Jimmy Rashed

Programming II (420-2P6-AB sec.1)

Talib Hussain

May 6, 2025

# Project Design Document

# Game Rules

The card game I have chosen to implement is Blackjack with a betting system. Here are the rules of blackjack:

In blackjack, each player competes against the dealer. A player must have a hand that is closer to 21 than the dealer’s without going over.

Here are the card values in blackjack:

Two =2, Three = 3, Four = 4, Five =5, 6 = Six, Seven = 7, Eight = 8, Nine = 9, Ten = 10, Jack = 10, King = 10, Queen = 10, Ace = 11 if counting it as such does not make the hand bust (exceed 21). If it does, then aces are counted as 1 instead.

Blackjack in played with a traditional 52 card deck (no jokers). In Blackjack, each player competes only against the dealer. Wins and losses are thus decided only by whether the player’s hand beats the dealer’s and is not affected by other players. Similarly, there are no betting pools in Blackjack. Betting happens individually, and payouts are calculated as such:

* If you win without a getting a Blackjack (21 in the **initial deal**), your winnings will be 100% of your bet plus your bet back(e.g. you bet 100 and win without blackjack, you get 100 and your bet back, for 200 total).
* If you win with a blackjack, you will get 150% of your bet back (e.g. you bet 100 and win with blackjack, you get 150 and your bet back, for 250 total).
* If you lose, you lose your entire bet.
* If you tie, your bet is refunded.

Turns are played as such in blackjack:

An initial deal happens, and both you and the dealer get 2 cards. However, the dealer’s second card is hidden. The player then plays their turn first. They can:

* Hit – Take another card
* Stand – End your turn
* Double down –doubles your bet, hits once, and stands
* Forfeit – Automatically lose

A player’s turn only ends if they choose to stand or if they bust (hand goes over 21)

Then, it is the dealer’s turn. The dealer’s second card is revealed. The dealer must keep hitting until the value of their hand is equal to or greater than 17, at which point they must stand. If the dealer busts, everyone playing who played and didn’t bust wins.

After the dealer’s turn, the player and the dealer compare hands. If the player’s hand is greater, they win. If the dealer’s hand is greater, the player loses. If the player’s hand and the dealer’s hand are equal, a tie occurs.

In my project, the game persists across multiple rounds. The player profiles and balances persists, allowing them to save, load, and manage their balance.

# OOP Design

You may need to zoom in to the word document a substantial amount to be able to read the OOP design chat.

# A screenshot of a computer screen AI-generated content may be incorrect.Pseudocode

Rank and Suit enums, and Card and Hand classes work near-identically to their assignment 6 counterparts. Please refer to that assignment’s pseudocode.

The deck and hand classes work near-identically to their assignment 5 counterparts, with sorting logic revised to use the rank and suit enums. Please refer to that assignment’s pseudocode.

## GameRules class

The GameRules class contains the following methods:

**Bust (Player player){**

If (player.hand.Value > Constants.BLACKJACK){

Return true;

}

Else{

Return false

}

**BlackJack(Player player){**

If (player.hand.Size greater than 2 && player.hand.Value equal Constants.BLACKJACK){

Return true

}

Else{

Return false;

}

**Outcome(Player player, Player dealer){**

If (Bust(player) || (player.hand.Value less than dealer.hand.Value){

Return “Lose”

}

Else if (BlackJack(Player player){

Return “Blackjack”

}

Else if (player.hand.Value is greater than deal.hand.Value){

Return “Win”

}

Else{

Return “Tie”

}

}

**Payout(Player player, string outcome){**

Switch (outcome){

Case (“Blackjack”)

return 250% of player bet

}

Case(“Win”){

return 200% of player bet

}

Case (“Lose”){

Return 0

}

Default {

Return 0

}

**}**

### Sorting algorithms

Hand class sorting uses near-identical logic to assignment 5

Leaderboard sorting uses a selection sort algorithm with player win/loss ratio as its metric

### Searching

The Leaderboard player search algorithm uses a linear search algorithm to locate the player.

### Complicated constructors

The PlayerStats constructor uses a ternary operator to check for null to avoid division by zero:

winRatio = (wins + losses + ties == 0) ? 0 : (wins / wins + losses + ties);

The card constructor derives its color based on its suit

if ((int)Suit%2 == 0)

{

Color = ConsoleColor.Black;

}

else

{

Color = ConsoleColor.Red;

}

Deck constructor functions identically to assignment 5 (default constructor generates 52 cards, then shuffles)

### Complicated dealing methods

The initial deal deals 2 cards to both the player and the dealer, but hides the last card of the dealer

It does so by adding a card to the hand using Hand.AddCard() with Deck.Draw() as an argument 2 times for each player and for the dealer

It then manually sets the dealer’s last card (logically the second one) to

### Complex variations

There are no major complex variations to this assignment. I’m reusing parts of nearly every assignment with light modifications to ensure coherence between different assignments

I’ve added a bool hidden to my Card class. If true, [Hidden] will be printed instead of the card value upon ToString() call. This is used to hide the dealer’s second card during the initial draw.