SYST 17796 DELIVERABLE 3

Shushank Raj Pariyar, Wendell Destang, Ejokirhie Joshua

Sheridan College

SYST 17796

Professor Paym Bergson

December 4, 2024

Abstract

This is a report that builds on deliverable 1 and 2 for the Blackjack game project. It shows and discusses the use case and UML diagrams, OOD principles and the game functionality and playability.

Table of Contents

SYST 17796 DELIVERABLE 3

The game is the Blackjack card game. Tieperman (2024) describes the game in a sentence – "A card game where players aim to beat the dealer with a hand that most closely totals 21 points". In this project, the game is between one dealer and one player. Dealer deals two cards facing up to the player and two cards to the dealer with one card being faced down. Player chooses to 'hit' or 'stand' based on the cards they were dealt. Dealer flips their faced down card and must 'hit' until their hand total is at least 17, then they must 'stand'. Both hand values are compared, and the winner is whoever is closest to 21 without exceeding that amount. By the end of the project, we expect to have a functional and playable BlackJack card game built entirely on the given framework and applying the principles studied throughout the course.

Team Member Roles –

Joshua Ejokirhie –

Deliverable 1: Defining the project scope and deliverable report.

Deliverable 2: Implementation and discussion of OOD principles and Deliverable report.

Deliverable 3: Deliverable report.

Wendell Destang –

Deliverable 1: Description and analysis of the project background, final vision and project high level requirements.

Deliverable 2: Use case diagrams and narratives.

Deliverable 3: Coding and pushing to GitHub.

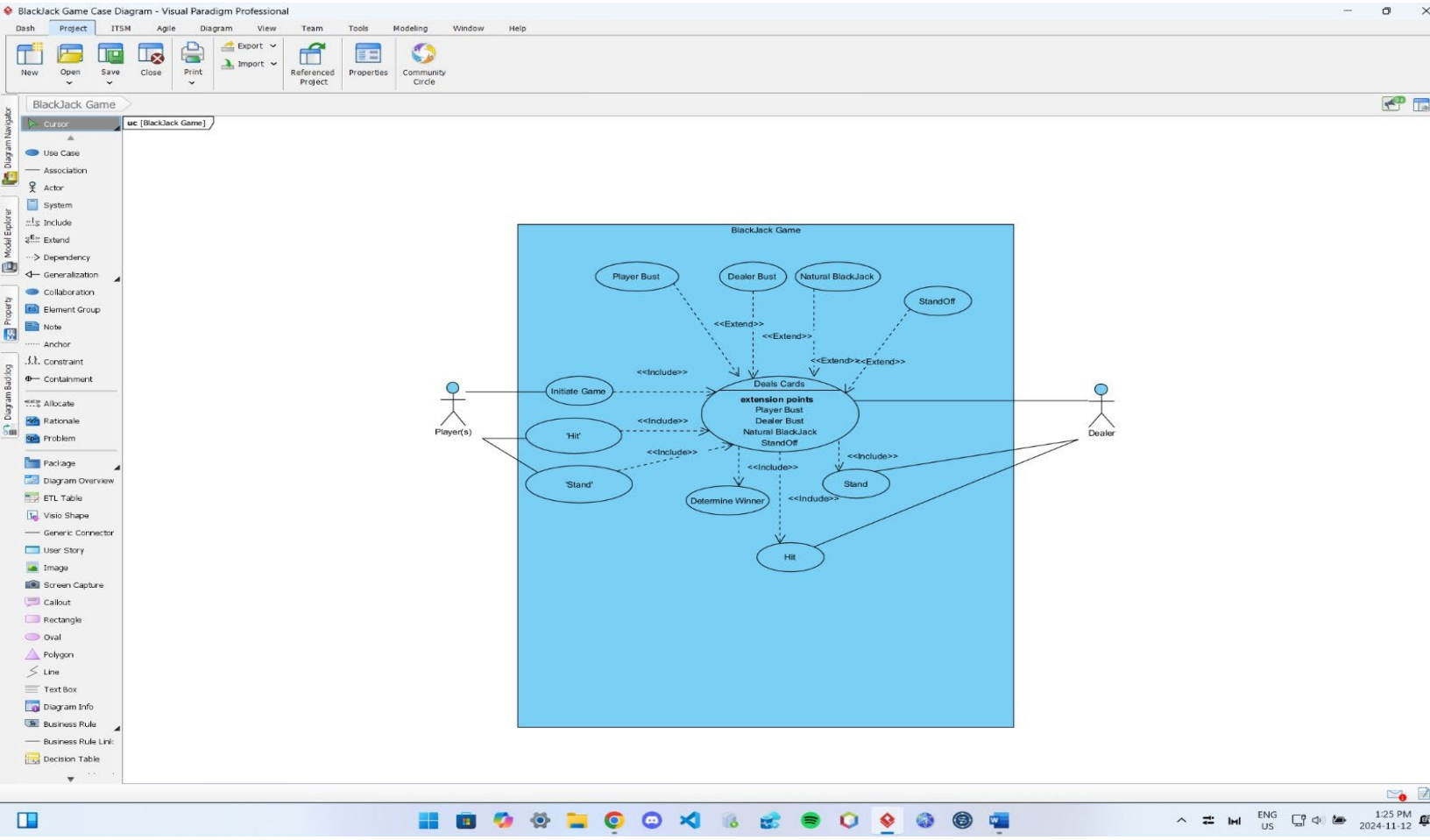
Shushank Raj Pariyar –

Deliverable 1: Analysis of design considerations and setting up team GitHub repository.

Deliverable 2: Class Diagram .

Deliverable 3: Coding the project and pushing to GitHub.

Use Case Diagram:



Main Path: Play Blackjack Game

1. Player initiates game.
2. Dealer deals two face up cards to the player and two cards to the dealer with one card being faced down.
3. Player chooses to ‘hit’ or ‘stand’ based on the cards they were dealt.
4. Dealer flips their faced down card and must ‘hit’ until their hand total is at least 17, then they must ‘stand’.

5. Both hand values are compared, and the winner is whoever is closest to 21 without exceeding that amount.

Alternate Path: Player Bust

1. Player initiates game.
2. Dealer deals two face up cards to the player and two cards to the dealer with one card being faced down.
3. Player chooses to ‘hit’ or ‘stand’ based on the cards they were dealt.

3.1. Player’s hand exceeds 21.

3.1.1. Player immediately ‘busts’ ending game with dealer as winner.

Alternate Path: Dealer Bust

1. Player initiates game.
2. Dealer deals two face up cards to the player and two cards to the dealer with one card being faced down.
3. Player chooses to ‘hit’ or ‘stand’ based on the cards they were dealt.
4. Dealer flips their faced down card and must ‘hit’ until their hand total is at least 17, then they must ‘stand’.

4.1. Dealer’s hand exceeds 21.

4.1.1. Dealer immediately ‘busts’ ending game with player as winner

Alternate Path: Natural Blackjack

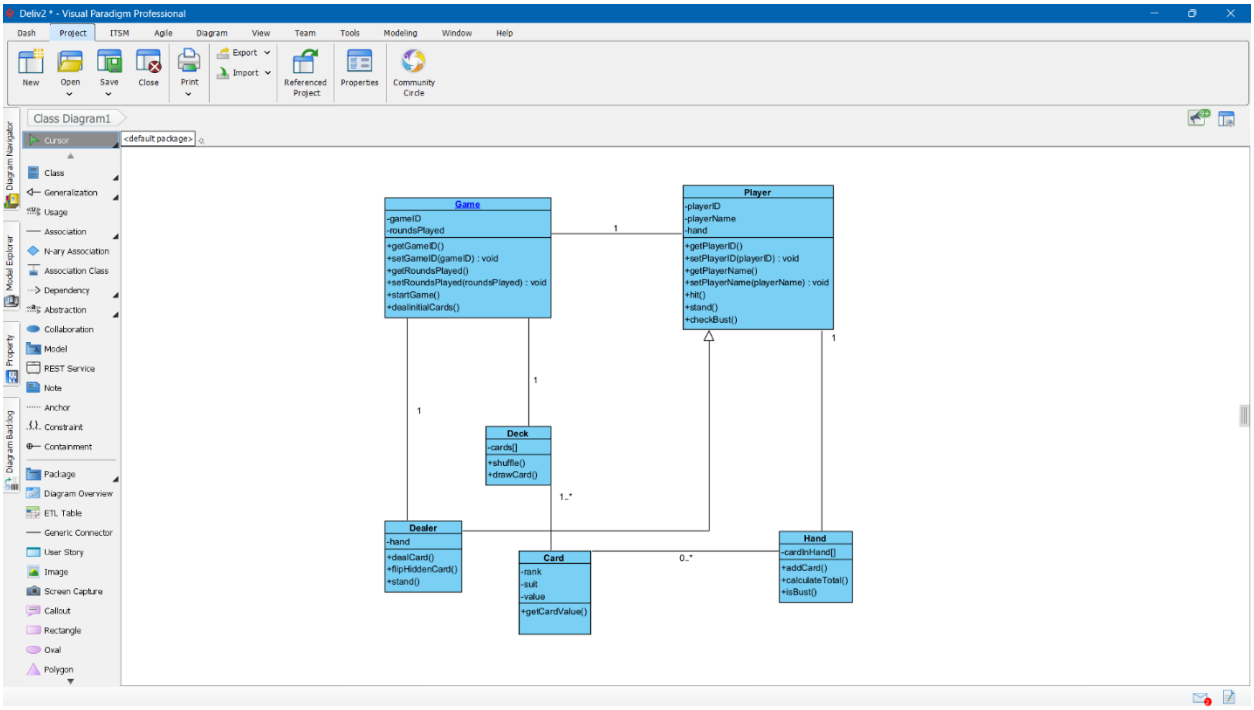
1. Player initiates game.
2. Dealer deals two face up cards to the player and two cards to the dealer with one card being faced down.
3. Player chooses to ‘hit’ or ‘stand’ based on the cards they were dealt.
4. Player’s initial cards total 21.
    4.1. Player’s hand is a natural blackjack and automatically wins.
5. Dealer’s initial cards total 21.
    5.1. Dealer’s hand is a natural blackjack and automatically wins.

Alternate Path: Stand Off

1. Player initiates game.
2. Dealer deals two face up cards to the player and two cards to the dealer with one card being faced down.
3. Player chooses to ‘hit’ or ‘stand’ based on the cards they were dealt.
4. Dealer flips their faced down card and must ‘hit’ until their hand total is at least 17, then they must ‘stand’.
5. Both player and dealer’s hand are the same total not exceeding 21.
    5.1. Game ends in a ‘stand off’ with no declared winner.
6. Both player and dealer have a natural blackjack.

6.1. Game ends in a 'stand off with no declared winner.

Class Diagram



Classes and Their Roles

1. Game Class:

- Attributes: gameID, roundsPlayed
- Methods:
- startGame(): Initiates a new game.
- endGame(): Concludes the game.
- dealInitialCards(): Distributes initial cards to Player and Dealer.
- Associations:

- Connects to Player and Dealer, handling their interactions.
- Connects to Deck for card distribution.

2. Player Class:

- Attributes: playerID, name, hand
- Methods:
- hit(): Adds a card to the player's hand.
- stand(): Ends the player's turn.
- checkBust(): Checks if the player's hand exceeds 21.
- Association with Hand: Represents the player's collection of cards.

3. Dealer Class:

- inheritance: Extends the Player class
- Attributes: hand
- Methods:
- dealCard(), flipHiddenCard(), stand()
- Bust Condition: If total > 21, dealer busts.
- Association with Hand: Contains the dealer's hand of cards.

4. Deck Class:

- Attributes: cards[]

- Methods: shuffle(), drawCard()
- Associations: Contains multiple Card objects and is used by the Game class for dealing.

5. Card Class:

- Attributes: rank (Enum), suit (Enum), value
- Methods: getCardValue()
- Association with Deck: Multiple Card instances make up the deck.

6. Hand Class:

- Attributes: cardsInHand[]
- Methods: addCard(), calculateTotal(), isBust()
- Associations: multiple Card objects.

Associations and Multiplicities

1. Game - Player:

- Multiplicity: 1 game has 1 player.
- Purpose: Represents the player's participation in the game.

2. Game - Dealer:

- Multiplicity: 1 game has 1 dealer.
- Purpose: Dealer controls the deck and responds to player actions.

3. Game - Deck:

- Multiplicity: 1 deck per game.
- Purpose: Provides cards for gameplay.

4. Deck - Card:

- Multiplicity: 1..* cards per deck.
- Purpose: Represents the collection of cards in the deck.

5. Player - Hand:

- Multiplicity: Each Player has 1 hand.
- Purpose: Holds cards that the player is dealt.

6. Dealer - Hand:

- Multiplicity: Each Dealer has 1 hand.
- Purpose: Contains the dealer’s hand of cards.

7. Hand - Card:

- Multiplicity: 0..* cards in a hand.
- Purpose: Represents the player’s or dealer’s cards.

Principles of OO Design:

Encapsulation: Most attributes, such as gameID, roundsPlayed, playerID, and cards[], are private (indicated by -), which prevents unauthorized access. Public getter and setter methods allow controlled access and modification of these attributes. This approach protects the integrity of the objects' data and ensures they can only be changed in controlled ways, which is essential for data integrity.

Delegation: Cohesion refers to how closely related the responsibilities of a class are. This is effectively used in the Game class, which relies on Dealer to manage interactions with the Deck (e.g., dealCard()). Similarly, the Hand class is responsible for managing the cards within a player's hand, allowing Player and Dealer classes to delegate responsibilities related to the hand's composition and value calculation. This separation of responsibilities leads to a cleaner design where each class performs specific functions and relies on others for complementary tasks.

Cohesion: The classes in this design demonstrate high cohesion. Each class has a well-defined role:

- Game handles the game flow and initiates important actions like starting, ending, and dealing cards.
- Player and Dealer handle the player-specific and dealer-specific actions (e.g., hitting, standing).
- Deck manages the collection of cards and shuffling, and Card encapsulates card-specific data.

- Hand aggregates the cards in each hand and performs hand-related operations, like calculating totals.

High cohesion improves the readability of each class and simplifies debugging, as each class has a single purpose.

Coupling: This depends on how dependent on each other the classes are. The classes are interdependent due to their specific roles in a card game. For instance, Game needs to interact with Player, Dealer, and Deck. However, the associations are necessary for the functionality of the system and are minimized to essential interactions. To reduce coupling, consider indirect interactions where possible, such as making Dealer manage Deck exclusively, instead of Game also accessing Deck. Overall, the coupling is appropriate, but decoupling could further enhance flexibility.

Inheritance: Inheritance allows for creating a new class based on an existing class. There is no inheritance in the current design, as each class has a unique role and doesn't require a hierarchy.

Aggregation: Aggregation is a type of association that represents a "whole part" relationship where the part can exist independently of the whole. Aggregation is represented between Hand and Card, where multiple Card objects form a Hand. This relationship reflects that Hand depends on Card objects, but cards do not solely belong to one hand and can exist independently. Additionally, you might consider representing the Deck-Card relationship as aggregation to clarify that the deck holds multiple cards but does not "own" them exclusively.

Composition: This is a stronger form of association where the part cannot exist independently of the whole. Composition is present in the relationship between Deck and Card. The Deck manages a collection of cards (cards[]), and the Card objects exist only as part of the Deck.

Similarly, Hand is composed of multiple cards (cardsInHand[]), which depend on Hand for their place within the gameplay context. This ensures that when a Deck or Hand is removed, the cards associated with it are also removed.

Flexibility/Maintainability: Flexibility and maintainability refer to the ease with which the system can be modified or extended with minimal impact on existing code. The design demonstrates flexibility by using encapsulation, delegation, and high cohesion, which keeps classes modular.

Inheritance

Inheritance is a core principle of object-oriented design (OOD) that enables a class (called the subclass or child class) to inherit attributes and methods from another class (called the superclass or parent class). In the context of this project, inheritance is utilized to establish a hierarchical relationship between the Player and Dealer class. The Player class serves as the parent class, encapsulating general attributes (playerID, name, hand) and methods (hit(), stand(), checkBust()) that are applicable to all participants in the game. The Dealer class extends the Player class, inheriting its attributes and methods while adding behavior unique to the dealer. For example, the Dealer class introduces specific methods such as dealCard() and flipHiddenCard(), which are not relevant for regular players.

Game Playability and Functionality

The game is entirely playable and users are able to follow the above outlined user paths. WE did good, bad and boundary tests. We tested what happens when a player either decides to hit or stand in any given situation during the game, and the randomness of playing cards so it doesn't keep spinnong up the same cards. We also tested that all prompts and outputs matched with the given situation during gameplay.

Git repository URL – https://github.com/shushankraj/game5project

Screenshots

```
run:
Enter your name:
w
Starting Blackjack Game...
Player's hand: [TWO of SPADES, QUEEN of CLUBS]
Dealer's hand: [hidden], TEN of DIAMONDS
Do you want to 'hit' or 'stand'?
hit
Player's hand: [TWO of SPADES, QUEEN of CLUBS, NINE of DIAMONDS]
Do you want to 'hit' or 'stand'?
stand
w stands.
Dealer reveals the hidden card.
Dealer's hand: [QUEEN of DIAMONDS, TEN of DIAMONDS]
Player's total: 21
Dealer's total: 20
Player wins!
BUILD SUCCESSFUL (total time: 6 seconds)
```

```
run:
Enter your name:
w
Starting Blackjack Game...
Player's hand: [TEN of SPADES, KING of CLUBS]
Dealer's hand: [hidden], SIX of DIAMONDS
Do you want to 'hit' or 'stand'?
stand
w stands.
Dealer reveals the hidden card.
Dealer's hand: [FOUR of DIAMONDS, SIX of DIAMONDS]
Dealer's hand: [FOUR of DIAMONDS, SIX of DIAMONDS, FOUR of HEARTS]
Dealer's hand: [FOUR of DIAMONDS, SIX of DIAMONDS, FOUR of HEARTS, JACK of HEARTS]
Dealer busts! Player wins.
BUILD SUCCESSFUL (total time: 8 seconds)
```

```
run:
Enter your name:
w
Starting Blackjack Game...
Player's hand: [ACE of SPADES, THREE of CLUBS]
Dealer's hand: [hidden], ACE of HEARTS
Do you want to 'hit' or 'stand'?
stand
w stands.
Dealer reveals the hidden card.
Dealer's hand: [QUEEN of CLUBS, ACE of HEARTS]
Dealer has a Blackjack! Dealer wins!
BUILD SUCCESSFUL (total time: 9 seconds)
```

```
run:
Enter your name:
w
Starting Blackjack Game...
Player's hand: [ACE of SPADES, THREE of CLUBS]
Dealer's hand: [hidden], FOUR of DIAMONDS
Do you want to 'hit' or 'stand'?
hit
Player's hand: [ACE of SPADES, THREE of CLUBS, SIX of HEARTS]
Do you want to 'hit' or 'stand'?
hit
Player's hand: [ACE of SPADES, THREE of CLUBS, SIX of HEARTS, ACE of DIAMONDS]
Do you want to 'hit' or 'stand'?
hit
Player's hand: [ACE of SPADES, THREE of CLUBS, SIX of HEARTS, ACE of DIAMONDS, FOUR of HEARTS]
Do you want to 'hit' or 'stand'?
hit
Player's hand: [ACE of SPADES, THREE of CLUBS, SIX of HEARTS, ACE of DIAMONDS, FOUR of HEARTS, KING of DIAMONDS]
Player busts! Dealer wins.
BUILD SUCCESSFUL (total time: 48 seconds)
```

```
Player's hand: [ACE of HEARTS, SEVEN of HEARTS]
Dealer's hand: [hidden], JACK of DIAMONDS
Do you want to 'hit' or 'stand'?
hit
Player's hand: [ACE of HEARTS, SEVEN of HEARTS, KING of CLUBS]
Do you want to 'hit' or 'stand'?
stand
w stands.
Dealer reveals the hidden card.
Dealer's hand: [KING of HEARTS, JACK of DIAMONDS]
Player's total: 18
Dealer's total: 20
Dealer wins!
BUILD SUCCESSFUL (total time: 31 seconds)
```

Conclusion

The project was successfully completed. At the start, we expected to apply our knowledge of Git operations, UML and USE case idaigrams along with how to define user paths, OOD principles and how to test programs and we have been successfully able to do that with the base code we were given. The game works and is fully functional.

References

Tieperman, I. (2024, August). *A beginner's Guide to Blackjack*

https://www.wikihow.com/Play-Blackjack