Software Design Specification

for

Black Y Poker

Version 1.0

Prepared by Casey Munga

The Red Howler

2-16-2020

Table of Contents

Table of Contents ii
1. Introduction 1
1.1 Purpose 1
1.2 Document Conventions 1
1.3 Intended Audience and Reading Suggestions 1
1.4 Product Scope 1
1.5 References 1
2. Overall Description 2
2.1 Product Perspective 2
2.2 Product Functions 2
2.3 User Classes and Characteristics 2
2.4 Operating Environment 2
2.5 Design and Implementation Constraints 2
2.6 User Documentation 2
2.7 Assumptions and Dependencies 2
3. External Interface Requirements 3
3.1 User Interfaces Overview 3
3.2 Hardware Interfaces 3
3.3 Software Interfaces 3
4. Design Considerations 3
4.1 Assumptions and Dependencies 3
4.2 General Constraints 4
4.3 Development Methods 4
5. System Architecture 4
5.1 Architectural Strategies 4
5.2 High level Overview of System Architecture 5
6. Human Interface Design 5
6.1 Screen Images 5
6.2 Screen Objects and Actions 6
7. Detailed System Design 7
7.0 Data Structures 7
7.1 Initial Screen Module 7
7.2 Menu 10
7.3 Rules 12
7.4 Play Round 16
7.5 Betting 20
7.6 Deal Cards 22
7.7 Generate Cards 23
7.8 ValidateHands 24
7.9 validateWinner 28
7.10 Component 2 Name 29
7.11 PlayRound 30
Appendix A: Glossary 31

Revision History

Name	Date	Reason For Changes	Version
Casey Munga	**2-16-2020**	Initial Software specifcation conversion	1.0

1. Introduction

1.1 Purpose

The purpose of this document is to convert the software specifications to the software design specification for Black Y Poker Company

1.2 Document Conventions

Highlighting is used to bring attention to the importance of the words so that the functionality will br focused on. The glossary also aids in helping to define the meanings of the terms found herein.

1.3 Intended Audience and Reading Suggestions

The document is intended for, agents such as developers, project managers, and other IT personnel who may be involved in the production of this software.

1.4 Product Scope

1.4.1 Purpose

This product is an online variation and simplified version of a real poker game. The dealer is an AI and the gamePlayer is the user. The program will allow betting and will determine a winner.

1.4.2 High Level Flow

The dealer is an AI and the game Player is the user. The user will be invited to input his name and will be given points if they are a new user or will have their points displayed if they are a returning player. The user may choose from a menu system to do various things such as viewing their scores, or playing a round of poker.
The software will allow betting, folding, matching and raising bets in the program. The interface will be GUI and the program Ai is the dealer. A winner will be declared when the All In button is selected. To quit the program the player may select and exit button at which time the player and dealer stats will be saved to a test file.

1.5 References

The document Simplified Software Specification Black Y Poker was used as the reference to gain the materials needed produce this document.

2. Overall Description

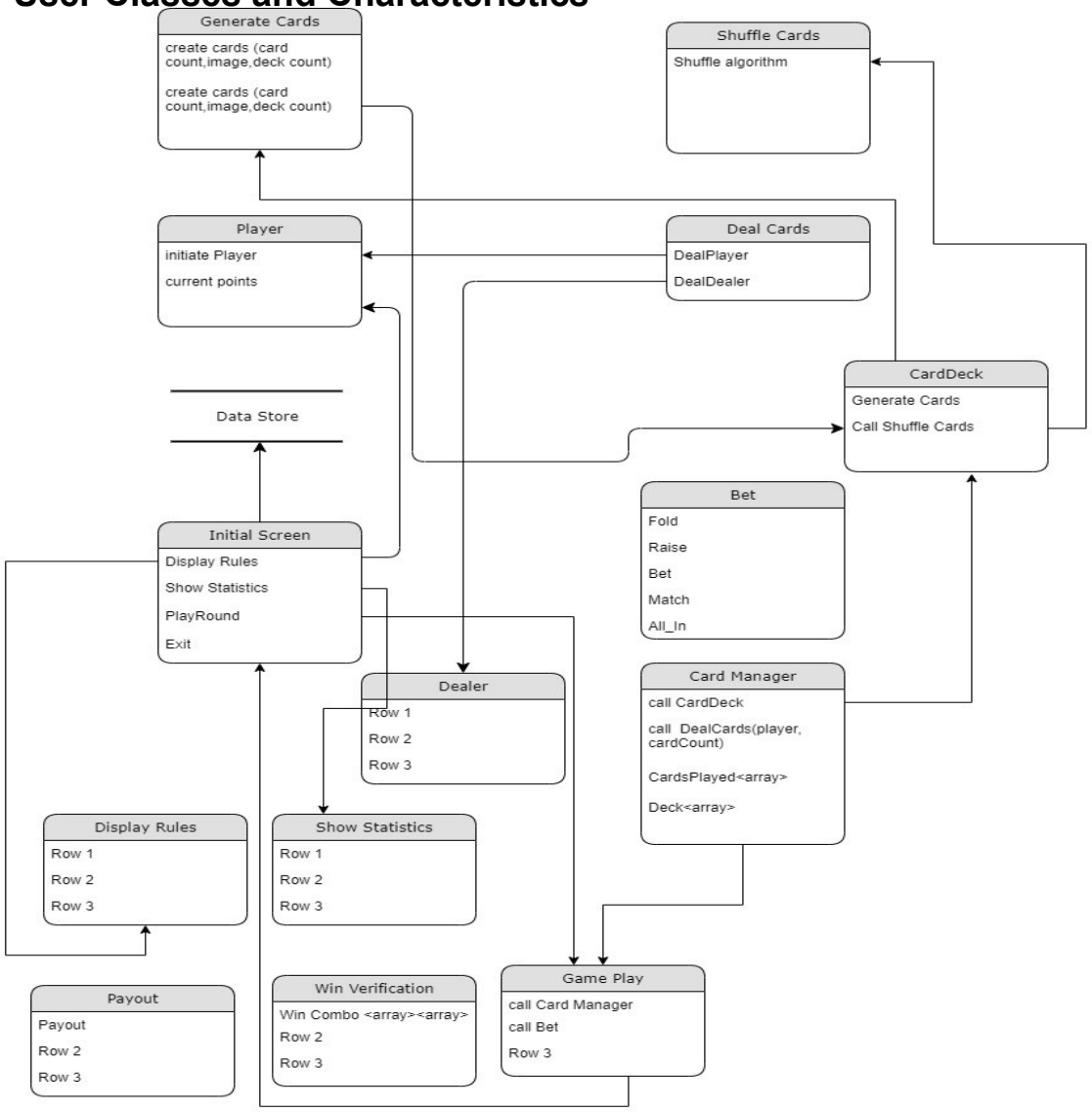
2.1 Product Perspective

This product is a new program, that is a single user GUI interface for a single sign on deskop computer

2.2 Product Functions

<Summarize the major features the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here.>

2.3 User Classes and Characteristics



2.4 Operating Environment

The Operating Systems. will be a desktop windows model computer, single sign-on.

2.5 Design and Implementation Constraints

This program is to be used as a single user sign on. It does not have the capacity to host mutlple players at the same time. The program must be written in python. The program file that keeps the game data must be named as **stats.txt** and must be housed in the same directory of the game (this software).

2.6 User Documentation

A set of Rules will be listed on the screen of how to play the game. As well the glossary of this document shows how to win the game.

2.7 Assumptions and Dependencies

The program will assume that the python software is installed and up to date.

3. External Interface Requirements

3.1 User Interfaces Overview

The details of each screen and the messages will be described in Human Interface section.

3.2 Hardware Interfaces

none

3.3 Software Interfaces

none

4. Design Considerations

4.1 Assumptions and Dependencies

None that is different than has already been described.

4.2 General Constraints

none

4.3 Development Methods

This program will be written as a procedural language and use that format. There will be no OOP

5. System Architecture

5.1 Architectural Strategies

The program is to be developed in python. Needs are computer that is equipped with the software, and peripherals as well as a software to write the software and compile it.

5.2 High level Overview of System Architecture

The program has several modules. A screen function, that displays the initial screen Generate cards shuffle and deal cards. A dealer AI module, And a module that validates the winning hands and the winners. There is an statistics file to save the player name, points and game stats and a Menu function that drives the program from a high level.

6. Human Interface Design

6.1 Screen Images

1.	10♥ J♥ Q♥ K♥ A♥	**Straight Flush**
2.	2♥ 7♦ 7♣ 7♠ 7♥	**Four of a Kind**
3.	2♦ 2♥ 3♣ 3♦ 3♥	**Full House**
4.	4♣ 6♣ 7♣ T♣ Q♣	**Flush**
5.	A♥ 2♦ 3♣ 4♣ 5♦	**Straight**
6.	2♦ 2♣ 2♥ Q♦ K♣	**Three of a Kind**
7.	2♦ 2♥ 3♣ 3♦ 7♥	**Two Pair**
8.	4♠ 4♥ 5♦ T♠ K♠	**One Pair**

```
9.      2♥ 5♥ 7♦ 8♣ 9♠          High Card
```

6.2 Screen Objects and Actions

7. Detailed System Design

7.0 Data Structures

The data types as this program is to be written in a procedural language are primitative data types, enumerated types, string and text types, composite types and pointers as well as other datatypes not covered here commonly used.

7.1 Initial Screen Module

This module is the starting point for the program.

7.1.1 Responsibilities

This module will:

- Receive input for the user name.
- Verify that the statistic file exists.
- Read the file to search for the user name
- Verify that the user name exist and get the information to generate player and the dealer
- Create a new player with the user name and gives the user and dealer 500 points
- Define and Initialize the major data structures. Such as the cards, deck, dealer player.

7.1.2 Uses/Interactions

7.1.3 Constraints

- Name input must be alphanumeric
- Name input length cannot exceed 20 characters
- Statistics file and path must already exist in same location as program,
- A clickable button labeled Okay must be presented to the user in order to accept player Name input.

7.1.4 Composition

Method	Description
Main()	Runs the program: Loops to validate player input; loops to generate Menu. Create player Struct and initialize player and dealer.
void validatePlayerInput()	Prompt for input and validate player constraints
struct validateFile()	Open Statistics File .
Player createPlayer(Player player)	check for player name Sets player name games and points and returns player If a player point is 0 then award 500 points to player. Sets game = new
Player createDealer(Player dealer)	Sets dealer games won, games lost and Dealer points
int generateMenu()	Displays menu on screen and prompt for player input. Returns player menu choice

Resources
Statistics data file will be read by the program to validate player name and get points if applicable.

7.1.5 Processing

Main()

1. Start of program
2. Loop prompt for input validatePlayerInput() until the constraints are met.
3. OpenStats File
4. Call validateFile() and check if player exist and returns player information
5. Calls ValidateFile() to find dealer and return dealer information
6. Calls createPlayer(struct player) an generate a player
7. Calls createPlayer(struct player) and generate a dealer
8. Loop Calls generateMenu – to display a menu of options to the screen until Exit option is selected
9. Closes program

validatePlayerInput():None

Loop until constraints have been satisfied
1. Prompts for playerName
2. Checks for PlayerName Length constraint of no greater than 20 characters.
3. Checks for alphanumeric character by using regular expressions format.
4. Displays appropriate error message defining constraints if they have been violated.

Int validateFile():None
1. Open Statistics file using a try statement
2. If file does not exist then throw an error message and stop program.

;

Player createPlayer(Player player)
1. Sequential search in stats file for playerName
2. If playerName exists then assign data to the player and return a Player structure
3. If playerName does not exist then set games won and games lost to 0, set points to 500, set gameStatus = new;

Player createDealer(Player dealer)
1. If gameStatus is New then Set dealer games won and games lost to 0 and set points to 500
2. If gameStatus is continued, get dealerName and date and assign data to dealer
3. Return dealer

7.2 Menu

This module describes the menu system that will drive the program.

7.2.1 Responsibilities

This module is responsible for displaying the four menu options

7.2.2 Uses/Interactions

This module is used by the Main program.

7.2.3 Constraints

Preconditions:

- The player and dealer data must be loaded and displayed.

Other constrainsts

- A menu must be display with 4 options
- The first option must be labeled 1. Display Rules
- The second option must be labeled 2. Show Statistics
- The third option must be labeled 3. Play Round
- The fourth option must be labeled 4. Exit
- Menu options must be clickable.
- Player may only select one option at a time by clicking on the appropriate item.
- Stats file must already be opened.

7.2.4 Composition

Function	Description
generateMenu():none	Runs the menu module, loops to display the 4 menu option on the screen.
displayRules():none	Displays the rules to the screen and how the player can win.
showStats():none	Displays the games won and lost by the player.
None:playRound	Player will be allows to play a single round of the game.
None: exitGame()	Ends the game, close the stats file and Ccoses the software.
generateMenu():int	Displays menu on screen and prompt for player input. Returns player menu choice

7.2.5 Resources

Statistics File names stats.txt

7.2.6 Processing

…
generateMenu():None

1. Loop to display 4 menu choices
2. Call displayRules()
3. Call showStats()
4. Call playRound()
5. Call exitGame()

displayRules():None

1. Display rules of game and how player can win.
2. Present player with a player button to return to the menu
3. Return to the menu.

showStats():None

1. Display games player won and lost.
2. Present player with a player button to return to the menu
3. Return to the menu.

playRound():None

1. Player will play a single round of the game.
2. Present player with a player button to return to the menu after the game is completed.
3. Return to the menu.

exitGame():None

1. Present player with confirmation message and button.
2. Call UpdateStats().
3. Close program

7.3 Rules

This module will display the rules of the game and will present the winning combinations

```
10♥ J♥ Q♥ K♥ A♥        Straight Flush
2♥ 7♦ 7♣ 7♠ 7♥         Four of a Kind
2♦ 2♥ 3♣ 3♦ 3♥         Full House
4♣ 6♣ 7♣ T♣ Q♣         Flush
A♥ 2♦ 3♣ 4♣ 5♦         Straight
2♦ 2♣ 2♥ Q♦ K♣         Three of a Kind
2♦ 2♥ 3♣ 3♦ 7♥         Two Pair
4♠ 4♥ 5♦ T♠ K♠         One Pair
2♥ 5♥ 7♦ 8♣ 9♠         High Card
```

7.3.1 Responsibilities

This module is responsible for:

- Displaying the rules of the game and the wining combinations
- Presenting a clickable button to the player to return to the Menu option.
- Verification of a win for the current player hand

7.3.2 Uses/Interactions

This module is called by the Menu option

7.3.3 Constraints

- The menu must be displayed
- The player must have selected the Display rules option.
-

7.3.4 Composition

Function	Description
None displayRules()	This module will display the written rules with visuals
handType validateHand(Hand currentHand)	Process the winning Hand,
boolean straightFlush(Hand currentHand)	Validate and verify that the cards of the current hand are of the same suit and will place the values in consecutive numerical order (rank)
boolean fourOfaKind(Hard currentHand)	Validate and verify that 4 of the cards of the current hand are of the same numerical order(rank)
boolean fullHouse(Hand currentHand)	Validate and verify that 3 cards are of the same rank and the remaining 2 cards are of another rank
boolean Flush(hand currentHand)	Validate that all cards are of the same suit
boolean straight(hand currentHand)	Validate that all cards are in numerical consecutive order
boolean threeOfaKind(hand currentHand)	Validate there are 3 cards of the same rank
boolean twoPairs(hand currentHand)	Validate that there are 2 sets of 2 cards of the same rank
booleano onePair(hand currentHand)	Validate that there are one set of 2 cards of the same rank
Card highCard(hand currentHand)	Validate the highest card in the hand

7.3.5 Resources

none

7.3.6 Processing

DisplayRules()

1. Displays rules on screen
2. Player is presented with a clickable Menu button
3. If button is clicked the program returns to the Main Menu

validateHand(Hand currentHand) : HandType

1. Initalize winning indicator to false.
2. Selection statement for each function
    a. If win Indicator is false then call **straightFlush**(…) returns win Indicator
    b. If win Indicator is false then call **fourOfaKind**(…) returns win Indicator
    c. If win Indicator is false then call **fullHouse**(…) returns win Indicator
    d. If win Indicator is false then call **Flush**(…)returns win Indicator
    e. If win Indicator is false then call **straighHand**(…)returns win Indicator
    f. If win Indicator is false then call **threeOfaKind**(…) returns win Indicator
    g. If win Indicator is false then call **twoPairs**(…)returns win Indicator
    h. If win Indicator is false then call **onePair**(…)returns win Indicator
    i. If win Indicator is false then call **highCard** (…)returns win Indicator
3. Set and Return HandType
4.

straightFlush(hand) : sorted Hand

1. Sort hand

   a. Sort on cardName
2. Initialize Boolean control variable = true
3. get the first card in the sorted hand
   a. **loop** until length of hand or Won is true (Boolean control variable)
      i. get next card in the sorted hand
      ii. if the previous card value is > current card value and previous card suit = current card suit
      iii. else set Boolean controlled variable Won to false
4. **return win Indicator**

fourOfaKind(hand) : sorted Hand

1. Sort hand
   a. Sort on cardValue
2. Initialize the win indicator variable = false
   a. Select second card of the hand, and compare with the fourth card of the hand.
   b. If they are both the same then set win indicator = true
3. **return win Indicator**

fullHouse(hand) : sorted Hand

1. Sort hand
   a. Sort on cardValue
2. Initialize win indication to false initialize Boolean control variable to false,
3. Slice first four cards from the card dictionary
   a. Compare first and fourth card element for equality
   b. If they are the same then set win indication to true
   c. If they are not the same then slice last four cards from the hand
   d. Compare the first and fourth resulting card element from that slice
   e. If they are the same set the win indicator to true
4. Return win indicator

straight(hand) : sorted Hand

1. Sort hand
   a. Sort on cardValue
2. Initialize the Boolean control variable = false
   Initialize win indicator to true

3. Get first card of the hand
4. **Loop** until Boolean control is true or length of hand
   a. Select next card of the hand.
   b. If the previous card's value <= current card value
      i. Set Boolean control value to true (hand is not a
      ii. Set win indicator to false
   c. Else set the current card to be the previous card
5. **Return win indicator**

threeOfaKind(hand) : sorted Hand

1. Sort hand
   a. Sort on cardValue
2. Initialize the Boolean control variable = false,
3. Get the first card in the hand and initialize the previous and current card
4. **Loop** until Boolean control variable (number of cards in the hand)
   a. Get the next card in the hand

        i. Compare for equality the card values
        ii. If they are the same increment counter
    b. set the current card to be the previous card
5. If the increment counter is greater than or equal to three then set win indicator to true
6. Else set win indicator to false
7. **Return win indicator**

twoPair(hand) : sorted Hand

1. Sort hand
    a. Sort on cardValue
2. Initialize the Boolean control variable = false, initialize counter, initialize pair count
3. Get the first card in the hand and initialize the previous and current card
4. **Loop** until Boolean control variable (length cards in the hand) or pair count < 2
    a. Get the next card in the hand
        i. Compare for equality the card values
        ii. If they are the same increment counter
            1. If counter = 2 (pair of same card)
            2. Increment pair count
            3. Reset increment counter
    b. set the current card to be the previous card

5. If the pair count = 2
    a. Set win indicator to true
6. If the pair count < 2
    a. Set win indicator to false
7. Return win indicator

onePair(hand) : sorted Hand

1. Sort hand
    a. Sort on cardValue
2. Initialize the Boolean control variable = false, initialize counter, initialize win indicator to false
3. Get the first card in the hand and initialize the previous and current card
4. **Loop** until Boolean control variable (length cards in the hand) or counter >= 2
    a. Get the next card in the hand
        i. Compare for equality the card values
        ii. If they are the same increment counter
            1. If counter = 2 (pair of same card)
    a. set the current card to be the previous card
5. If counter >= 2
    a. Set win Indicator to true
6. **return win Indicator**

highCard(hand):card

1. Sort hand
    a. Dort hand on card Value
2. Get the first card in the deck
3. Return card

7.4 Play Round

This module will process the input of the player,

7.4.1 Responsibilities

This module will be responsible for:

- Processing the screen input as it pertains to the Play round of the game.
- Ensuring that they player has enough points to bet
- Player betting, Folding, Matching raising and exchanging cards.
- Ensuring that the cards are shuffled and dealt
- Calls the startGame() function which will initialize the UI, create the player, dealer,
- Calls generateDeck()
- Call ShuffleDeck()
- Dealing the initial hand to the player
- Dealing the initial hand to the dealer

6.1.1 Uses/Interactions

This process will be used by the Menu round module.

7.4.2 Constraints

- The player must have enough money to place the bet.
- Initial screen must have been run and player name must be validated
- Dealer information and stats and points must be available
- Player information and stats and points must be available
- Two rounds of betting per gsme
-

7.4.3 Composition

Function	Description
startGame(Player, Dealer)	This module initializes the game area and calls the user interface . It calls the generateCard() and shuffleDeck functions and DealCards()
generateDeck()	Creates a deck of 52 cards made uo of 13 cards for each suit
shuffleDeck():Deck	Randomly shuffle deck
dealCards(Deck,number):hand	Deals the amount of cards from the deck
dealPlayers();	Deal each player a hand
Bet5(Player, points, pot)	Player bets5 points
fold(Player,ponts,pot)	Player folds and forfeits the pot
raise5(player,points,pot)	Player raises
allIn(hand currentHand,pot)	Player runs out of points, betting stops
Match(player,points,pot)	Player matches the adversary points
Cards	Calls dealCards(Deck,number)

7.4.4 Resources

< A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.>

7.4.5 Processing

startGame : None

1. Call Initialize Screen()
2. Call generateDeck() : returns Deck
3. Call ShuffleDeck(Deck): returns shuffled Deck
4. Call dealCards
5. Call gamePlay

InitializeScreen :

1. Display game Player
    a. Display gamePlayerName
    b. Display game player games, Won and lost
    c. Display game player points
    d. Initialize gamePlayerLastMove

2. Display dealer
    a. Display dealer Name
    b. Display dealer games, won and lost
    c. Display Dealer points
    d. Initialize dealerLastMove

3. Display GameTable
    a. Display 5 buttons for betting
        i. Button BET5: Status (active)
        ii. Button FOLD: Status(active)
        iii. Button RAISE5 : Status (inactive)
        iv. Button MATCH : Status(inactive)
        v. Button ALL IN : Status(inactive)
        vi. Button CARDS : Status(Inactive)
    b. Display current pot value (initialized to 0)
    c. Hand for Dealer (cards placed face down)
    d. Hand for gamePlayer (cards placed face up)

generateDeck(): deck

1. This function will create a deck of 52 card of 13 rank cards from (A,K,Q,J,10,9,8,7,6,5,4,3,2) of 4 suits (♥, ♦ ,♣ ,♠)
2. A deck will be returned

shuffleDeck(deck):deck()

1. This function will shuffle the deck of 52 cards
2. A randomly generated shuffled deck till be returned.

dealCards(deck, 5):hand

1. This function will deal a hand of 5 cards from the deck and pop the cards off the deck.
2. The hand will be returned and assigned to a player.

DealPlayers()

1. Call dealCards() and assign to game player.
2. Call dealCards() and assign to dealer.
3.

// Game play starts

1. Initalize **rounds** to 2 –limit of betting per game

Loop until gameplayerpoints is zero or Dealer points is 0

2. If gameplayer select Bet5 and points > 0
    - **a.** Call Bet5(gamePlayer,points)
    - **b.** Update gamePlayerLastMove to Bet5

3. If gameplayer select FOLD – game stops
    - **a.** Call Fold(gamePlayer, points)
    - **b.** decrement **rounds**
    - **c.** update gamePlayerlastMove to Fold

4. If gamePlayerLastMove is Bet5 and Rounds = 2
    - **a.** Set CARDS button status to Active

5. if gamePlayer selects CARDS
    - **a.** pop 3 cards from gamePlayer Hand
    - **b.** call dealCards(deck, selectedCards)
    - **c.** Call dealerRules(Cards) – dealer may also exchange up to 3 cards
    - **d.** Prompt gamePlayer to bet again

6. If gamePlayerLastMove is “Bet5”
    - **a.** Prompt dealer for gamePlay
    - **b.** If dealer selects MATCH5
        - **i.** Call dealerRules(Match5)
        - **ii.** Update dealerLastMove to Match5
    - **c.** If dealer selects RAISE5
        - **i.** Call dealerRules(Raise5)
        - **ii.** Update dealerLastMove to Raise5
    - **d.** If dealer selects FOLD – game Stops
        - **i.** Call Fold(dealer, points)
        - **ii.** Update dealerLastMove to Fold
        - **iii.** decrement **rounds**

7. If dealerLastMove is Raise5
    - **a.** If gamePlayer selects MATCH5
        - **i.** Call Match5(gamePlayer,points)
        - **ii.** Update gamePlayerLastMove to Match5
    - **b.** If gamePlayer selects RAISE5
        - **i.** Call Raise5(gamePlayer,points)
        - **ii.** Update gsmrPlayerLastMove to Raise5
    - **c.** If gamePlayer selects FOLD
        - **i.** Call Fold(gamePlayer,points)

   ii. Update gamePlayerLastMove to Fold
   iii. decrement **rounds**

8. If Dealer points <= 0
   a. Display Message " dealer is out of points and game must be restarted"
   b. Restart game.
9. If gamePlayer points <= 0
   a. Select All_In Button
   b. Call All_In()
10. Carda
11. If rounds < 1
   a. Reset rounds to 2

7.5 Betting

Player bets for a single game os the game session

7.5.1 Responsibilities

This module will be responsible for:

- Processing the screen input as it pertains to the Play round of the game.
- Ensuring that they player has enough points to bet
- Player betting, Folding, Matching raising and exchanging cards.
-

7.5.2 Uses/Interactions

7.5.3 Constraints

- Preconditions
  - The player must have been in the program and selected the menu option Play Round.
  - If a new game has been initiated the gameplayer and dealer must have a 500 points
  - If this is not a new game the stats file for both gameplayer and dealer must be already on the screen
  - The screen UI with BET5, Fold, MATCH, RAISE5 and ALL IN buttons must be on the screen
  - Cards must be dealt to both gameplayer and dealer

7.5.4 Composition

Function	Description
bet5(player,points ,pot)	If player has enough points player can bet5 points
fold (player, points ,pot)	Player decides discard the cards and end betting and forfeits the pot
match(player ,points,pot)	Player plays same bet as adversary
raise5(player, points,pot)	Player raises the bet by 5 points for a new bet at 10 points
All_In(player, points,pot)	If Player of Adversary runs out of points the betting stops calls validateWinner()

7.5.5 Resources

none

7.5.6 Processing

Bet5(player, points):None

1. If player is the gamePlayer and points > 0
    a. Decrement gameplayer points by 5
    b. Increment pot by 5
2. If player is dealer and points > 5
    a. Decrement dealer points
    b. Increment pot by 5

Match(player, points):None

1. If player is the gamePlayer and points > 0
    a. Decrement gameplayer points by 5
    b. Increment pot by 5
2. If player is dealer and points > 5
    a. Decrement dealer points by 5
    b. Increment pot by 5

Raise5(player, points):None

1. If player is the gamePlayer and points > 10
    a. Decrement gameplayer points by 10
    b. Increment pot by 10
2. If player is dealer and points > 10
    a. Call PlayerRules(points,status)
    b. Decrement dealer points by 10
    c. Increment pot by 10

Fold(player ,points):None

a. If player is the gamePlayer
    i. Increment dealer points by pot
    ii. Reset pot to 0

b. If player is dealer
    i. Increment the dealer points by the pot
    ii. Reset pot to zero

All_In(player ,points):None

a. If player is the gamePlayer
    iii. Increment dealer points by pot
    iv. Reset pot to 0

b. If player is dealer
    v. Increment the dealer points by the pot
    vi. Reset pot to zero

7.6 Deal Cards

This module will shuffle the cards at a the beginning of a new game and will deal the cards to the player and dealer.

7.6.1 Responsibilities

The responsibility of this module :

- Deal the cards
- Keep a list of the cards that are left in the deck

7.6.2 Uses/Interactions

This module is called by the Play Round menu option

7.6.3 Constraints

- Cards must be generated shuffled
- Deck must contain 52 distinct cards
- Game must already be in play
- No more than 5 cards can be dealt at one time
- Player hand cannot be greater than 5

7.6.4 Composition

Function	Description
dealCards(Deck, number)	This function will deal the cards that are and create a hand.

7.6.5 Resources

This module will be used by the PlayRound module

7.6.6 Processing

dealCards(deck, numberOfCards):Hand

1. **Loop** until numberOfCards and deck still has enough cards
    a. Get card from top of deck
    b. pop card from deck
    c. Add card to player hand
2. Return hand to caller

7.7 Generate Cards

This module will generate the cards at the beginning of a new game, will generate a deck of cards and shuffle them.

7.7.1 Responsibilities

The responsibility of this module:

- Create a deck of cards
- Shuffle the cards

7.7.2 Uses/Interactions

This module is called by the Play Round Module.

7.7.3 Constraints

- Play round Module must be selected open and engaged
- Deck must contain 52 distinct cards
- Shuffle algorithm must be randomly generated
- Suit of cards must be 4 and distinct (Heart, Spades, Diamonds, Clubs)
- Deck must be made up of 4 suits of 13 Rank Cards each
- Rank is as follows {A, K,Q,J,10,9,8,7,6,5,4,3,2}
- Must use lists

7.7.4 Composition

Function	Description
Deck generateDeck()	This function will generate cards according to the constraints
shuffleCards(Deck)	This function will shuffle the cards and place them in a deck

7.7.5 Resources

This module will be used by the PlayRound module.

```
deft Player {

 // player structure
  string playerName
  int gamesWon
  int gamesLost
  int points
};

def Card {

// card structure
        def cardName {
                A=14 ,K=13,Q=12, J=11, 10=10,
                9=9, 8=8, 7=7, 6=6, 5=5, 4=4, 3=3, 2=2
```

```
                }
        Def suit {hearts, spades, diamonds, clubs}
};


Def  Deck {
// 52 cards
        Card gameDeck [52];
};


Def  gameStatus{

// sets is new or continued
        New,
        Continued

};

Def  Hand{
//Hand: consists of 5 cards
 Cards[5]
```

7.7.6 Processing

generateDeck() : Deck

1. **Loop**
   a. For every cardName and suit in the cardName structure and the suit create a card.
   b. Add Card to the gameDeck.

shuffleDeck(deck): Deck (already shuffled)

1. Call internal library function random.shuffle (deck) to shuffle the deck of cards.

7.8 ValidateHands

This module validates the hands that are dealt after the

7.8.1 Responsibilities

This module takes a player hand sorts it and validate the type of hand the player and dealer has. The resultant sorted and validated hand will be used in the the comparison of who the actual winning hand

7.8.2 Uses/Interactions

This module is called by ValidateWins()

7.8.3 Constraints

- The program must be open and the Play

7.8.4 Composition

Function	Description
None displayRules()	This module will display the written rules with visuals
handType validateHand(Hand currentHand)	Process the winning Hand,
boolean straightFlush(Hand currentHand)	Validate and verify that the cards of the current hand are of the same suit and will place the values in consecutive numerical order (rank)
boolean fourOfaKind(Hard currentHand)	Validate and verify that 4 of the cards of the current hand are of the same numerical order(rank)
boolean fullHouse(Hand currentHand)	Validate and verify that 3 cards are of the same rank and the remaining 2 cards are of another rank
boolean Flush(hand currentHand)	Validate that all cards are of the same suit
boolean straight(hand currentHand)	Validate that all cards are in numerical consecutive order
boolean threeOfaKind(hand currentHand)	Validate there are 3 cards of the same rank
boolean twoPairs(hand currentHand)	Validate that there are 2 sets of 2 cards of the same rank
booleano onePair(hand currentHand)	Validate that there are one set of 2 cards of the same rank
Card highCard(hand currentHand)	Validate the highest card in the hand

7.8.5 Resources

none

7.8.6 Processing

DisplayRules()

4. Displays rules on screen
5. Player is presented with a clickable Menu button

6. If button is clicked the program returns to the Main Menu

validateHand(Hand currentHand) : HandType

5. Initalize winning indicator to false.
6. Selection statement for each function
    a. If win Indicator is false then call **straightFlush**(…) returns win Indicator
    b. If win Indicator is false then call **fourOfaKind**(…) returns win Indicator
    c. If win Indicator is false then call **fullHouse**(…) returns win Indicator
    d. If win Indicator is false then call **Flush**(…)returns win Indicator
    e. If win Indicator is false then call **straighHand**(…)returns win Indicator
    f. If win Indicator is false then call **threeOfaKind**(…) returns win Indicator
    g. If win Indicator is false then call **twoPairs**(…)returns win Indicator
    h. If win Indicator is false then call **onePair**(…)returns win Indicator
    i. If win Indicator is false then call **highCard** (…)returns win Indicator
7. Set and Return HandType
8.

straightFlush(hand) : sorted Hand

5. Sort hand
    a. Sort on cardName
6. Initialize Boolean control variable = true
7. get the first card in the sorted hand
    a. **loop** until length of hand or Won is true (Boolean control variable)
        i. get next card in the sorted hand
        ii. if the previous card value is > current card value and previous card suit = current card suit
        iii. else set Boolean controlled variable Won to false
8. **return win Indicator**

fourOfaKind(hand) : sorted Hand

4. Sort hand
    a. Sort on cardValue
5. Initialize the win indicator variable = false
    a. Select second card of the hand, and compare with the fourth card of the hand.
    b. If they are both the same then set win indicator = true
6. **return win Indicator**

fullHouse(hand) : sorted Hand

5. Sort hand
    a. Sort on cardValue
6. Initialize win indication to false initialize Boolean control variable to false,
7. Slice first four cards from the card dictionary
    a. Compare first and fourth card element for equality
    b. If they are the same then set win indication to true
    c. If they are not the same then slice last four cards from the hand
    d. Compare the first and fourth resulting card element from that slice
    e. If they are the same set the win indicator to true
8. Return win indicator

straight(hand) : sorted Hand

6. Sort hand
    a. Sort on cardValue
7. Initialize the Boolean control variable = false
   Initialize win indicator to true

8. Get first card of the hand
9. **Loop** until Boolean control is true or length of hand
    a. Select next card of the hand.
    b. If the previous card's value <= current card value
        i. Set Boolean control value to true (hand is not a
        ii. Set win indicator to false
    c. Else set the current card to be the previous card
10. **Return win indicator**

threeOfaKind(hand) : sorted Hand

2. Sort hand
    a. Sort on cardValue
3. Initialize the Boolean control variable = false,
4. Get the first card in the hand and initialize the previous and current card
5. **Loop** until Boolean control variable (number of cards in the hand)
    a. Get the next card in the hand
        i. Compare for equality the card values
        ii. If they are the same increment counter
    b. set the current card to be the previous card
6. If the increment counter is greater than or equal to three then set win indicator to true
7. Else set win indicator to false
8. **Return win indicator**

twoPair(hand) : sorted Hand

8. Sort hand
    a. Sort on cardValue
9. Initialize the Boolean control variable = false, initialize counter, initialize pair count
10. Get the first card in the hand and initialize the previous and current card
11. **Loop** until Boolean control variable (length cards in the hand) or pair count < 2
    a. Get the next card in the hand
        i. Compare for equality the card values
        ii. If they are the same increment counter
            1. If counter = 2 (pair of same card)
            2. Increment pair count
            3. Reset increment counter
    b. set the current card to be the previous card

12. If the pair count = 2
    a. Set win indicator to true
13. If the pair count < 2
    a. Set win indicator to false
14. Return win indicator

onePair(hand) : sorted Hand

4. Sort hand
5. Sort on cardValue

6. Initialize the Boolean control variable = false, initialize counter, initialize win indicator to false
7. Get the first card in the hand and initialize the previous and current card
8. **Loop** until Boolean control variable (length cards in the hand) or counter >= 2
9. Get the next card in the hand
10. Compare for equality the card values
11. If they are the same increment counter
12. If counter = 2 (pair of same card)
    a. set the current card to be the previous card
13. If counter >= 2
14. Set win Indicator to true
15. **return win Indicator**

highCard(hand):card

16. Sort hand
    a. Dort hand on card Value
17. Get the first card in the deck
18. Return card

7.8.7 Resources

< A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.>

7.8.8 Processing

< A description of precisely how this components goes about performing the duties necessary to fulfill its responsibilities. This should encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.>
…

7.9 validateWinner

< A one line brief description of this component.>

7.9.1 Responsibilities

<Describe in detail all the primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements.>

7.9.2 Uses/Interactions

none

7.9.3 Constraints

< Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, postconditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.).>

7.9.4 Composition

< A description of the use and meaning of the subcomponents (e.g. functions) that are a part of this component.
Include all the functions that make up this components to include: function names, its parameters, its return data type if any, the basic flow and what each function will do, what other functions it calls, and what main data types it may use or define>

7.9.5 Resources

< A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.>

7.9.6 Processing

< A description of precisely how this components goes about performing the duties necessary to fulfill its responsibilities. This should encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.>
…

7.10 Dealer Rules

This is the rules AI that that is used by the dealer

7.10.1 Responsibilities

This module is responsible

- Rules that the dealer uses to bet
-

7.10.2 Uses/Interactions

none

7.10.3 Constraints

- The dealer must have points in order to bet

- The game must be in play
- The player must have to select Bet5 or Raise5

7.10.4 Composition

Function	Description
dealerRules(points Status, hand)	This function will determine what the dealer will do
handType validateHand(Hand currentHand)	Process the winning Hand,
boolean straightFlush(Hand currentHand)	Validate and verify that the cards of the current hand are of the same suit and will place the values in consecutive numerical order (rank)
boolean fourOfaKind(Hard currentHand)	Validate and verify that 4 of the cards of the current hand are of the same numerical order(rank)
boolean fullHouse(Hand currentHand)	Validate and verify that 3 cards are of the same rank and the remaining 2 cards are of another rank
boolean Flush(hand currentHand)	Validate that all cards are of the same suit
boolean straight(hand currentHand)	Validate that all cards are in numerical consecutive order
boolean threeOfaKind(hand currentHand)	Validate there are 3 cards of the same rank
boolean twoPairs(hand currentHand)	Validate that there are 2 sets of 2 cards of the same rank
booleano onePair(hand currentHand)	Validate that there are one set of 2 cards of the same rank
Card highCard(hand currentHand)	Validate the highest card in the hand

7.10.5 Resources

none

7.10.6 Processing

dealerRules(points, status, hand)

1. If the Status = “BET5”
    a. If points = 5
        i. Call threeOfaKind(hand) : win Indicator
            1. If win Indicator is false then return FOLD.
            2. If win indicator is true then reurn MATCH
    b. If Points > 5
        i. Call straightFlush(hand) win Indicator
            1. If win indicator is true then call Betting Match 5()
        ii. Call fourOfaKind(hand) win Indicator
            1. If win indicator is true then call Betting Match 5()
        iii. Call Flush(hand) win Indicator
            1. If win indicator is true then call Betting Match 5()
        iv. Call fullHouse (hand) win Indicator
            1. If win indicator is true then call Betting Match 5

Appendix A: Glossary

<Define all the terms necessary to properly interpret the specification, including acronyms and abbreviations.>

Rank:
Cards of the same numerical value or face such as 4 cards with number 3; 2 cards with face of Ace

Hand:
A set of 5 cards

***HandType**:*
The type of hand that is defined;

Hand	HandType	Value(default is None)
10♥ J♥ Q♥ K♥ A♥	**Straight Flush**	**1**
2♥ 7♦ 7♣ 7♠ 7♥	**Four of a Kind**	**2**
2♦ 2♥ 3♣ 3♦ 3♥	**Full House**	**3**
4♣ 6♣ 7♣ T♣ Q♣	**Flush**	**4**
A♥ 2♦ 3♣ 4♣ 5♦	**Straight**	**5**
2♦ 2♣ 2♥ Q♦ K♣	**Three of a Kind**	**6**
2♦ 2♥ 3♣ 3♦ 7♥	**Two Pair**	**7**
4♠ 4♥ 5♦ T♠ K♠	**One Pair**	**8**
2♥ 5♥ 7♦ 8♣ 9♠	**High Card**	**9**