Functional Requirements Specification



## Use Cases

|  |  |
| --- | --- |
| UC1 | Enter Player Name |
| UC2 | Play Game |
| UC3 | Play Card |
| UC4 | Deal Cards |
| UC5 | Shuffle Deck |
| UC6 | Display Hand |
| UC7 | Sort Hand |
| UC8 | Display Scores |
| UC9 | Legal Move |
| UC10 | Get Leading Suit |
| UC11 | Suit In Hand |
| UC12 | Update Scores |
| UC13 | Winning Card |
| UC14 | Played Card |

### Use Case Flow of Events (Function Specification)

Using the Use Case flow-of-events format described by Quantrani (Quantrani, 1998) and Williams (Williams 2004).

#### UC1 Flow of Events for the Enter Player Name Use Case

* 1. **Preconditions:**

None.

* 1. **Main Flow:**

This use case begins as soon as the game starts. The Player Name dialogue box will be displayed, prompting the player to enter their name[E1].

* 1. **Subflows:**

None:

* 1. **Alternative Flows:**

[E1] The name must have a length of greater than zero. If a player enters an empty string, the game then asks the player to re-enter their name.

#### UC2 Flow of Events for the Play Game Use Case

* 1. **Preconditions:**

The Enter Player Name use case must execute before this use case begins.

* 1. **Main Flow:**

This use case begins as soon as the Player Name dialogue box disappears. The Start Game dialogue box will be displayed, prompting the player to enter the ip address and port number for the game server[E1] [S1].

* 1. **Subflows:**

[S1] After the player enters a valid ip address the cards will be dealt [UC4], the hands will be displayed to the players [UC6] and the scoresheet will be displayed[UC8].

* 1. **Alternative Flows:**

[E1] If the player enters an address which the game client is unable to connect to then the game will ask the player to enter the ip address and port number for an active server.

#### UC3 Flow of Events for the Play Card Use Case

* 1. **Preconditions:**

The Play Game use case and all use cases included in that use cases functionality (UC4, UC5, UC6, UC7 and UC8) must have executed.

It must be the player’s turn before this use case begins.

* 1. **Main Flow:**

The game is turn based. The player that goes first is selected at random. On the player’s turn they select a card to be played [E1] [S1]. The player’s turn is then over[S2].

* 1. **Subflows:**

[S1] After the player selects a card to be played, the card is checked to see if it is a legal move [UC9].

[S2] After a card has been legally played to the trick, if the player was the last player to take their turn then the scores must be updated [UC12].

* 1. **Alternative Flows:**

[E1] When playing a card, the player must follow suit if they can. A player cannot play a card from another suit if they are capable of following suit. If the player attempts to do so the game will ask the player to pick a card which follows suit.

#### UC4 Flow of Events for the Deal Cards Use Case

* 1. **Preconditions:**

The Play Game use case has been executed.

There are four players in the game.

* 1. **Main Flow:**

As soon as four players (human and AI) are connected to the game, the cards must be dealt to the players [S1].

* 1. **Subflows:**

[S1] After the deck of cards is created it must be shuffled [UC5].

* 1. **Alternative Flows:**

None.

#### UC5 Flow of Events for the Shuffle Cards Use Case

* 1. **Preconditions:**

The deck of cards has been created.

* 1. **Main Flow:**

Once the deck of cards is created, before it is dealt, it must be shuffled such that the order of the cards is random.

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

None.

#### UC6 Flow of Events for the Display Hand Use Case

* 1. **Preconditions:**

The Play Game use case has been executed.

There are four players in the game.

The cards have been dealt into four hands of equal size.

* 1. **Main Flow:**

Once the cards have been dealt into hands, each player’s hand must be displayed [S1] so that only they can see their own hand.

* 1. **Subflows:**

[S1] As the hands are displayed they must be done so in order of suit and rank [UC7].

* 1. **Alternative Flows:**

None.

#### UC7 Flow of Events for the Sort Hand Use Case

* 1. **Preconditions:**

The cards have been dealt into four hands of equal size.

* 1. **Main Flow:**

Once the cards have been dealt they will be completely unordered they must be sorted by suit and rank.

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

None.

#### UC8 Flow of Events for the Display Scores Use Case

* 1. **Preconditions:**

The Play Game use case has been executed.

There are four players in the game.

Each player has a hand of 13 cards.

* 1. **Main Flow:**

Before the gameplay can begin the scoreboard must be displayed with each player’s name and their score, set to zero.

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

None.

#### UC9 Flow of Events for the Legal Move Use Case

* 1. **Preconditions:**

It is the player’s turn.

The player has selected a card to be played.

* 1. **Main Flow:**

After the player chooses a card to be played, the card must be checked to see if it is a legal move to play[S1] [S2].

* 1. **Subflows:**

[S1] In order to determine whether or not the card can be played, the suit of the first card played in the trick must be checked [UC10].

[S2] Once the leading suit is known, if the card the player chose to play is not of that suit then the players hand must be checked for any cards of that suit [UC11].

* 1. **Alternative Flows:**

None.

#### UC10 Flow of Events for the Get Leading Suit Use Case

* 1. **Preconditions:**

None.

* 1. **Main Flow:**

When a played card is being checked if it is a legal move, the suit of the first card played must be known [E1].

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

[E1] If no cards have been played in this round then there will not be a leading suit, therefore the card the player chose to play must be legal.

#### UC11 Flow of Events for the Suit In Hand Use Case

* 1. **Preconditions:**

At least one card has already been played this round.

* 1. **Main Flow:**

When a played card does not follow suit, the players hand is checked to see if it contains any cards of the leading suit[E1].

* 1. **Subflows:**
  2. **Alternative Flows:**

[E1] If the card being played is the last card of a player’s hand then it is a legal move.

#### UC12 Flow of Events for the Update Scores Use Case

* 1. **Preconditions:**

The last player has played a card and the trick now contains four cards.

* 1. **Main Flow:**

After the fourth card has been played to the current trick, the scores must be updated by adding a point to the player who played the winning cards score [S1] [S2].

* 1. **Subflows:**

[S1] Before the points can be updated the winning card must be selected (as per the rules of the game).

[S2] Once the winning card is selected, the player that played the card must be known.

* 1. **Alternative Flows:**

None.

#### UC13 Flow of Events for the Winning Card Use Case

* 1. **Preconditions:**

The last player has played a card and the trick now contains four cards.

* 1. **Main Flow:**

When the points are being allocated at the end of a round, the winning card must be selected as per the rules of the game. The winning card is the highest rank card of the leading suit unless a trump suit card is played in which case the winner is the highest rank trump suit card.

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

None.

#### UC14 Flow of Events for the Played Card Use Case

* 1. **Preconditions:**

The last player has played a card and the trick now contains four cards.

The winning card has been selected.

* 1. **Main Flow:**

Once the winning card has been selected from the current trick, the player who played the winning card must be found.

* 1. **Subflows:**

None.

* 1. **Alternative Flows:**

None.

## Non-functional Requirements

### Performance

The system shall wait for all user inputs and execute any necessary functions based upon those inputs:

* User response – The system must be responsive, in particular the AI players must not take longer that a human player would to take a turn.
* Setup time – the game should generate quickly and reset the hands and update the scoresheet quickly.

### Usability

A user shall quickly be able to determine what options they have to perform:

* The system will let the user interact with it via the mouse on cards. The user keyboard input is restricted to the initial setup, once the game has started all of the interface will be through the mouse.
* The way that the user interacts with the system will be obvious, through use of standardised format for card games.

Design Documentation

# Design Documentation

## Class Diagram



Testing Documentation

# Testing

## Testing Strategy

### Unit Testing

The unit testing in this project will be done using junit testing within the Eclipse IDE. The unit testing will be used to confirm that the individual methods in the classes work correctly.

### Integration Testing

Following the unit testing, the integration testing will be done using the Functional Incremental approach. This testing approach involves existing classes and methods being integrated into functions; these provide the functionality specified in the functional requirements specification.

I chose to take the functional incremental approach for the following reasons:

* The fact that this approach is incremental allows me to test during the development of the software.
* Starting at the lowest level functionality and continuing incrementally means that the critical function will be built and tested first, therefore any error or mistakes in these functions are found early in the process.

## Testing Plan

The following test plan is for the deal cards function.

The deal cards function uses the following class methods:

* getCard() from the Deck class
* getHand() from four separate Player classes
* addCard() from four separate Hand classes (encapsulated within the Player class)

Using test cases, I would prove that the methods work without error.

#### getCard() Unit Test Pseudocode

Create card object for the Ace of Spades

Create deck object

Assert that getCard(51) is equal to the card object.

The suits are ordered Hearts, Clubs, Diamonds and Spades last in the Suit Enumeration (see Class Diagram). The cards would be in rank order of 2 to Ace, therefore the Ace of Spades should be the last card in the deck. If the getCard() method works correctly, getCard(51) should return the Ace of Spades because the ace of spades would be the 52nd card in the deck.

#### getHand() Unit Test Pseudocode

Create a hand of specific cards

Create a player object with that hand

Create a list of cards with the same cards in the same order

Assert that player.getHand() is equal to the list of cards.

If the getHand() method works correctly then it should return the hand of the player, which is implemented as a List of cards and comparable to the other List of cards.

#### addCard() Unit Test Pseudocode

Create a card object

Create an empty hand object

Use hand.addCard() to add the card object to the empty hand

Assert that hand.getCard(0) is equal to the card object.

If addCard()works correctly then it will add the new card to the end of the hand (list), in this example the hand is empty before addCard() is used so the card will be at index 0.

This unit test relies on the getCard() method also working correctly, it will have been tested earlier on in the development stage that this unit test.

#### deal() Test Case Pseudocode

The deal() function will require the four player objects and the deck to be passed as parameters. As per the functional requirements, this function includes the use of the shuffle function which is lower level functionality and would have already been tested.

The following pseudocode shows how the deal() function will use the methods tested above:

Create a list of type Player

Add the four Player objects to the list

Shuffle the deck

Create an index variable with value zero

For each card in the deck

get the player at index variable, get the players hand with getHand(), add the card with addCard()

increment the index variable

modulo the index variable by four.

This function will deal the cards evenly between the four players. Each Player object should have a hand of 13 cards and each card should only appear once.

To test this function, I will first assert that all four Player objects contain a hand of length 13. Then I will assert that each hand is unique, I will do this by checking each card in one hand against every card in the other three hands.