DELIVERABLE 2

For

Nagma Nagma

Instructor of Software Design and Development

Sheridan College

Brampton, Ontario

By

Kaitlin Saqui: 991723734
Arshvir Singh: 991709700
Mandeep Kaur: 991701614
Manpreet Kaur: 991680973

July 25, 2023

TABLE OF CONTENTS

INTRODUCTION

This document is focused on the blackjack card game project. This is the second part of the project that is going to include the details about the project. First, we will have the Use Case Diagram. Then, there is the Use Case Narratives. Next is the Class Diagrams. And at last, the Design document template that will include the project background and details about the class diagrams.

USE CASE DIAGRAM

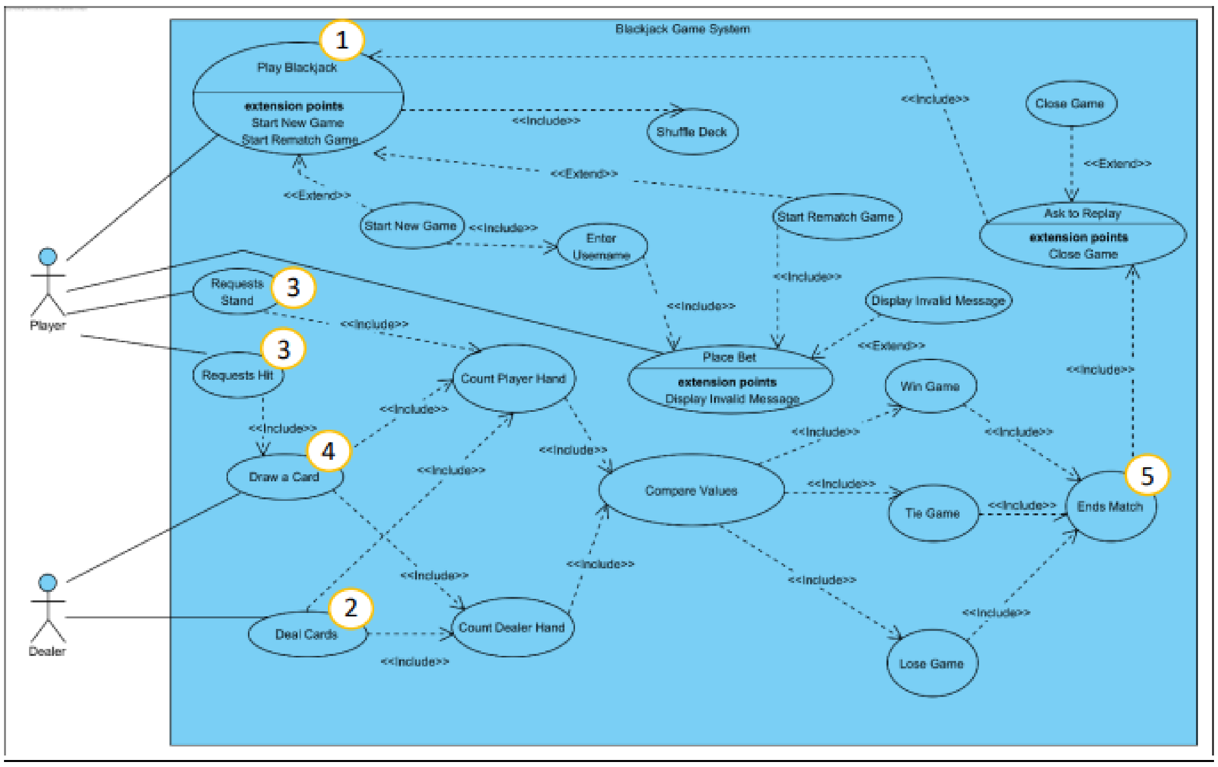
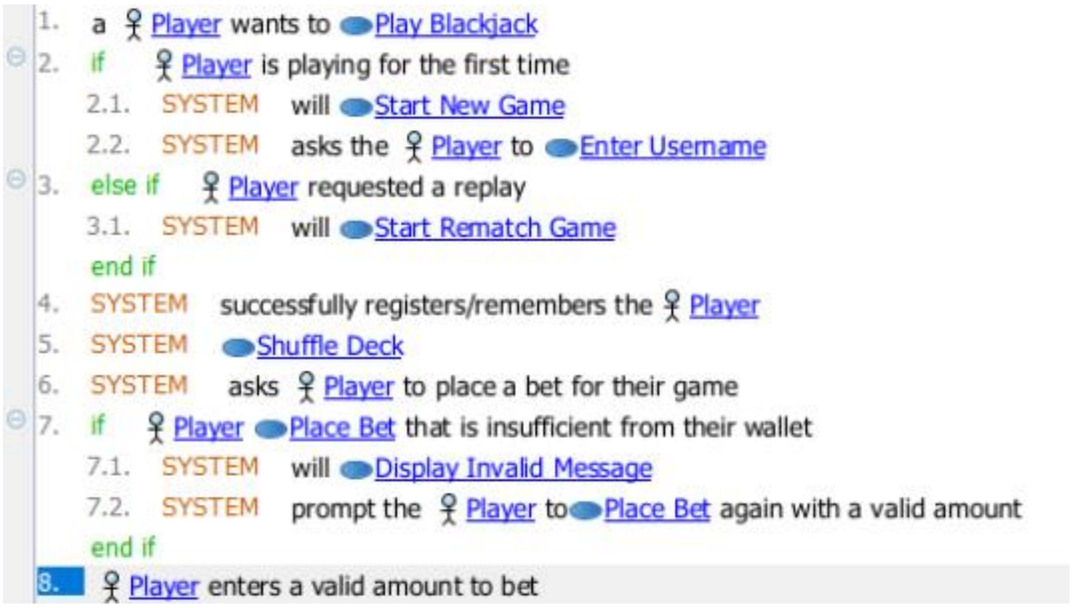


Figure 1 - UML use case diagram for a Blackjack Game System

This use case diagram was designed in Visual Paradigm and is used to model the dynamic behaviour of the Blackjack game system. It is used to display the relationships between actors and use cases, along with the system's functionality. In Figure 1, it presents the following use cases: how a Player would start playing Blackjack, how a Player can bet, how a Player can hit or stand, and how a Player would win, lose, or tie a game. This diagram also displays how a Dealer deals the cards to the Player and themselves, how a Dealer draws a card, and how a Dealer would win, lose, or tie a game.

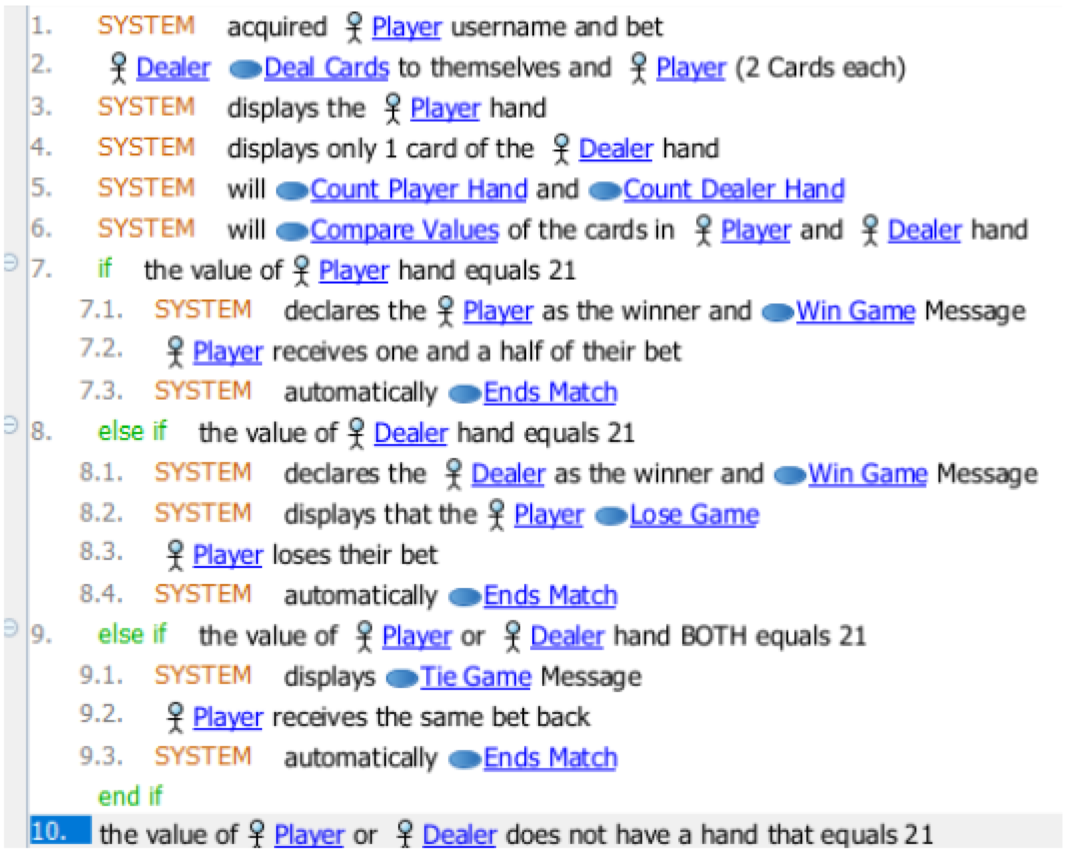
USE CASE NARRATIVES

CASE 1: PLAYER WANTS TO PLAY BLACKJACK AND BET



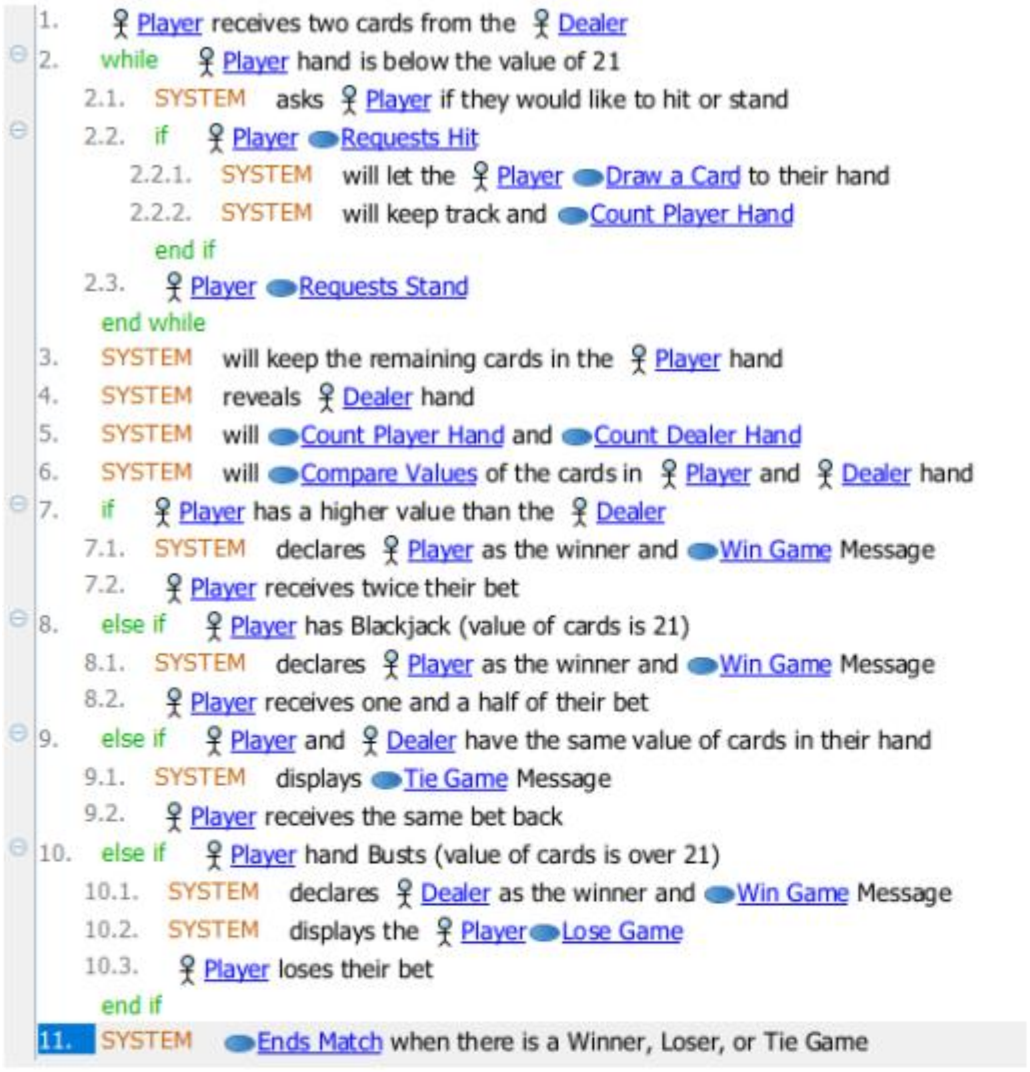
```
1. a Player wants to Play Blackjack
2. if Player is playing for the first time
    2.1. SYSTEM will Start New Game
    2.2. SYSTEM asks the Player to Enter Username
3. else if Player requested a replay
    3.1. SYSTEM will Start Rematch Game
    end if
4. SYSTEM successfully registers/remembers the Player
5. SYSTEM Shuffle Deck
6. SYSTEM asks Player to place a bet for their game
7. if Player Place Bet that is insufficient from their wallet
    7.1. SYSTEM will Display Invalid Message
    7.2. SYSTEM prompt the Player to Place Bet again with a valid amount
    end if
8. Player enters a valid amount to bet
```

CASE 2: DEALER DEALS CARDS AND THERE IS AN AUTOMATIC BLACKJACK



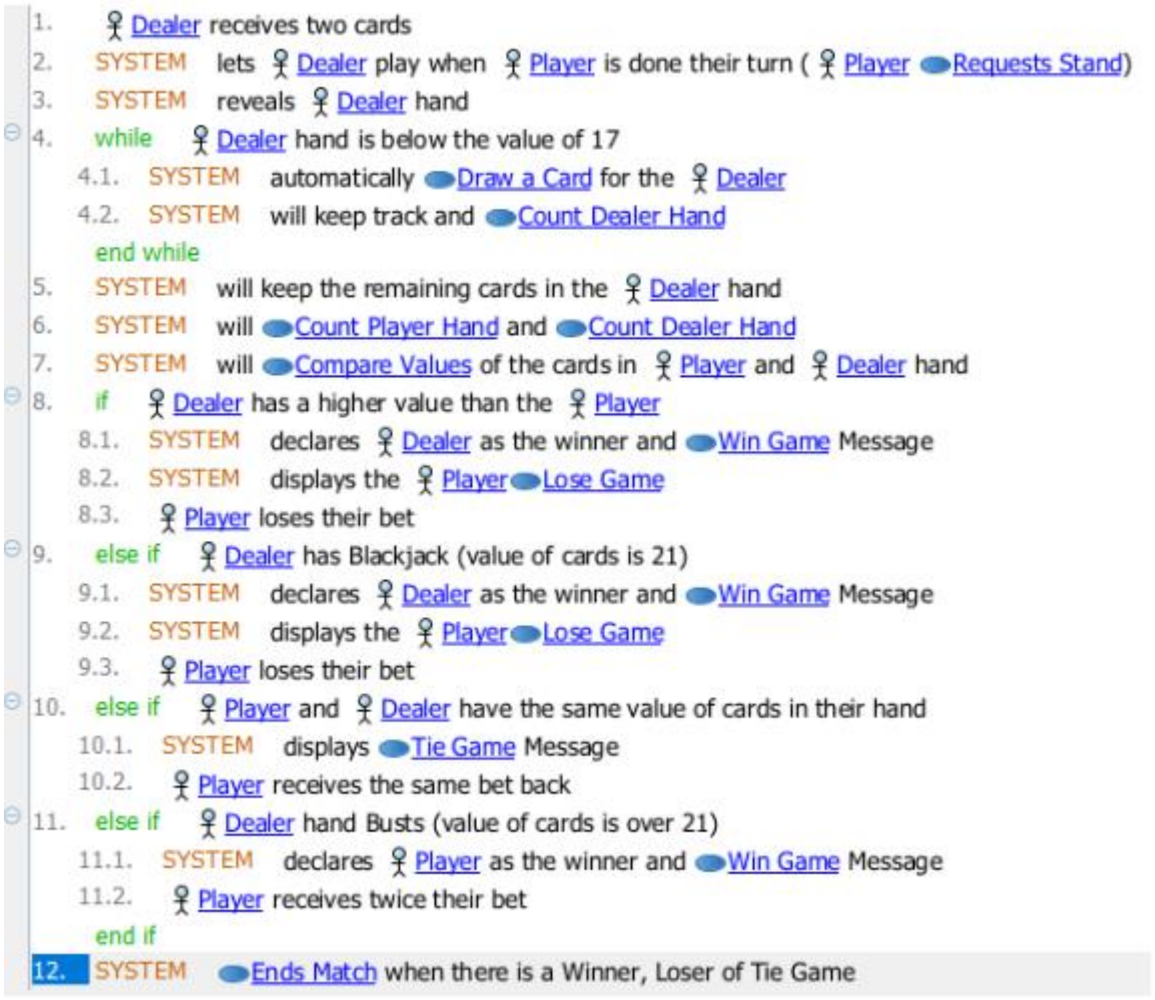
```
1.    SYSTEM   acquired Player username and bet
2.    Dealer Deal Cards to themselves and Player (2 Cards each)
3.    SYSTEM   displays the Player hand
4.    SYSTEM   displays only 1 card of the Dealer hand
5.    SYSTEM   will Count Player Hand and Count Dealer Hand
6.    SYSTEM   will Compare Values of the cards in Player and Dealer hand
7.    if   the value of Player hand equals 21
    7.1.   SYSTEM   declares the Player as the winner and Win Game Message
    7.2.   Player receives one and a half of their bet
    7.3.   SYSTEM   automatically Ends Match
8.    else if   the value of Dealer hand equals 21
    8.1.   SYSTEM   declares the Dealer as the winner and Win Game Message
    8.2.   SYSTEM   displays that the Player Lose Game
    8.3.   Player loses their bet
    8.4.   SYSTEM   automatically Ends Match
9.    else if   the value of Player or Dealer hand BOTH equals 21
    9.1.   SYSTEM   displays Tie Game Message
    9.2.   Player receives the same bet back
    9.3.   SYSTEM   automatically Ends Match
      end if
10.   the value of Player or Dealer does not have a hand that equals 21
```

CASE 3: PLAYER HIT OR STAND



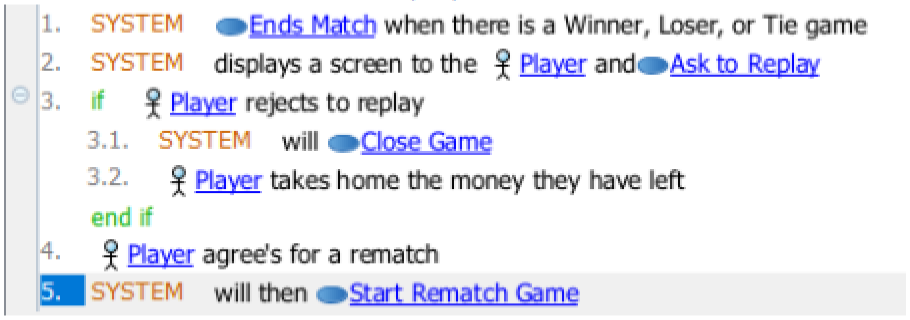
```
1.    Player receives two cards from the Dealer
2.    while   Player hand is below the value of 21
   2.1.  SYSTEM   asks Player if they would like to hit or stand
   2.2.  if   Player Requests Hit
      2.2.1.  SYSTEM   will let the Player Draw a Card to their hand
      2.2.2.  SYSTEM   will keep track and Count Player Hand
         end if
   2.3.  Player Requests Stand
      end while
3.    SYSTEM   will keep the remaining cards in the Player hand
4.    SYSTEM   reveals Dealer hand
5.    SYSTEM   will Count Player Hand and Count Dealer Hand
6.    SYSTEM   will Compare Values of the cards in Player and Dealer hand
7.    if   Player has a higher value than the Dealer
   7.1.  SYSTEM   declares Player as the winner and Win Game Message
   7.2.  Player receives twice their bet
8.    else if   Player has Blackjack (value of cards is 21)
   8.1.  SYSTEM   declares Player as the winner and Win Game Message
   8.2.  Player receives one and a half of their bet
9.    else if   Player and Dealer have the same value of cards in their hand
   9.1.  SYSTEM   displays Tie Game Message
   9.2.  Player receives the same bet back
10.   else if   Player hand Busts (value of cards is over 21)
   10.1.  SYSTEM   declares Dealer as the winner and Win Game Message
   10.2.  SYSTEM   displays the Player Lose Game
   10.3.  Player loses their bet
      end if
11.   SYSTEM   Ends Match when there is a Winner, Loser, or Tie Game
```

CASE 4: DEALER DRAWS CARD(S)



```
1.    Dealer receives two cards
2.    SYSTEM   lets Dealer play when Player is done their turn ( Player Requests Stand)
3.    SYSTEM   reveals Dealer hand
4.    while    Dealer hand is below the value of 17
      4.1.  SYSTEM   automatically Draw a Card for the Dealer
      4.2.  SYSTEM   will keep track and Count Dealer Hand
      end while
5.    SYSTEM   will keep the remaining cards in the Dealer hand
6.    SYSTEM   will Count Player Hand and Count Dealer Hand
7.    SYSTEM   will Compare Values of the cards in Player and Dealer hand
8.    if    Dealer has a higher value than the Player
      8.1.  SYSTEM   declares Dealer as the winner and Win Game Message
      8.2.  SYSTEM   displays the Player Lose Game
      8.3.  Player loses their bet
9.    else if    Dealer has Blackjack (value of cards is 21)
      9.1.  SYSTEM   declares Dealer as the winner and Win Game Message
      9.2.  SYSTEM   displays the Player Lose Game
      9.3.  Player loses their bet
10.   else if    Player and Dealer have the same value of cards in their hand
      10.1.  SYSTEM   displays Tie Game Message
      10.2.  Player receives the same bet back
11.   else if    Dealer hand Busts (value of cards is over 21)
      11.1.  SYSTEM   declares Player as the winner and Win Game Message
      11.2.  Player receives twice their bet
      end if
12.   SYSTEM   Ends Match when there is a Winner, Loser of Tie Game
```

CASE 5: AFTER A MATCH ENDS – ASK TO REPLAY:



```
1.    SYSTEM   Ends Match when there is a Winner, Loser, or Tie game
2.    SYSTEM   displays a screen to the Player and Ask to Replay
3.    if    Player rejects to replay
      3.1.  SYSTEM   will Close Game
      3.2.  Player takes home the money they have left
      end if
4.    Player agree's for a rematch
5.    SYSTEM   will then Start Rematch Game
```

Class Diagram

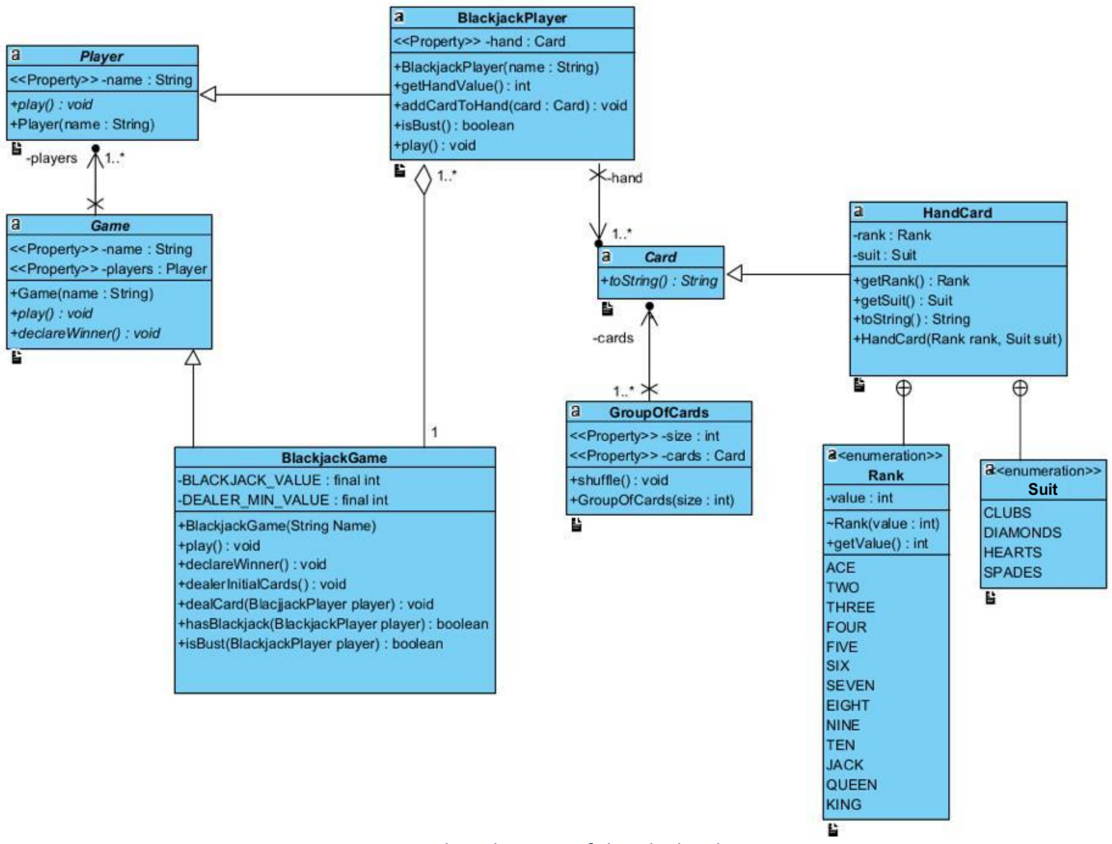


Figure 2 - UML class diagram of the Blackjack Game System

DESIGN DOCUMENT

PROJECT BACKGROUND

Scope of the Blackjack Game Project:

1) The game will start by asking the user to input the number of players and their names.
2) Before each round, players will place their bets.
3) The game will use a standard 52-card deck of playing cards (without jokers). The deck will be shuffled before each round.
4) Players and the dealer will be dealt two cards each at the beginning of each round.
5) Players will be able to see their two cards and one of the dealer's cards. The dealer's other card will remain hidden until it's their turn. Numbered cards (2 to 10) will have their face values (e.g., 2 is worth 2 points). Face cards (Jack, Queen, King) will be worth 10 points each. Aces can be worth either 1 or 11 points, depending on the player's choice and hand value.
6) During their turn, each player will have the option to "hit" (draw another card) or "stand" (keep their current hand).

7) If a player's hand value exceeds 21 after hitting, they will bust and lose the round. If a player's hand value reaches exactly 21, they get one and a half times their bet amount.

8) Once all players have finished their turns, the dealer will reveal their hidden card and hit until their hand value is 17 or greater.

If the dealer busts, all remaining players win twice their bet amount.

If the dealer doesn't bust, players whose hand value is greater than the dealer's hand value win twice their bet amount.

9) The game will track each player's balance, deducting their lost bet amount and adding the winnings at the end.

10) Players can continue playing until they decide to quit or their balance reaches zero. The game will end after a certain number of rounds (e.g., four rounds). Alternatively, players can choose to quit the game at any time. The final score of each player (remaining balance) will be displayed at the end.

11) The game will provide appropriate prompts and messages to guide players during their turns and at the end of each round.

Players' choices (hit/stand) and the dealer's actions will be clearly displayed.

12) The game will handle input validation and errors gracefully to avoid crashing or unexpected behavior.

DESIGN CONSIDERATIONS

1. CLASSES

- **Player**: Represents a general player with attributes and methods related to player behavior.
- **BlackjackPlayer**: Inherits from the Player class, representing a player specifically for a blackjack game.
- **Game**: Represents a general game with attributes and methods related to game behavior.
- **Card**: Represents a single playing card with attributes like suit and rank. **BlackjackGame**: Inherits from the Game class, representing a blackjack-specific game.
- **GroupOfCards**: Represents a collection of cards, possibly a deck or a hand of cards.
- **HandCard**: Inherits from the GroupOfCards class, representing a player's hand in a card game.

- **Rank**: An enumeration representing the ranks of cards (e.g., Ace, Two, Three, etc.).
- **Suit:** An enumeration representing the suit of cards (e.g., Clubs, Diamonds, Hearts, Spades)

2. ASSOCIATION AND MULTIPLICITIES

- Player has an association with BlackjackPlayer with a multiplicity of 0..1 (optional), implying that a general player can be a blackjack player, but not all players necessarily are.
- Game has an association with BlackjackGame with a multiplicity of 0..1 (optional), indicating that a game can be a blackjack game, but not all games are blackjack games.
- Player has an association with HandCard with a multiplicity of 0..1 (optional) to represent that a player may have a hand of cards in a game. BlackjackGame has an association with GroupOfCards with a multiplicity of 1..* (at least one), suggesting that a blackjack game must have a deck of cards.

- BlackjackGame has an association with BlackjackPlayer with a multiplicity of 1..* (at least one), indicating that a blackjack game must have at least one player.
- HandCard has an association with Card with a multiplicity of 1..* (at least one), representing that a hand of cards consists of one or more individual cards.

3. ENCAPSULATION

The class diagram demonstrates encapsulation as each class hides its internal implementation and provides public methods to interact with the data. For example, each of the classes contain data members/data fields are private and protects the data from unauthorized access or manipulation. This can be seen in the HandCard class, the Rank and Suit data fields are private and can only be accessed from their getters (getRank(), getSuit()). The Rank class encapsulates the value data field with the getValue() method. GroupOfCards encapsulates the size and cards data fields by setting the size within the constructor and the cards is accessible within the HandCard class. The BlackjackPlayer has a hand data field that is only accessible through the HandCard class. The Player class has a name data field that can only be set through the constructor. The Game class has a name and player data fields that can only be set when the Game is instantiated. And lastly, the BlackJackGame class has the BLACKJACK_VALUE and DEALER_MIN_VALUE as private as they are constants that cannot be changed at all.

4. DELEGATION

Delegation can be observed in the association between BlackjackGame and BlackjackPlayer. The BlackjackGame class can delegate specific tasks or responsibilities to the BlackjackPlayer class without having to do it directly by asking another object to handle or be a part of the task. In Figure 2 we can see methods in the BlackjackGame perform the following tasks such as, dealing cards (dealCard(BlackjackPlayer player), determining if a player has Blackjack (hasBlackjack(BlackjackPlayer player) and if a player has Bust (isBust(BlackjackPlayer player). This makes the code more reusable by letting each object focus on its own functionality by letting any BlackjackPlayer that is created in the BlackjackGame perform any of the tasks independently.

5. COHESION

Cohesion represents how closely the elements within a class are related to each other. From the class diagram, each class seems to have high cohesion, as they are designed to represent specific entities and their related functionalities.

The Game class serves as a generic base for games, providing abstract methods for playing the game and declaring a winner. The GroupOfCards class represents a collection of cards and includes methods for shuffling the cards. The Card class, an abstract class, acts as the base for various types of cards and defines an abstract toString() method for displaying card information. The Player class represents a player in the game and contains the player's name and an abstract play() method that must be implemented by its subclasses. Finally, the BlackjackGame, BlackjackPlayer, HandCard, Rank, and Suit classes are tailored specifically for the Blackjack game, each with their respective responsibilities.

6. COUPLING

Coupling refers to the interdependence between classes. For instance, Coupling between BlackjackGame and BlackjackPlayer. The BlackjackGame class interacts with the BlackjackPlayer class by calling its methods, such as play() and isBust(this), to manage the player's actions during the game. for (Player player : getPlayers()) { player.play(); }.Another instance, between BlackjackGame and GroupOfCards. The BlackjackGame class calls methods from the GroupOfCards class, such as initialize() and shuffle(), to manage the deck of cards deck.initialize(); and deck.shuffle();.

7. INHERITANCE

Inheritance is demonstrated by classes like BlackjackPlayer and BlackjackGame inheriting from the more general Player and Game classes, respectively. This allows the specific classes to reuse and extend functionalities from their parent classes.

The BlackjackGame class extends the Game class, which enables it to inherit the generic game functionality and define its specific implementations for playing the Blackjack game and declaring a winner. The BlackjackPlayer class extends the Player class, inheriting the name attribute and the abstract play() method. This allows the BlackjackPlayer class to implement its own logic for a player's turn in the Blackjack game

8. AGGREGATION AND COMPOSITION

Aggregation is a type of association where one class contains references to other classes, but the referenced objects maintain an independent lifecycle. The code demonstrates aggregation in a couple of instances: The BlackjackGame class contains two attributes: deck, which refers to a GroupOfCards instance, and dealer, which refers to a BlackjackPlayer instance representing the dealer. This relationship indicates that a BlackjackGame "has-a" deck of cards and a dealer. The BlackjackGame class doesn't control the lifecycle of these instances; they are independent entities that exist within the context of the game. Composition can be represented if, for example, the HandCard class "owns" the individual Card objects in a player's hand, meaning the cards are created and destroyed with the HandCard.

9. FLEXIBILITY/MAINTAINABILITY

The class diagram's design decisions support flexibility and maintainability. For example, the use of inheritance allows for code reuse and easy extension of functionalities. The associations between classes enable the components to interact efficiently and provide modularity in the design.

Flexibility of the BlackjackGame class:

- The BlackjackGame class can accommodate any number of players by prompting the user for the number of players and dynamically creating player objects. Example: int numPlayers = scanner.nextInt(); and getPlayers().add(player);

Flexibility of the BlackjackPlayer class:

- The BlackjackPlayer class has a flexible way of calculating the hand value, considering the value of aces based on whether they cause a bust or not. Example: public int getHandValue()

Maintainability of the GroupOfCards class:

- The GroupOfCards class encapsulates the deck of cards used in the game, making it easier to modify or replace the deck implementation without affecting the game logic. Example: public class GroupOfCards.

Use of Enums for Maintainability:

- The code uses enums (Rank and Suit) to represent card ranks and suits. Enums provide a maintainable and concise way to define a fixed set of constants for the card values. Example: enum Rank and enum Suit.