KOÇ ÜNİVERSİTESİ
COMPUTER ENGINEERING

COMP132: Advanced Programming
Programming Project Report

UNO Card Game

Ozan Emre ARIKAN, 83993

Spring 2024

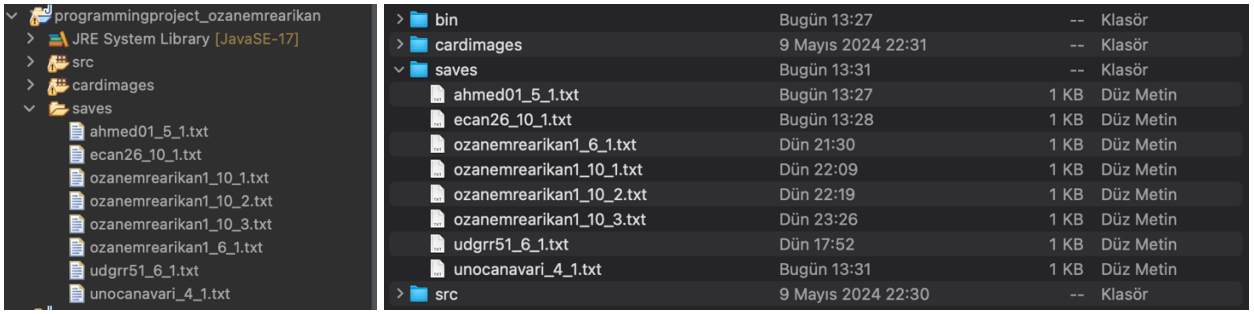
Part 1 – User and Player Guide

General Demo Information

List of users and their information:

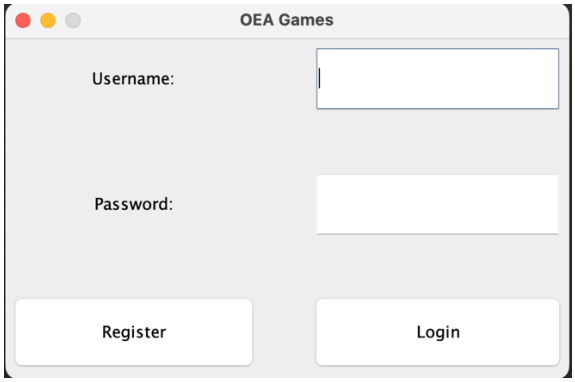
Username	Password	Name	Surname	Email	Age
ozanemrearikan1	ozanemre1+	Ozan Emre	Arıkan	ozanemrearikan1@gmail.com	21
ahmed01	ahmed01+	Ahmet	Hakan	ahmedhakan@outlook.com	49
unocanavari	YigitM1903.	Yiğit	Maşaoğlu	ymasaoglu04@gmail.com	19
ecan26	2626123456e.	Enes	Can	enesca26@gmail.com	21
udgrr51	12345umut.	umut	dogruer	dogruerumut265@gmail.com	21

List of created game sessions saved by each user:

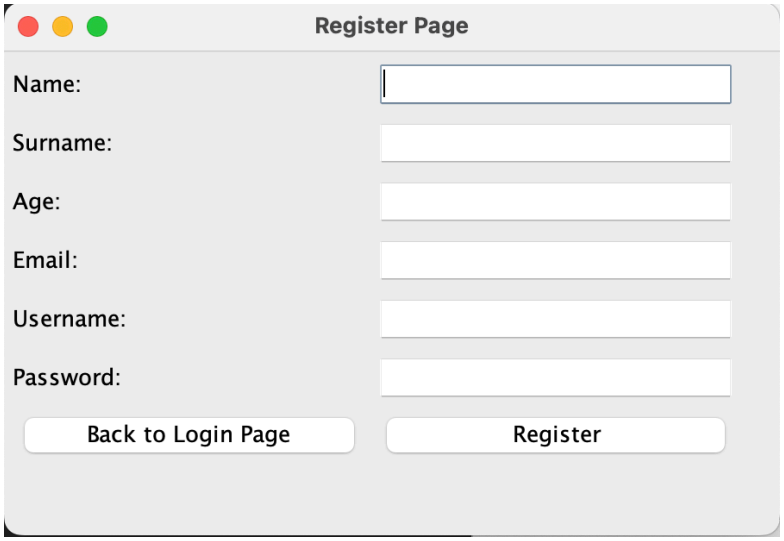


Application usage information

- **Sign up/Login Guide**

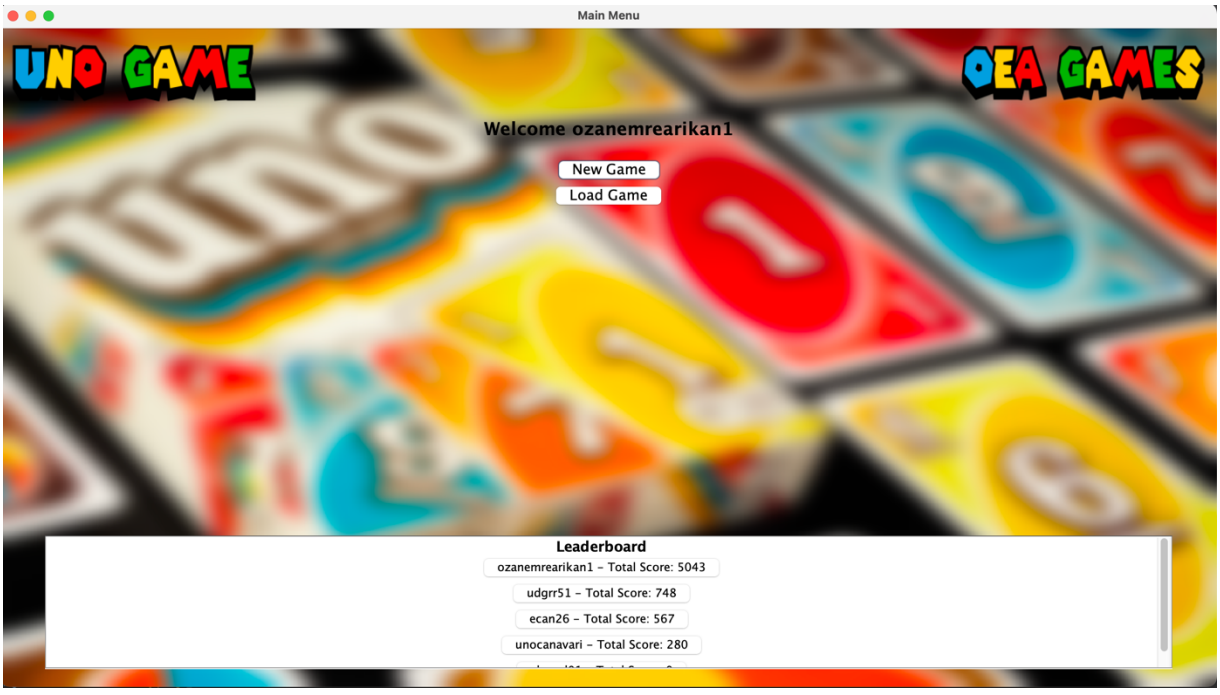


This login frame has basically two buttons one of which is for login and other to open register page. There are also two labels and corresponding frames such as text and password fields. If the login frame class can authenticate the user fields, user is able to open main menu frame. “RegistrationLoginFrame” which opens this frame has a method for authentication and for returning user from text file “String” data.



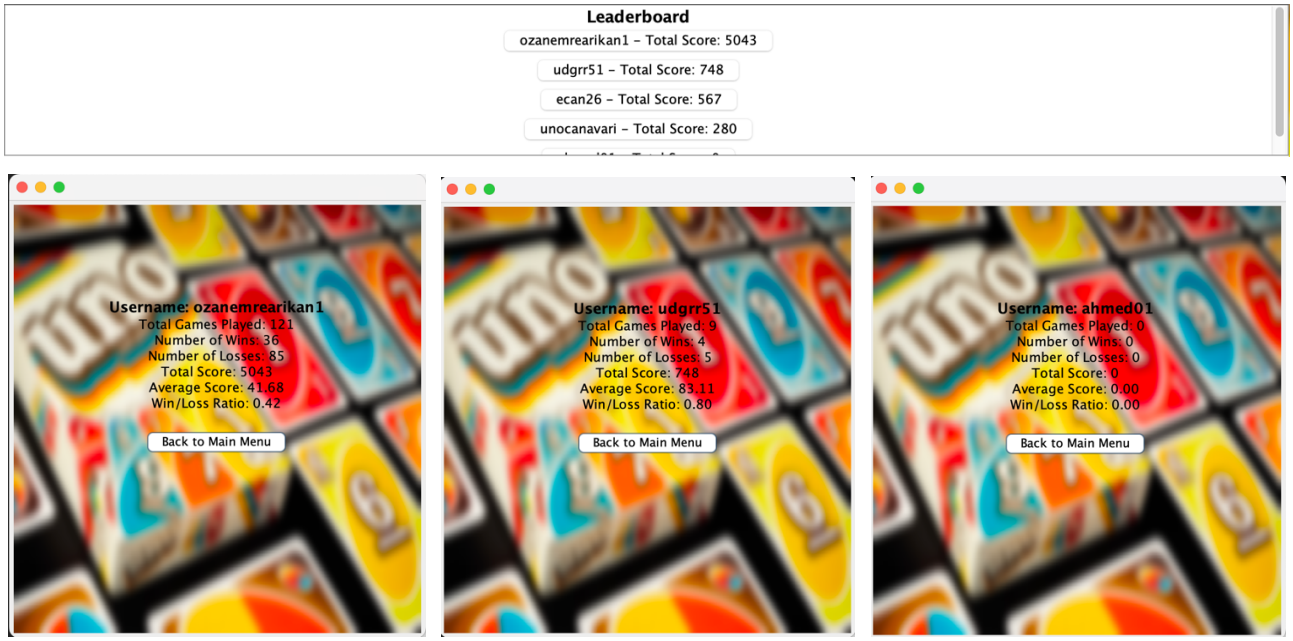
If user wants to sign up, they might access this register page below with clicking “Register” button on login page frame. Six corresponding labels and their text field (for password it is password field) are shown on register frame. There are two buttons under labels and fields, one of which is for turning back to login page and other successfully registers user and write data to “users.txt” text file which is in “src/user/users.txt”. “RegistratoryFrame” which opens this “Register Page” has controller method because there might be existing user with same username and email. Moreover, “RegistratoryFrame” has saving method with use of “PrintWriter, BufferedWriter, and FileWriter java.io” utilities. Furthermore, there is an invisible status label at bottom which prints any problem with registering.

- **User’s Guide for Main Menu Frame**



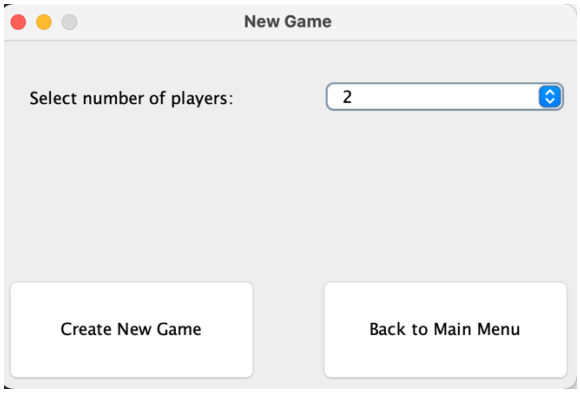
When user successfully logs in, main menu opens with welcoming message shown to user to feel them an authentic game experience. This frame displays logo of game at left and so-called my game studio at right. That creates an aesthetic appearance is the background image, on this frame. A blurred "UNO Cards" image is put which is in "src/images/blurred_uno_cards.png". "PanelForBackground" extending "JPanel" is used in other frames related to main menu as well as main menu frame. Other components as header and footer are added to this background panel thus everything is shown in frames properly. "MainFrame" class, which opens main menu, includes "resizeIcon()" method as well, used in many times in the project is created during trying to put logo at each side of screen. Moreover, statistics text file which is called "stats.txt" in "src/user/" is created in "MainFrame" first time if not created thanks to "appendToStats()" method. This method creates new line in text file with "0,0,0,0" data if new user is registered. This statistics is used for leaderboard creation at bottom.

- **Leaderboard**



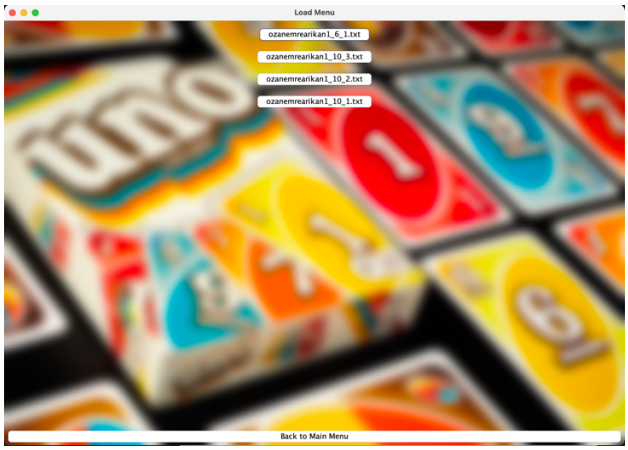
Leaderboard panel has all existing users from "stats.txt" using "loadLeaderboard()". The method puts clickable buttons for all users even if they did never play. This method includes connection to "UserStats" class and "Collections.sort(userStatsList, new StatsComparator())" method that will need to use "Comparator" afterwards. "StatsComparator" implementing "Comparator" is created due to the need for arranging users from highest to lowest on main menu frame. If "StatsOfUserFrame" is opened and "MainFrame" disposed, the user accesses the all statistics of clicked user. Total games, total score, and win/loss information are shown as the project instructions require.

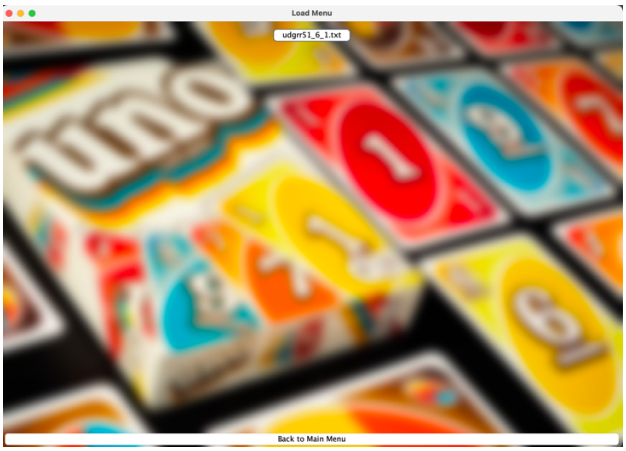
- **Number of Players Selection**



If current user clicks on “New Game” button, this number of player selection frame opens. “NumberOfBotsFrame” has “JComboBox” which shows a list of numbers between 2 and 10. “Game” class constructor has user and number of bots parameters thus selected number is saved with subtracting 1 from the number. Session name will be shown as “<username> & friends” but if the game is saved, the save name will be shown as “<username>_<numberOfPlayers>_<i>” (“i” is for increasing the number if the save exists with same name at “saves/” direction.

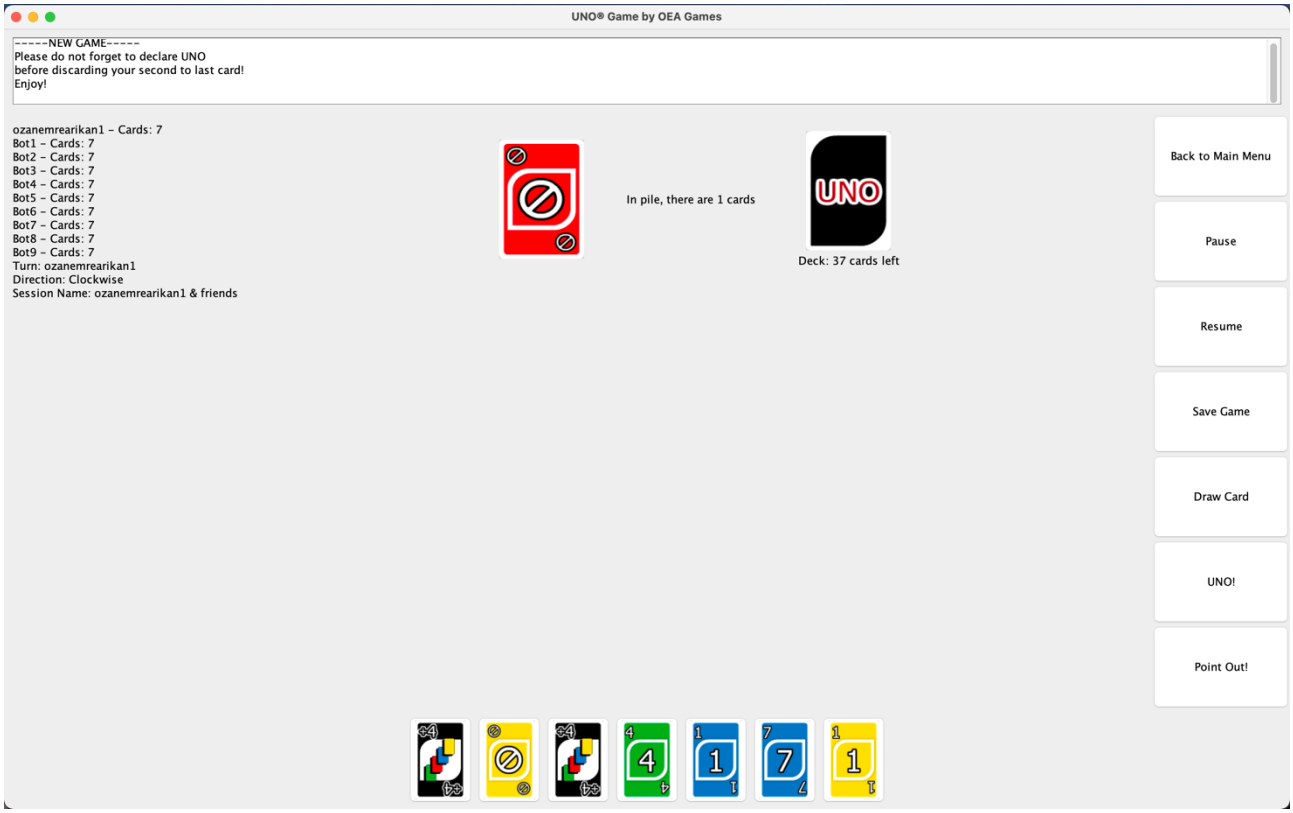
- **Loading Game**





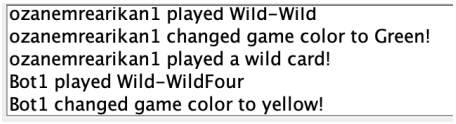
If user has saved game, they might load their game by clicking “Load Game” button on main menu frame. As indicated above, the save name will be shown as “<username>_<numberOfPlayers>_<i>” (“i” is for increasing the number if the save exists with same name at “saves/” direction. User must click “Resume” if the turn appears on bots at left panel when “GameFrame” opens. If opposite, they must play at bottom cards panel.

- **Game Session Gameplay**

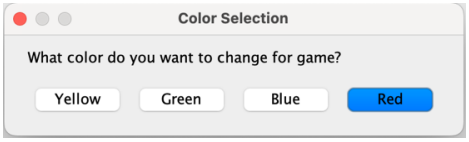


When “GameFrame” opens, user faces with “UNO® Game by OEA Games” title at top of window. In the frame of window, at top, there is a log area. Log area is interrelated with “LogGame” class via “logNow()” method in game. Log area is a scrollable and refreshable “JTextArea”. “LogGame” class has “logMessage()” method which appends each action to “loggings.txt” text file in “src/game/” and refreshes screen thanks to the Swing utility and “invokeLater()” method including runnable object. At left, user may see existing players with human player at top. Human always plays first. At bottom of left panel, there are refreshing labels through game for showing whose turn is, which direction the game continues to, and what simply the session name is (However, saved algorithmically in saving process). Moreover, at right panel, user has “Back to Main Menu”, “Pause”, “Resume”, “Save Game”, “Draw Card”, “UNO!” declaration, “Point Out!” declaration (for bot who did not declare UNO) buttons. User must save the game before turning back to main menu if they want to play again later. Pause stops the timer created in the back end and resume continues the timer. “Save Game” button successfully saves the game. Furthermore, the user must remember to declare UNO before discarding their second to last card at hand as indicated in log area at the beginning of the game. Also, user can click on “Point Out!” button. Remember that bots are sufficiently intelligent to declare UNO before they remain with one card at hand; however, do not hesitate to try!

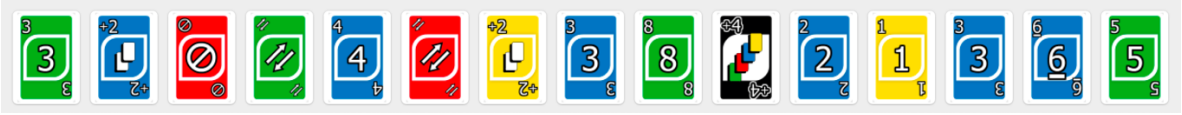
At bottom, human user has their card. When the user plays a card, the bottom panel refreshes. For every card at hand, there is a button with resized image icon of the card.



If wild card is discarded, bot automatically with randomness chooses a colour. User chooses the colour with "JOptionPane.showOptionDialog()" method:



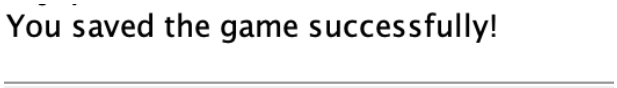
User can draw card as many as they want. However, pay attention to the fact that if there is not any card in the deck. Game finishes automatically and scores are calculated. Player with lowest points of card at hand becomes winner. At the end, be sure that statistics are registered to "stats.txt".

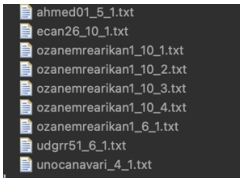


At centre panel, left card is "currentCardOnTop". User must play the card accordingly to this card on top. Value or colour of card must match to play. If not, "JOptionPane.showOptionDialog()" opens and writes that "You cannot play this card!". Also, username and card are logged on log area as invalid move. Remaining cards in "LinkedList" for "cards" of deck and "pile" are shown as label. Pile, discarded cards, is shown with UNO card back image icon at centre panel as well.

- **Saving Game**

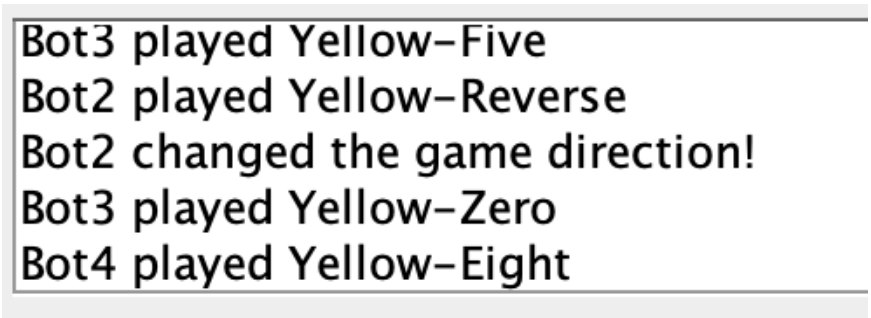
  When a user saves a game, log area is refreshed and writes:





In saved project there will not be "ozanemrearikan1_10_4.txt", it is just added for showing how save button works.

- **Logs into Text File and Log Area**



Part 2

Project Design Description:

- **Class Relations, Inheritances, Type Hierarchies, Interfaces, and Abstract Classes**
    - "frames" package
        - GameFrame, MainFrame, LoadGameFrame, StatsOfUserFrame, NumberOfBotsFrame, RegistrationLoginFrame, RegistratoryFrame as subclass of JFrame
        - PanelForBackground as subclass of JPanel
        - UserStats class for simpler manipulation of data
        - StatsComparator implementing Comparator<UserStats> is used for sorting scores from highest to lowest in leaderboard on main menu frame.
    - "game" package
        - Game class includes "User" class as type hierarchy. This user is the user logged in at the beginning and becomes first player (HumanPlayer) in the game. Bots become "AIPlayers" in the game. New deck will be initialised for each new game as field. This method can save and load game thanks to "FileWriter" and "FileReader" as well as "BufferedReader" and "PrintWriter" java.io classes.
        - Deck class simply creates "cards" of deck and "pile" as "LinkedLists".
        - Card class includes "enum" type for colours called "Couleur" and values as "Valeurs". The class contains a method to access card images in "cardimages/unocards/". Moreover, another method in the class might convert "String card information" from save text file to playable "Card" object.
        - HumanPlayer and AIPlayer as subclass of Player
        - Player class is an abstract class as superclass, including playing, drawing and point calculating card methods as well as an "ArrayList" for cards at hand.
        - LogGame class takes the path of "loggings.txt" and "JTextArea" log area as type hierarchy parameters in constructor.
    - "main" package
        - Main class starts the game from "RegistrationLoginFrame" and includes pledge of honour.

- “user” package
  - InvalidUserException as subclass of Exception
  - User is generated with name, surname, username, password, email, and age data in constructor.
  - Validation throws “InvalidUserException” if a user tries to register with invalid inputs. Regular expressions are used to simplify the process.

All frame classes except registration and login frame classes, has “User currentUser” information in constructor to turn back easily between frames as well as registering data into save and statistics file. Furthermore, GameFrame has a type hierarchy between Game and takes all information thanks to this hierarchy. However, both GameFrame and Game needs the user information; Game and GameFrame are initialised in “NumberOfBotsFrame” or “LoadGameFrame” separately, but Game is connected to GameFrame successively.

- **GUI Components**

1. **Login Frame**
   a. This frame permits user to log in or register by opening register frame.
2. **Register Frame**
   a. This frame permits user to register or turn back to login frame.
   b. If a problem occurs during registration, “JLabel statusLabel” is set with corresponding text and becomes shown.
3. **Main Menu Frame**
   a. This frame permits user to start a new game, load a paused game, observe leaderboard, and access to detailed user statistics.
   b. The frame includes “PanelForBackground” class as background panel and adjust another panel with adding in it. For logos, file handling helped. Image, ImageIcon, BufferedImage, and ImageIO classes are used in resizeIcon() method with getScaledInstance() -Image class method- and reader methods.
   c. Leaderboard is “JScrollPane” as our scrollable log panel. loadLeaderboard() method adds buttons for each user from “stats.txt” text file and sort them using “StatsComparator”.
4. **Player Number Frame**
   a. This frame includes “JComboBox” which permits to select number of players between 2 and 10. The selected number will be subtracted by 1 and recalls the Game(user, numberOfBots) object.

5. **Game Loading Frame**
    a. “LoadGameFrame” takes current user as its parameter. Additionally, it controls the “saves/” file and if it finds the saves which contains the user.getUsername() String, it arranges them on screen.
    b. That is aesthetically perfect, this frame uses the “backgroundPanel” as well. In many place, “setAlignmentX(Box.CENTER_ALIGNMENT)” is used for aligning buttons in centre. Also, Box class, with its methods, helps to leave space between objects.
    c. “Back to Main Menu” button at the bottom.
6. **User Statistics Frame**
    a. As I prepared in loading frame, this frame takes wanted user’s “String username” as parameters and of course, as well as “currentUser” to back to main menu.
    b. This string is used to find the wanted user in “stats.txt”.
    c. Since the data is string, “Integer.parseInt()” must be used.
    d. Comma is shown in my computer because of program is Europe-oriented, I’ve added “Locale.US” at the beginning of “format()” method. You may change it as you like!
    e. All stats might be zero thus it is controlled in the method.
7. **Game Session Frame**
    a. This frame is most extensive and comprehensive frame in the project.
    b. The frame mainly includes logPanel, leftPanel, rightPanel, centralPanel, and bottomPanel.
    c. “logPanel” includes “JScrollPane logScrollPane” which is scrollable text area. To not lean on edges too much, empty borders are added. Since setText() method clears previous writings, setCaretPosition() method is used from “Swing JTextComponents”.
    d. “leftPanel” includes labels for turn, direction, and session name. Left panel is refreshed after each player’s turn. Direction shows whether the game is clockwise or counterclockwise.
    e. “rightPanel” includes non-changing buttons of “Back to Main Menu”, “Pause”, “Resume”, “Save Game”, “UNO!”, “Point Out!”, and “Draw Card”. All buttons logic is explained but I need to add that pause and resume utilise “Swing Timer” class. Timer helps us to stop when bots are playing. When user clicks on pause, “isItPaused” boolean becomes true and timer.stop() method activates. If user clicks on resume, aiBotsTurn() method is recalled and timer starts again with 1.3 seconds delay of bot decision. It is vital because as expected game would be extremely rapid otherwise.

f. “centralPanel” includes “ImageIcon” of “currentCardOnTop”. It will be refreshed until game finishes. There is non-changing “ImageIcon” for game aesthetic indicating deck. Under this icon, there is an information of remaining cards. Also, centralPanel informs user how many cards there are in the pile.

g. “bottomPanel” includes buttons and resized image of cards on the buttons. This panel refreshes accordingly after human user plays or a card is drawn. “For loop” controls the human user’s hand and manipulate the bottom screen. Game session loop implementation and bot implementations are found below.

- **.txt File Processing Details**

In this project, there are mainly three text files named “users.txt”, “loggings.txt”, and “stats.txt” and save text files. First one I’ve created is “users.txt” on the first day of project. Java NIO package is used at the beginning. To read data, Files.readAllLines() and Paths.get() methods are used properly. Basically, it controls whether the user trying to register exists in the file. If not, PrintWriter, BufferedWriter, and FileWriter classes creates an “out” object and “println()” it into text file. Logging was the work of four and fifth days of the project. Log manipulations did include a writing-out process thus we use here PrintWriter, BufferedWriter, and FileWriter classes as well. During statistics and save text file manipulations, I realised that we can simply use Java IO package. Generally, line containing specific username is found and not found new line will be created.

However, I had to add complicated saving game method which contains a logic. The save name is shown as “<username>_<numberOfPlayers>_<i>” (“i” is for increasing the number if the save exists with same name at “saves/” direction) as indicated above.

```
/**
 * This method creates a save which include number of players in game to facilitate loading game,
 * if there is a save with same name, both will be there and saved.
 * @return String savename
 */
private String nextSaveName() {
    int i = 1;
    while (new File("saves/" + game.getPlayers().get(0).getName() + "_" + game.getPlayers().size() + "_" + i + ".txt").exists())
        i++;
    }
    return "saves/" + game.getPlayers().get(0).getName() + "_" + game.getPlayers().size() + "_" + i + ".txt";
}
```

This method returns new save game automatically with controlling if there exists a save with same name in the path. When save button is clicked, nextSaveName() method and game.saveGame(String nextSaveName) methods are recalled successively. “saveGame()” method separates each information with a line “---INFO---” like that. Load will read and evaluate each information until the line appears. I want to add that if I could think before write all Game and GameFrame classes, I could have added a map which holds all the information in it.

- **Game Session Loop Implementation**

  "updateHandOfUser()" method in the GameFrame constitutes the first part of the game session loop. Buttons are added at the beginning as well and refreshes with the method I mentioned. The method "getHand()" of human player and if the current player is "instanceof HumanPlayer", buttons become clickable. Also, if the card is "!(isPlayable)", card is not discarded and opens a "JOptionPane.showMessageDialog()". Corresponding logs are logged during game thanks to LogGame class and logNow() method. After human user plays a card, button recalls aiBotsTurn() method.

- **Bot Implementations**

  "aiBotsTurn()" method constitutes the second part of the game session loop and includes bot implementations. The method contains a timer which is field of the GameFrame class. This flexibility permits us to stop timer from "Pause" button and to continue it from "Resume" button as well. "aiBotsTurn() Timer" continues until current player is bot or isItPaused boolean did become true. Timer.setRepeats() permits us to repeat this until above statements are changed. Not in a short way, I write all possible scenarios which are logging to text file and top of the frame at the same time because I tried whether the game works. Both aiBotsTurn() timer and buttons of human user control if the game is finished.

  Game might finish with two ways. First way is the scenario where one player played all cards at their hand and remained with zero card. Second way is the scenario where there is not any card in the deck. In the first way game controls in every turn if previous player has zero card at their hand with "getPreviousPlayer()" method. If yes, game is finished and winner logged onto screen. If the winner is human user, "additionDesPoints()" (Means addition of points in French) method has parameter true and in it "totalScore" becomes a number other than 0. Statistics will be updated accordingly. Since previous player has zero card at hand, game is logged easily. Nevertheless, we need "determinerGagnantParPoints()" (Means determining winner by points) method for second way of finishing the game. This method determines winner by calculating points of every player's hand and log proper message.

References

Apart from the slides and documentation on the course website, I enhanced my knowledge and used the information there:

https://docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html -> to learn file manipulations

http://www.java2s.com/Code/Java/Swing-Components/ImagePanel.htm/ -> to learn how to user panels

http://www.java2s.com/Tutorials/Java/Swing_How_to/Basic/Add_Background_image_to_JPanel.htm -> to learn how to use background image as panels

https://www.javatpoint.com/java-jscrollpane -> to learn how to use “JScrollPane”

https://docs.oracle.com/javase/8/docs/api/javax/swing/JScrollPane.html -> to access method information

https://www.youtube.com/watch?v=vcHDscPR9Vw&t=710s -> to learn WindowBuilder

https://www.geeksforgeeks.org/stream-in-java/ -> to learn about stream operations

https://www.geeksforgeeks.org/java-util-timer-class-java/ -> Timer

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html -> Stream operations