Project 1 – Card Game Blackjack

Course:

CIS-17C-43673

Instructor:

Mark. E. Lehr

Due Date:

05/05/2024

Author:

Humza Saulat

Introduction

For this card game project, I decided to create a game of Blackjack with a few additives, coded in C++. The game of Blackjack is simple with a goal of reaching a total “hand” value of 21 with different methods of doing so. The reasoning for deciding to create this game was due to a few different reasons, the first being that the game itself has a simple premise of reaching 21 with the player’s hand. The next reason was being able to ignore one of the biggest premises of other card games, different suits having rules. The final reason for deciding on creating this game of Blackjack was that the “building upon” aspect of different parts of the game. With the rules and actions of Blackjack being simple, it was enticing to code Blackjack.

The overall length of the coding project from start to finish was about three weeks consisting of constant debugging, logic difficulties, and creating a simple console-based interface for the user. The project has 754 lines of code with 6 classes, 6 header files, and 1 main file. The project can be located on GitHub through my personal profile’s repository labeled CIS-17C_Project_Blackjack, the link is as follows https://github.com/Hsaulat729/CIS-17C_Project_Blackjack.

Approach to Development

The beginning concept of the game was to be able to fulfill the basic game in that the user needs to be able to reach a “hand” value of 21 with a deck of cards. This was immediately followed by the user “losing” or the game and the program terminating if the user’s hand reached above 21 which would result in a “bust”. This was then followed by addressing whether an ace would be equal to 1 or 11 whichever benefitted the user more. This was done by checking the user’s hand value if it is over 21 and if there are any aces in the hand. This leads to the hand value being subtracted by 10 for how many aces there are. When the hand value is below 21 then the aces are worth still worth 11. The next concept to cover was to allow the user to “hit” or “stand” which means to take another card or stop taking cards respectively. Once the user hits enough times to either get exactly 21 or over 21, or if the user stands, the game would terminate which led up to be the first version of the game.

Moving on to version 2 of the game consisted of covering the “Dealer” concept of the game Blackjack. I first began with the dealer taking their turn immediately before any inputs from the user so that the dealer would have a hand value that the user needs to beat without busting. With this the logic of greater than or less than became a monumental logic keystone throughout the game as one of the main premises of the game came to light. After solving logic issues with the dealer’s and the user’s hand values, I then implemented the dealers turns to coincide with the actions of the user. The dealer will draw one card from the deck not showing the user the card they have pulled then follows by the user hitting or standing with their hand until satisfied, then the dealer also plays which results in the user winning or losing.

The third version of my game of Blackjack added on the concepts of other choices the user would be able to make in a real-life version of the game. This included splitting, doubling down and the gambling system. This involved checking the user’s initial hand with the first two cards against both classes of splitting and doubling down. The implementation of the gambling system was one of the easier concepts to add to the game as the gambling rules of Blackjack are quite simple where winning gives the bet amount to the user, losing results in the user losing their bet, and a tie results in the user being returned their bet.

The final version of my Blackjack game added the concept of a “help” feature, amping up the intrigue of the dialogue of the game and debugging the gamble system’s winning bet. The help feature's concept consists of the rules of Blackjack and beginner tips that the user can view or not, at the beginning of the game. Should the user input that they would like to view the rules then a block of text is shown displaying all rules and beginner tips followed by the game beginning with the user being asked for their initial bet. If the user decided to not view the rules of the game, the user’s game immediately begins with being asked their initial bet. This was followed by adding more catching dialogue and scenarios to the game such as the dealer seeming more “alive” with text describing the dealer’s facial cues and more supportive text for the user when winning or losing. Finally, the gambling system was bugged with the winning bet resulting in 0 dollars being outputted for how much the user has won, but the correct amount still being added to the user’s total money. This was a problem of the user’s bet being refreshed before the gambing section of the game was completed. This was fixed by removing any calls for the user’s bet within the main file and was restricted to only being called within the player header file.

Game Rules

The rules of my game of Blackjack consist of the same rules for regular casino Blackjack except for the dealer pulling one card instead of two for their initial hand. The premise of the game is to beat the dealer by pulling cards from the deck and creating a hand value greater than the dealer. Cards are given values with 2 through 9 being as they are, an ace being 1 or 11 depending on if it benefits the player or not, and all face cards are worth 10. The game begins with the dealer pulling one card for themselves and keeping it “face down” while the user is given two cards and then prompted to either hit or stand. Staying allows the user to stop pulling cards and forces the dealer to then play. Hitting results in the player receiving another card. If the player’s hand exceeds the value of 21 then the player busts and loses their hand and bet. Another rule is that if the player initially receives a “Blackjack” with their first two cards and the dealer is not able to get a Blackjack immediately after then the player wins, but if the dealer does have a Blackjack, then it is a tie, and the bets are returned.

This leads to other options that the player has depending on the value of their hand. If the player pulls two cards that are of the same value, then they are asked if they wish to “split” their hand resulting in two different hands where cards can be pulled for both. This applies the same bet to both hands which can lead to the user doubling their profits if they win or losing double the initial bet if they lose. The player is then able to hit or stand on the two different hands and play the game similarly to having a single hand. Another rule is doubling down, which occurs only when the initial hand is equal to 9, 10 or 11, but without an ace. This prompts the user to double down on their bet as it is more likely for them to reach blackjack and win against the dealer. This also runs the risk of losing even more money if the player busts.

Description of Code

The organization of the code was kept straightforward with the different aspects of the game being split into different header files with classes and functions that are then called into the main.cpp file. This organization of the code allowed for easy access to debugging and overall neatness. The first header file created was the Deck.h file that consists of the Deck class and functions that draw cards for the players, checks the player's hands to see if they need a card, and if a card is already in use. This header file also consists of the actual "Deck of Cards" being created, the cards being shuffled before being played, and the initialization of the deck.

The next header file created was the player header file consisting of the Player class with private and public functions that hold the cards for the user and allow interaction. This header file is the biggest and rightfully so as this header file controls most of the play that occurs within the game and controls the betting that the user inputs. The class contains functions for the user such as placing bets, receiving cards, checking hand values and bets, and retrieving won and lost bets. The player class also utilizes the Deck.h header file variables in order for arguments to be taken. The player class also handles displaying the user's hand and split hands.

Moving on to the dealer header file with the Dealer class, this header file also utilizes the Deck.h header file to be able to take arguments. The dealer class works similarly to the player class but instead controls the Dealer's action within the game. The dealer similarly has the logic for controlling the ace value of 1 or 11 along with also being able to tell if a hand is "soft" with the ace. With this the dealer also has functions such as newGame and displayHand to clear for a new game and also output to the user what the dealer currently has. The dealer also deals cards to themself using the dealCardToSelf function within the class.

The next 3 header files allow the user to make decisions on how the game is played. This includes the Splitting.h header file, Double_Down.h header file, and the Help.h header file. The splitting.h header consists of the Splitting class which contains the function offerSplit that allows the user to split their hand. Within this function is the program's logic of having two of the same cards in their initial hand to split. This functionality also applies to the Double_Down header file which contains the DoubleDown class. This class holds the logic of allowing the user to double down on their initial bet if the user's hand is equal to 9, 10, or 11 without an ace within the offerDoubleDown function. Lastly is the Help.h header file which includes the Help class that contains the displayRules function that allows the user to view the rules and tips of Blackjack.

With the numerous functions and header files, the culmination of the logic and functionality is then pulled into the main.cpp file where the game is ultimately played. This begins with the introduction to the programmed game of Blackjack followed by the displayRules function from the Help class which allows the user to view the rules and tips of Blackjack. The program then moves on to initialize the deck from the Deck class and “shuffles” the cards. The user is then given 1000 dollars to gamble with and the dealer from the Dealer class is then initialized, then the game begins with the user inputting their initial bet. This sequence utilizes functions from the Player class that control the bet amount followed by the dealer being initialized for a new game. As the user pulls their first two cards for their initial hand the program pulls the functions from the Splitting and DoubleDown classes to check if the user can split or double down. The user is then able to keep hitting or standing until the user wins by a Blackjack or beating the dealer, or by busting and thus losing the round. After the user wins or loses the round, the user is then prompted if they would like to play again with their new total of their betting money. This loop is done until the user no longer wishes to play or the user completely runs out of betting money.

Sample Input/Output

User inputs are bolded:

Welcome to the Spectacular Virtual Blackjack Table!

The cards are shuffling, the bets are mounting, and the stakes are as high as the sky!

Would you like to see the rules of Blackjack? (y/n): ***n***

You currently have $1000. How daring do you feel today? What's your bet? ***100***

The dealer smirks as he reveals his initial hand:

Your cards whisper a tale of fortune or despair:

Player's hand:

King of Diamonds

10 of Hearts

Will you hit (h) or stand (s)? Choose wisely: ***s***

A calm decision to stand. Let's see how the cards fall.

Dealer's hand:

4 of Diamonds

Dealer's hand:

4 of Diamonds

```
King of Diamonds
Dealer's hand:
4 of Diamonds
King of Diamonds
10 of Diamonds
The dealer fumbles and busts with a hand value of 24!
Congratulations! Your winning current bet: $100

Your fortune now stands at $1100.
Do you wish to tempt fate again? (y/n): n

Thank you for playing at our esteemed table. Return when the cards call you again!
```

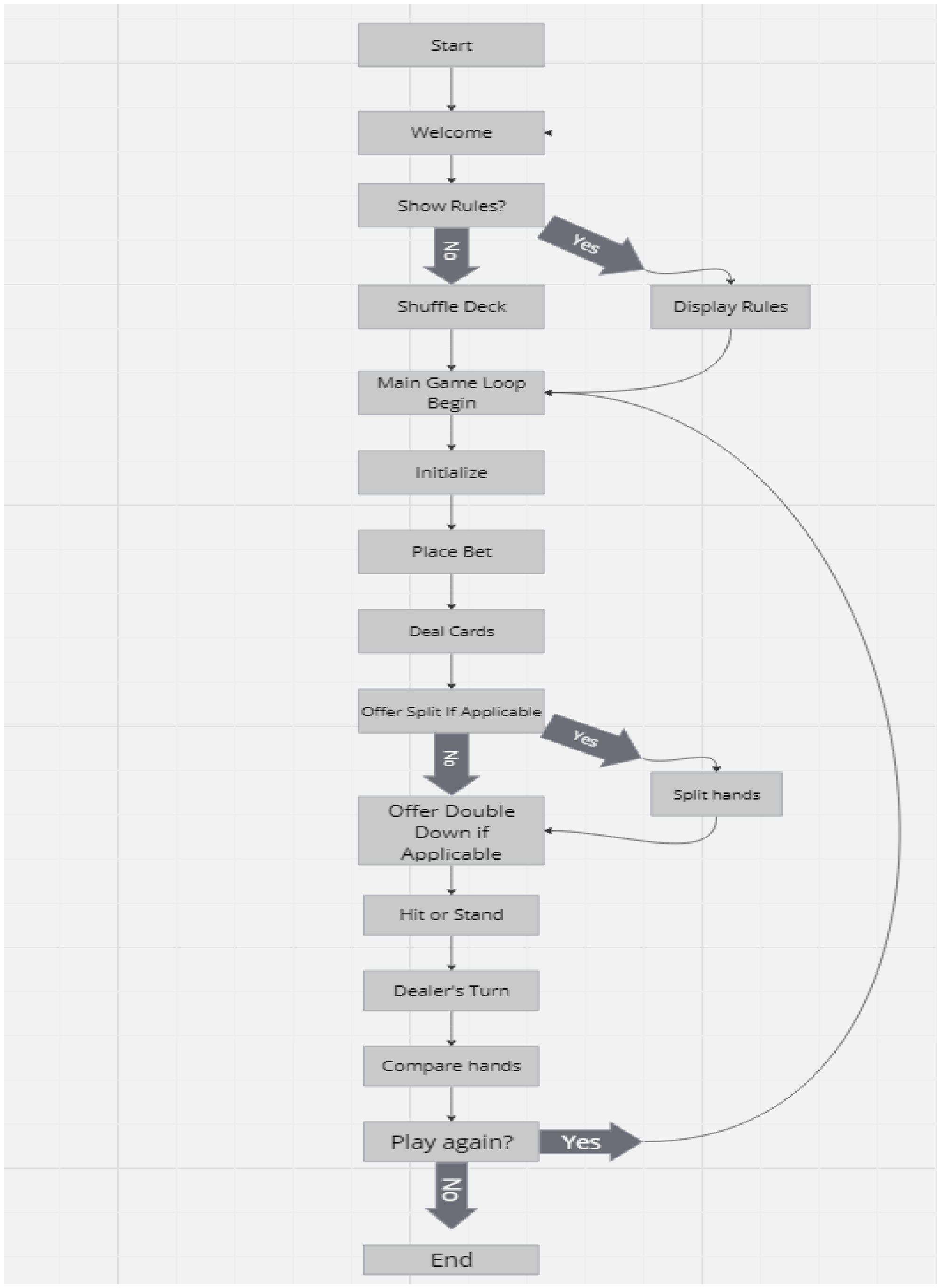
Checkoff Sheet

Beginning with the sequences section of the STL library, std::list was used multiple times with a prime example being within the Deck.h header file in the Deck class. A private member of the class is declared: cards, then the list is used to represent the deck of cards where each element of the list is an object of type Card. Then the initialize() method the cards list is cleared to remove any existing cards which prepares the deck for a new set of cards. This then moves onto the use of std::set within the same Deck class for the associative containers section of the checkoff sheet. In the Deck class std::set is used for usedCards which tracks cards that are already in use and therefore cannot be drawn again for that round of the game. When a card is drawn from the deck in the drawCard method then the drawn cards are stored in the set and any subsequent attempts to draw that card are unsuccessful due to sets only storing unique elements. The other example of the associative containers is the std::map container which occurs in the Dealer header file within the Dealer class. I used std::map to associate each rank of the playing cards with their corresponding values in the context of Blackjack. Here the std::map named cardValues is declared where the ranks are strings for the name of each card and the values are inputted as ints to represent the numerical value for each rank.

Moving on to the container adaptors, an example of std::stack is used within the Dealer class in the Dealer.h header file. Std::stack<card> deck is declared to represent the dealer's deck of cards. The line declares a private member variable named "deck" and the stack represents the dealer's deck of cards where each element of the stack is an object of type Card. Within the dealer header file iterators are used within the calculateHandValue() method that utilizes the std::list container and the implicitly functional forward iterators. This occurs again within the Deck class in the drawcard method where the cards.front is used to access the first element of the std::list where the iterator is used implicitly as forward iterators. This leads to the use of std::list and std::map within the player header file and Player class. The first is the std::list <Card> hand where the container holds the cards in the player's current hand and the iterator is an implicit forward iterator used when adding cards to the hand, clearing the hand, or checking the size of the hand. Then follows the std::map <std::string,int> cardValues where the container maps each card rank to its corresponding value and uses the bidirectional iterator when adding key value pairs or accessing values given a key.

This now leads into the algorithms section of the check off list where the for_each algorithm is used within the player header file and the Player class. Within this class the displayHand method is responsible for showing the current hand of the player. This starts by printing the player's hand to indicate the beginning of the hand display which then the std::for_each is called with three arguments. The hand.begin and the hand.end methods are used to specify the range over which the algorithm will operate. Iterating over all the cards in the player's hand. For the std::for_each algorithm is used to iterate over the cards in the player's hand and display their ranks and suits one by one which provides an easy and concise way to read each element from the std::list container. In the Deck class the std::sort algorithm is used indirectly through the cards.sort function in the shuffle method. The function sorts the cards in the deck based on a comparator function provided as an argument. The original idea was to use std::shuffle but std::sort was helpful in terms of being able to fill out the section of the check off sheet.

Documentation of Code, FlowChart



Pseudo Code

```
    Welcome message is displayed to user for the game of Blackjack
    Player is prompted to enter yes or no to display rules of Blackjack
    If player wishes, display rules
```

Deck of cards is initialized and shuffled

Player is assigned $1000 for betting

Instances of Player and Dealer class are created

Beginning of Game Loop

Player is prompted to enter bet

Dealer is initialized to begin new game and round of playing Blackjack taking player's bet

Dealer takes single card placed "face down" and player is given two face up cards

Player's hand is checked if splitting is possible, offer option to split

Player's hand is checked if double downing is possible, offer option to double down

Player is now allowed to either stand or hit; repeated until player stands or busts

If player loses/busts display loss message and prompt for new round of game is offered with new money total – bet.

If player stands, dealer's turn is taken and dealer hits unstil hand value is 17 or greater

If dealer busts, player wins and is awarded bet amount back to total amount, then a new round of the game is prompted for the player to bet with their new money total

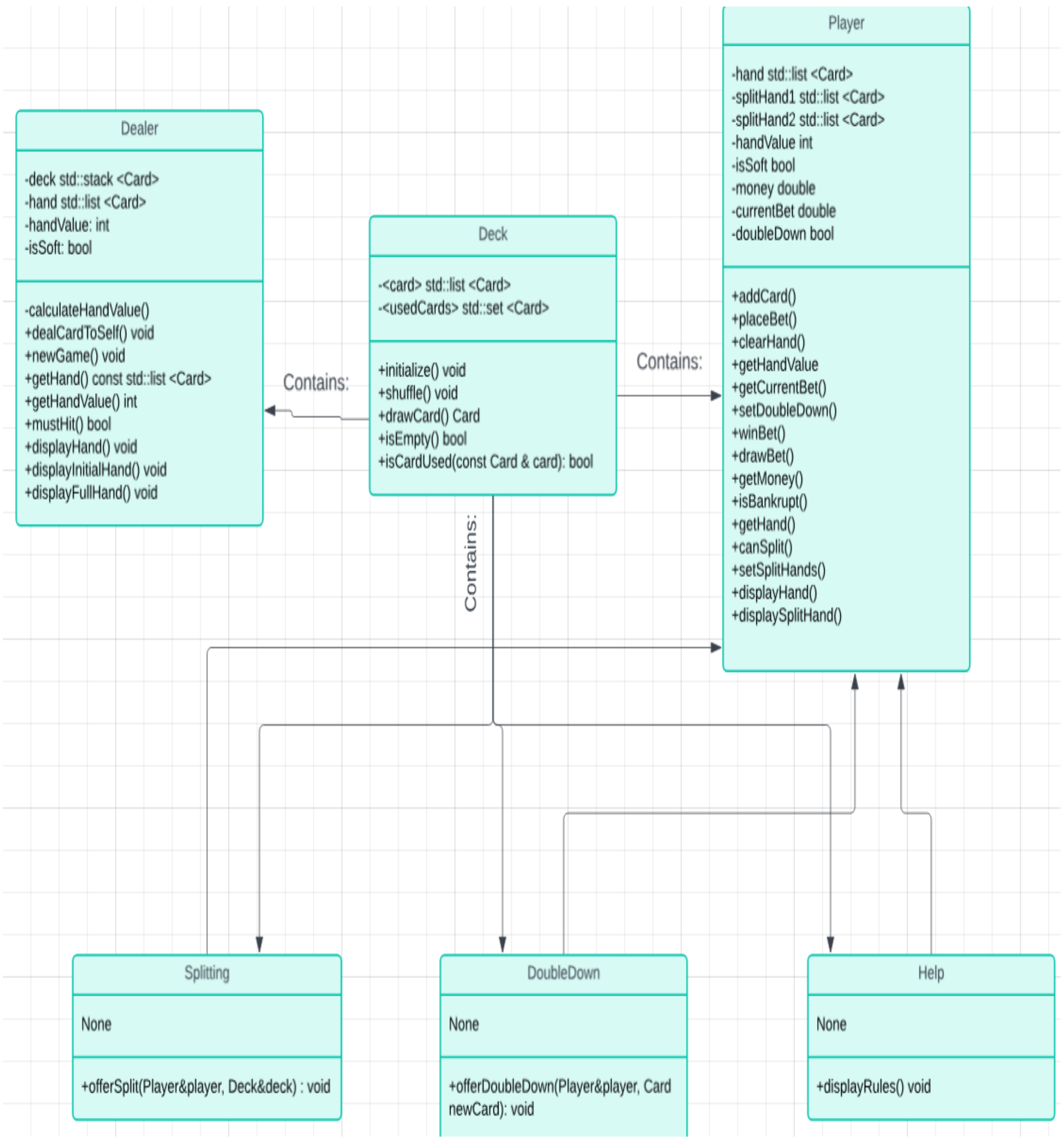
If neither player nor dealer busts, compare hands and calculate whether player wins, loses, or ties. Then update total money.

Ask player if they wish to play again.

Continue game loop if yes, terminate program if no.

End of main

UML Class Diagrams for Each Class


Dealer
-deck std::stack <Card>
-hand std::list <Card>
-handValue: int
-isSoft: bool
-calculateHandValue()
+dealCardToSelf() void
+newGame() void
+getHand() const std::list <Card>
+getHandValue() int
+mustHit() bool
+displayHand() void
+displayInitialHand() void
+displayFullHand() void
Contains:
Deck
-<card> std::list <Card>
-<usedCards> std::set <Card>
+initialize() void
+shuffle() void
+drawCard() Card
+isEmpty() bool
+isCardUsed(const Card & card): bool
Contains:
Contains:
Player
-hand std::list <Card>
-splitHand1 std::list <Card>
-splitHand2 std::list <Card>
-handValue int
-isSoft bool
-money double
-currentBet double
-doubleDown bool
+addCard()
+placeBet()
+clearHand()
+getHandValue
+getCurrentBet()
+setDoubleDown()
+winBet()
+drawBet()
+getMoney()
+isBankrupt()
+getHand()
+canSplit()
+setSplitHands()
+displayHand()
+displaySplitHand()
Splitting
None
+offerSplit(Player&player, Deck&deck) : void
DoubleDown
None
+offerDoubleDown(Player&player, Card newCard): void
Help
None
+displayRules() void