Programmer’s Manual

By

Jerry Parng
Benjamin Nix
Ryan Mecir

Requirements Documentation

1. Introduction

1.1 Scope Statement

For our group project, we have decided to create a card game called War. The general idea of the game involves each player holding a deck of a set amount of cards. Each player will draw and flip over their top card. Whoever's card has the higher value, that player will take all those cards and add it to their "spoils" pile. If there is a tie, then one card will be drawn and placed face down. The next card drawn will decide who wins all the cards. If a player runs out of cards, but has cards won from previous "battles", the cards will be shuffled and put into the player's deck. The winner is decided when only one player has all the cards.

This game will work in Windows and will act as a standalone game that will not require an internet connection. The game will likely make use of images to display the cards and a class to randomize the cards collected when the initial deck runs out of cards. We will also need to use some frameworks– such as Swing and AWT for Java– to create a graphical user interface for the game. The game will support at least 2 players, the user and a computer. The programming language that we will use will be Java. The game will also use a graphical interface to show the cards in play. Sound effects will be played for certain in game actions such as shuffling, winning a game, or losing.

1.2 Definitions, acronyms, and abbreviations

1.2.0 **JDK™:** Java™ Platform, Standard Edition Development Kit

1.2.1 **GUI** : Graphical User Interface

1.2.2 **JAR**: Java archive file

1.2.3 **Bg**: Background

1.3 References

1.3.0 Wikimedia Foundation. (2022, November 22). *War (card game)*. Wikipedia. From https://en.wikipedia.org/wiki/War_(card_game)

1.3.1 *Java SE 19 archive downloads*. Java Archive Downloads - Java SE 19. (n.d.). From https://www.oracle.com/java/technologies/javase/jdk19-archive-downloads.html

2. General Description

2.1. Product Perspective

This game was developed to enjoy an old children’s game with simple rules, but a match can take anywhere from a few minutes to more than 10. It is a fun way to pass the time.

2.2. Product Functions

When starting the game, 52 cards are distributed equally to two players. Each player will draw a card and the player whose card has the higher value will take both cards. In the event of a tie, one card will be taken from each player’s deck and added to the tie pool as well as the card that tied. The next card drawn will determine who wins all the cards in the tie pool. The winner will be whoever collects all the cards.

2.3. User Characteristics

The user will be anyone who wants to play a game and pass the time.

2.4. General Constraints

The game will not run if the user has an older version of Java. The game has not been tested on Linux or any other operating systems so far.

2.5. Assumptions and Dependencies

The game requires the latest version of Java (JDK 19 from Oracle at the time of development) to be installed on the end user’s computer. The game will assume that the user is running Windows 10.

3. Requirements

Playing Cards and Displays

1.0.0 The game will have 52 playing cards sorted into 2 decks of 26 cards, with no jokers.

1.1.0 Non-face cards (2 - 10) will have their respective numerical values.

- 1.1.1 Non-face cards will have specific numerical values. (Ace = 1, Jack = 11, Queen, 12, and King = 13)

1.2.0 When the game is loaded, a title, a “Start Game” button, and a text box with instructions on how to play the game will appear.

1.3.0 Clicking on the “Start Game” button will change the display to the layout of the game.

- 1.3.1 Each player will have an active deck of 26 cards, a spoils deck zone, and an active card zone.
- 1.3.2 A “Tie pool” will be displayed in the event of a tie.
- 1.3.3 Each round will have a note based on if the player wins the round, the computer wins the round, or if there is a tie.
- 1.3.4 There will be 2 active card zones, one for the player and one for the computer.
- 1.3.5 A Draw button will be used to draw the cards.

1.4.0 Each player’s deck and spoils deck will display the image of the back of a card.

- 1.4.1 Cards placed in the active card zone will display the image of that specific card.

1.5.0 Spoils decks from both players will be linked lists holding the card values.

1.6.0 A Tie pool linked list will hold cards that are tied, a card taken from both players' decks.

- 1.6.1 Repeat **Requirement 1.6.0** in the event of another tie.

1.7.0 If the Player runs out of cards, then the player will reach a "You Lose" screen.

1.8.0 If the Computer runs out of cards then the player will reach a "You Win" screen.

1.9.0 A reset button will be located on the top right corner of the game and can be used to restart the game.

1.10.0 A title screen will appear when the game is opened.

- 1.10.1 The screen will show the instructions of how the game will be played, and underneath will display a "Start Game" button.

Game Mechanics

2.0.0 Clicking the “Start Game” button will initialize a full 52-card deck and sort the cards into 2 decks of 26 cards.

2.1.0 Clicking the “Draw” button will take the top card in each deck and place them into the active card zone in the middle.

2.2.0 Cards that are placed in the active card zone are compared.

- 2.2.1 The card with the higher value will win, and both cards will be moved to the victor’s spoils deck
- 2.2.2 When a tie occurs, the cards that tie as well an additional card from each player’s deck will be placed into the Tie pool.
- 2.2.3 The next draw will decide who wins the extra cards in the Tie pool, but if another tie occurs, repeat **Requirement 2.2.2.**

2.3.0 If either the Player or Computer run out of cards in their respective decks, then the winner will be whoever still has cards.

- 2.3.1 If the decks of either the Player or Computer run out, but they still have cards available in their spoils decks, then those cards will be used to replenish their decks.

2.4.0 Resetting the game will clear all decks, spoils decks and tie pool values as well as return to the main title screen.

Sound Effects

3.0.0 Initialize sound effects and music, and prepare to play them when the game is opened.

3.1.0 When the game is first loaded, music will be played until the “Start Game” button is clicked.

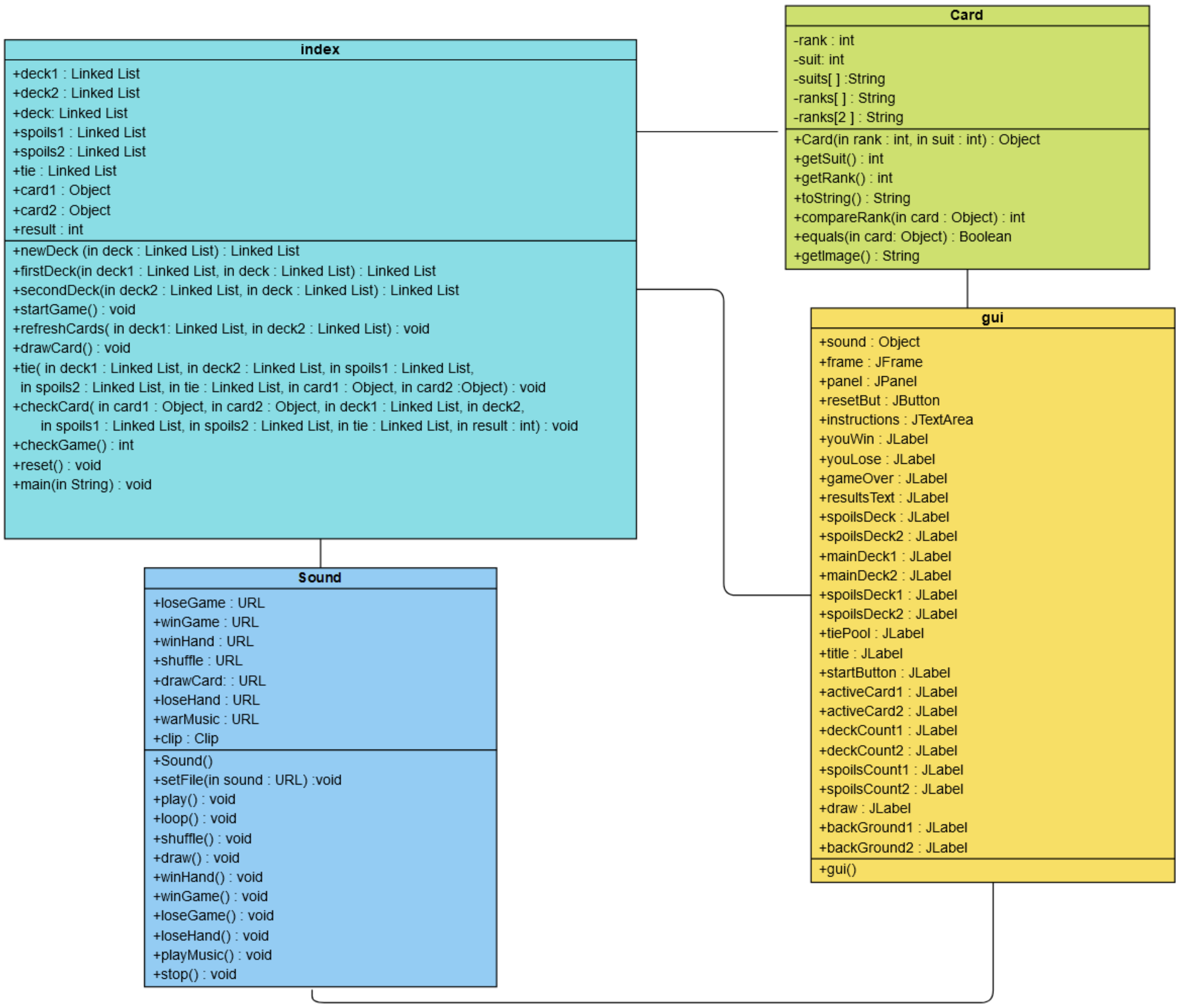
3.2.0 A shuffle sound effect will play after “Start Game” is clicked to let the user know that the decks have been created and shuffled.

3.3.0 Clicking “Draw” will create a “draw card” sound effect

- 3.3.1 If the user’s card value is higher than the computer’s card value, a “winning” sound effect will play.
- 3.3.2 If the user’s card value is lower than the computer’s card value, a “losing” sound effect will play.
- 3.3.3 If the “Draw” button is pressed when the user has no more cards, a “game lost” sound effect will play.
- 3.3.4 If the “Draw” button is pressed when the computer has no more cards, a “game win” sound effect will play.

2. Design Documentation

2.1 UML Diagram



2.2 Pseudocode

<u>Case: Player Starts Up Application</u>

[index.java]
Call upon the gui method from gui.java:

[gui.java]

```
Create a new sound “sound”
Create a new JFrame “frame”, Make frame not resizable, default to 1280x720

Create a new JPanel “panel”, set it’s bounds to 1280x720
Create a button “resetBut” with text “Reset”, not visible, at bounds (115,11,91,35)
New JTextArea “instructions”, w/ a white foreground foreground and dark green background
Instructions cannot be edited.
Set instructions font to bold 18pt Arial, with word wrap and line wrap, with tab size at 12.
Set instructions with the desired text, set bounds to (324,219,642,306)
Add instructions to panel

New JLabel “youWin” with desired text
Set youWin’s color to white, 48pt bold Arial, set its bounds to the center
Add youWin to panel and set its visibility to false

New JLabel “youLose” with desired text, repeat the subsequent lines from youWin

New JLabel “gameOver” with text as an empty string
Set gameOver’s icon with the resource “bg.png”, set bounds to (0,1,1270,690)
Add gameOver to panel, set visibility to false

New JLabel “resultsText” with desired text
Set resultText’s foreground color to white and the background color to a lighter gray
Set resultText’s text to 25pt bold Arial, and set its bounds at (90,23,1031,135)
Add resultsText to panel and set its visibility to false

New JLabel “spoilsDeck2”, image uses resourced “Card-Back-Spoils.jpg”
Set spoilsDeck2’s bounds to (1121,242,121,189)
Add spoilsDeck2 to panel and set it’s visibility to false
```

New JLabel “spoilsDeck1”, everything's the same as spoilsDeck2, but has a centered horizontal alignment and has bounds set at (42,242,123,189).

New JLabel “mainDeck2”, set with text “ “ and uses resource “Card-Back-Resized.jpg” as its icon
Set mainDeck2’s bounds to (958,219,165,239)
Add mainDeck2 to panel and set visibility to false

New JLabel “tiePool”, set with text “ “ and centered horizontal alignment
Set tiePool’s foreground color to white, its font to 20pt bold Arial, and bounds to (585,582,103,47)
Add tiePool to panel and set visibility to false

New JLabel “mainDeck1”, everything’s the same as mainDeck2, but its bounds are instead (175,219,156,230)

New JLabel “title” as “Game of War!”
Set title’s foreground color and background color as white, it’s horizontal alignment centered, its font at 50pt bold Arial and its bounds as (419,79,422,50)
Add title and resetBut to panel

Set the title of frame as “Card Game of War”, set its layout to null and add panel into the content pane

New JButton “firstButton” that reads “Start Game”
firstButton’s text is in 14pt plain Arial, and firstButton’s bounds are (545,566,165,47)
Add firstButton to panel

New JLabel “activeCard1”, uses resource “Card-Back-Active.jpg” as its icon
activeCard1’s bounds are (370,169,231,329)
Add activeCard1 to panel and set its visibility to false

New JLabel “activeCard2”, identical to functions regarding activeCard1, but instead setting its bounds to (680,169,231,329)

New JLabel “deckCount1”
Set deckCount1’s foreground color to white, its font as 24pt bold Arial and its bounds to (185,460,123,47)
Add deckCount1 to panel and set visibility to false

New JLabel “deckCount2”, identical to functions regarding deckCount1, but instead setting its bounds to (968, 469, 137, 52)

New JLabel “spoilsCount1”, which displays “Spoils: “ plus the size of spoils1
Set spoilsCount1’s color to black, its font to 20pt bold Tahoma, and set its bounds to (52,437,97,47)
Add spoilsCount1 to panel and set visibility to false

New JLabel “spoilsCount2”, identical to functions regarding spoilsCount1, but instead setting its bounds to (1131, 442, 111, 42)

New JButton “draw”, which displays the text “Draw”
Set bounds for draw as (575,524,123,47)
Add draw to panel and set visibility to false

New JLabel “backGround1”, using resource “bg.png” as Icon
Set backGround1’s bounds as (0,1,1270,690)
Add backGround1 to panel

New JLabel “backGround2”, identical to functions regarding backGround1, but instead setting its bounds to (0,0,1270,691)
Set backGround2’s visibility to false

Add action listener for the draw button (pseudocode for said listener will be shown in the “Game Has Started, Player has pressed Draw Button” case)

Set frame so it will exit on close
Set frame’s visibility to true

Add action listener for firstButton (pseudocode for said listener will be shown in the “Player Presses the ‘Start Game’ Button” case)

Add action listener for resetBut (pseudocode for said listener will be shown in the “Player Presses the Reset Button” case)

[index.java Part 2]
Call the playMusic method from sound.java

Case: Player Presses the "Start Game" Button

[gui.java]

Start a game by calling the method "startGame" from index.java:

[index.java]

```
Try newDeck to initialize a 52 card deck "deck":
    For every suit
        For every value
            Add the corresponding card to the linkedlist.
Once done, shuffle and return deck.
Use firstDeck and secondDeck to split deck into two new decks "deck1" and "     "deck2":
    For [half of deck size] amount of times
        Add a card into deck 1 (same for deck 2's method)
Add remaining ace and king cards to each deck
Stop any sounds
Catch any errors with printSackTrace

Output "Deck: " plus the deck
Output "deck1: " plus deck1, along with "deck1 count: " + deck1 size
Output "deck2: " plus deck2, along with "deck2 count: " + deck2 size
```

[gui.java Part 2]

Set the visibility of the following to ***true:*** draw, resetBut, backGround2, mainDeck1, mainDeck2, spoilsDeck1, spoilsDeck2, activeCard1, activeCard2, deckCount1, deckCount2, spoilsCount1, spoilsCount2, tiePool

Set the visibility of the following to ***false:*** instructions, firstButton, title, backGround1, youLose, youWin, gameOver and resultsText

Set the text for deckCount1 to "Deck: " plus the size of deck1. Do the same for deckCount2 and the size of deck2

Set the text for spoilsCount1 as "Spoils: " plus the size of spoils1. Do the same for spoilsCount2 and the size of spoils2

Set tiePool's text to "Tie Pool: " plus the size of tie.

Play the sound assigned to "shuffle", called from Sound.java

Case: Game Has Started, Player has pressed Draw Button

[gui.java]

Call the checkGame method from index.java:

[index.java]

```
If deck1 and spoils1 are empty,
        Play the sound assigned to “loseGame”
        Return 1
Else if deck2 and spoils2 are empty,
        Play the sound assigned to “winGame”
        Return 2
else return 0
```

[gui.java Part 2]

```
If checkGame is 0,
        Try to call the method “drawCard” from index.java:
```

[index.java Part 2]

Call upon the refreshCards method for deck1 and 2:
If deck1 is empty but spoils1 is not, add each of the spoils1 cards to deck1, clear spoils1 and shuffle deck1
Do the same for deck2 if it meets the same conditions

If deck1 and deck2 are not empty, pop a card from each (card1 and card2 respectively)
Call the “checkCard” method for card1, card2, deck1, deck2, spoils1, spoils2, tie and result:

Variable “outcome” is the result of using card1 against card2 in the “compareRank” method from Card.java (returns the first card’s rank minus the second card’s rank)

```
If outcome is greater than 0,
        Play the sound assigned to winHand
        Add card1 and card2 to spoils 1.
        Additionally, if the tie size is greater than 0, add each card from the tie pool into spoils1
        and clear tie.
        Print out “Player 1 wins! P1’s card total increased to “ plus the size of spoils1.

If outcome is less than 0,
        Add card1 and card2 to spoils 2.
        Additionally, if the tie size is greater than 0, add each card from the tie pool into spoils2
        and clear tie.
        Print out “Player 2 wins! P2;s card total increased to “ plus the size of spoils2.

Else,
        Print out “Tie has occurred, one extra card has been added to spoils. draw again”
        Call method “tie” for deck1, deck2, spoils1, spoils2, tie, card1, card2 (method pops a
        card from deck1 and 2 and adds them to tie if neither of them are null. If either of them
        are null, only pop the other deck and print “Player 1/CPU rand out of cards”)

Print out “card1: “ with card1, as well as “card2: “ with card2
```

[gui.java Part 3]

```
Else if checkGame returns 1,
        Set the visibility for gameOver and youLose to true, and draw, backGround1 and 2,
        mainDeck1 and 2, spoilsDeck1 and 2, activeCard 1 and 2, deckCount 1 and 2, spoilsCount
        1 and 2, tiePool and resultsText to false
Else if checkGame returns 2,
        Set the visibility for gameOver and youWin to true, and draw, backGround1 and 2,
        mainDeck1 and 2, spoilsDeck1 and 2, activeCard 1 and 2, deckCount 1 and 2, spoilsCount
        1 and 2, tiePool and resultsText to false

Call compareRank for card1 and card2.
If the result returns 0,
        Set resultsText as "Tie has occurred. Tied card + 1 more card placed into the tie pool.
        Draw again!" and set its visibility to true
Else if the returned result is greater than 0
        Set resultsText as "Player 1 wins! Card total increased to " plus the size of spoils1,
         and set its visibility to true
Else if the returned result is less than 0
        Set resultsText as "Player 2 wins! Card total increased to " plus the size of spoils2,
         and set its visibility to true
Else, set resultsText visibility to false

Play the sound assigned to draw

If deck1 is empty, set the visibility for mainDeck1 to false, else set it to true.
```

If deck2 is empty, set the visibility for mainDeck2 to false, else set it to true.
If spoils1 is not empty, set spoilsDeck1's visibility to true, else set it to false.
If spoils2 is not empty, set spoilsDeck2's visibility to true, else set it to false.
Try to set activeCard1's icon to a string version of card1's image url. Any errors should be caught with a printStackTrace exception. Do the same for activeCard2 with card2.

Set the text for deckCount1 and deckCount2 as "Deck: " plus the size of deck1 and deck2 respectively
Set the text for spoilsCount1 and spoilsCount2 as "Spoils: " plus the size of spoils1 and spoils2 respectively
Set the text for tiePool as "Tie Pool: " plus the size of tie.

Set the visibility of activeCard1 and activeCard2 to true.

<u>Case: Player Presses the Reset Button</u>
[gui.java]
Call the reset method from index.java:

[index.java]
Clear out deck, deck1, deck2, spoils1, spoils2 and tie
Call the method playMusic from sound.java

Print out "deck1: " along with deck1 to the system
Print out "deck2: " along with deck2 to the system
Print out "full deck: " along with the size of deck to the system

[gui.java Part 2]
Set the visibility of the following to ***true***: instructions, firstButton, title, backGround1
Set the visibility of the following to ***false:*** draw, resetBut, backGround2, mainDeck1, mainDeck2, spoilsDeck1, spoilsDeck2, activeCard1, activeCard2, deckCount1, deckCount2, spoilsCount1, spoilsCount2, tiePool, youLose, youWin, gameOver and resultsText
setEnabled set to true for draw

2.3 Decision Table

Case	**Condition**	**Computer's card > Player's card**	**Computer's card < Player's card**	**Computer's card == Player's card**
1	Player draws an "Ace".	Player loses the round. (**Requirement 2.2.1)**		Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
2	Player draws a "2"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
3	Player draws a "3"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
4	Player draws a "4"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
5	Player draws a "5"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
6	Player draws a "6"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
7	Player draws a "7"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
8	Player draws an "8"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
9	Player draws a "9"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
10	Player draws a "10"	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**

11	Player draws a “Jack”	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
12	Player draws a “Queen”	Player loses the round. (**Requirement 2.2.1)**	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
13	Player draws a “King”	▨	Player wins the round.(**Requirement 2.2.1)**	Tie Occurs. **(Requirements 2.2.2 and 2.2.3)**
14	Player has no cards in the deck, but has cards in their spoils deck.	Player loses the round. **(Requirement 2.3.1)**	Player wins the round. **(Requirement 2.3.1)**	Tie Occurs. **(Requirements 2.2.2, 2.2.3, and 2.3.1)**
15	Computer has no cards in the deck, but has cards in their spoils deck.	Computer loses the round. **(Requirement 2.3.1)**	Computer wins the round. **(Requirement 2.3.1)**	Tie Occurs. **(Requirements 2.2.2, 2.2.3, and 2.3.1)**
16	Player has no cards left.	Player cannot draw and loses the game. **(Requirement 2.3.0)**	Player cannot draw and loses the game. **(Requirement 2.3.0)**	Player cannot draw and loses the game. **(Requirement 2.3.0)**
17	Computer has no cards left.	Computer cannot draw and loses the game. **(Requirement 2.3.0)**	Computer cannot draw and loses the game. **(Requirement 2.3.0)**	Computer cannot draw and loses the game. **(Requirement 2.3.0)**

3. Source Code Documentation

3.1 Index Java File :

```
/**
  * User defined package containing all game assets.
 * /
package WarGamePack;

import java.io.IOException;
import java.util.Collections;
import java.util.LinkedList;

public class index extends javax.swing.JFrame {

   /**
    *
    */
   private static final long serialVersionUID = 1L;

   public static LinkedList<Card> deck1 = new LinkedList<Card>();
   public static LinkedList<Card> deck2 = new LinkedList<Card>();
   public static LinkedList<Card> deck = new LinkedList<Card>();
   public static LinkedList<Card> spoils1 = new LinkedList<Card>();
   public static LinkedList<Card> spoils2 = new LinkedList<Card>();
   public static LinkedList<Card> tie = new LinkedList<Card>();
   public static Card card1;
   public static Card card2;
   public static int result;

   //Sound.java file (Requirement 3.0.0)
   static Sound sound = new Sound();


   // Initializes the full deck of 52 cards. (Requirement 1.0.0)

           public static LinkedList<Card> newDeck(LinkedList<Card> deck) throws IOException {
           // 4 suits
           for(int i = 0; i < 4; i++) {
                      //13 ranks
                      for(int j = 1; j <= 13; j++) {
                                 deck.add(new Card(j, i));
```

```
                }
        }
        Collections.shuffle(deck);
         return deck;
        }



//Splits deck into 26 cards for Player 1 (Requirement 1.3.1)
public static LinkedList<Card> firstDeck(LinkedList<Card> deck1, LinkedList<Card> fullDeck){
        //takes 1st half of deck and adds to the Player 1's deck
        for(int i = 0; i < fullDeck.size() / 2; i++) {
                deck1.add(fullDeck.get(i));
        }
        return deck1;
}
//Splits deck into 26 cards for Player 2 (Requirement 1.3.1)
public static LinkedList<Card> secondDeck(LinkedList<Card> deck2, LinkedList<Card> fullDeck) {
        //takes  2nd half of deck and adds to Player 2's deck
        for(int i = fullDeck.size() - 1; i >= fullDeck.size() / 2; --i) {
                deck2.add(fullDeck.get(i));
        }
        return deck2;
}




public static void startGame() {

        try {
        //initializes 52 card deck and the splits into two separate decks (Requirements 1.0.0 and 1.3.1)
                newDeck(deck);
                firstDeck(deck1, deck);
                secondDeck(deck2, deck);

        //Sound will stop playing when Start Game button is clicked (Requirement 3.1.0)
                sound.stop();

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        System.out.println("Deck: " + deck);

        System.out.println("deck1: " + deck1);
        System.out.println("deck1 count: " + deck1.size());

        System.out.println("deck2: " + deck2);
        System.out.println("deck2 count: " + deck2.size());



    }
```

/**If either active decks are empty, add all cards from the spoils deck to the active decks (**Requirement**
* **2.3.1**)
*/

```
    public static void refreshCards(LinkedList<Card> deck1, LinkedList<Card> deck2) {
        if(deck1.size() == 0 && spoils1.size() > 0) {
            for(int i = 0; i < spoils1.size(); i++) {
                deck1.add(spoils1.get(i));
            }
            spoils1.clear();
            Collections.shuffle(deck1);
        }
        if(deck2.size() == 0 && spoils2.size() > 0) {
            for(int i = 0; i < spoils2.size(); i++) {
                deck2.add(spoils2.get(i));
            }
            spoils2.clear();
            Collections.shuffle(deck2);
        }
    }
```

// Checks if active decks are empty and if spoils decks have cards (**Requirement 2.3.1**). If active decks have cards, then one card is taken from each deck (**Requirement 2.1.0**)

```
    public static void drawCard() throws IOException {
        refreshCards(deck1, deck2);
        if(deck1.size() > 0 && deck2.size() > 0) {
            card1 = deck1.pop();
            card2 = deck2.pop();
```

```
                checkCard(card1, card2, deck1, deck2, spoils1, spoils2, tie, result);
        }
        System.out.println("card1: " + card1);
        System.out.println("card2: " + card2 );


    }
    /*In the event of a tie, take  one card from both decks and place them into the tie pool (Requirement
 * 2.2.2)
 */
    public static void tie( LinkedList<Card> deck1, LinkedList<Card>deck2, LinkedList<Card> spoils1,
            LinkedList<Card> spoils2, LinkedList<Card> tie, Card c1, Card c2) {

        refreshCards(deck1, deck2);
        if(deck1.size() > 0 && deck2.size() > 0) {
                Card temp = deck1.pop();
                Card temp2 = deck2.pop();
                tie.add(c1);
                tie.add(c2);
                tie.add(temp);
                tie.add(temp2);

        } else if(deck1.size() == 0) {
                Card temp = deck2.pop();
                tie.add(c2);
                tie.add(temp);
                System.out.println("Player 1 ran out of cards.");
        }
        else if(deck2.size() == 0) {
                Card temp = deck1.pop();
                tie.add(c1);
                tie.add(temp);
                System.out.println("CPU ran out of cards.");
        }


        System.out.println("Tie deck: " + tie);


    }

    // Evaluates cards and determines which card has higher value (Requirement 2.2.1)
```

```
public static void checkCard(Card card1, Card card2,LinkedList<Card> deck1, LinkedList<Card> deck2,
        LinkedList<Card> spoils1, LinkedList<Card> spoils2, LinkedList<Card> tie, int outcome) {


        outcome = card1.compareRank(card2); //in Card.java

        if (outcome > 0) {
                sound.winHand();
                spoils1.add(card1);
                spoils1.add(card2);

                if(tie.size() > 0) {
                        for(int i = 0; i <= tie.size() - 1; i++) {
                                spoils1.add(tie.get(i));
                                }

                tie.clear();


                }
                System.out.println("Player 1 wins! " + "Card total increased to " + spoils1.size());
        } else if (outcome < 0) {
                sound.loseHand();
                spoils2.add(card1);
                spoils2.add(card2);

                if(tie.size() > 0) {
                        for(int i = 0; i <= tie.size() - 1; i++) {
                                spoils2.add(tie.get(i));
                                }

                tie.clear();

                }
                System.out.println("Player 2 wins! " + "Card total increased to " + spoils2.size());
        }else {
               System.out.println("Tie has occurred, one extra card has been added to spoils. draw
               again");

                tie(deck1, deck2, spoils1, spoils2,tie, card1, card2);
        }
}
```

```
//Checks the active decks and spoils decks if they have run out (Requirement 2.3.0)
  public static int checkGame() {
        //Player 1 loses
        if(deck1.size() == 0 && spoils1.size() == 0) {
                sound.loseGame();
                return 1;
        //Player 1 wins
        }else if(deck2.size() == 0 && spoils2.size() == 0) {
                sound.winGame();
                return 2;
        }
        return 0;
  }

 //Resets all values and returns to the main title screen (Requirement 2.4.0)
  public static void reset() {
        //Clears the cards from the deck
        deck.clear();
        deck1.clear();
        deck2.clear();
        spoils1.clear();
        spoils2.clear();
        sound.playMusic();
        tie.clear();

        System.out.println("deck1: " + deck1);
        System.out.println("deck2: " + deck2);
        System.out.println("full deck: " + deck.size());

  }




  public static void main(String args[]) {
        new gui();
        sound.playMusic();

  }
}
```

3.2 Card Java File :

```
package WarGamePack;


import java.io.IOException;




public class Card{
            private int rank;
            private int suit;
            String cardImage;

            private static String[] suits = {"clubs", "diamonds", "hearts", "spades"};
            public static final int CLUBS        = 0;
            public static final int DIAMONDS = 1;
            public static final int HEARTS   = 2;
            public static final int SPADES   = 3;

            private static String[] ranks = {null, "ace", "2", "3", "4", "5", "6", "7", "8",
                                    "9", "10", "jack", "queen", "king"};
            public static final int JACK  = 11;
            public static final int QUEEN = 12;
            public static final int KING  = 13;
            public static final int ACE   = 1;

   /** Constructor
     * rank can be 2, 3, ..., 10, JACK, QUEEN, KING, ACE
     * suit can be CLUBS, DIAMONDS, HEARTS, SPADES (Requirement 1.1.0)
    * @throws IOException
     */
            public Card (int rank, int suit) throws IOException {
            if (rank < ACE || rank > KING)
            throw new IllegalArgumentException("invalid rank: " + rank);
            if (suit < 0 || suit > 3)
            throw new IllegalArgumentException("invalid suit: " + suit);
            this.rank = rank;
            this.suit = suit;
```

```
        }

  /** Returns the suit of the card:
   *  CLUBS, DIAMONDS, HEARTS, SPADES (Requirement 1.1.0 and 1.1.1)
   */
        public int getSuit() {
        return this.suit;
        }

  /** Returns the rank of the card:
   *   2, 3, ..., 10, JACK, QUEEN, KING, ACE  (Requirement 1.1.1)
   */
        public int getRank() {
        return this.rank;
        }

  /** Returns a string representation of the card  (Requirement 1.1.0 and 1.1.1)
  */
        public String toString() {
        return ranks[rank] + " of " + suits[suit];
        }

  /** Returns a negative number if this object is lower in rank than c,
   *  0 if the cards are equal rank, and a positive number if this object
   *  is higher in rank than c. Aces are considered 'low'. (Requirement 2.2.1)
   */
        public int compareRank(Card c) {
        return rank - c.rank;
        }



// Gets the rank and suit of cards in the active card zone and returns it as a string (Requirement 1.4.0)
        public String getImage() throws IOException {
                cardImage = "/Cards/" + ranks[rank] + "_of_" + suits[suit] + ".png";
                return cardImage;
        }


  }
```

***3.3 Sound Java File**:*

```
package WarGamePack;

import java.net.URL;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.FloatControl;


public class Sound {

    URL loseGame;
    URL winGame;
    URL winHand;
    URL shuffle;
    URL drawCard;
    URL loseHand;
    URL warMusic;
    Clip clip;


    public Sound() {

            winHand = getClass().getResource("/Sound_Effects/winHand.wav");
            winGame = getClass().getResource("/Sound_Effects/GameWon.wav");
            shuffle = getClass().getResource("/Sound_Effects/shuffle.wav");
            drawCard = getClass().getResource("/Sound_Effects/card.wav");
            loseHand = getClass().getResource("/Sound_Effects/loseHand.wav");
            warMusic = getClass().getResource("/Sound_Effects/warMusic.wav");
            loseGame = getClass().getResource("/Sound_Effects/GameLose.wav");

    }

//Finds the URL sound file and will prepare the sound effect ( Requirement 3.0.0)
    public void setFile(URL s) {

                    try {
                            AudioInputStream sound = AudioSystem.getAudioInputStream(s);
```

```
                        clip = AudioSystem.getClip();
                        clip.open(sound);
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }
//Plays the sound effect (Requirement 3.0.0)
        public void play() {

                clip.start();
        }
 //Loops the sound effect (Requirement 3.1.0)
        public void loop() {
                clip.loop(Clip.LOOP_CONTINUOUSLY);
        }
 //Plays the shuffle sound effect when a new game is played (Requirement 3.2.0)
        public void shuffle(){
                setFile(shuffle);
                play();
        }
//Plays the draw sound effect when the draw button is clicked (Requirement 3.3.0)
        public void draw() {
                setFile(drawCard);
                play();
        }
//Plays the winHand sound effect when the player wins a round (Requirement 3.3.1)
        public void winHand() {
                setFile(winHand);
                FloatControl gainControl = (FloatControl)
               clip.getControl(FloatControl.Type.MASTER_GAIN);
                gainControl.setValue(-10.0f);
                play();
        }
//Plays the winGame sound effect when the player wins the game (Requirement 3.3.4)
        public void winGame() {
                setFile(winGame);
                FloatControl gainControl = (FloatControl)
               clip.getControl(FloatControl.Type.MASTER_GAIN);
                gainControl.setValue(-10.0f);
                play();
        }
        //Plays the loseGame sound effect when the player loses the game (Requirement 3.3.3)
        public void loseGame() {
                setFile(loseGame);
                FloatControl gainControl = (FloatControl)
               clip.getControl(FloatControl.Type.MASTER_GAIN);
                gainControl.setValue(-10.0f);
                play();
        }

    //Plays the loseHand sound effect when the player loses the round (Requirement 3.3.2)
        public void loseHand() {
                setFile(loseHand);
                FloatControl gainControl = (FloatControl)
               clip.getControl(FloatControl.Type.MASTER_GAIN);
                gainControl.setValue(-10.0f);
                play();
        }
    //Plays music during the title screen of the game (Requirement 3.1.0)
        public void playMusic() {
                setFile(warMusic);
                FloatControl gainControl = (FloatControl)
               clip.getControl(FloatControl.Type.MASTER_GAIN);
                gainControl.setValue(-10.0f);
                play();
                loop();
        }
    //Stops the sound effect or music (Requirement 3.1.0)
        public void stop() {
                clip.stop();
        }

    }
```

3.4 Gui Java File :

```
package WarGamePack;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

import javax.swing.*;
import javax.swing.border.BevelBorder;



public class gui {

   public gui() {
//Sound java file (Requirement 3.0.0)
          Sound sound = new Sound();

//Initialize the frame for the game (Requirement 1.10.0)
          JFrame frame = new JFrame();
          frame.setResizable(false);
          frame.setBounds(0, 0, 1280, 720);


 //Initialize panel and panel layout (Requirement 1.10.0)
          JPanel panel = new JPanel();
          panel.setBounds(0, 0, 1280, 720);
          JButton resetBut = new JButton("Reset");
          resetBut.setVisible(false);
          resetBut.setBounds(1151, 11, 91, 35);
          panel.setLayout(null);
 //Initializes the instructions (Requirement 1.10.1)
          JTextArea instructions = new JTextArea();
          instructions.setForeground(new Color(255, 255, 255));
          instructions.setBackground(new Color(0, 128, 64));
          instructions.setEditable(false);
          instructions.setFont(new Font("Arial", Font.BOLD, 18));
          instructions.setWrapStyleWord(true);
          instructions.setLineWrap(true);
```

```
        instructions.setTabSize(12);
        instructions.setText("  Welcome to the game of War! In this game, each player will start
       with 26 cards in their respective decks. Click the draw button to start each battle by
       drawing 1 card. The player with a card value that is higher (ACE - KING, with ACE being
       the lowest and KING being the highest) will take both cards and place them into their
       spoils deck. \r\n\r\n  In the event of a tie, the tied card and 1 extra card will be moved to
       the Tie Pool, and the next card drawn will determine who wins all of the cards. In case of
       multiple ties, the same steps will be repeated until one side wins the round.\r\n\r\n
       The winner is declared when one player has no more cards left. If a tie occurs and one
       player has no more cards to wager, they will lose. ");
        instructions.setBounds(324, 219, 642, 306);
        panel.add(instructions);

//When the player wins, this screen will appear. (Requirement 2.3.0)
        JLabel youWin = new JLabel("Player 1 Wins! Game Over!");
        youWin.setForeground(new Color(255, 255, 255));
        youWin.setFont(new Font("Arial", Font.BOLD, 48));
        youWin.setBounds(314, 276, 735, 173);
        panel.add(youWin);
        youWin.setVisible(false);

  //When the player loses, this screen will appear. (Requirement 2.3.0)
        JLabel youLose = new JLabel("CPU Wins! Game Over!");
        youLose.setForeground(new Color(255, 255, 255));
        youLose.setFont(new Font("Arial", Font.BOLD, 48));
        youLose.setBounds(370, 276, 636, 173);
        panel.add(youLose);
        youLose.setVisible(false);

//This background image will appear when a winner is declared. (Requirement 2.3.0)
        JLabel gameOver = new JLabel("");
        gameOver.setIcon(new ImageIcon(gui.class.getResource("/Backgrounds/bg.png")));
        gameOver.setBounds(0, 1, 1270, 690);
        panel.add(gameOver);
        gameOver.setVisible(false);
```

//This text will appear when a tie occurs (**Requirement 2.2.2**)

```
        JLabel resultsText = new JLabel("Tie has occurred. Tied cards and 1 more card are placed
        in the tie pool. Draw again!");
         resultsText.setForeground(new Color(255, 255, 255));
         resultsText.setBackground(new Color(240, 240, 240));
         resultsText.setFont(new Font("Arial", Font.BOLD, 25));
         resultsText.setBounds(90, 23, 1031, 135);
         panel.add(resultsText);
         resultsText.setVisible(false);
```

/*Displays the total number of cards in the spoils deck for the Computer (**Requirement 1.5.0**)

```
         JLabel spoilsDeck2 = new JLabel("");
         spoilsDeck2.setIcon(new
         ImageIcon(gui.class.getResource("/Cards/Card-Back-Spoils.jpg")));
         spoilsDeck2.setBounds(1121, 242, 121, 189);
         panel.add(spoilsDeck2);
         spoilsDeck2.setVisible(false);
```

//Displays the total number of cards in the spoils deck for the Player (**Requirement 1.5.0**)

```
         JLabel spoilsDeck1 = new JLabel("");
         spoilsDeck1.setHorizontalAlignment(SwingConstants.CENTER);
         spoilsDeck1.setIcon(new
        ImageIcon(gui.class.getResource("/Cards/Card-Back-Spoils.jpg")));
         spoilsDeck1.setBounds(42, 242, 123, 189);
         panel.add(spoilsDeck1);
         spoilsDeck1.setVisible(false);
```

//Displays image for the Computer's deck (**Requirement 1.4.0**)

```
         JLabel mainDeck2 = new JLabel("");
         mainDeck2.setIcon(new
        ImageIcon(gui.class.getResource("/Cards/Card-Back-Resized.jpg")));
         mainDeck2.setBounds(958, 219, 165, 239);
         panel.add(mainDeck2);
         mainDeck2.setVisible(false);
```

//Displays the Tie pool (**Requirement 1.3.2**)

```
         JLabel tiePool = new JLabel("Tie Pool: ");
```

```
            tiePool.setHorizontalAlignment(SwingConstants.CENTER);
            tiePool.setForeground(new Color(255, 255, 255));
            tiePool.setFont(new Font("Arial", Font.BOLD, 20));
            tiePool.setBounds(585, 582, 103, 47);
            panel.add(tiePool);
            tiePool.setVisible(false);

//Displays the player's deck (Requirement 1.4.0)
            JLabel mainDeck1 = new JLabel("");
            mainDeck1.setIcon(new
           ImageIcon(gui.class.getResource("/Cards/Card-Back-Resized.jpg")));
            mainDeck1.setBounds(175, 219, 156, 230);
            panel.add(mainDeck1);
            mainDeck1.setVisible(false);

//Displays the title screen name (Requirement 1.10.0)
            JLabel title = new JLabel("Game of War!");
            title.setForeground(new Color(255, 255, 255));
            title.setBackground(new Color(255, 255, 255));
            title.setHorizontalAlignment(SwingConstants.CENTER);
            title.setFont(new Font("Arial", Font.BOLD, 50));
            title.setBounds(419, 79, 422, 59);
            panel.add(title);
            panel.add(resetBut);


  //Displays frame title for the game (Requirement 1.10.0)
            frame.setTitle("Card Game of War");
            frame.getContentPane().setLayout(null);
            frame.getContentPane().add(panel);


  //Displays the Start Game button in the title screen (Requirement 1.2.0)
            JButton startButton = new JButton("Start Game");
            startButton.setFont(new Font("Arial", Font.PLAIN, 14));
            startButton.setBounds(545, 566, 165, 47);
            panel.add(startButton);
```

```
//Displays active card for the player (Requirement 1.3.4)
        JLabel activeCard1 = new JLabel("");
        activeCard1.setIcon(new
       ImageIcon(gui.class.getResource("/Cards/Card-Back-Active.jpg")));
        activeCard1.setBounds(370, 169, 231, 329);
        panel.add(activeCard1);
        activeCard1.setVisible(false);

 //Displays active card for the computer (Requirement 1.3.4)
        JLabel activeCard2 = new JLabel("");
        activeCard2.setIcon(new
       ImageIcon(gui.class.getResource("/Cards/Card-Back-Active.jpg")));
        activeCard2.setBounds(680, 169, 231, 329);
        panel.add(activeCard2);
        activeCard2.setVisible(false);

 //Displays the deck size for the player (Requirement 1.3.1)
        JLabel deckCount1 = new JLabel("");
        deckCount1.setForeground(new Color(255, 255, 255));
        deckCount1.setFont(new Font("Tahoma", Font.BOLD, 24));
        deckCount1.setBounds(185, 460, 123, 47);
        panel.add(deckCount1);
        deckCount1.setVisible(false);

 //Displays the deck size for the computer (Requirement 1.3.1)
        JLabel deckCount2 = new JLabel("");
        deckCount2.setForeground(new Color(255, 255, 255));
        deckCount2.setFont(new Font("Tahoma", Font.BOLD, 24));
        deckCount2.setBounds(968, 469, 137, 52);
        panel.add(deckCount2);
        deckCount2.setVisible(false);

 //Displays the spoils deck size for the player (Requirement 1.3.1)
        JLabel spoilsCount1 = new JLabel("Spoils: " + index.spoils1.size());
        spoilsCount1.setForeground(new Color(255, 255, 255));
        spoilsCount1.setFont(new Font("Tahoma", Font.BOLD, 20));
        spoilsCount1.setBounds(52, 437, 123, 47);
```

```
            panel.add(spoilsCount1);
            spoilsCount1.setVisible(false);

    //Displays the spoils deck for the computer (Requirement 1.3.1)
            JLabel spoilsCount2 = new JLabel("Spoils: " + index.spoils2.size());
            spoilsCount2.setForeground(new Color(255, 255, 255));
            spoilsCount2.setFont(new Font("Tahoma", Font.BOLD, 20));
            spoilsCount2.setBounds(1131, 442, 111, 42);
            panel.add(spoilsCount2);
            spoilsCount2.setVisible(false);

    //Displays the Draw button (Requirement 1.3.5)
            JButton draw = new JButton("Draw");
            draw.setBounds(575, 524, 123, 47);
            draw.setVisible(false);
            panel.add(draw);

  //Displays the title screen background (Requirement 1.2.0)
            JLabel backGround1 = new JLabel("");
            backGround1.setIcon(new ImageIcon(gui.class.getResource("/Backgrounds/bg.png")));
            backGround1.setBounds(0, 1, 1270, 690);
            panel.add(backGround1);

  //Displays the background for when the player is playing the game. (Requirement 1.3.0)
            JLabel backGround2 = new JLabel("");
            backGround2.setIcon(new
           ImageIcon(gui.class.getResource("/Backgrounds/bg_1play.jpg")));
            backGround2.setBounds(0, 0, 1270, 691);
            panel.add(backGround2);
            backGround2.setVisible(false);

     // Clicking the draw button will communicate with index.java, Card.java, and Sound.java
    // (Requirement 1.3.5)
            draw.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            //draws the cards
                            if(index.checkGame() == 0) {
                                    try {
```

```
                index.drawCard();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
    } else if(index.checkGame() == 1){

            gameOver.setVisible(true);
            youLose.setVisible(true);
            draw.setVisible(false);
            backGround1.setVisible(false);
            backGround2.setVisible(false);
            mainDeck1.setVisible(false);
            mainDeck2.setVisible(false);
            spoilsDeck1.setVisible(false);
            spoilsDeck2.setVisible(false);
            activeCard1.setVisible(false);
            activeCard2.setVisible(false);
            deckCount1.setVisible(false);
            deckCount2.setVisible(false);
            spoilsCount1.setVisible(false);
            spoilsCount2.setVisible(false);
            tiePool.setVisible(false);
            resultsText.setVisible(false);

    }else if(index.checkGame() == 2) {

            gameOver.setVisible(true);
            youWin.setVisible(true);
            draw.setVisible(false);
            backGround1.setVisible(false);
            backGround2.setVisible(false);
            mainDeck1.setVisible(false);
            mainDeck2.setVisible(false);
            spoilsDeck1.setVisible(false);
            spoilsDeck2.setVisible(false);
            activeCard1.setVisible(false);
            activeCard2.setVisible(false);
```

```
			deckCount1.setVisible(false);
			deckCount2.setVisible(false);
			spoilsCount1.setVisible(false);
			spoilsCount2.setVisible(false);
			tiePool.setVisible(false);
			resultsText.setVisible(false);
		}
```

//Compares the cards (**Requirement 2.2.0**)

```
		if(index.card1.compareRank(index.card2) == 0) {
			resultsText.setText("Tie has occurred. Tied card + 1 more card
			placed into the tie pool. Draw again!");
			resultsText.setVisible(true);
		}else if(index.card1.compareRank(index.card2) > 0) {
			resultsText.setText("Player 1 wins! " + "P1's Card total increased
			to " + index.spoils1.size());
			resultsText.setVisible(true);
		}else if(index.card1.compareRank(index.card2) < 0 ) {
			resultsText.setText("Player 2 wins! " + "CPU's Card total increased
			to " + index.spoils2.size());
			resultsText.setVisible(true);
		}
		else {
			resultsText.setVisible(false);
		}
```

//Plays the draw sound effect (**Requirement 3.3.0**)

```
		sound.draw();
```

//Checks for card visibility (**Requirement 1.4.0**)

```
		if(index.deck1.size() == 0) {
			mainDeck1.setVisible(false);
		}else {
			mainDeck1.setVisible(true);
		}
		if(index.deck2.size() == 0) {
			mainDeck2.setVisible(false);
		} else {
			mainDeck2.setVisible(true);
		}
```

```
if(index.spoils1.size() > 0 ) {
        spoilsDeck1.setVisible(true);
}else if(index.spoils1.size() == 0) {
        spoilsDeck1.setVisible(false);
}
if (index.spoils2.size() > 0) {
        spoilsDeck2.setVisible(true);
}else if(index.spoils2.size() == 0) {
        spoilsDeck2.setVisible(false);
}
```

//Displays the new card every draw (**Requirement 1.4.1**)

```
try {
        activeCard1.setIcon(new
       ImageIcon(gui.class.getResource(index.card1.getImage().toString()
       )));
} catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
}

try {
        activeCard2.setIcon(new
       ImageIcon(gui.class.getResource(index.card2.getImage().toString()
       )));
} catch (IOException e2) {
        e2.printStackTrace();
}

deckCount1.setText("Deck: " + index.deck1.size());
deckCount2.setText("Deck: " + index.deck2.size());
spoilsCount1.setText("Spoils: " + index.spoils1.size());
spoilsCount2.setText("Spoils: " + index.spoils2.size());
tiePool.setText("Tie Pool: " + index.tie.size());
activeCard1.setVisible(true);
activeCard2.setVisible(true);
```

```
			}
		});
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);


 //if start game button is clicked, hide buttons and create second layout with decks and a draw *
//button (Requirement 1.3.0)
		startButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				index.startGame();
				instructions.setVisible(false);
				startButton.setVisible(false);
				title.setVisible(false);
				draw.setVisible(true);
				resetBut.setVisible(true);
				backGround1.setVisible(false);
				backGround2.setVisible(true);
				mainDeck1.setVisible(true);
				mainDeck2.setVisible(true);
				deckCount1.setVisible(true);
				deckCount2.setVisible(true);
				spoilsCount1.setVisible(true);
				spoilsCount2.setVisible(true);
				tiePool.setVisible(true);
				youLose.setVisible(false);
				youWin.setVisible(false);
				gameOver.setVisible(false);
				deckCount1.setText("Deck: " + index.deck1.size());
				deckCount2.setText("Deck: " + index.deck2.size());
				spoilsCount1.setText("Spoils: " + index.spoils1.size());
				spoilsCount2.setText("Spoils: " + index.spoils2.size());
				tiePool.setText("Tie Pool: " + index.tie.size());

//Plays the shuffle sound effect (Requirement 3.2.0)
				sound.shuffle();

			}
```

```
        });

//Resets all buttons and title(Requirement 1.9.0)
        resetBut.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        index.reset();
                        instructions.setVisible(true);
                        startButton.setVisible(true);
                        title.setVisible(true);
                        draw.setVisible(false);
                        resetBut.setVisible(false);
                        backGround1.setVisible(true);
                        backGround2.setVisible(false);
                        mainDeck1.setVisible(false);
                        mainDeck2.setVisible(false);
                        spoilsDeck1.setVisible(false);
                        spoilsDeck2.setVisible(false);
                        activeCard1.setVisible(false);
                        activeCard2.setVisible(false);
                        deckCount1.setVisible(false);
                        deckCount2.setVisible(false);
                        spoilsCount1.setVisible(false);
                        spoilsCount2.setVisible(false);
                        tiePool.setVisible(false);
                        youLose.setVisible(false);
                        youWin.setVisible(false);
                        gameOver.setVisible(false);
                        draw.setEnabled(true);
                        resultsText.setVisible(false);
                }
        });

   }
}
```

4. Testing Documentation

Test Case #	Requirement Tested	Rationale	Input(s)	Expected output	Pass/Fail
1	**2.1.0**	When clicking the draw button, the active card image should be refreshed.	User clicks “Draw” button.	Image refreshes.	Pass
2	**1.3.1**	When clicking the “Start Game” button, the gui image should update with the card fields and player titles.	User clicks “Start Game” button.	GUI updates with labels and spaces for spoils decks, main decks, and active cards for both players.	Pass
3	**2.0.0 and 3.1.0**	When clicking the “Start Game” button, the shuffle sound effect should play and decks should be initialized.	User clicks “Start Game” button.	“Shuffle” sound is played when GUI displays the game with decks for both players.	Pass
4	**1.4.0**	When cards are in the players’ decks or spoils decks, the “Card-back” image displays.	User clicks the “Start Game” button. User clicks the draw button.	“Card-back” image is displayed when cards are in those decks.	Pass
5	**1.2.0**	When the game is loaded, a title, instructions, and start game button should appear.	User clicks the JAR file to launch the game.	Game launches and displays a game screen with title, an instructions box with text, and a “Start Game” button.	Pass

6	**3.2.0**	When clicking the draw button, the draw card sound should play.	User clicks draw button.	“Draw card” sound is played.	Pass
7	**3.0.0**	When the game loads, music should loop until the user clicks the start game button.	User launches the game but does not press the start game button.	Game launches and music loops.	Pass
8	**1.4.0**	When clicking the draw button when the active deck has one card left will cause the card back image to be removed.	User’s deck has only one card left and clicks the draw button.	Deck image of the back of the card will disappear.	Pass
9	**1.9.0**	When clicking the reset button at any time, the game will reset.	User clicks on the reset button.	Game will return to the title screen and all values will be cleared.	Pass
10	**2.2.2**	When a Tie occurs, User is notified that a card has been taken from the deck and told to draw again.	User and Computer cause a tie.	Both are notified via text with tied cards and one card from each deck being placed in tie pool.	Pass
11	**1.7.0**	When the User runs out of cards completely, they will reach a “Game Over” screen with a losing sound effect.	User runs out of cards and clicks the “Draw” button.	User reaches “Game Over” screen with “losing” sound effect.	Pass
12	**1.8.0**	When the Computer runs out of cards, the User will reach the “You	Computer has no	User reaches the “You Win”	Pass

		win” screen with a winning sound effect.	more cards and the User clicks the “draw” button.	screen with “winning” sound effect.	
12	**1.4.1**	Cards drawn from the deck will be placed in active card zone and the image corresponding to that card’s value will be displayed	User clicks “Draw” button to pop the card from the deck.	Cards popped off the deck will have its image displayed in active card zone.	Pass
13	**2.2.1**	Cards are compared, and the winner is determined by the higher card value.	User draws a “5” and Computer draws a “10”.	Computer wins the round. A sound effect for losing will play and both cards will be moved to Computer’s spoils deck.	Pass
14	**2.3.1**	When either players’ decks have run out and their spoils decks still have cards, those cards will be used to replenish the players’ decks.	User draws a card when their deck has run out when their spoils deck has cards.	Cards from the User’s spoils deck will be moved to the User’s deck.	Pass
15	**1.2.0**	When the JAR file is executed, the game will load and a title screen will display with music.	JAR file is executed.	Title screen will display with music playing.	Pass

5. Bugs and Issues

Currently, there are no glaring issues at the moment. Something that can be improved upon in the future could be to update how the sound effects have more time to complete before the player continues the next round for each draw. The timing for the sound effects for each win and loss can also be improved upon as well, but are not completely detrimental to the game.