Blackjack Report

# How to run

This program can be most easily run by opening ‘B00746055 OOP CW.sln’, and then building and running the program with Visual Studio 2017.  
On launch, a logo will greet the user and they will be prompted to play or change settings. To play the game, the user must input ‘1’ and then select how much money they have to play with, and how much money they want to bet on each round. The player will then play blackjack against the dealer until they run out of money, or until they decide to stop playing. The user receives double their bet if they win a round, they lose their bet if they lose the round, and their bet is returned on a draw. If a user runs out of money, they have the opportunity to return to the main menu if they wish to play again.

# Additional features implemented

This program implements a number of additional features. Upon running the program, if the user selects ‘2’ to change settings, they can change the number of decks being shuffled at once. This can be used if the user wants to simulate French blackjack for example, in which 4 decks are often played at once.   
The program includes a betting system; users may input how much money they bring with them to the casino, and then can use this as a pool of money from which they may draw to make bets on each round.  
There are checks for blackjack, so he round is not played if the player or dealer has a natural 21.

# Requirements table

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Implemented | Filename | Line |
| Use of classes/objects | Yes | Hand.cpp Card.cpp Deck.cpp | All over.  Main.cpp line 27 |
| Applications of Pointers | Yes | Hand.cpp | 56 Hand contains many applications of pointers, especially with c-style arrays |
| Manipulation of Vectors/Arrays | Yes | Deck.cpp | 65 Vectors are manipulated many times across the program |
| Using functions | Yes | Main.cpp | This entire file is composed of many functions. main() method only has one statement |
| Exception Handling | Yes | Main.cpp | 33 – many try/catches implemented throughout this file |
| File Manipulation | Yes | Main.cpp Deck.cpp | Main - 440 Deck - 36 |
| Inheritance | No | In my design stage, I decided against using inheritance as it would not have added any value to my program. Include statements for header files provided all the functionality I needed | |
| Polymophism | Yes | Main.cpp Hand.cpp | Main 109,110,171,194 |
| Advances Functionality | Yes | In Deck.cpp, a random generator was used.  A Map was used in Deck.cpp Object destructors were used throughout for clean-up Functionality for soft-aces was implemented throughout, as well as a betting system and the ability to use multiple decks | |

# Class Diagram

Aside from the standard main class, 3 other classes were developed and implemented for this project. These are as follows:

|  |
| --- |
| Deck |
| engine : default\_random\_engine |
| Deck() setupDeck() getRandomEngine() |

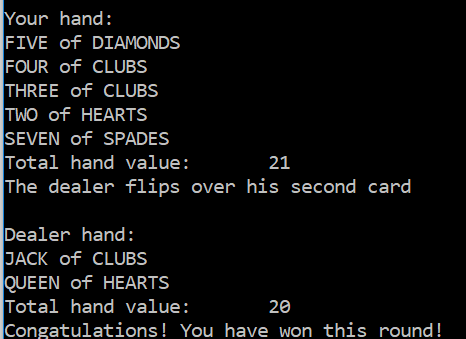
|  |
| --- |
| Hand |
| player : bool cards : vector<Card> value : int[2] |
| Hand() isPlayer() clear() dealCard() displayHand() calculateHandValue() checkBust() checkForNaturals() displayTopCard() clearCards() getFirstValue() getSecondValue() |

|  |
| --- |
| Card |
| value : int suit : string cardNumber : string secondValue : int |
| Card() getValue() getSecondValue() getSuit() getValueString() |

# Test plan

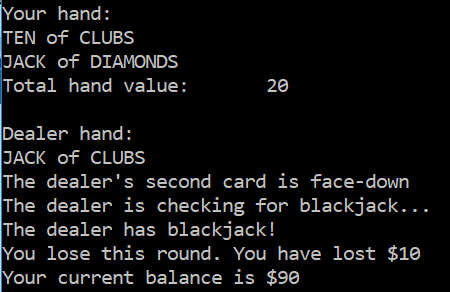
# 1.

Test that if the player has a better hand than the dealer, they win the round

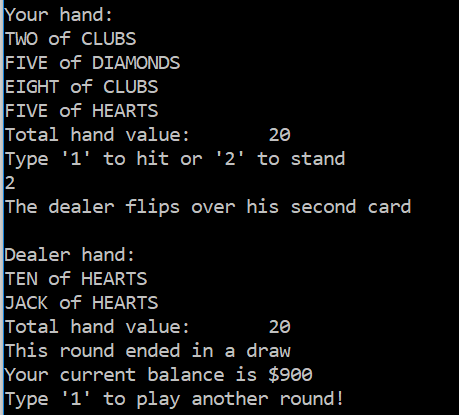


# 2.

Test that if the player has a worse hand than the dealer, they lose the round

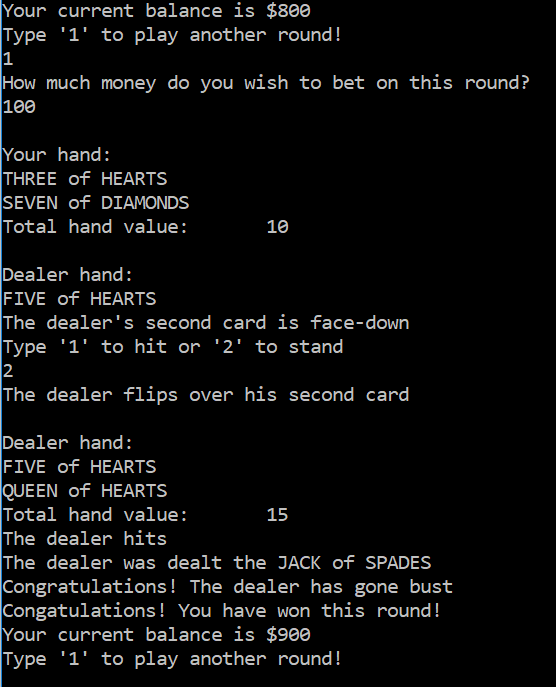


# 3.

Test that if the player and the dealer have hands of the same value, the round ends in a draw  


# 4.

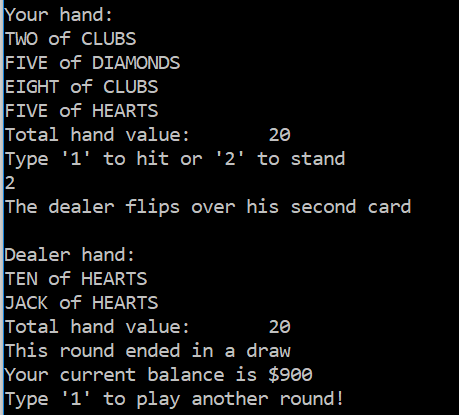
Test that upon winning a round, the player’s balance is increased by 2 \* the bet



Note that once this bet is made, the bet amount ($100) is deducted from the balance so now the balance is $700

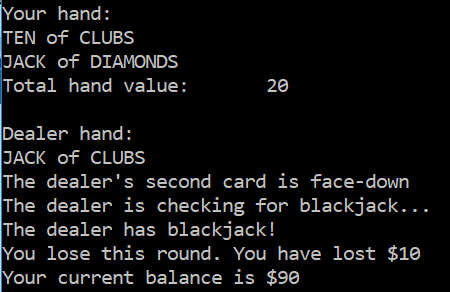
# 5.

Test that upon a draw, the player’s balance is returned to its original level by returning the bet (balance += bet)

  
(Original balance was $900)

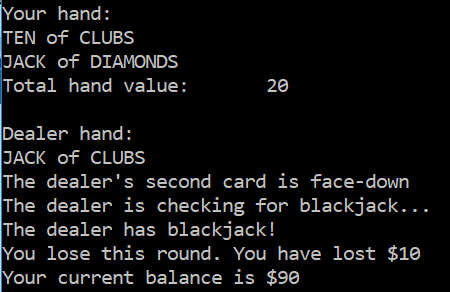
# 6.

Test that upon losing a round, the player’s balance is decreased by their bet.



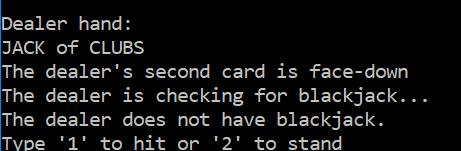
# 7.

Test that if the dealer’s first card is an ace or a spade, they check for blackjack



# 8.

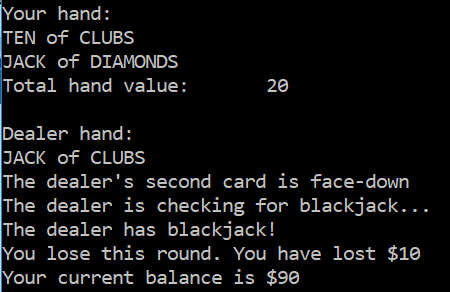
Test that if the dealer does not have blackjack, the game continues



>>Game continues from here

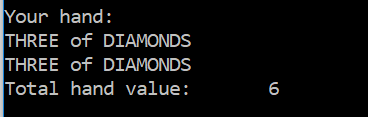
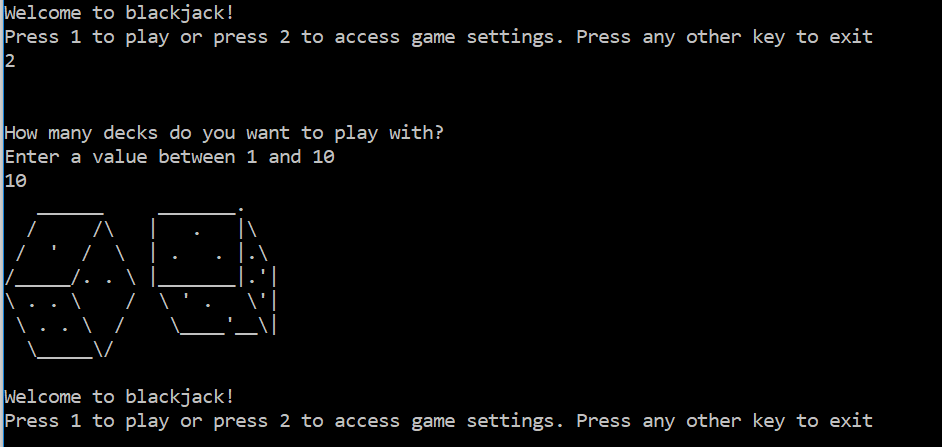
# 9.

Test that if the dealer or player has blackjack, the round is not played



# 10.

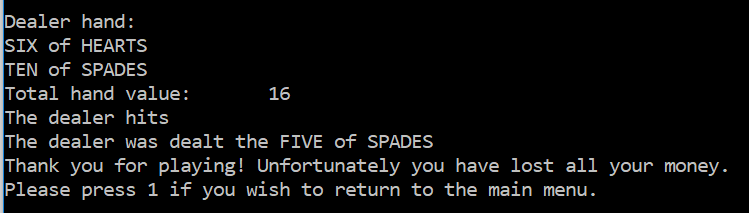
Test that multiple decks can be used at once



With a single deck in use, it would be impossible to have 2 “THREE of DIAMONS” cards since the deck would only have 1

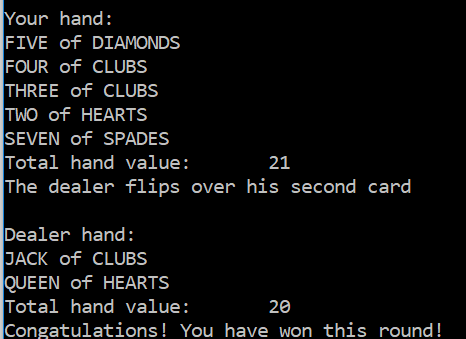
# 11.

Test that upon running out of money, the player may exit the application or return to the main menu



# 12.

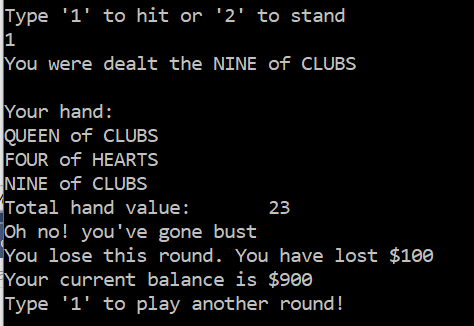
Test that if the user “hits” and reaches 21, they are no longer prompted for moves and the dealer’s turn begins



No prompt to hit/stand after hand value hits 21

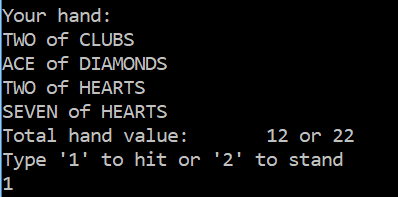
# 13

Test that if the player hits and goes bust, they lose the round



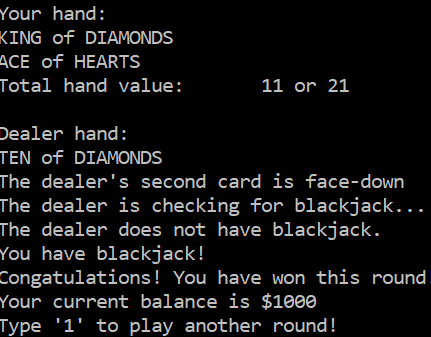
# 14

Check that if dealt an ace, the player will have 2 possible hand values



# 15

Check that if the player has blackjack and the dealer does not, the player wins



# 16.

Test that upon running out of money, player is returned to the main menu

