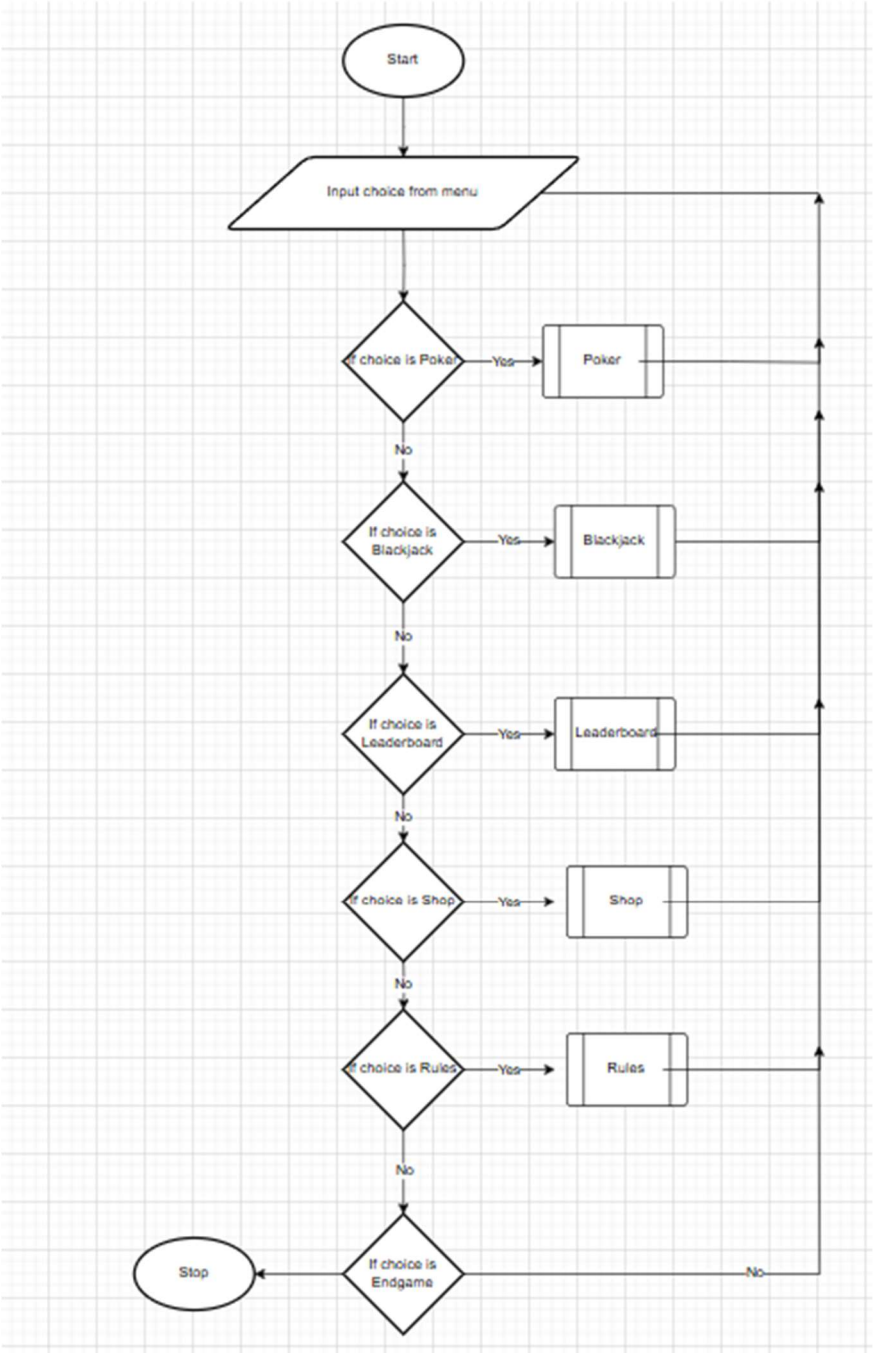
Structure Diagram



I have laid out the structure diagram so that it follows the natural flow of the program and effectively represents the different branches when the user first opens the program the menu will be displayed providing options of leaderboard, blackjack and accessory shop. If either is clicked it will call upon the subroutine for the program to the display e.g Blackjack(). On the next layer of the program it shows the nested sub routines that will be used within the primary ones.

Flowchart

Menu



GUI

First draft

Gui drafting notes:

- I had an idea where the table background is the scenery of a restaurant or kitchen. Table is made of stainless steel (like a kitchen worktop) – matches "fries" and high "steaks" gameplay
- I also thought of another table originating from spongebob (krusty krab restaurant table) for the same reason but also showing that the project is suitable for all ages as using referencing kids cartoon matching my primary stakeholders
- Thought of having a picnic table with the flooring as grass which would also imply that the project is suitable for all ages which matches my primary stakeholders as usually families go on picnics
- Thought of having an antique table not sure why
- I may implement multiple of these different tables if I have time giving users the option to change the look of the game

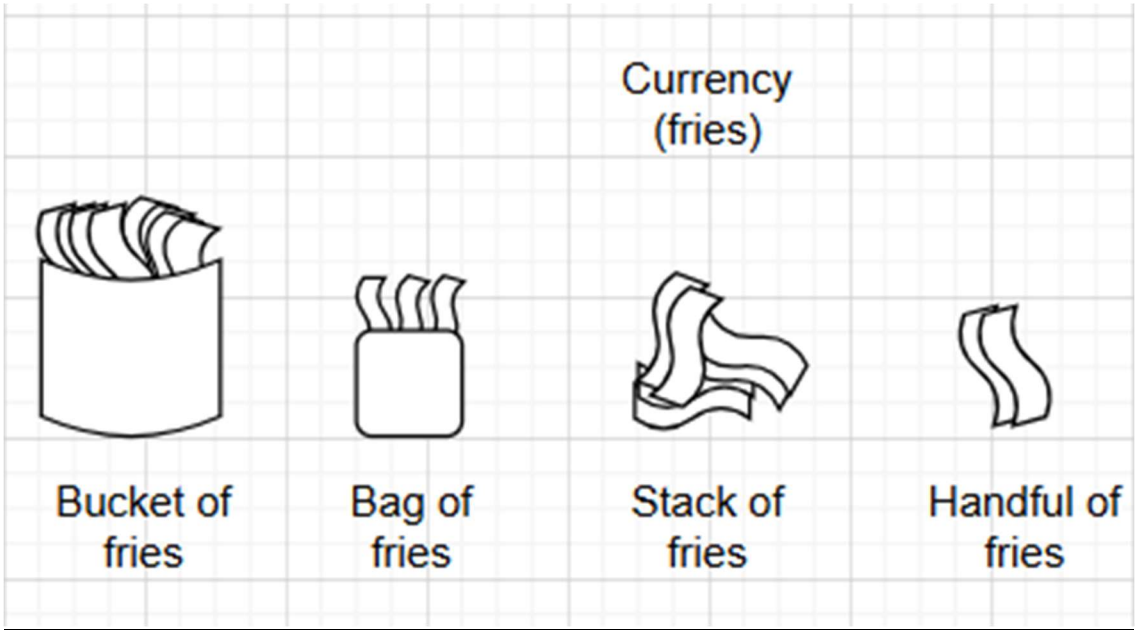
Positioning etc notes:

Cramped cards,

Shuffler to make it look more realistic

Cards move depending on which turn it is for the player like a conveyor belt

Currency:



Bucket of fries = 2000

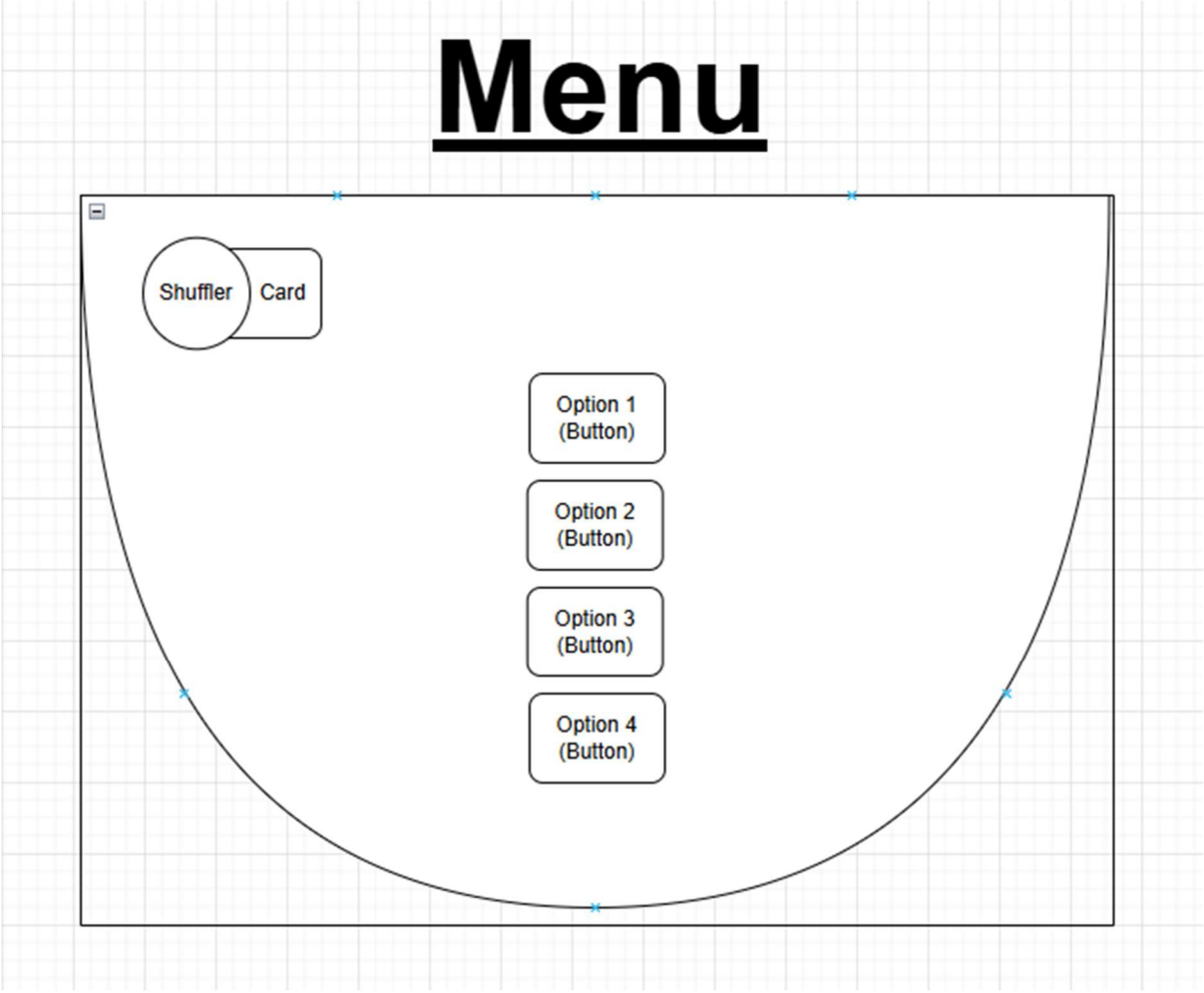
Bag of fries = 1000

Stack of fries = 200

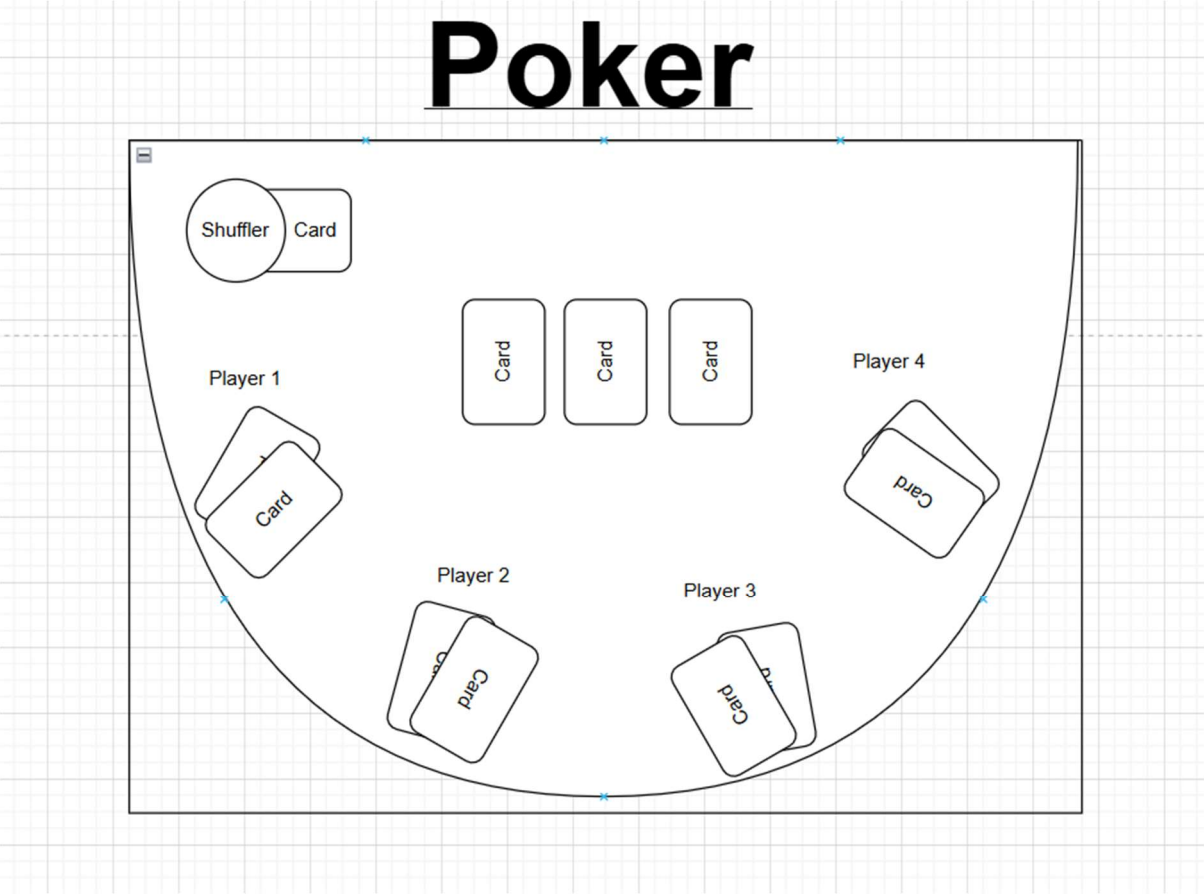
Handful of fries = 50

Will be in the centre of the table of poker depending on the total amount of currency invested in to the round

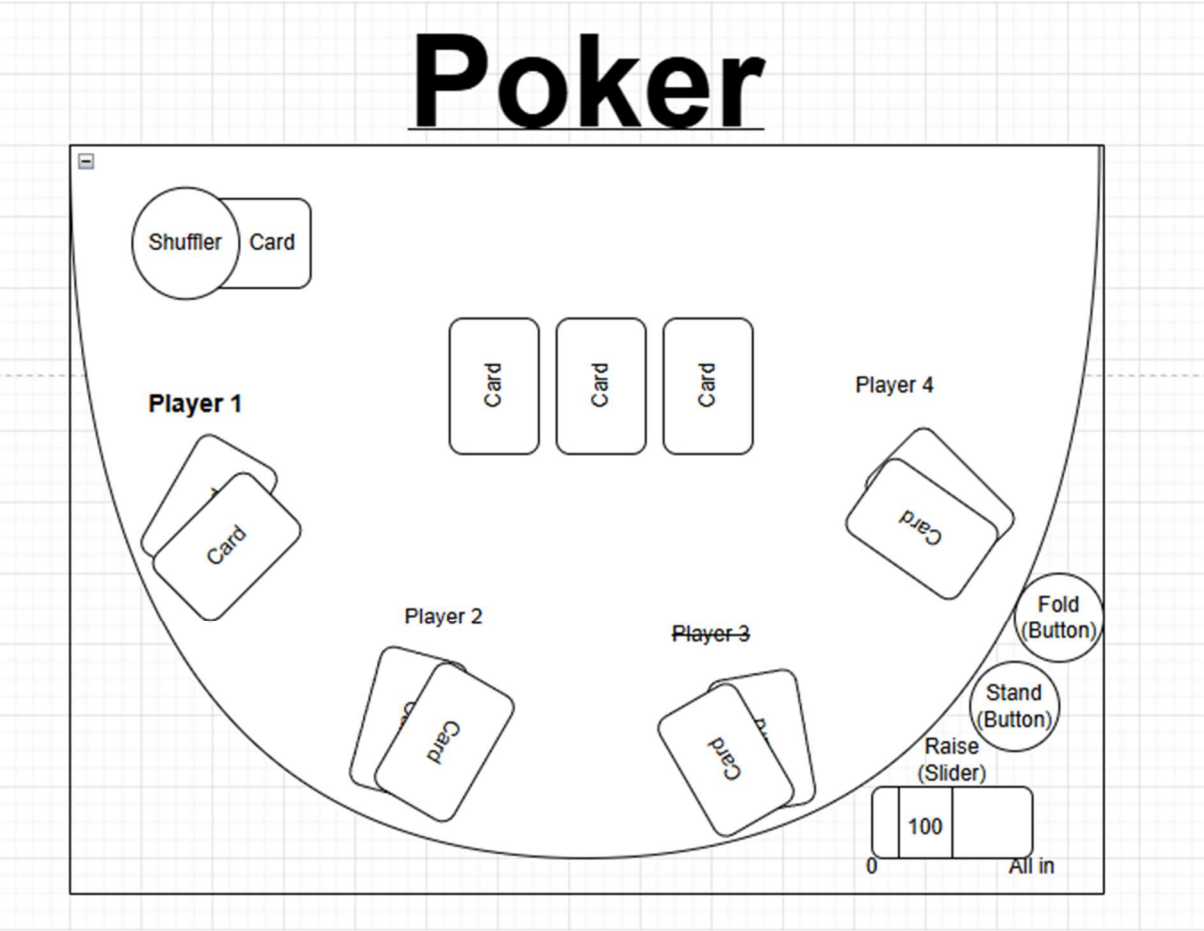
Menu


Menu
Shuffler
Card
Option 1 (Button)
Option 2 (Button)
Option 3 (Button)
Option 4 (Button)

Poker

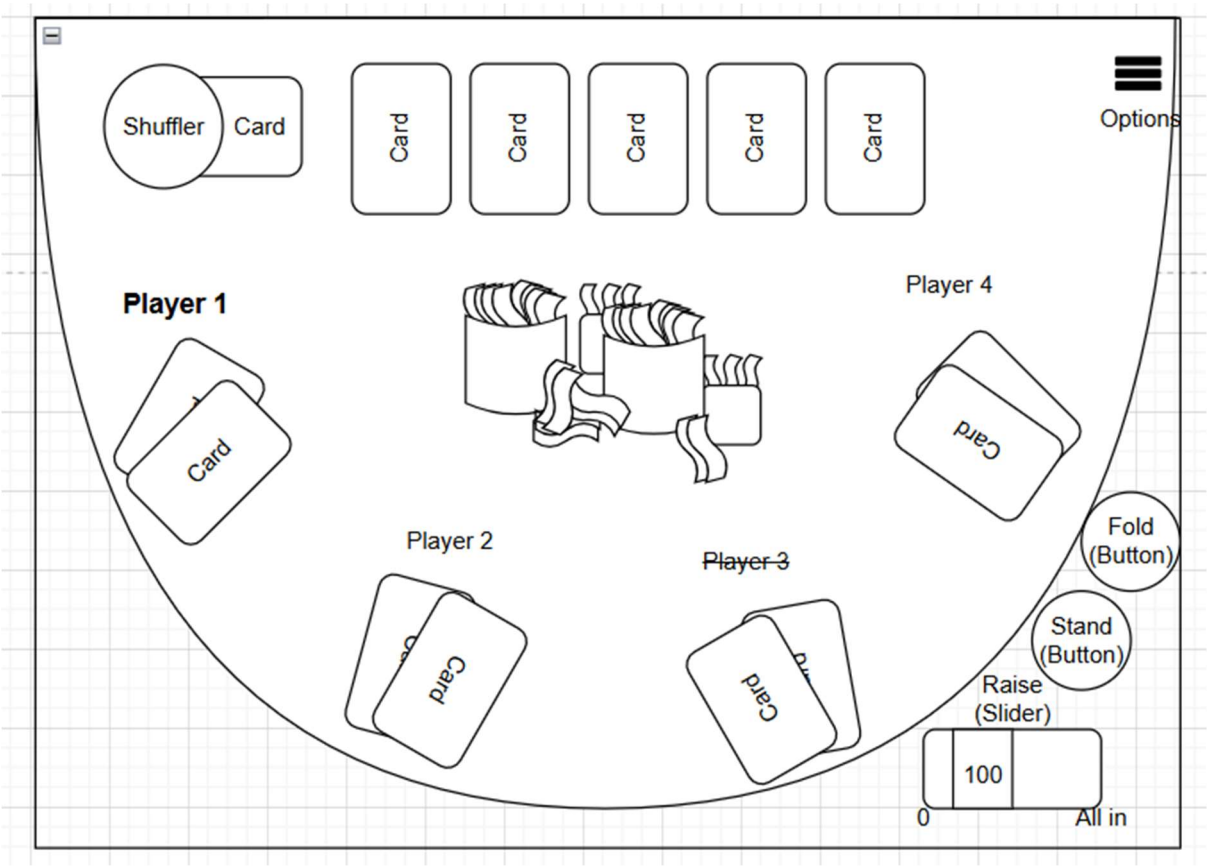


I was thinking of moving cards in the centre up to make space for a indicator in the centre of the board indicating which players turn it is like in blackjack but I ultimately decided the players username will increase in size and change to bold when it is their turn and return back to normal once their turn has ended and if a player is out their name will appear crossed out with a strikethrough line and this will be the same for blackjack.



I also added the different buttons as I forgot to add this to the last draft

- Raise (slider) is going up in increments of 50 from 0 to All in (Players full balance)
    - Releasing the slider will finalise their input and end their turn
    - 100 is an example of the sliders value
- Stand (Button) is a mouse click
- Fold (Button) is a mouse click and when clicked strikethrough line the player who clicked as they are now out

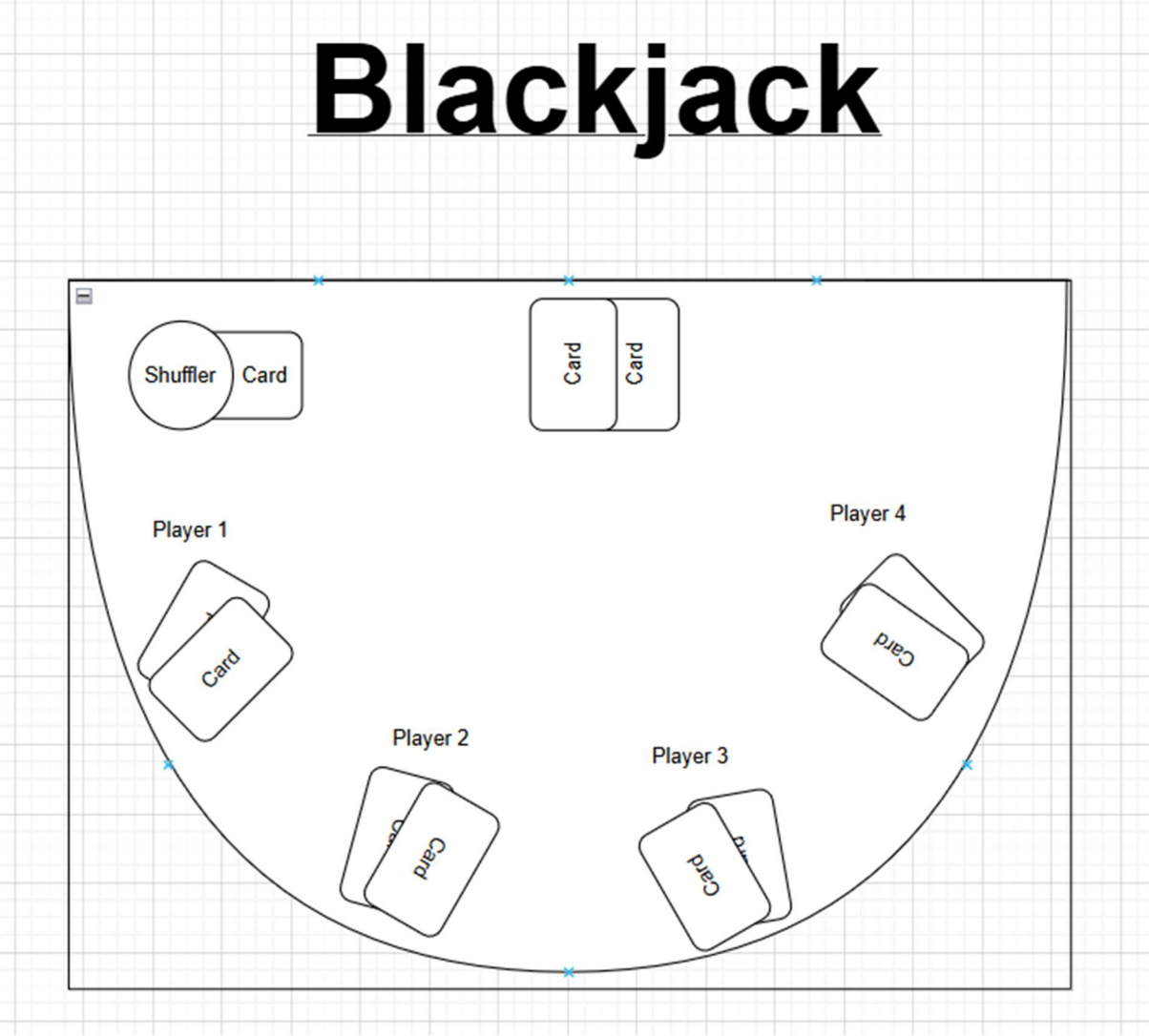


I added an options button in the top right and this is where the poker card rankings can be accessed. I ended up moving the cards in the centre upwards to make space for the currency pool that will be in the centre which will be the total of every players invested currency.

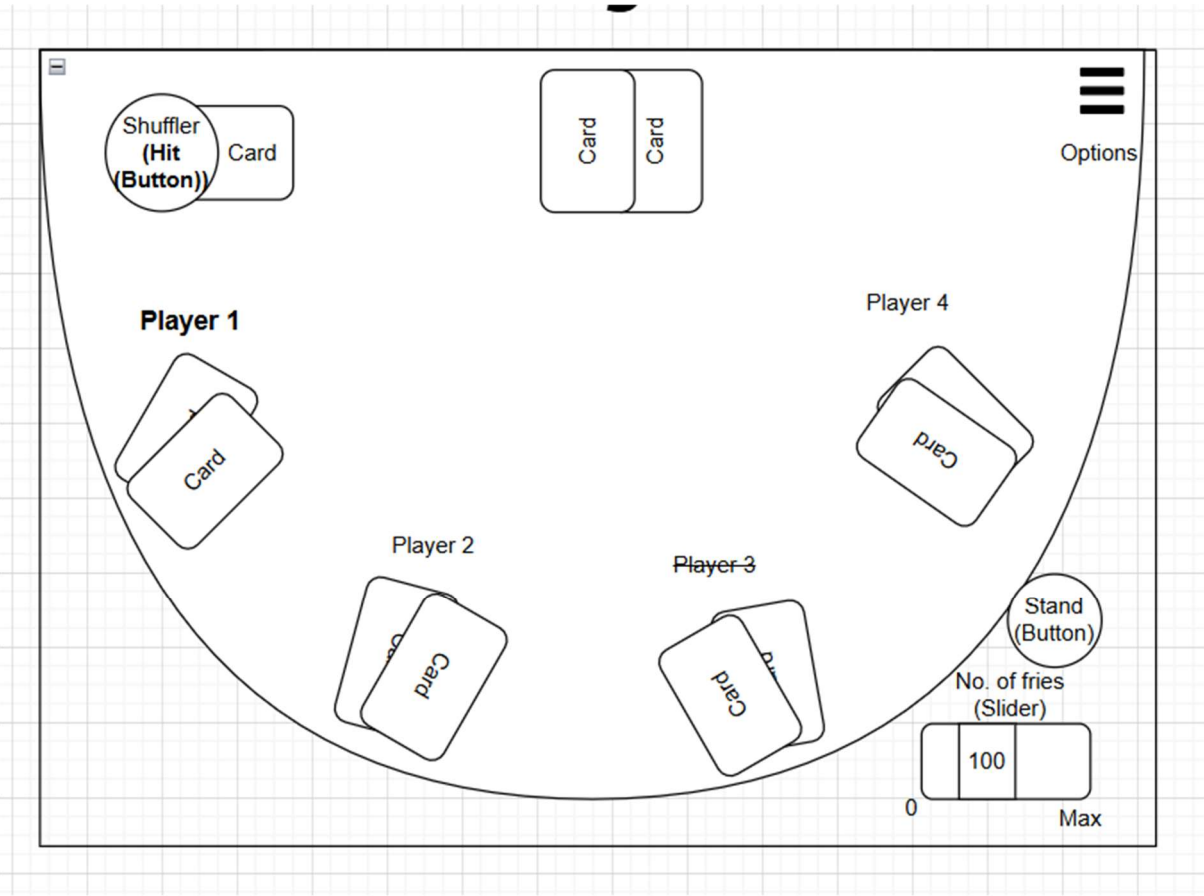
In this example of the draft the Cental Pot has a value of 2000 + 2000 + 1000 + 1000 + 200 + 50 = 6250 French Fries

Also I mistakenly put 3 community cards (the cards near the top of the screen) and have since changed them to 5

Blackjack



- The Menu, Poker, Blackjack all have the same background to allow the background to remain constant throughout the game reducing overcomplication. It is also for the realism of my project as it creates the feel/atmosphere of being at a table just as in real life poker/blackjack.
- The cards will be positioned the same in blackjack and poker so code can be reused between them
- The shuffler is on the table to add to the realism as it creates the illusion to the user that the cards are not popping out of thin air and are being produced from the shuffler.



I also forgot to add inputs in the last draft of the GUI.

I added the indicator to show which players turn it is (Name increases in size and bold) which will return to normal once their turn is over and which players are out (Name is strikethrough).

Buttons added:

- Stand (button) mouse click
- No. of fries (slider) using mouse select amount of currency to bet to start the round in intervals of 50 between 0 and Max (Max value has not been decided yet)
- Shuffler (Hit (Button)) mouse click on shuffler. I decided to make this the button to get another card as it adds to the realism of getting a card from the shuffler instead of having a separate button for Hit
- Options button

Cards

Playing Cards Icon Pack | Gradient fill | 78 .SVG Icons - Page 2

Development plan

Card (dealing system)

Deadline: Week 1-2

Requirements:

-
- (D) Physical card integration
- (E) Create a shuffled/randomised deck of 52 cards from ordered array cards
- (E) Cards can only be assigned to one player at a time (No duplicates)
    - Shuffled deck will be a queue and the dealing algorithm will dequeue the “card” at the front of the queue/”deck”
-

Justification:

For my project I have chosen to start by developing the Card dealing system as without it, the rest of the game will be unplayable making it one of the most essential parts of the game.

Blackjack gameplay Class

Deadline: Week 3

Requirements:

- Display each players cards value and their suit
- Players are given the option to Hit or Stand – Input should be handled accordingly
- Determine If player won, drew or loss
- Enter a value before each round

Justification:

I have chosen to start by developing the blackjack game as the currency system is dependent on inputs from the blackjack game.

Currency system

Deadline: Week 4

Requirements:

- Keep track of each players currency
- Add and subtract currency from individual players existing balance
- Calculate currency increase/decrease depending on how much each player has put in if its blackjack and depending on the outcome of a round: Win, Draw, Loss

Justification:

I have decided to do this next as then the blackjack section of the project will be finished early into development

Menu

Deadline: Week 5

Requirements:

- Displays different options:
    - Poker
    - Blackjack
    - Rules
    - Exit
- Players will be able to pick which option and output the desired option when chosen.
- Input number of players for blackjack/poker how many rounds etc

Justification:

I have decided to do this after the blackjack gameplay is developed and before the poker gameplay as it is an essential but easy and should be relatively quick part of the game to make.

Poker gameplay

Deadline: Week 6-7

Requirements:

- Display 5 cards from dealer at appropriate timings
- Allow player to input Raise, Stand, Fold
- Allow player to input of integer from each player before cards being displayed
- Determine win,draw or loss for each player based upon each players card rankings if player has not Folded

Justification:

I have chosen to do the poker gameplay next as it is another essential part of the game. This has purposely been done before the development of the GUI as the logic of the game will be completed after this point such that the game is “playable” if there are any unforeseen problems past this stage in development

GUI

Deadline: Week 8-9

Requirements:

- Take inputs via keyboard and mouse interaction
- Display background, cards, currency values etc in correct positions
- Simple animations when triggered (card moving across screen, card flipping etc)

Justification:

- This is done after the essential parts of the games logic as this will be the most time consuming part of this project and for reasons stated further up.

Save Player Data

Deadline: Week 10

Requirements:

- Save players data to an external file
- Update changes to player data in external file
- Load this data at the start of the project from the external file

Justification:

I have decided to do this stage after the GUI as it is not an essential part of the game but is a useful feature and will be needed for a leaderboard and player log in to check credentials and save credentials so players can carry on where they left off.

Data Dictionary

variables = camelCase
parameters = camelCase
bool variables = isCamelCase
constants = UPPERCASE

constant pointers = _UPPERCASE
subroutines/functions/procedures = PascalCase

((UPPER CASE) X = number 1-4 just a simplification in data dictionary)

Identifier	Data type	Description	Validation
players = [player1,player2,player3,player4]	Record	2D list holding 2D lists of playerUser, playerCards etc	
playerX = [playerXUser,playerXCurrency,playerXCards,playerXResult]	2D list	*(X = number 1-4 just a simplification)* All 2d lists holding data to be changed and used by different subroutines of lots of different things for players e.g Poker, Blackjack, Leaderboard etc	
dealer = [dealerUser,dealersPot,dealerCards, dealerResult]	2D list	All 2d lists holding data to be changed and used by different subroutines of lots of different things for dealer	
gamemode	String	Holds whether gamemode selected is “blackjack” or “poker”	
_USER	Pointer	index of playerXUser in playerX to make code more readable	
_CURRENCY	Pointer	index of playerXCurrency in playerX to make code more readable	
_CARDS	Pointer	Index of playerXCards in playerX to make code more readable	

_RESULT	Pointer	Index of playerXResult in playerX to make code more readable	
Menu			
playerXUser	String	Holds username for playerX	
Cards			
ShuffleDeck()	Subroutine	Randomises order of 52 cards and any cards delt are removed from it	
DealCards(playerCards,nCards)	Subroutine	Moves first cards in the list shuffledCards into playerCards and removes those cards from shuffledCards	
CardIntegration(playerCards,nCards)	Subroutine	Moves inputted card from user in the list shuffledCards into playerCards and removes those cards from shuffledCards	
CardValue(card)	Subroutine	Produces an integer value of from the first 2 characters of a card depending on the card	
CardSuit(card)	Subroutine	Extracts a single character from the third/last character in the array	
playerXCards	List	Holds cards assigned to playerX and reset to [] at the start of each round	

dealerCards	List	Holds cards assigned to dealer and reset to [] at the start of each round	
ORDERED[52]	2D array	Holds an ordered standard deck of (52) cards: cards[13][4]	
shuffledCards[52]	1D list	Randomised order of 52 cards and any cards delt are removed from it	
Currency			
Winnings()	Subroutine	Calculates which players are subject to how much currency depending on how many players won, the gamemode and size of the pot/their bet	
playerXCurrency		The amount of currency player has	
dealersPot	List	Holds each players currency separately in the order of the index of the players index of the players	

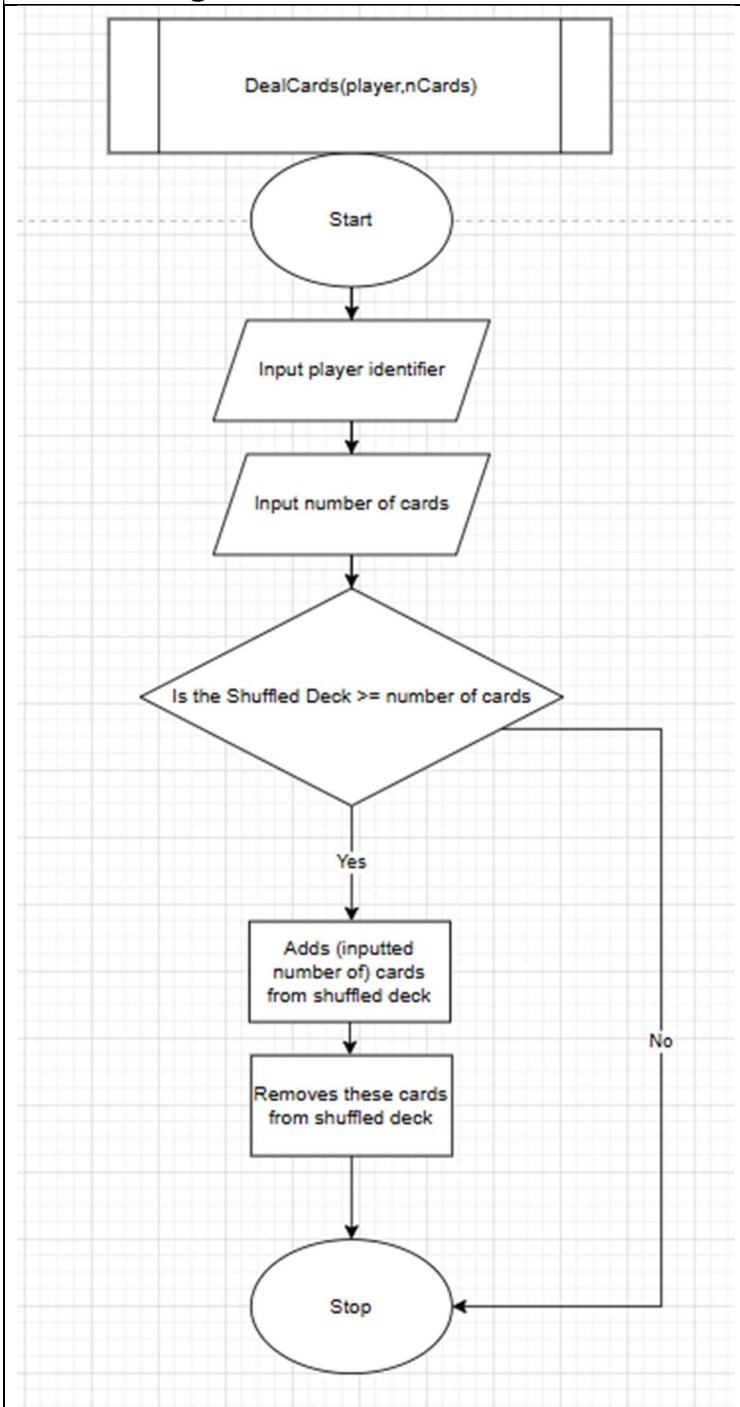
Blackjack			
BlackjackResult	Subroutine		
Poker			
	Subroutine		
GUI			
	Subroutine		
AcS[0] = “AcS”	array		
AcS[1] = screen.blit[image, “ace-of-spades.png”]	Array		

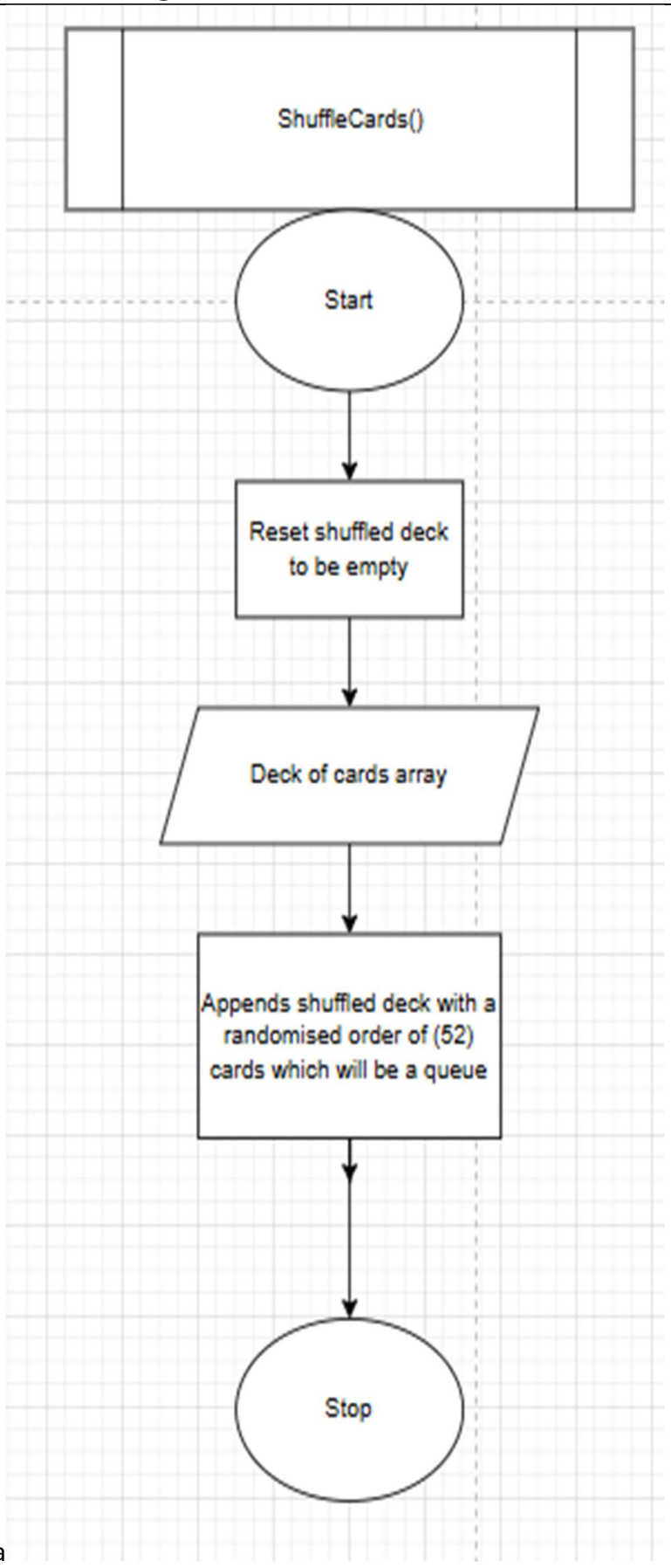
Development

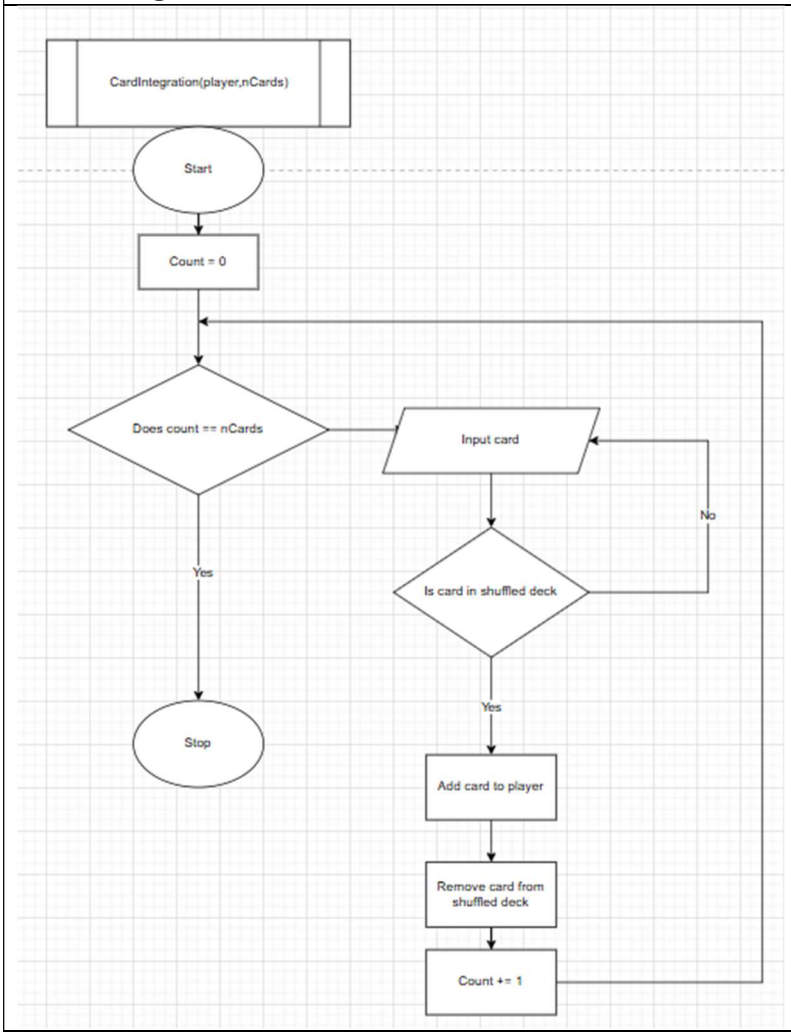
Stage 1 – Card (dealing system)

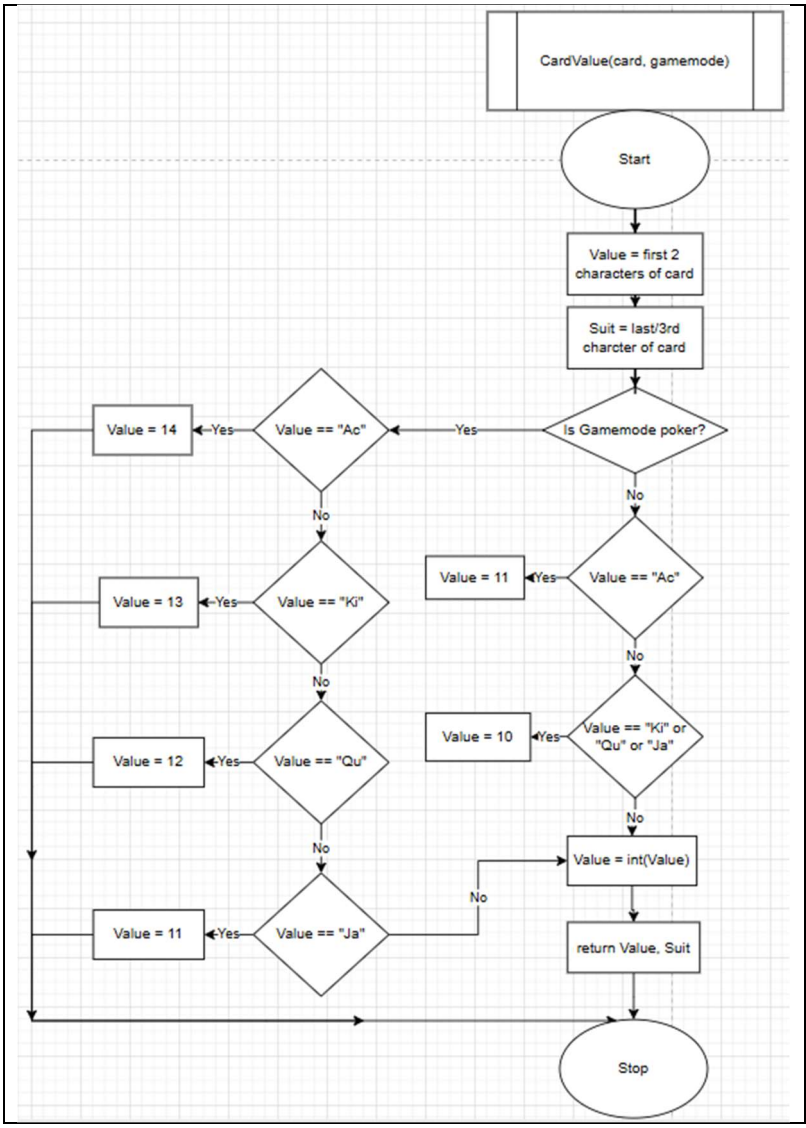
Design

Algorithms - Should contain flowcharts / pseudocode

Card Dealing subroutine	
	The DealCards subroutine takes 2 parameters which are used to know which player the cards are being credited to and "nCards" being the number of cards being given. Shuffled deck is in the global scope and not passed as a parameter. There is a check if there is enough cards to give out(is the number of cards more than the number of cards in shuffled deck) If there aren't enough cards then The subroutine ends without dealing any cards or removing any from the shuffled deck If there is enough cards the cards are added to the player (passed as a parameter) Then the cards added to the player are removed from the shuffled deck so there aren't any duplicates

Card shuffling subroutine	
a 	The card shuffling routine shuffles a fresh ordered deck of cards and produces a freshly shuffled deck with all 52 cards present. This resets the shuffled deck to empty to prevent any conflicts or any patterns that may occur if it isn't reset The ordered deck of cards is used to be shuffled and the shuffled version is the shuffled deck and the ordered deck remains the same

Card integration subroutine	
	This is the flowchart of the card integration it will be inputted a card from the user and that card will be checked to see if it is in the shuffled deck. If it is it will be removed from the shuffled deck and added to the specific player but if it isn't then the user will be prompted to re-enter a card until it is one existing in the shuffled deck. This process is repeated for the number of cards "nCards" which when all the cards have been added will end the subroutine
Card values(and suit) Subroutine	



Card value takes 2 parameters: card and gamemode

The first two characters of card is assigned to Value

The last character is assigned to suit
(to be used in poker) to calculate card rankings

Depending on gamemode different cards are assigned different values

Requirements:

- (D) Physical card integration
- (E) Create a shuffled/randomised deck of 52 cards
- (E) Cards can only be assigned to one player at a time (No duplicates)
    - Shuffled deck will be a queue and the dealing algorithm will dequeue the “card” at the front of the queue/”deck”

Data - data dictionary / class diagrams

Development

Show the code build over time

Justify your decisions

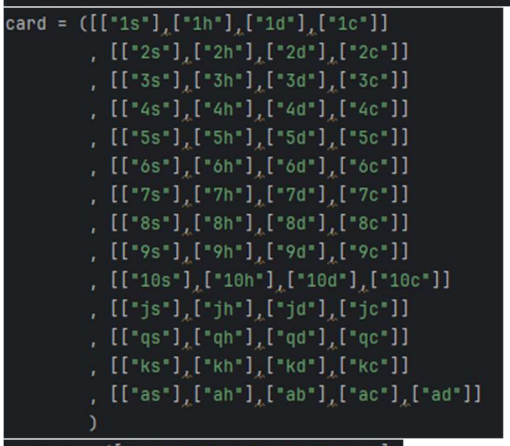
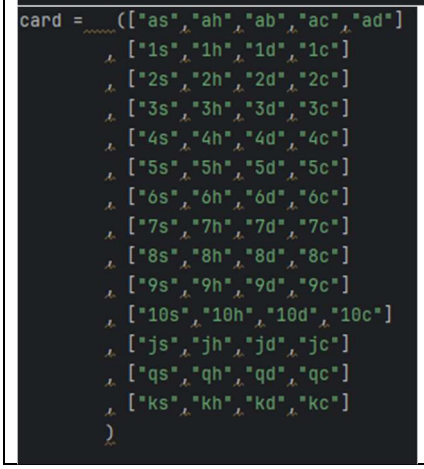
Screenshot errors and your fixes

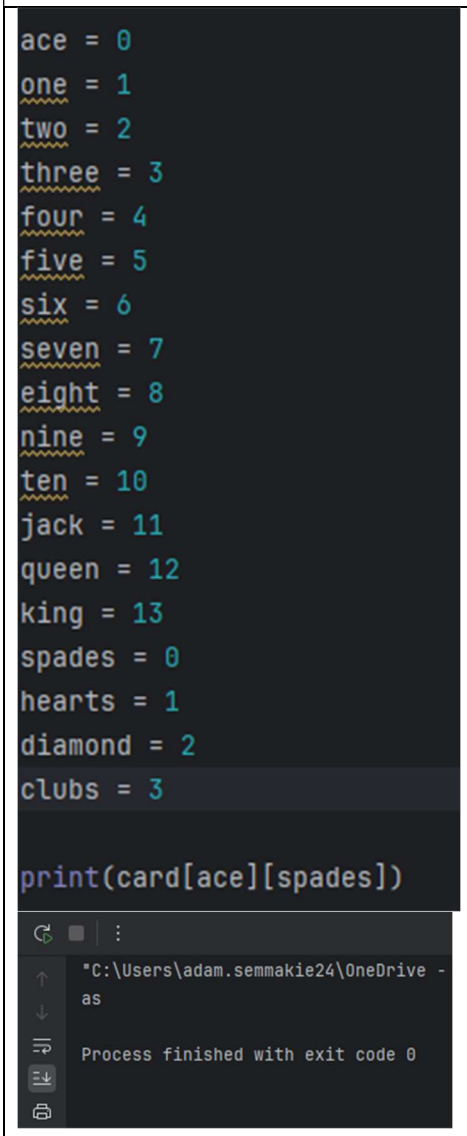
Videos or screenshots of what it looks like when run

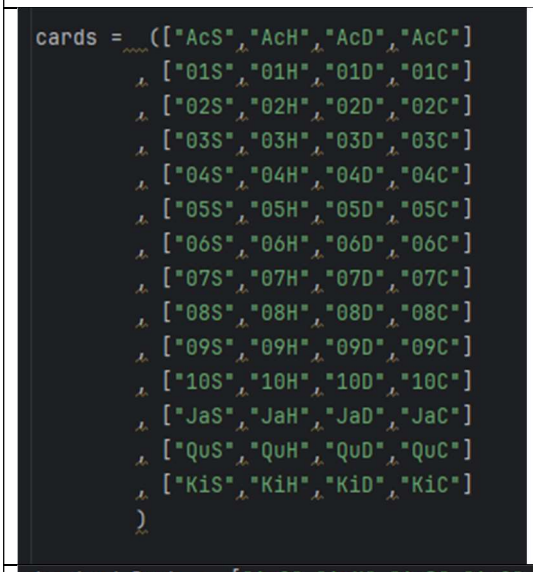
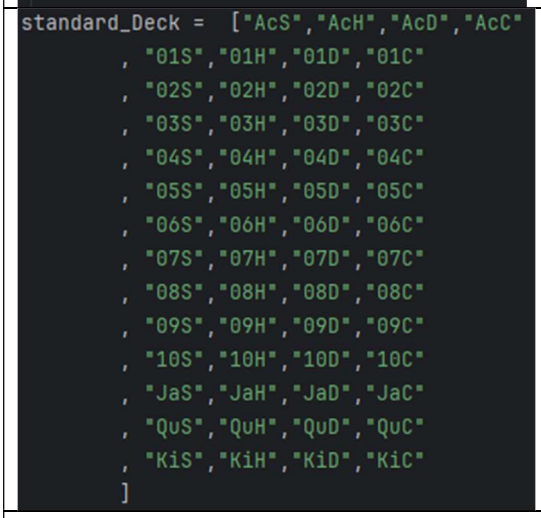
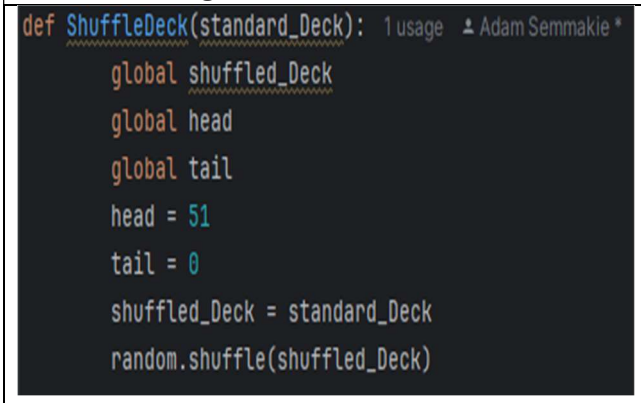
Reference tutorials used

Python Random shuffle() Method

Python - List Methods

Card lists	
  	I was stuck between these two ways of storing the cards list. In Picture 1 each card has is saved as its own variable while in Picture 2 the cards are split up by value and suits where the total combination of the value and suit produces the full deck of cards. Ultimately I opted with Picture one as the codes readability is much better and helping with future maintenance and debugging and ease of GUI creation each card will have an image of it assigned to it. E.g 1s image is 1s.png I altered the formatting and arrangement making it more logical such that index 0 is

	ace index can be seen in the 3rd screenshot
	Testing this highlighted problems that would of caused distruption later down the line. Problems discovered: - They weren't all the same string length e.g "1s" and "10s" ○ I was planning on holding the value of each card in the first character however in this example my algorithm would have thought that it was the same value ○ I changed the values such that the first 2 characters indicate the value and the last character indicates the suit the suit letter being capitalised as now the face cards and ace are two letters long which means I also capitalise the first letter of them e.g "KiS". ○ I changed the numbers such as 1 to

	"01S" instead of "1S" etc which solves this problem
	Here is the updated array
	I made a 1d list version of the organised card list so that the shuffle method in the built in python module will be compatible to be used on it. I have left the cards 2d array in so it can be used in future to compare cards based on their index.
Deck shuffling	
	This function takes standard deck as a parameter as standard deck will remain a constant. The global variables shuffled_deck , head , tail are changed within the function (resetting it to a full deck) and will be used within the other card subroutines. - random.shuffle() is a built-in method in the random module of python. I have decided to do this as it makes

	the code much more readable and saves me time coding a bare bone shuffling algorithm - Treating the shuffled cards with a queue like fashion using pointers head and tail where cards will be handed out from the front(head). The tail has no use as of this time but I have instantiated a variable for it for the common standard with a head comes a tail. ○ Realistically a stack would be a more accurate representation of the cards but there are stricter access constraints with a stack thus opting for a queue-like structure.
Card dealing	

```
def DealCards(player,nCards):  4 usages  new *
        global shuffled_Deck
        global head
        for i in range(nCards):
                card = shuffled_Deck[head]
                shuffled_Deck.pop(head)
                head -= 1
                player.append(card)
        return player
```

```
player1 = []
player2 = []
player3 = []
player4 = []
```

The DealCards function takes in parameters player and nCards allowing for it to be reused no matter the number of cards being delt and to who.
shuffled_Deck and head is globalised so their values can be used and edited.
The subroutine is internally looped for the number of cards (nCards) that will be delt each loop assigning the card to a buffer (local) variable called “card” then removed from the shuffled cards using the pop() built-in method for list handling.

- The pointer of the head of the queue is cremented/decreased by 1 as the card has been removed from the deck.

- The card is added to the playerCard list
- Essentially the code has moved a card out of the deck and is now in the specified players possession

<table>
<tr>
<td>

```
119
120     ShuffleDeck(standard_Deck)
121     DealCards(player1, nCards: 2)
122     DealCards(player2, nCards: 2)
123     DealCards(player3, nCards: 2)
124     DealCards(player4, nCards: 2)
125     print(player1)
126     print(player2)
127     print(player3)
128     print(player4)
129     print(shuffled_Deck)
130     print(len(shuffled_Deck))
```

Run Cards

```
C:\Users\adams\AppData\Local\Programs\Python\Pytho
['05D', '10D']
['02C', '07C']
['05S', '08S']
['05C', '03C']
['06C', 'KiS', '09C', '03H', 'AcH', '08D', '09S',
48

Process finished with exit code 0
```

</td>
<td></td>
<td>I tested the dealing of the cards and it passed without repeating any cards and removing the delt cards from the shuffled list. However it brought to my attention that the total amount of cards left was 48 even though 8 were removed and the total before dealing is supposed to be 52. It turned out I mistakenly included an extra value of cards ["01S","01H","01D","01C"] which do not exist in real life. I corrected this shortly after and the DealCards() and ShuffleCards() sub-routines were working as intended.</td>
</tr>
</table>

```
ordered_Deck =  [["AcS","AcH","AcD","AcC"]
        , ["02S","02H","02D","02C"]
        , ["03S","03H","03D","03C"]
        , ["04S","04H","04D","04C"]
        , ["05S","05H","05D","05C"]
        , ["06S","06H","06D","06C"]
        , ["07S","07H","07D","07C"]
        , ["08S","08H","08D","08C"]
        , ["09S","09H","09D","09C"]
        , ["10S","10H","10D","10C"]
        , ["JaS","JaH","JaD","JaC"]
        , ["QuS","QuH","QuD","QuC"]
        , ["KiS","KiH","KiD","KiC"]]
# Value
ace = 0
two = 1
three = 2
four = 3
five = 4
six = 5
seven = 6
eight = 7
nine = 8
ten = 9
jack = 10
queen = 11
king = 12
# Suit
spades = 0
hearts = 1
diamond = 2
clubs = 3
#ordered_Deck[value][suit]
print(ordered_Deck[ace][spades])
```

```
standard_Deck =  ["AcS","AcH","AcD","AcC"
        , "02S","02H","02D","02C"
        , "03S","03H","03D","03C"
        , "04S","04H","04D","04C"
        , "05S","05H","05D","05C"
        , "06S","06H","06D","06C"
        , "07S","07H","07D","07C"
        , "08S","08H","08D","08C"
        , "09S","09H","09D","09C"
        , "10S","10H","10D","10C"
        , "JaS","JaH","JaD","JaC"
        , "QuS","QuH","QuD","QuC"
        , "KiS","KiH","KiD","KiC"
        ]
```

```
ShuffleDeck(standard_Deck)
DealCards(player1, nCards: 2)
DealCards(player2, nCards: 2)
DealCards(player3, nCards: 2)
DealCards(player4, nCards: 2)
print(player1)
print(player2)
print(player3)
print(player4)
print(shuffled_Deck)
print(len(shuffled_Deck))
```

```
Run    Cards
C:\Users\adams\AppData\Local\Programs\Python\Python312\python.exe
AcS
['JaD', '04H']
['03H', 'KiS']
['10D', 'QuH']
['JaS', 'AcD']
['QuC', '02C', '10H', 'AcH', 'KiH', '03C', '07S', 'QuD', '05H', '0
44
```

Here is the updated lists:
ordered_Deck
standard_Deck

Total after removing the extra card values is correct 52-8 = 44 as can be seen at the bottom of the 3rd screenshot

Card Integration	
	CardIntegration() is the subroutine for the card integration feature for the user to handle their own physical deck of cards. It is abit similar to the DealCards() sub routine but instead of the system deciding which cards the players are delt the user is doing it. - The remove() is a another built-in python method for list handling. I have used it to remove the instance of that card in shuffled Deck which should replicate the physical deck of cards that the user has remaining. This is why the shuffled Cards is queue-like and not a standard queue as cards are removed from any position of the list and not just the head. - It also adds the card to the players possession in the program. "player.append(card)" - Player and nCards are passed as parameters so that the sub routine is reusable for all players and different amounts of cards if needed.

```python
def CardIntegration(player, nCards):
    global shuffled_Deck
    global head
    found = False
    for i in range(nCards):
        while not found:
            card = input("Input card")
            if shuffled_Deck.count(card) == 1:
                found = True
            else:
                print("Card does not exist in deck")
        shuffled_Deck.remove(card)
        player.append(card)
    head -= nCards
    return player
```

I improved the CardIntegration subroutine by adding a check that the card they are trying to add is in the shuffled_Deck and if not they will be prompted to renter a card catching typos, card reptetition and non existing cards. This check uses a while loop.

Card Value subroutine

Cards.py

```python
117  def CardValue(card, gamemode):
118      value = card[0:2]
119      if int(value) <= 10:
120          value = int(value)
121      elif gamemode == "blackjack":
122          if value == "Ac":
123              value = 11
124          else:
125              value = 10
126      elif gamemode == "poker":
127          if value == "Ac":
128              value = 14
129          elif value == "Ki":
130              value = 13
131          elif value == "Qu":
132              value = 12
133          elif value == "Ja":
134              value = 11
135          else:
136              print("card value error")
137      else:
138          print("gamemode error")
139
140      suit = card[2:]
141      return value, suit
```

Card value subroutine
The code uses captures the first 2 characters which represent the value of the card using [0:2] and captures the suit as the last character [2:]

```
140         suit = card[2:]
141         return value, suit

Cards ×

C:\Users\adam.semmakie24\Desktop\Coursework\pythonProject\.venv\Scripts\python.exe C:\Users\adam.semm
Traceback (most recent call last):
  File "C:\Users\adam.semmakie24\Desktop\Coursework\pythonProject\Cards.py", line 167, in <module>
    print(CardValue("AcS","poker"))
          ^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\adam.semmakie24\Desktop\Coursework\pythonProject\Cards.py", line 119, in CardValue
    if int(value) <= 10:
       ^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'Ac'

Process finished with exit code 1
```

There was an error trying to turn Ac into an integer so reordered the if statement branches and checks such that all string values (“Ac”,”Ki”,”Qu”,”Ja”) are assigned a integer value before value = int(value) which converts the string number to integer number e.g “10” to 10, “05” to 5 etc

```
#gamemode is either poker or blackjack
gamemode = "blackjack"

def CardValue(card, gamemode):  1 usage
    value = card[0:2]
    if gamemode == "blackjack":
        if value == "Ac":
            value = 11
        elif value in ["Ki", "Qu", "Ja"]:
            value = 10
        else:
            value = int(value)
    elif gamemode == "poker":
        if value == "Ac":
            value = 14
        elif value == "Ki":
            value = 13
        elif value == "Qu":
            value = 12
        elif value == "Ja":
            value = 11
        else:
            value = int(value)
    else:
        print("gamemode error")

    return value
def CardSuit(card):
    suit = card[2:]
    return suit

```

I have also separated it into two subroutines CardValue and CardSuit which are called upon separately. CardValue returning the Value as an integer to be used when deciding a high card in poker and a hand total in blackjack. CardSuit returns the first character of a suit e.g “H” for hearts “D” diamonds etc. The card values are also determined depending on the gamemode passed as a parameter when the subroutine is called upon. This allows this one subroutine to be used to get the value of any card in any gamemode and allows the easy integration for future possible gamemodes compared to having multiple subroutines for different gamemodes (poker and blackjack).

Player variables	

	I created these nested lists all within players list and dealer list and I have attempted to make a commented visualisation to make it easier to understand and comprehend what blocks of code hold what data and what classes will be accessing them. I was planning on making a bunch of separate lists or making a Class Player and having to mess with inheritance which can over complicate the code compared to simpler nested list statements this is much simpler and easier to adapt into all the other subroutines.
Naming conventions: # variables = camelCase # parameters = camelCase # bool variables = isCamelCase # constants = UPPERCASE # constant pointers = _UPPERCASE # subroutines/functions/procedures = PascalCase	I changed the naming conventions for all the code that has been produced so my code can stay consistent throughout
	Made a visualisation for the code and I have changed variable names across all screenshots to follow the naming convention mentioned above

```
STANDARDDECK = ["AcS", "AcH", "AcD", "AcC"
               , "02S", "02H", "02D", "02C"
               , "03S", "03H", "03D", "03C"
               , "04S", "04H", "04D", "04C"
               , "05S", "05H", "05D", "05C"
               , "06S", "06H", "06D", "06C"
               , "07S", "07H", "07D", "07C"
               , "08S", "08H", "08D", "08C"
               , "09S", "09H", "09D", "09C"
               , "10S", "10H", "10D", "10C"
               , "JaS", "JaH", "JaD", "JaC"
               , "QuS", "QuH", "QuD", "QuC"
               , "KiS", "KiH", "KiD", "KiC"
                ]
```

```
def CardIntegration(playerCards, nCards):
    global shuffledDeck
    global head
    found = False
    for i in range(nCards):
        while not found:
            card = str(input("Input card: "))
            if shuffledDeck.count(card) == 1:
                found = True
            else:
                print("Card does not exist in deck")
        shuffledDeck.remove(card)
        playerCards.append(card)
    head -= nCards
    return playerCards
```

```
def DealCards(playerCards, nCards):
    global shuffledDeck
    global head
    for i in range(nCards):
        card = shuffledDeck[head]
        shuffledDeck.pop(head)
        head -= 1
        playerCards.append(card)
    return playerCards
```

```python
def CardValue(card, gamemode):
    value = card[0:2]
    if gamemode == "blackjack":
        if value == "Ac":
            value = 11
        elif value in ["Ki", "Qu", "Ja"]:
            value = 10
        else:
            value = int(value)
    elif gamemode == "poker":
        if value == "Ac":
            value = 14
        elif value == "Ki":
            value = 13
        elif value == "Qu":
            value = 12
        elif value == "Ja":
            value = 11
        else:
            value = int(value)
    else:
        print("gamemode error")

    return value
```

```python
def CardSuit(card):
    suit = card[2:]
    return suit
```

Testing

Test table

Test number	Description	Type of test	Test data	Expected result	Actual result	Evidence
1.1.a	Physical input valid card	Valid	Input “07H” via card integration	Accepted and logged. Removes card from internal shuffled deck		
1.1.b	Physical input invalid string	Invalid	Input “11Z” or “AA”	Rejected: incorrect card format		
1.1.c	Duplicate card input	Invalid	Player already holds “7H” so input “7H” again	Rejected: duplicate card		
1.2.a	Check deck contains 52 cards	Valid	len(cards)	Returns 52		
1.2.b	Check every card string is unique	Valid	Convert cards to a set	Set length = 52 (no duplicates)		
1.2.c	Check naming conventions	Valid	Input "AcS","10H","KiD",etc	Each string follows pattern [Rank][Suit] where Rank ∈ {A,1–10,J,Q,K}, Suit ∈ {S,H,D,C}		
1.2.d	Shuffle deck order	Valid	Call ShuffleCards() then compare to original	Deck orders are different but all 52 unique cards still exist		
1.3.a	Deal one card	Valid	Call DealCards(p1,1)	Player1 length +1, deck length –1		
1.3.b	Deal full deck	Boundary	DealCards(p1,52)	Player1 has 52 unique cards, shuffled deck length is 0		
1.3.c	Deal more cards than available	Boundary	With 5 cards left call DealCards(p1,6)	Operation rejected: no deck change		

1.3.d	Deal 0 cards	Boundary	DealCards(p1,0)	No change to deck or player and no error		
1.3.e	Invalid number input	Invalid	DealCards(p1,"three") or DealCards(p1,-3)	Function rejects with clear message		
1.3.f	Empty deck deal	Boundary	Dequeue all 52 then deal again	“No cards remaining” message - no error		
1.2.post	Shuffle randomness	Robustness	Run ShuffleCards() 100× and compare all the deck orders	No duplicates within a deck, the greater majority of shuffled decks must be unique		

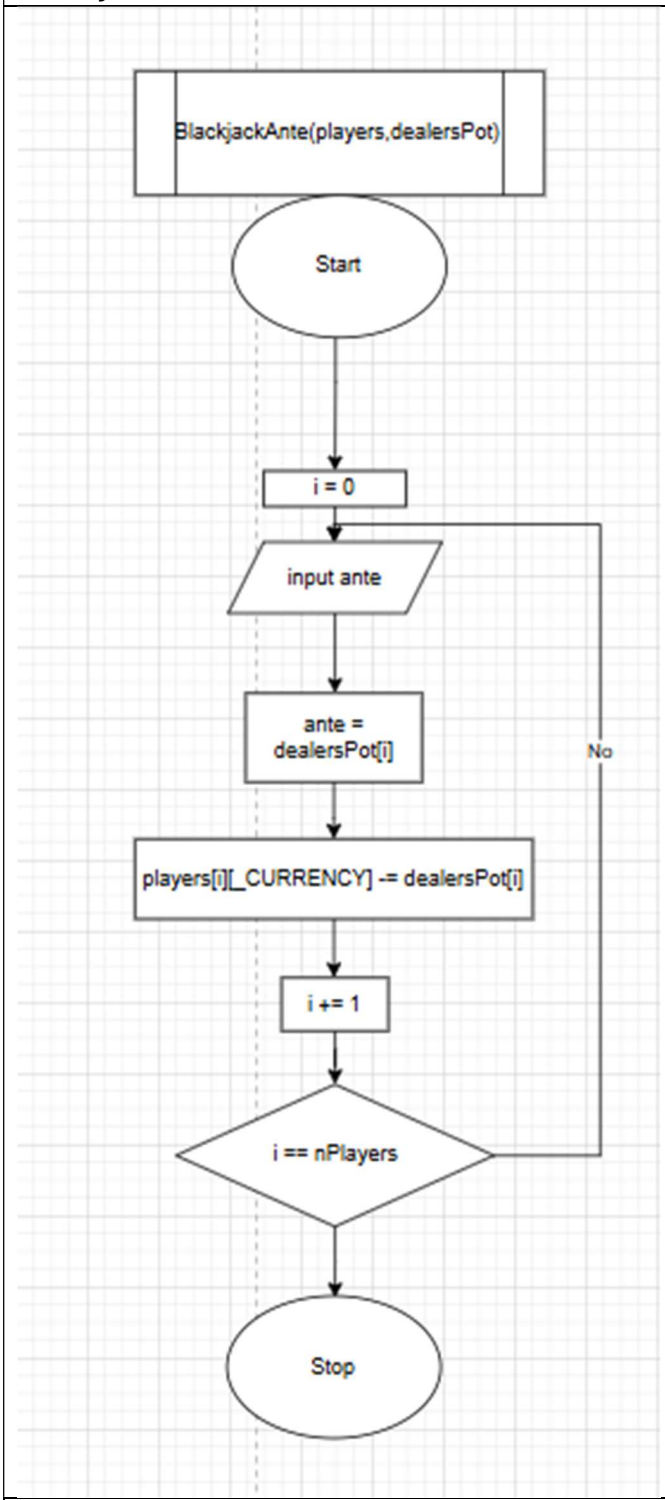
Evidence referenced - screenshots or videos (with timestamps)

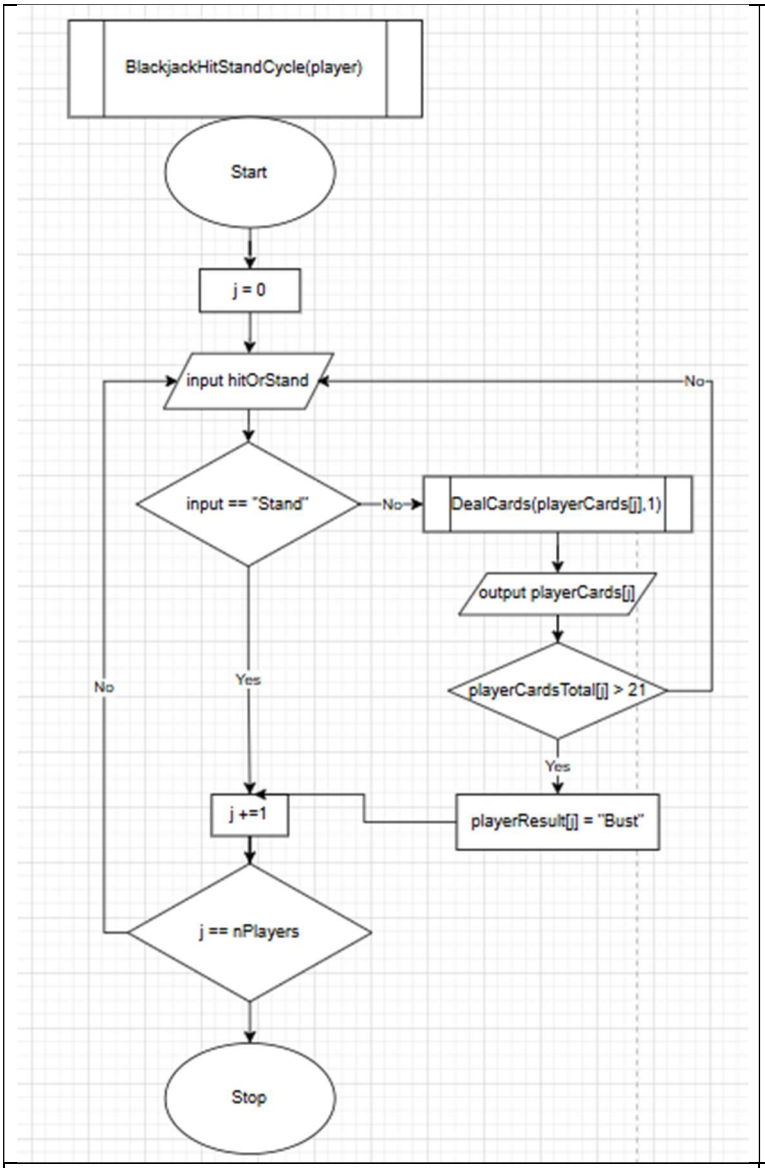
Review of how well your stage has gone

Stage 2 – Blackjack Gameplay

Design

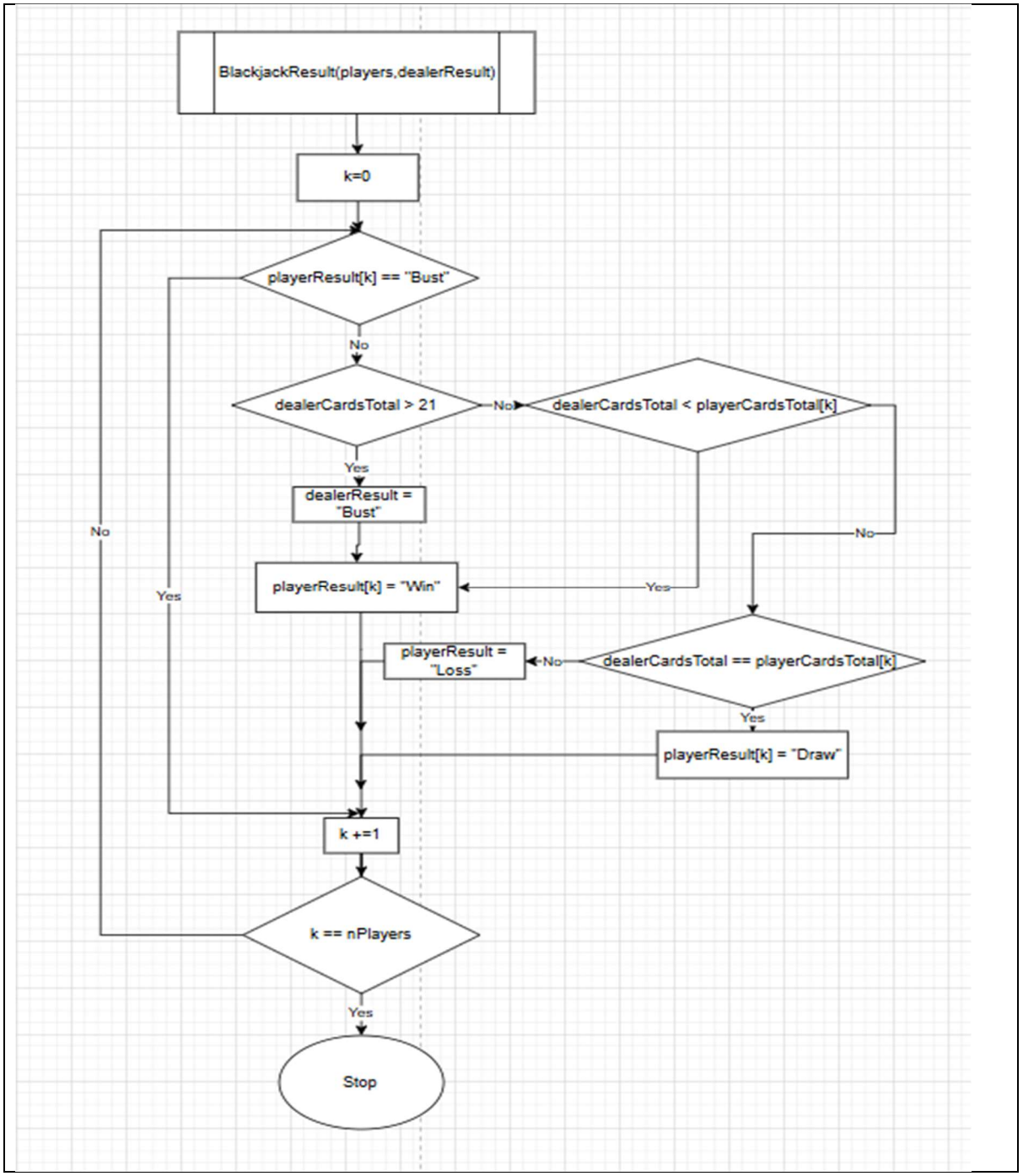
Algorithms - Should contain flowcharts / pseudocode

Blackjack Ante bet subroutine	
	Blackjack ante subroutine explanation
Blackjack Hit or stand inputs subroutine	



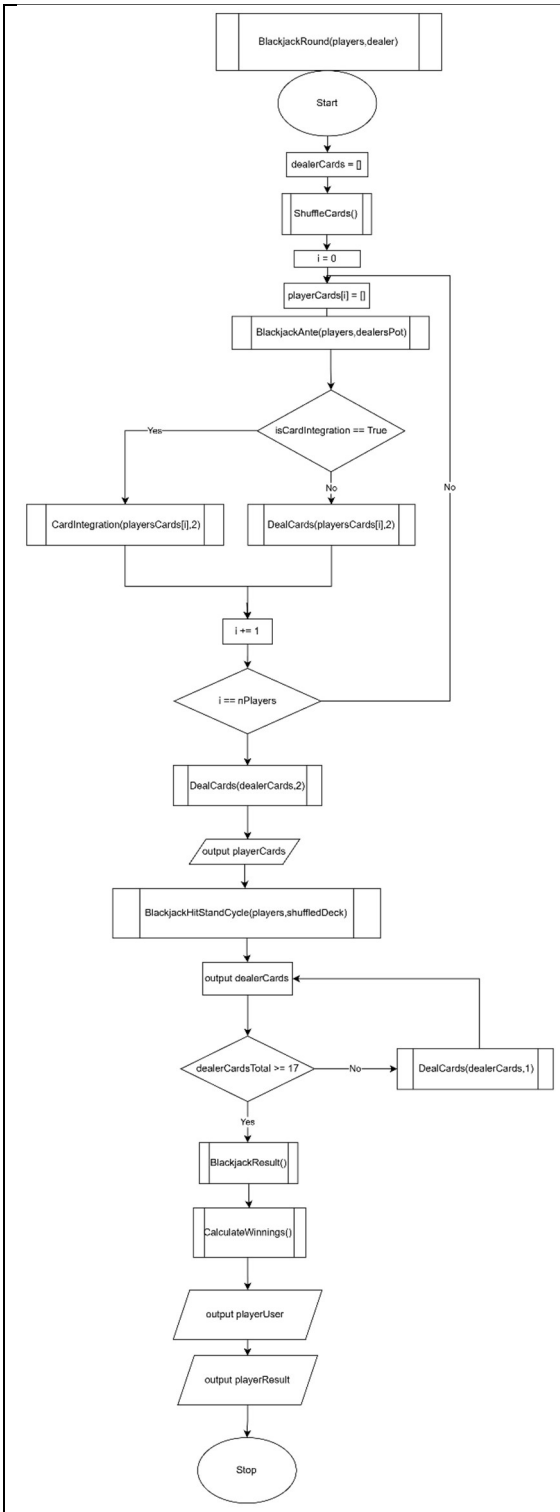
Blackjack hit stand routine explanation

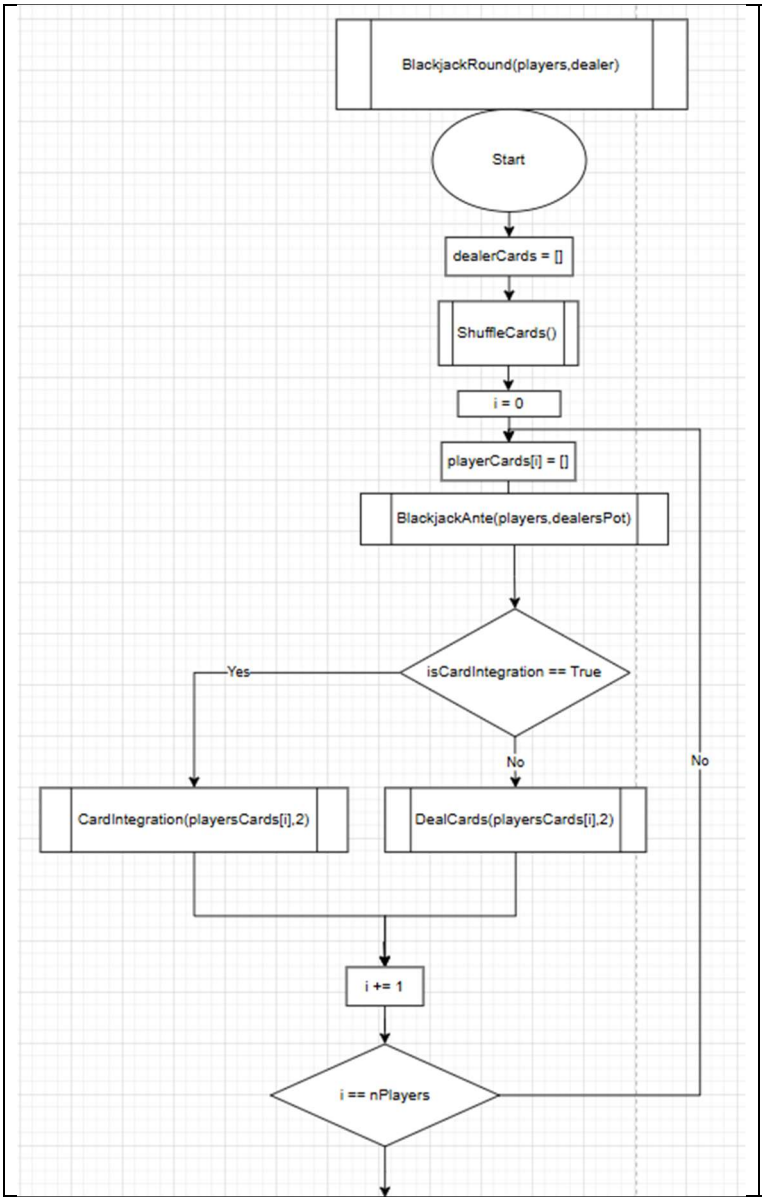
Blackjack end of round results


BlackjackResult(players,dealerResult)
k=0
playerResult[k] == "Bust"
No
dealerCardsTotal > 21
No
dealerCardsTotal < playerCardsTotal[k]
Yes
dealerResult = "Bust"
No
No
playerResult[k] = "Win"
Yes
Yes
playerResult = "Loss"
No
dealerCardsTotal == playerCardsTotal[k]
Yes
playerResult[k] = "Draw"
k +=1
k == nPlayers
Yes
Stop

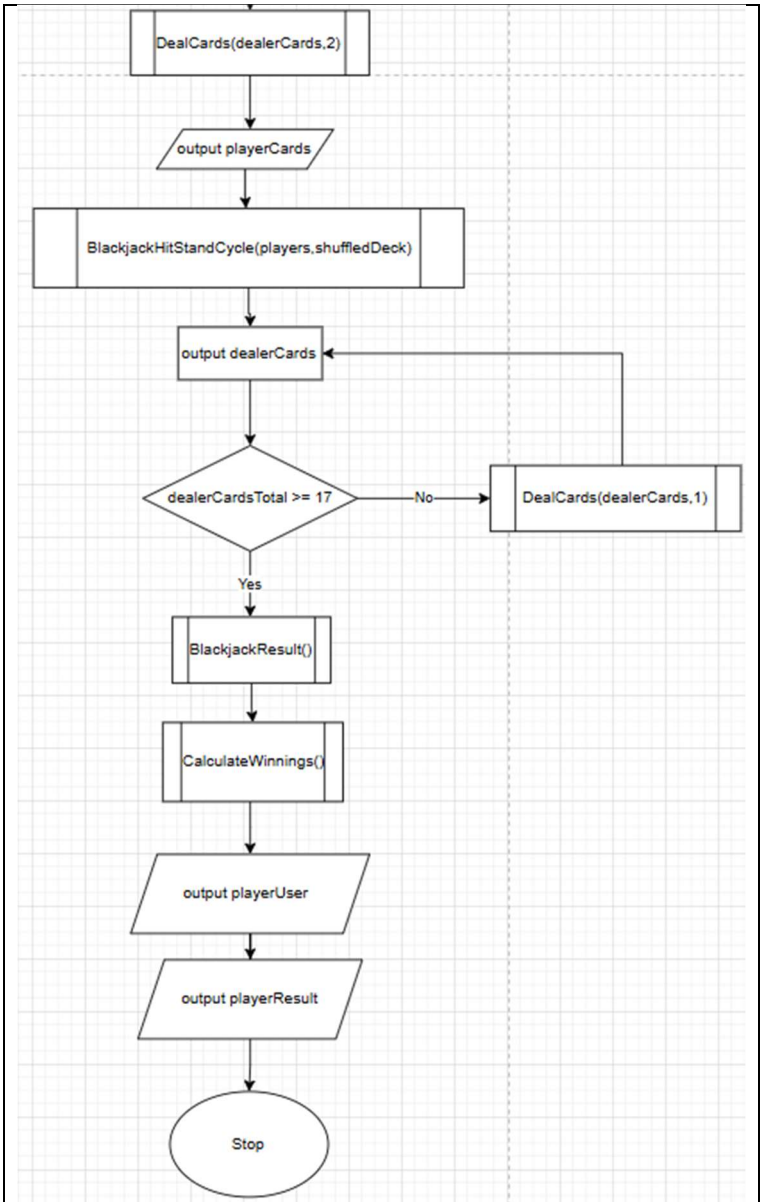
Result subroutine explanation

Blackjack Full round subroutine


BlackjackRound(players,dealer)
Start
dealerCards = []
ShuffleCards()
i = 0
playerCards[i] = []
BlackjackAnte(players,dealersPot)
isCardIntegration == True
Yes
No
CardIntegration(playersCards[i],2)
DealCards(playersCards[i],2)
i += 1
i == nPlayers
No
DealCards(dealerCards,2)
output playerCards
BlackjackHitStandCycle(players,shuffledDeck)
output dealerCards
dealerCardsTotal >= 17
No
DealCards(dealerCards,1)
Yes
BlackjackResult()
CalculateWinnings()
output playerUser
output playerResult
Stop



Blackjack Full round subroutine explanation



Development

Show the code build over time

Justify your decisions

Screenshot errors and your fixes

Videos or screenshots of what it looks like when run

Reference tutorials used

Testing

Test table

Test number	Description	Type of test	Test data	Expected result	Actual result	Evidence

Evidence referenced - screenshots or videos (with timestamps)

Review of how well your stage has gone

Evaluation