SINGAPORE MANAGEMENT UNIVERSITY

IS442 Object Oriented Programming

Group Project Report G1T08

Chang Ern Rae

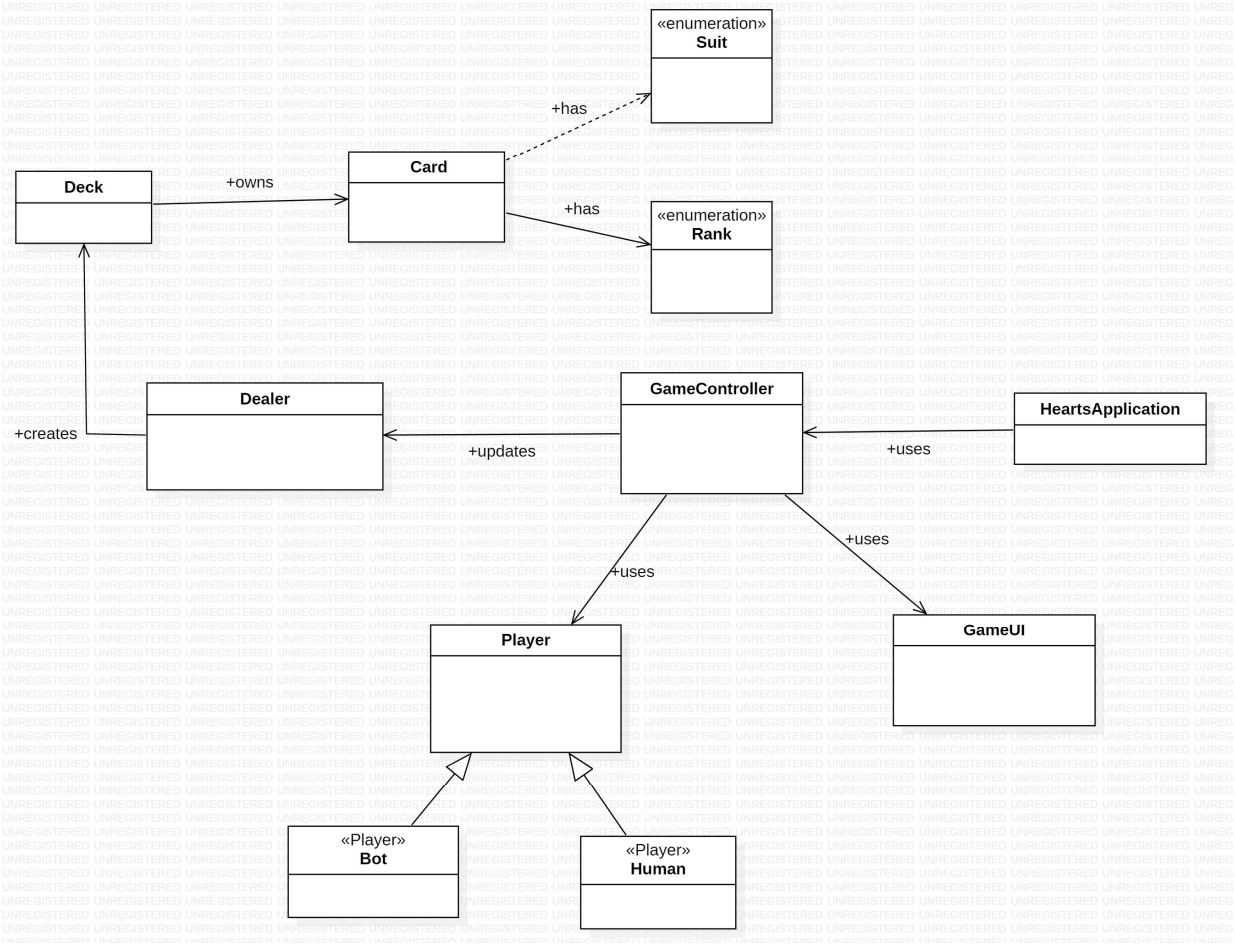
Lam Guolun Amos

Lim Zhao Wei

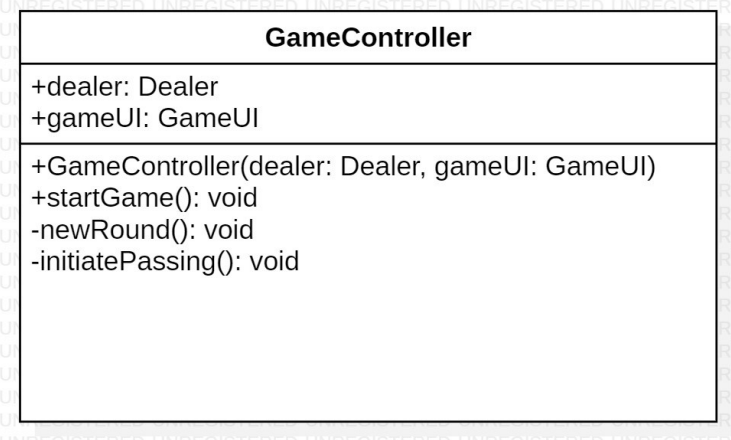
Contents

1. Class Diagram

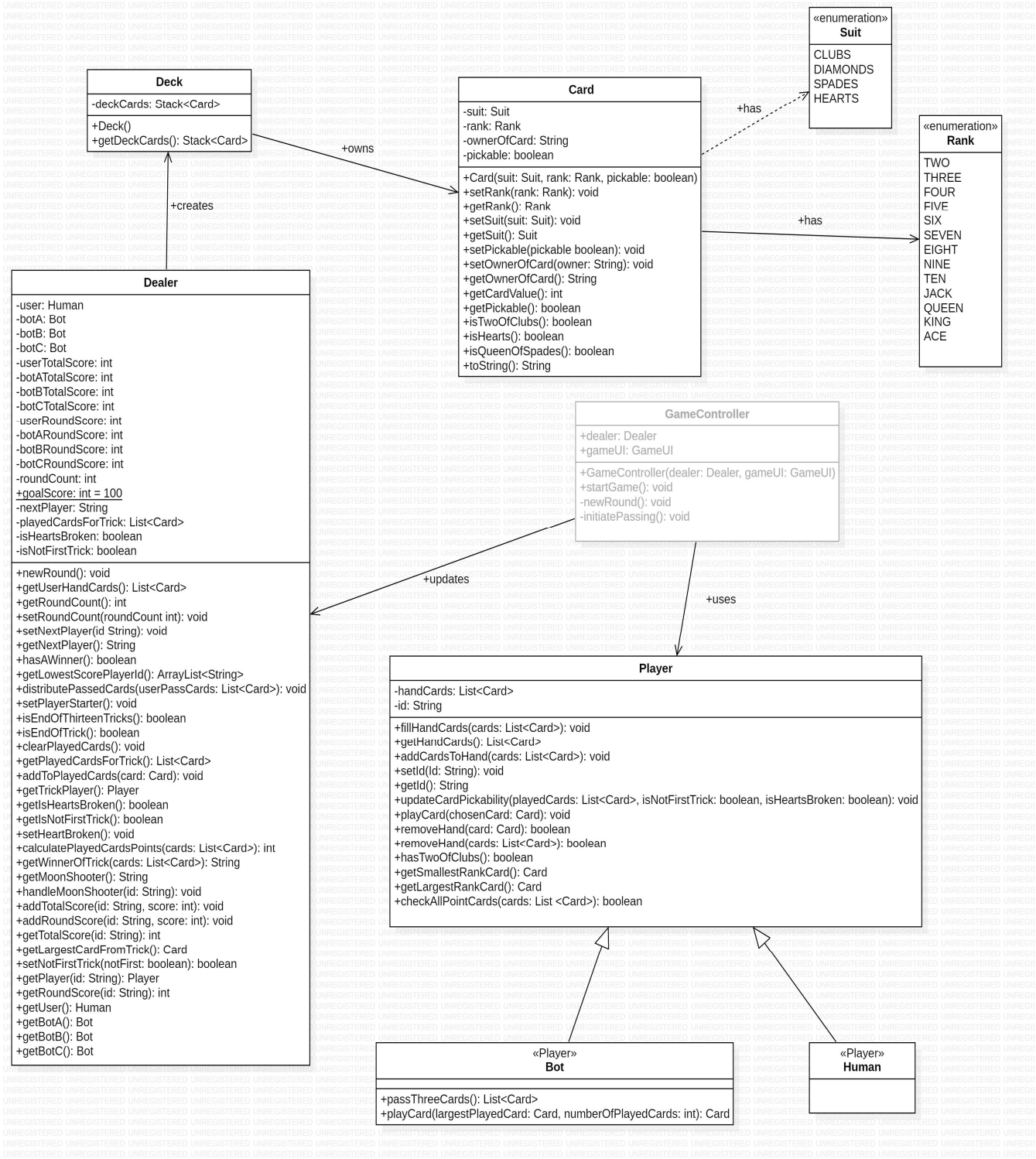
2.1. Main class diagram



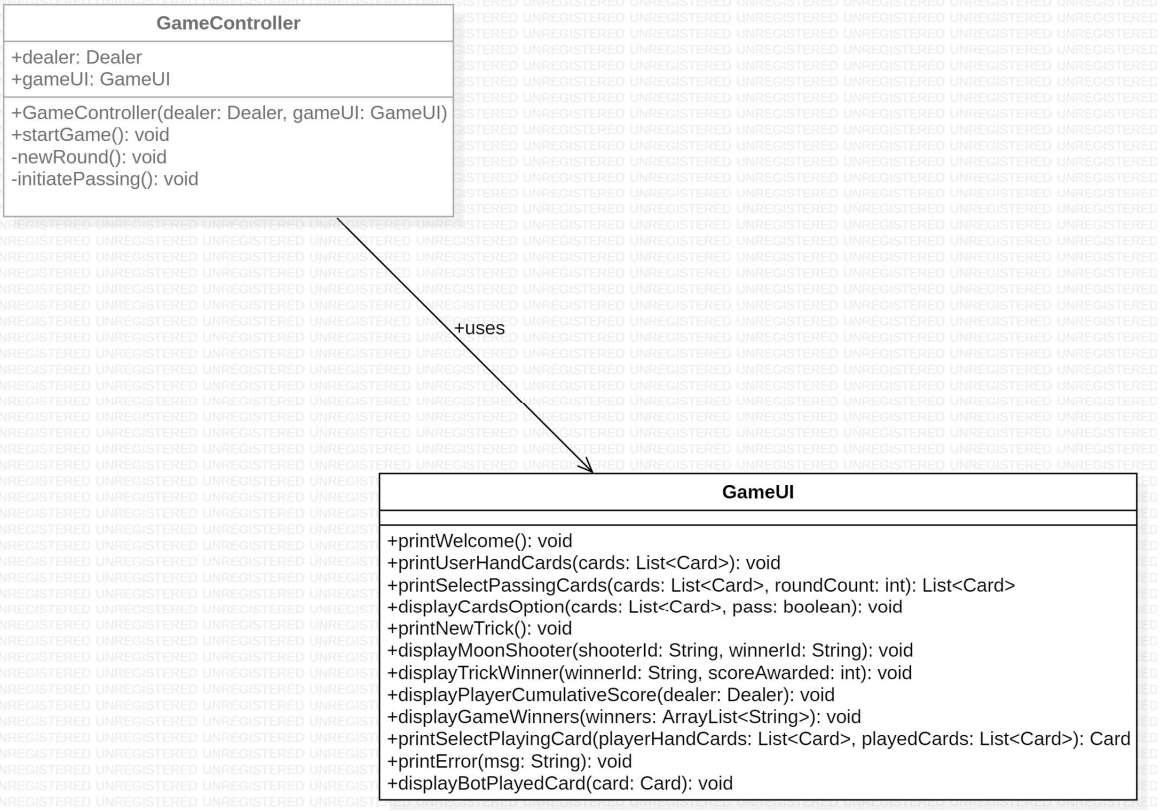
2.2. Controller Class



2.3. Model class



2.4. View Class



2. Object-Oriented Design Considerations

2.1. Approach Considerations

For the design choice, the textbook “A Journey Into the IT Jungle” was referenced for ideas. Database application design is not considered for this project as the requirements does not include long term persistence of data. Drawing ideas from both single responsibility and stereotype principle, we decided to model our classes according to the following clearly defined roles:

1) Dealer Class: shapes the behaviour of an actual dealer in a poker card game where the dealer distributes the cards, be the winner judge and others.

2) Player class: it shapes the behaviour of an actual Hearts Game player who can play and receive cards among many other actions.

These roles would allow us to create codes that are clear and simple since responsibilities of each classes are clearly defined. In addition, with the stereotype principle, we can better separate the front-end **(Boundary)** from the back-end **(Entity).** This suits our project requirements since user interface will be a big consideration in the project switching from a console interface to Graphical User Interface (GUI) -- Java Swing or JavaFx. With no experience in using Java Swing, we can leverage on the loose coupling advantage between entity and boundary to first code out our Hearts Application in console UI. This would allow for easy transition to GUI using Java Swing to improve the front-end should we have addition time. With these considerations in mind, we selected the MVC design pattern.

2.2. MVC (Model-View-Controller) Design Pattern

Stereotype	MVC	Description
Entity	Model	The model keeps domain specific information. This is similar to the entity stereotype.
Boundary	View	The view renders the information from the model in some user presentation format. This is similar to the boundary stereotype.
Control	Controller	The controller response and process specific to user actions from the user interface. It corresponds similarly to the control stereotype.

Figure 1. Definition taken from "A Journey Into the IT Jungle"

2.2.1. Reasons for decision

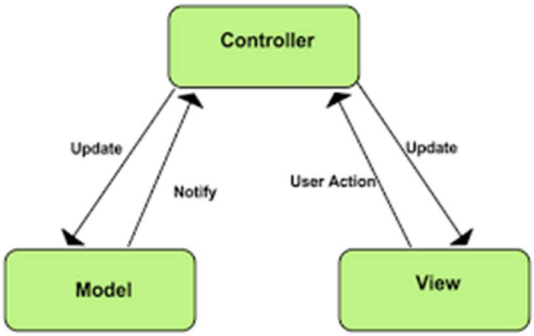


Figure 2. MVC Architecture Diagram. (Google Chrome, n.d.)

- Introduces flexibility in changing User Interface without affecting codes in other areas
- Suitable for applications that experience frequent changes in UI due to customer feedback while conceptual classes that capture information are normally less volatile to changes.

2.3. Usage Examples

MVC is implemented as much as possible throughout our codes and it can be observed through the sequence diagrams below.

One example will be the sequence diagram for Win Trick **(Section 4.9. Win Trick)** where GameController is the middle-man class that interacts with GameUI (View) and Dealer (Model) on behalf of each other. This creates a decoupling effect between model and view classes.

3. Use Cases

3.1. Use Case Diagram

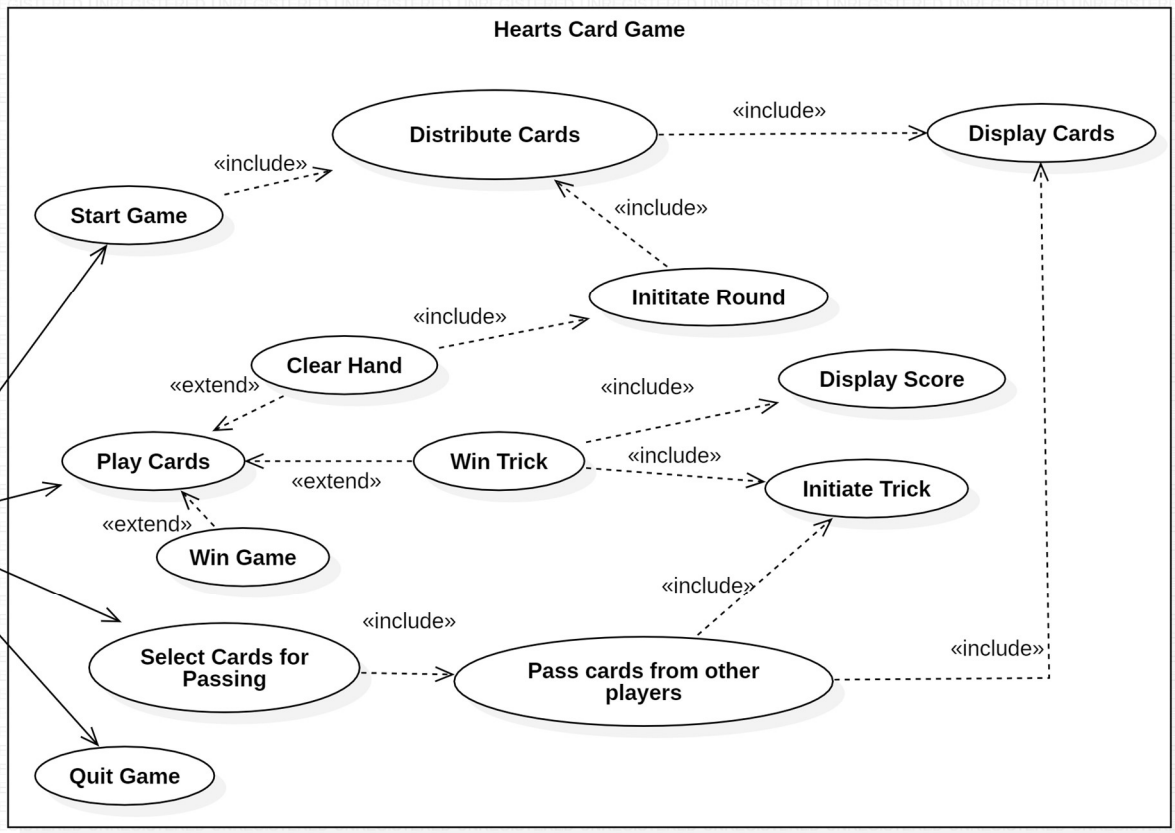


Figure 3. Use Case Diagram of Hearts Application

3.2. Use Case Specifications

Use Case	**Start Game**
Brief Description	The goal of Start Game is to start a game of Hearts.
Actor	User
Precondition	User has the application files and knows how to run them.
Main Flow of Events	1. This use case begins when the user begins the game 2. The system displays a welcome message and instructions.
Alternate Flow of Events	NIL
Use Case	**Initiate Round**
Brief Description	The goal of Initiate Round is to begin a new round.
Actor	User
Precondition	User has no cards on hand. Game is either the 1st round or 13 tricks have ended.

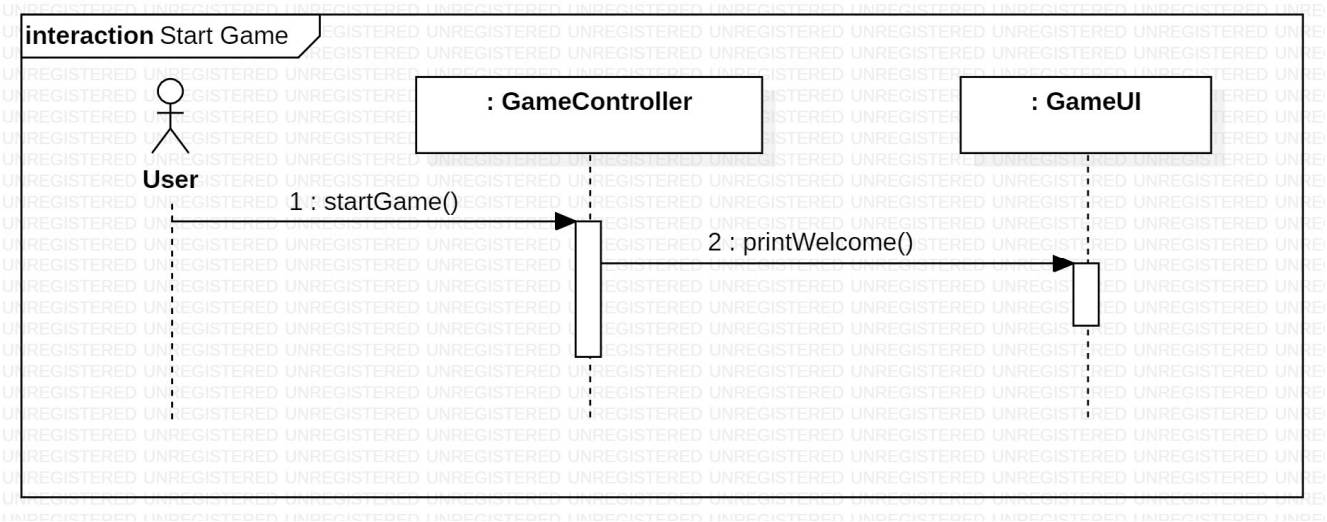
Main Flow of Events	1. This use case begins when all 13 tricks have been played and there are no more cards in the deck. System will reset the round scores, shuffle the cards, distribute them and randomly store 13 cards for each player.
Alternate Flow of Events	NIL
Use Case	**Display Cards**
Brief Description	The goal of Display Cards is for the User to view the cards they have on hand.
Actor	User
Precondition	User has no more than 13 playing cards on hand.
Main Flow of Events	1. This use case begins before the User needs to see their cards on hand. 2. The system displays the cards on User's current hand.
Alternate Flow of Events	NIL
Use Case	**Select Cards for Passing**
Brief Description	The goal of Select Cards for Passing is for the User to input the 3 cards he/she would like to pass.
Actor	User
Precondition	User has been distributed their cards (has 13 cards on hand)
Main Flow of Events	1. This use case begins when a pass is happening. 2. The system will prompt the User to select a card to be passed to the next player. 3. The user keys in their choice and presses enter The system will record the chosen card and repeats step 2-3 until 3 cards have been chosen.
Alternate Flow of Events	2a. Game is in Round 4 1. System will not prompt User for any cards.
Error Flow of Events	E1: User keys in an invalid input The system displays an error message and prompts for another input.
Use Case	**Pass Cards from Other Players**
Brief Description	The goal of Pass Cards from Other Players is to gather the chosen cards from other players and facilitate the pass between all players such that they each have 13 cards
Actor	User
Precondition	User has selected the 3 cards they want to pass.
Main Flow of Events	1. This use case begins when User initiates the pass by providing the 3 cards to be passed from his/her hand. The system will retrieve the cards passed from the player on the User's right and store it in the User's hand.
Alternate Flow of Events	2a. Game is in Round 2 1. The system will retrieve the cards passed from the player on the User's left and store it in the User's hand. 2b. Game is in Round 3 1. The system will retrieve the cards passed from the player across the User and store it in the User's hand. 2c. Game is in Round 4 3. The system will begin the trick.
Use Case	**Initiate Trick**

Brief Description	The goal of Initiate Trick is to trigger the start of a trick.
Actor	User
Precondition	Cards played stack must be empty User must have completed the passing round.
Main Flow of Events	1. This use case begins when there are no cards on the table. The system checks if User's hand has the starting card (2 of Clubs). If there is, the system will prompt the User to play the first card.
Alternate Flow of Events	2a. Game > Round 1 1. The System checks if the User meets the criteria to lead the trick. If User meets criteria, the system will prompt the User to play their card of choice.
Error Flow of Events	E1. User keys in an invalid input – Invalid Option/ Illegal Card The system displays an error message and prompts for another input.
Use Case	**Win Trick**
Brief Description	The goal of Win Trick is to determine the winner of the trick by calculating the score accumulated from the trick
Actor	User
Precondition	All 4 players have played their card.
Main Flow of Events	1. This use case begins when all 4 players have played their card for the current trick. 2. The system checks for the highest card of the suit during the trick.
Alternate Flow of Events	NIL
Use Case	**Play Cards**
Brief Description	The goal of Play Cards is to facilitate the playing of a trick by prompting for the User's desired choice.
Actor	User
Precondition	Trick is ongoing User has viewed his/her cards
Main Flow of Events	1. This use case begins when the User needs to decide which cards to play. 2. The user keys in their choice of card. The system will display the User's choice
Error Flow of Events	E1. User keys in an invalid input – Invalid Option/ Illegal Card 1. The system displays an error message and prompts for another input.
Use Case	**Display Score**
Brief Description	The goal of Display Score is to display the current scores
Actor	User
Precondition	User has won a trick
Main Flow of Events	1. This use case begins after the User wins a trick. 2. The system displays the scores of all players at the end of the current trick.
Alternate Flow of Events	NIL
Use Case	**Quit Game**
Brief Description	The goal of Quit Game is to exit the application
Actor	User

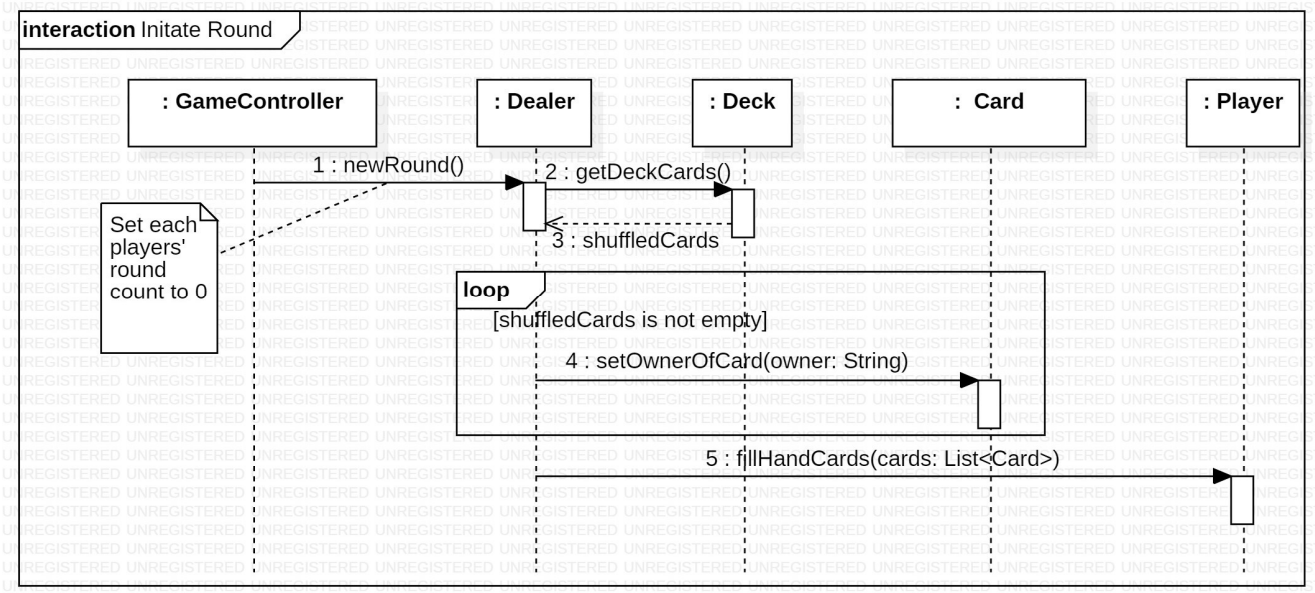
Precondition	User is being prompted to key in an input.
Main Flow of Events	1. This use case begins when the user keys in “Quit” or “quit” as an input. 2. The system will end the application immediately.
Alternate Flow of Events	NIL
Use Case	**Win Game**
Brief Description	The goal of Win Game is to determine the winner
Actor	User
Precondition	A player has hit 100 points
Main Flow of Events	1. This use case begins when any player hits an accumulated score of 100 at the end of the trick. 2. The system checks for the lowest scoring player, displays that player as the winner and ends the application.
Alternate Flow of Events	NIL

4. Sequence Diagrams

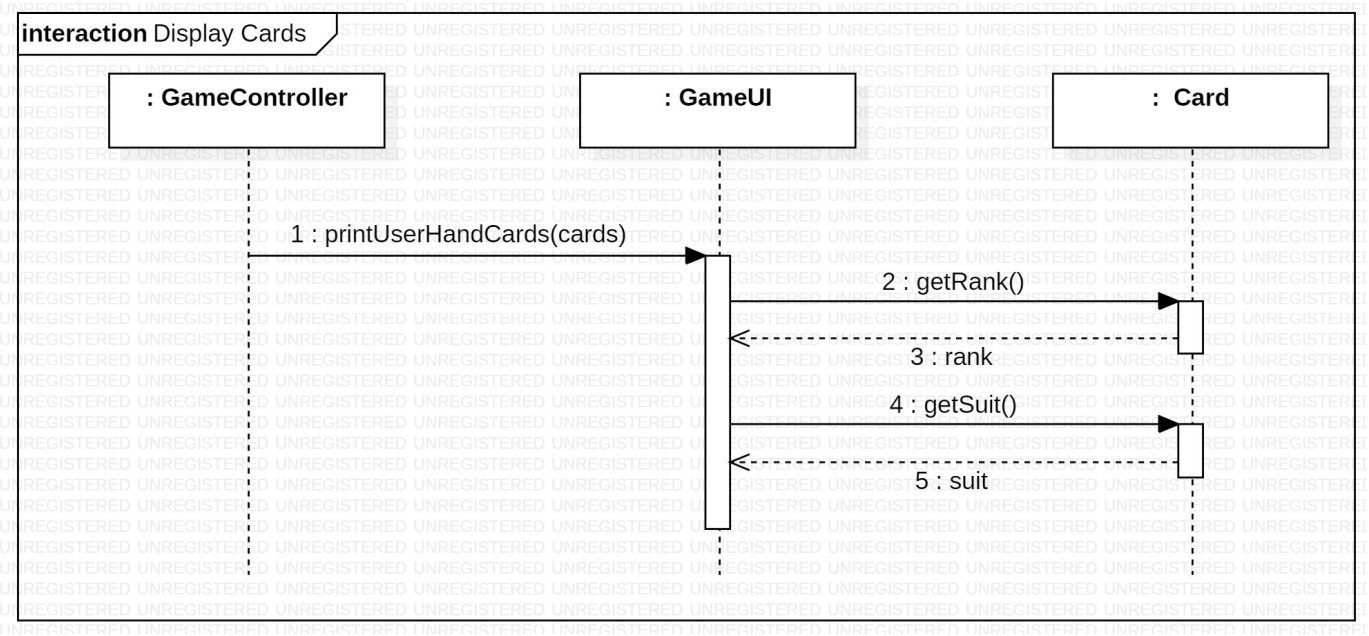
4.1. Use Case: Start Game



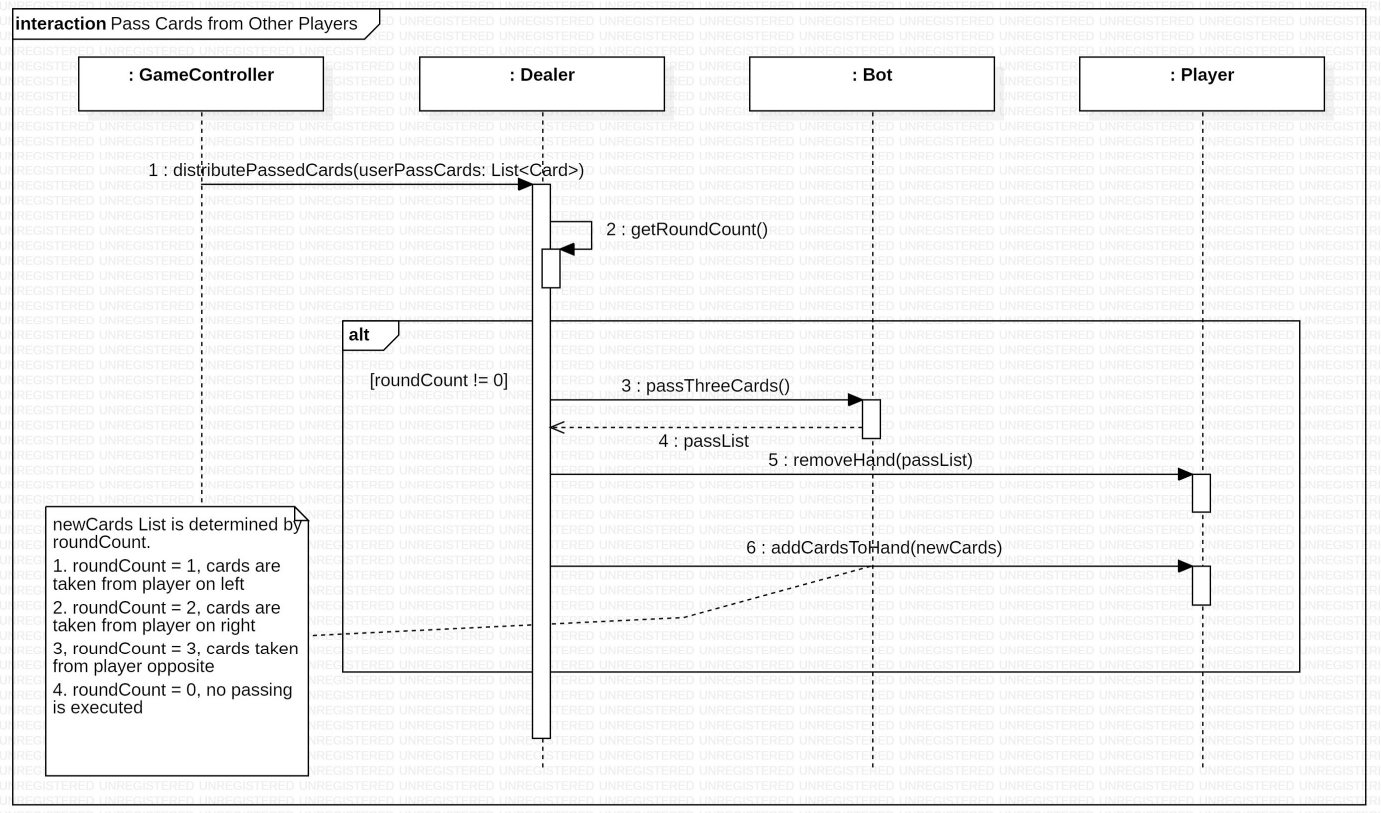
4.2. Use Case: Initiate Round



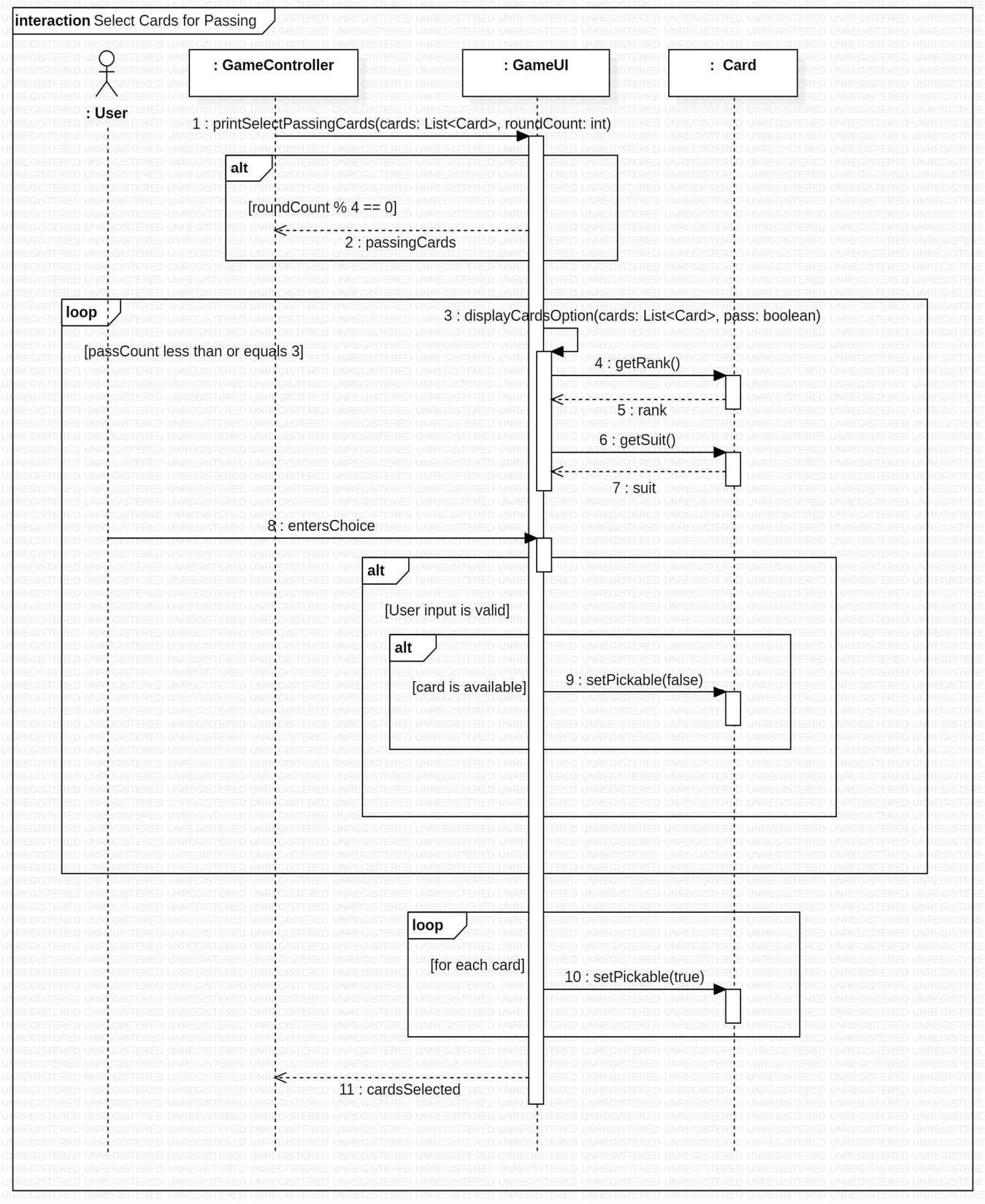
4.3. Use Case: Display Cards



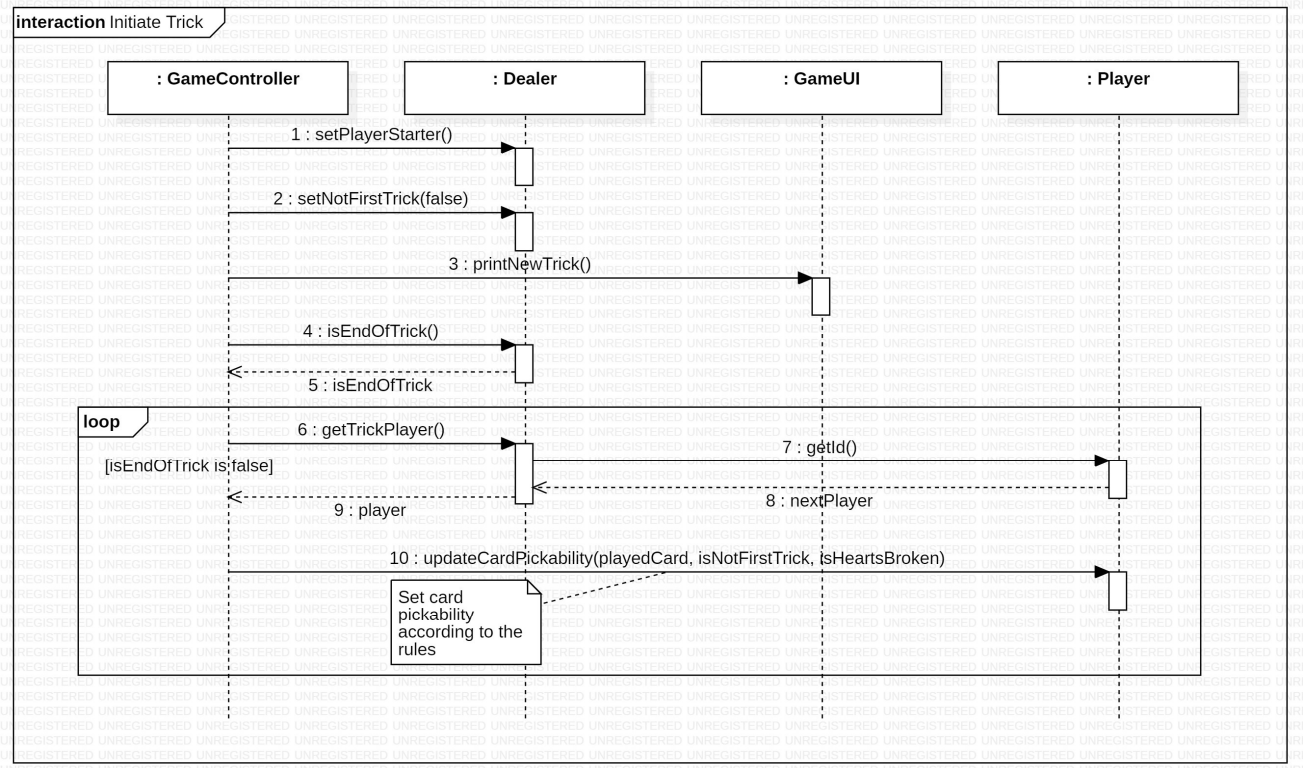
4.4. Use Case: Pass Cards from Other Players



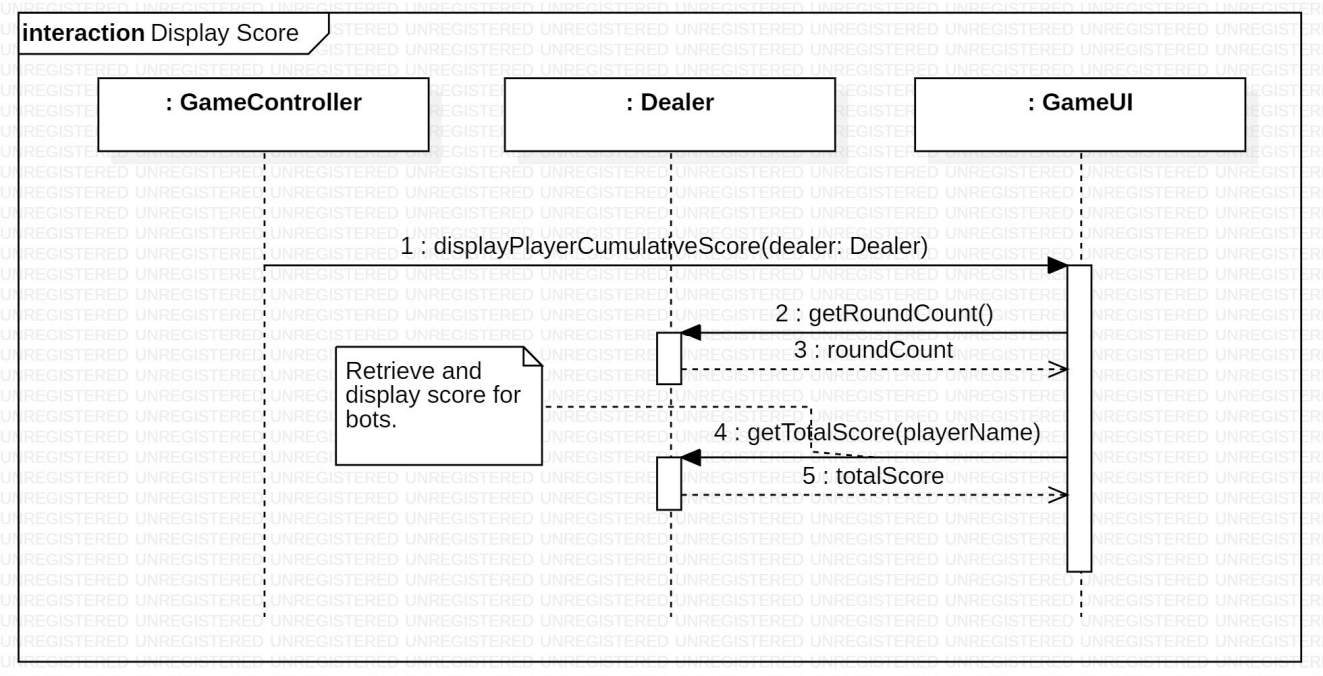
4.5. Use Case: Select Cards for Passing



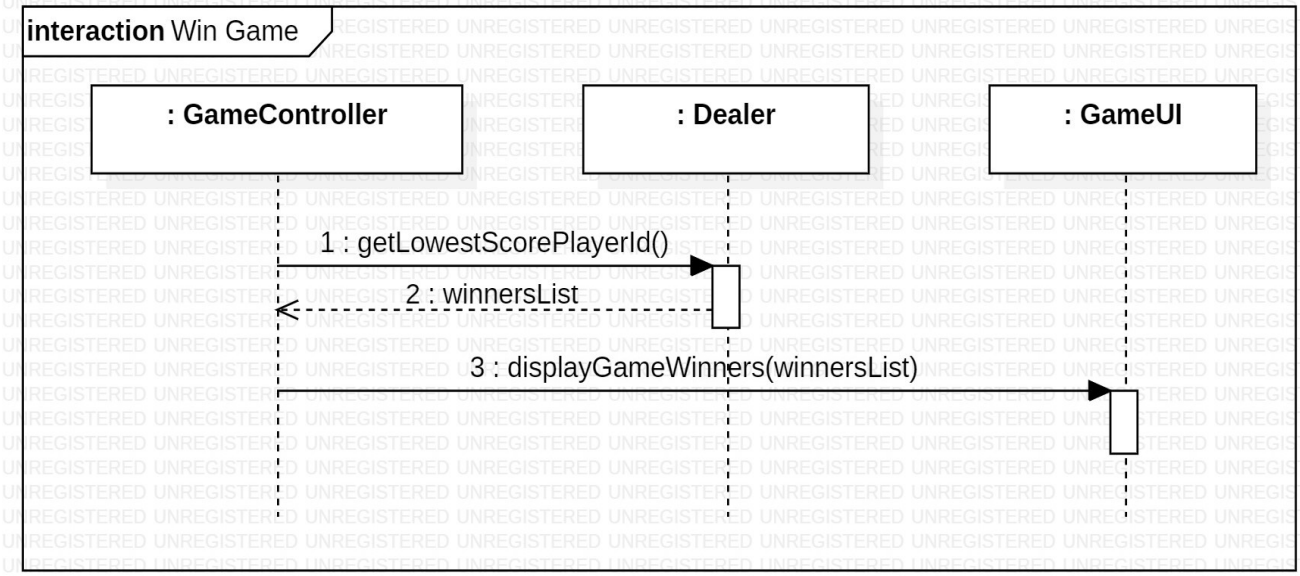
4.6. Use Case: Initiate Trick



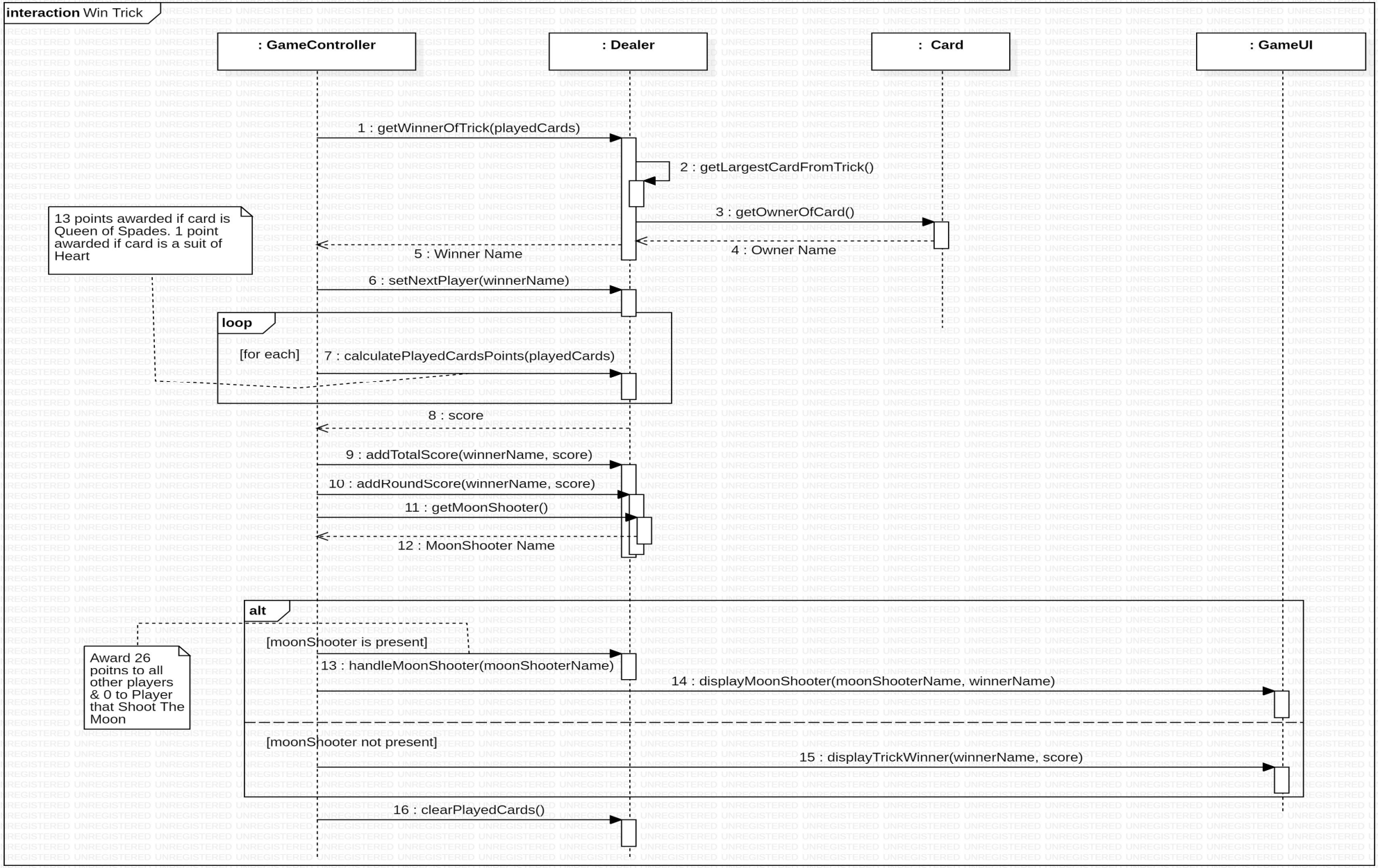
4.7. Use Case: Display Score



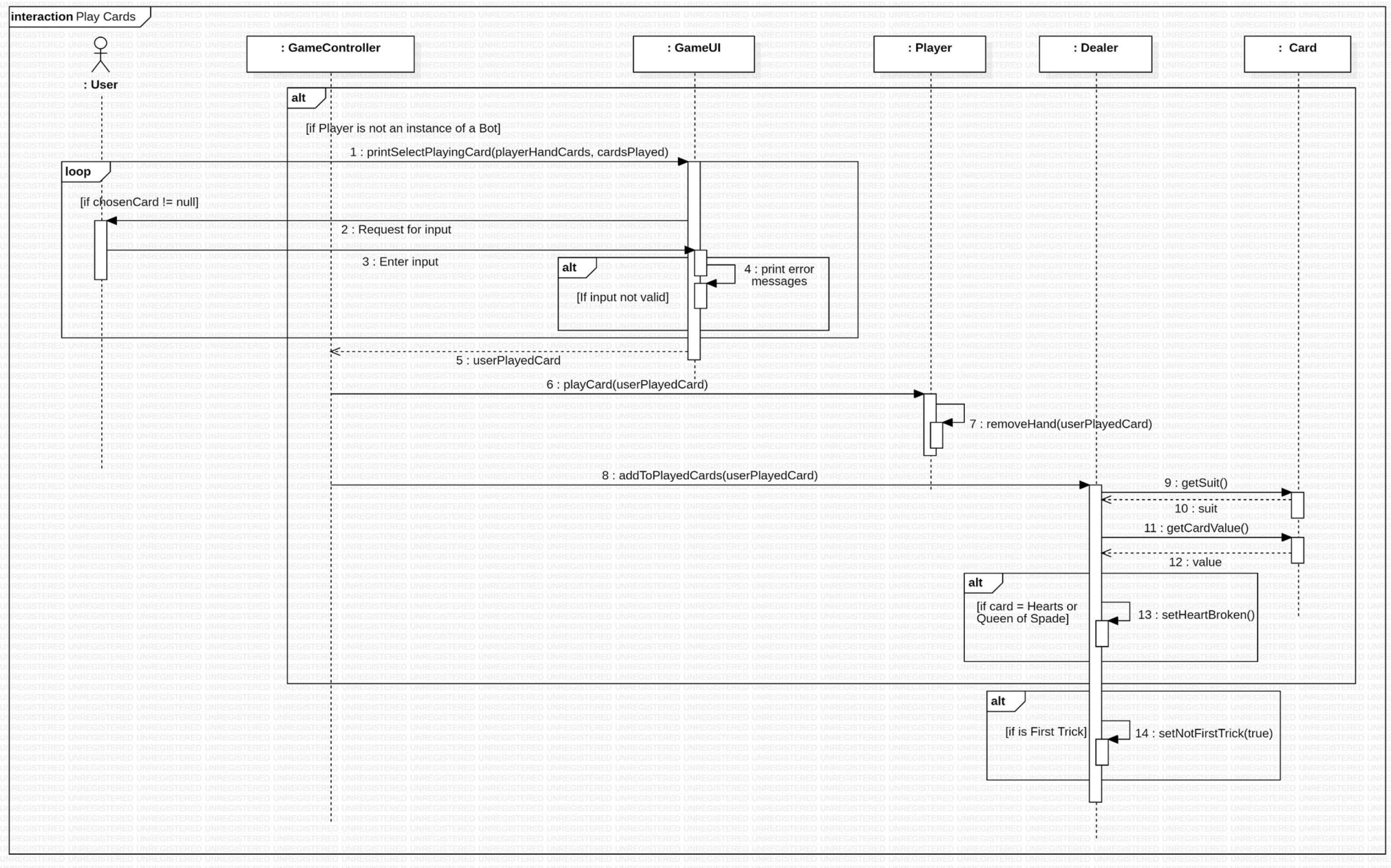
4.8. Use Case: Win Game



4.9. Use Case: Win Trick



4.10. Use Case: Play Cards



5. Test Cases

5.1. Testing of Bot Class

5.1.1. Objectives

a. Test the passing of top 3 cards
b. Test the passing sequence according to round
c. Test the throwing of 2 Clubs in first trick
d. Test if the card thrown follows the rules of either largest rank card or smallest rank card depending on the order.

5.1.2. Functions Tested

- playCard()
- passThreeCards()

Test for top 3 cards to pass: checkTop3Cards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for top 3 cards in hand	Hand of all spades	Ace, King ,Queen picked	Pass	passThreeCards()
TC2: Test for top 3 cards in mixed hand	Random hand	Three Clubs, Diamonds, Spades	Pass	passThreeCards()

Check bot pass each other cards: passedCards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for passing of 3 cards	A bot with hand all hearts and B bot with hand all spades. Resulting A bot should have spades in his hand and B, hearts	Pass	Pass	passThreeCards()
TC2: Test for passing of 3 cards in mixed hand	1 bot with 4 suits of TWOs and another with 4 suits of THREES	Pass Spades	Pass	passThreeCards()

Bot will throw 2 of clubs: throws2clubs				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for bot throw 2 clubs	Bot with random hand a 2 of clubs	Two of Clubs	Pass	playCard()

Bot will throw smallest rank after first trick: throwsSmallestRankTest				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for bot throws smallest rank	Bot with fixed hand with smallest rank being FOUR of CLUBS	FOUR of CLUBS	FOUR of CLUBS	playCard()
TC2: Test for bot throws smallest rank, hearts broken	Bot with fixed hand with smallest rank being THREE HEARTS	THREE HEARTS	THREE HEARTS	playCard()

Bot will throw second highest rank following a suit: playCard				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for second bot	Bot with SPADES QUEEN, TEN SPADES and thrown card is JACK SPADES.	TEN OF SPADES	Pass	playCard()
TC2: Test for third bot	Bot with THREE DIAMONDS, JACK DIAMONDS and thrown card is TEN DIAMONDS.	THREE OF DIAMONDS	Pass	playCard()

Bot will throw highest card of the suit if no other card: throwCardFollowSuitHigherThanCardsPlayed				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for bot throw highest and only card of suit	Bot has only 1 JACK DIAMONDS and suit is diamonds	JACK DIAMONDS	Pass	playCard()

Last bot will always throw largest rank card				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for last bot to throw highest following suit	Bot has fixed hand and highest rank card is KING SPADES. Suit is Spades	KING SPADES	Pass	playCard()
TC2: Test for last bot to throw highest when no suit	Bot has fixed hand, no clubs and highest rank card is KING SPADES. Suit is Clubs	KING SPADES	Pass	playCard()

Bot will throw highest rank if cant follow a suit: playCard				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for second bot	Bot has fixed hand and highest rank card is KING SPADES. Suit is Spades	KING SPADES	Pass	playCard()
TC2: Test for third bot	Bot has fixed hand, no clubs and highest rank card is KING SPADES. Suit is Clubs	KING SPADES	Pass	playCard()

5.2. Testing of Player Class

5.2.1. Objectives

a. Test that the Hand has only 13 cards after distribution
b. Test the removal of cards from passing and playing
c. Test the checking of 2 Clubs present in hand
d. Test the update of card pick-ability on leading suit, hearts broken or not broken, or all point cards
e. Test whether starting hand is all point cards

5.2.2. Functions Tested

- hasTwoOfClubs()
- getPickable()
- removeHand()
- getUserHand()
- getHandCards()
- getSmallestRankCards()
- getLargestRankCards()
- checkAllPointCards()

Check hand for 13 cards after distribution: checkHandHas13Cards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for hand size	New dealer and init round. Test for user's hand	13	Pass	getUserHandCards()
TC2: Test for Bot A size	New dealer and init round. Test for Bot A's hand	13	Pass	getUserHandCards()
TC3: Test for Bot B size	New dealer and init round. Test for Bot B's hand	13	Pass	getUserHandCards()
TC4: Test for Bot C size	New dealer and init round. Test for Bot C's hand	13	Pass	getUserHandCards()

Test hand size after removing a card: checkHandAfterRemovingACard				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for hand size	Init hand and remove a card	12	Pass	removeHand()
TC2: Test for hand size	Init hand and remove top 3 cards	10	Pass	removeHand()

Check whether 2 of clubs present in hand: checkfor2Clubs				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: In a hand full of clubs	A clubs hand	TRUE	Pass	hasTwoOfClubs()
TC2: In a hand full of spades	A spades hand	FALSE	Pass	hasTwoOfClubs()

Check whether card is pickable: getPickable				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: First round when player has 2 of clubs	Random hand and a 2 of clubs	All cards pickability false, only 2 clubs true	Pass	getPickable()
TC2: Check pickability for leading suit and hearts not broken	Random hand	All false only for leading diamonds are true	Pass	getPickable()
TC3: Check pickability for no leading suits and hearts broken	Random hand	All True	Pass	getPickable()
TC4: Check pickability for no leading suits and hearts not broken	Random hand	All True	Pass	getPickable()
TC5: Check pickability for starting trick with hearts broken	Random hand	All True	Pass	getPickable()
TC6: Check pickability for starting trick with hearts not broken	Random hand	All True, except Queen Spades and Hearts	Pass	getPickable()
TC7: Check pickability for following the first trick with no leading suit	Random hand	All True, except Queen Spades and Hearts	Pass	getPickable()
TC8: Check pickability when hand is 12 hearts and 1 Queen Spades	12 Hearts + 1 Queen Spades	Any cards playable	Pass	getPickable()
TC9: Check pickability when hand is 13 hearts	13 Hearts	Any hearts playable	Pass	getPickable()

Check whether card is pickable: getSmallestRankTest()				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Get smallest rank from random hand	Random hand	TWO DIAMONDS	TWO DIAMONDS	getSmallestRankCard()

Check whether card is pickable: getLargestRankTest()				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Get largest rank from random hand	Random hand	QUEEN SPADES	QUEEN SPADES	getLargestRankCard()

Check whether card is removed: testRemoveCard				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Remove a card	Random hand	QUEEN SPADES	QUEEN SPADES	removeHand()
TC2: Remove a list of cards	Random hand	3 Cards	Pass	removeHand()

Check whether starting hand is all point cards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Check for all point cards	13 Hearts test	Pass	Pass	checkAllPointCards()
TC2: Check for all point cards	12 Hearts and a Queen Spades	Pass	Pass	checkAllPointCards()

5.3. Testing of Card Class

5.3.1. Objectives

a. Test if card is 2 of Clubs
b. Test whether card is a point card (Queen Spades or any Heart)
c. Test the update of card pick-ability on leading suit, hearts broken or not broken, or all point cards
d. Test whether starting hand is all point cards

5.3.2. Functions Tested

- isHearts()
- isTwoOfClubs()
- isQueenOfSpades()
- getCardValue()

Test for card value: cardValueTest				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test card value of 2 cards	TWO of SPADES and ACE of CLUBS	26 and 12	Pass	isTwoOfClubs()

Test for 2 of clubs: cardIsTwoOfClubsTest				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for different rank and different suit	Check THREE OF SPADES	FALSE	Pass	isTwoOfClubs()
TC2: Test for different rank and same suit	Random hand	FALSE	Pass	isTwoOfClubs()
TC3: Same rank and same suit	Random hand	TRUE	Pass	isTwoOfClubs()
TC4: Same rank and different suit	Random hand	FALSE	Pass	isTwoOfClubs()

Test for card is hearts: cardIsHearts				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test card of clubs	ACE of CLUBS	FALSE	Pass	isHearts()
TC2: Test card of hearts	ACE of HEARTS	TRUE	Pass	isHearts()

Test for Queen of Spades: cardIsQueenOfSpades				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for different rank and different suit	Check TWO of CLUBS	FALSE	Pass	isQueenOfSpades()
TC2: Test for different rank and same suit	Check TWO of SPADES	FALSE	Pass	isQueenOfSpades()
TC3: Same rank and same suit	Check QUEEN of SPADES	TRUE	Pass	isQueenOfSpades()
TC4: Same rank and different suit	Check QUEEN of CLUBS	FALSE	Pass	isQueenOfSpades()

Test for Get Card Value				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for different rank and different suit	Check TWO of SPADES	0	0	getCardValue()
TC2: Test for highest card in deck ace hearts	Check ACE of HEARTS	51	51	getCardValue()
TC3: Test for card in deck 2 SPADES	Check TWO of SPADES	26	26	getCardValue()

5.4. Testing of Dealer Class

5.4.1. Objectives

a. Test whether dealer deals exactly 13 cards
b. Test if the right player starts the trick and follows a clockwise sequence of playing
c. Test for first player
d. Test if the trick points are calculated correctly
e. Test if the cards are passed in the right sequence based on rounds (Left, Right, Opposite, None)
f. Test “MoonShooter” case handling
g. Test the recording of total round score and getting the lowest score of player when the game ends

5.4.2. Functions Tested

- getUserHandCards()
- getLargestCardFromTrick()
- getTrickPlayer()
- getNextPlayer()
- getWinnerOfTrick()
- calculatePlayedCardsPoints()
- getMoonShooter()
- handleMoonShooter()
- getLowestScorePlayerId ()

Test for 13 cards: testUserHandDistributed				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for distributed hand	Create dealer object from init_dealer() and create new round new_round()	User hand should get 13	Pass	getUserHandCards()
TC2: Test for Bot A distributed hand	Create dealer object from init_dealer() and create new round new_round()	Bot A hand should get 13	Pass	getUserHandCards()
TC3: Test for Bot B distributed hand	Create dealer object from init_dealer() and create new round new_round()	Bot B hand should get 13	Pass	getUserHandCards()
TC4: Test for Bot C distributed hand	Create dealer object from init_dealer() and create new round new_round()	Bot C hand should get 13	Pass	getUserHandCards()

Test largest card in trick: testForLargestCardInTrick (diamond leading suit)				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for largest card in trick	4 cards: TWO OF DIAMONDS, KING OF SPADES, ACE OF SPADES, FIVE OF DIAMONDS	Five of diamonds	Pass	getLargestCardFromTrick()
TC2: Test for largest card in trick, comparing rank	Return Ace diamonds as highest of diamond leading suit	Ace of diamonds	Pass	getLargestCardFromTrick()

Test next player: nextPlayer				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for next player	Sets player as first player and tests next trick player is indeed Bot A	Bot A	Pass	getTrickPlayer(), getNextPlayer()

Test next player clockwise of player who has 2 clubs: getTrickPlayer				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for Bot A	Bot A has 2 of clubs	Bot B	Pass	getTrickPlayer()
TC2: Test for Bot B	Bot B has 2 of clubs	Bot C	Pass	getTrickPlayer()
TC3: Test for Bot C	Bot C has 2 of clubs	User	Pass	getTrickPlayer()
TC4: Test for User	User has 2 of clubs	Bot A	Pass	getTrickPlayer()

Test first player who has 2 of clubs: testFirstPlayer				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for first player	3 bots and user with 4 sorted hands according to suit. User has 2 of clubs	User	Pass	getNextPlayer()

Test winner of trick: testWinnerOfTrick				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for winner of a trick	3 bots - BotA has KING CLUBS, B has FIVE CLUBS, C has FOUR SPADES First card being played is FOUR CLUBS and order of play is A->B->C	botA	Pass	getWinnerOfTrick()
TC2: Test for winner of a trick following leading suit	Leading suit is clubs and only Bot A has clubs	botA	Pass	getWinnerOfTrick()

Calculate points at end of whole round: calculatePoints				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: All hearts	Finishing hand of 13 hearts	13	Pass	calculatePlayedCardsPoints()
TC2: All hearts and queen spades	Random hand of 3 hearts, 1 queen spades and 9 random suits	16	Pass	calculatePlayedCardsPoints()

Check bot pass each other cards: passedCards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for passing order round = 1	3 bots and 1 users. Passed order is to the Left	Return passed card	Pass	distributePassedCards()
TC2: Test for passing order round = 2	3 bots and 1 users. Passed order is to the Right	Return passed card	Pass	distributePassedCards()
TC3: Test for passing order round = 3	3 bots and 1 users. Passed order is to the opposite	Return passed card	Pass	distributePassedCards()
TC4: Test for passing order round = 4	3 bots and 1 users. No passing	Return passed card	Pass	distributePassedCards()
TC5: Test for passing order round = 99	3 bots and 1 users. Passed order is to the opposite	Return passed card	Pass	distributePassedCards()

Test for moon shooter				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for moon shooter	User starts with 0 points and moon shooter. End with 0 points and all other bots get 26	User	Pass	getMoonShooter(), handleMoonShooter()

Test for getting lowest score player id				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for a single lowest score	Bot C has the lowest score	BotC	BotC	getLowestScorePlayerId()
TC2: Test for multiple lowest score	Bot C and B has the lowest score	BotC and BotB	BotC and BotB	getLowestScorePlayerId()

5.5. Testing of Deck Class

5.5.1. Objectives

a. Test if the deck has 52 cards
b. Test if each suit has 13 cards

5.5.2. Functions Tested

- getDeckCards ()
- shuffledCards ()

Test Deck has 52 cards: testNumberofCards				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for deck size	New deck object and get size	52	Pass	getDeckCards()

Test for getting lowest score player id				
Test Cases	**Input**	**Expected Result**	**Actual Result**	**Tested Functions**
TC1: Test for suits and cards	New deck object sort cards based on their suits into 4 hand, each has 13	13	Pass	shuffledCards

6. Additional Rules and Definitions

Apart from the rules specified in the project requirements document, we also accounted for cases which are not mentioned in the document. These cases are ones we feel we should account for as it can cause our program to either fail or not create logical results.

First, we define a round to consist of 13 tricks and one trick consist of 4 played cards. Hence, for each round, each player will play 13 tricks and empty their hands of 13 cards at the end of the round. This definition runs contrary to the project document which defines a round to be a trick. This is done for our own clarity when coding out the application.

Second, we added logic to allow for ties, in other words, there is more than 1 winner when a player exceeds 100 points. This happens when more than one player has the same minimum score.

Third, we added logic to handle the situation where a player happens to play the first trick or the player starts a trick other than the first when hearts is not broken with all point cards on hand. We needed to account for this as if a player has all point cards, the console application will crash since all cards are unpickable. Hence, our logic was that if a player has all point cards and is starting a trick without hearts broken or playing the first trick, the player can throw any of the point cards and break hearts on the spot. This is the **new rule** that we included in our application.