Project 1

Title

Blackjack Game

Course

CIS-17C

Section

47541

Due Date

November 10, 2024

Author

Quanyu Liu

# Introduction

For this project, I made a Blackjack game. The reason why I chose the Blackjack game was because this was the same game I made when I was in CIS-5. Although at that time, the game I made was really straightforward, or “beginner’s level”. But it helped me to be familiar with the game rules and the basic concepts of the game. I spent about 2 hours every day for about one and a half weeks on this project. The project is about 300 lines. And I created 4 classes for the project: Card, Deck, Player, Gameplay.

Where on git hub is it located?

<https://github.com/qliu579/CIS-17C/tree/main>

# Approach to Development

The development of this Blackjack game was done step by step. First, I created a basic game functions like dealing cards, calculating hand values, and letting the player to decide whether hit or stand. After getting these basics right, I added more features, like keeping track of dealt cards and improving the game rules.

# Game Rules

Blackjack is a card game where you try to get as close to 21 points as possible without going over. Players are dealt two cards, and they can choose to “hit” (get another card) or “stand” (keeping their current cards). Face cards are worth 10 points, Aces can be 1 or 11 points, and the other cards are worth their number value. The goal is to beat the dealer’s hand without going over 21 points.

# Description of Code

The code is organized into several classes, each representing a different part of the game. I made Card, Player, Deck and Gameplay classes, which makes it easy to understand the responsibilities of each part of the game.

Classes:  
 Card: Represents a playing card without a suit and value. It also includes a function to display the card.

Player: Represents a player (or dealer) in the game. It keeps track of the player’s hand and includes functions to show the hand, calculate its value, and count Aces.

Deck: Represents the deck of cards used in the game. It includes functions to create the deck and shuffle the cards.

Gameplay: Handles the flow of the game, including dealing cards, handling player decisions, and determining the winner.

# Sample Input/Output

A screenshot of a computer

Description automatically generated

In this situation, the player has the option to either “hit” to draw an additional card or “stand” to keep their current hand. Let’s choose to his and continue the game.

A screenshot of a computer game

Description automatically generated

In this example, the player chooses to “hit”. They draw a Jack of Hearts (JH), which brings their total to 26 points, resulting in a “bust”, meaning they lose the round.

# Checkoff Sheet Contents

1. Container classes

1.Sequences

List: Used in Deck class to store and iterate through the suits and values of cards before creating a stack (deckLst).

1. Associative Containers

Set: Used in the Deck class to keep track of dealt cards (dealtCards).

Map: Used in Gameplay and Player classes to store the values of each card type (cardVal)

1. Container Adaptors

Stack: Used in the Deck class to represent the deck of cards (deck).

Queue and priority\_queue: Not used in the project. I found it hard to fit in my codes, I couldn’t really think of and usage of these two. Maybe I could’ve added another player and made them take turns in order to use Queue, but then I had to modify the AI player’s behavior, which I don’t really have a clue how to achieve that.

2. Iterators

1. Trivial Iterator: Not used.

2. Input Iterator: Used in functions like crtDeck() when iterating over the suits and values lists to create cards for the deck. Also used in the shwHand() function to traverse the player’s hand (list<card>).

3. Output Iterator: used in ostream& operator << for displaying card information.

4. Forward Iterator: The list container used in the Player class and Deck class. Forward iterators are used when iterating through cards in a player’s hand or during deck creation.

5. Bidirectional Iterator: The list container also supports bidirectional iteration, which is used in Player::shwHand() when displaying the player’s hand.

6. Random Access Iterator: Not used in this project.

3. Algorithms

1. Non-mutating algorithms:

Count: The function countAces() in the player class counts the number of Aces in a player’s hand. This helps determine the value of the Aces in calculating the total hand value.

2. Mutating algorithms:

Random\_shuffle: The Deck class uses random shuffling when creating the deck to randomize the order of cards.

3. Organization algorithms:

The project did not include sorting, merging, or binary searching as these did not fit into the game logic. I found it hard to put in my code.