

**COMP132: Advanced Programming**

**Programming Project Report**

**UNO Game Simulation**

**Mert TUNA   
  
0080363**

**SPRING24**

****

## 1. General Demo Information

### 1.1 List of Users

The following is a list of users currently in the system, including their nicknames and passwords:

* **Nickname:** mert, **Password:** mert1234\_
* **Nickname:** cago, **Password:** cago\_!123
* **Nickname:** baldeniz, **Password:** denom\_12345
* **Nickname:** onur, **Password:** apcag!123
* **Nickname:** baho, **Password:** bahadır\_swinger33
* **Nickname:** mengu, **Password:** penguin\_2002
* **Nickname:** tahaekim, **Password:** taha\_ekim123456
* **Nickname:** plutoc, **Password:** lol123\_lol123

### 1.2 Information about Each User

For each user, detailed information including wins, losses, total score, and other metrics is maintained.

#### Nickname: mert

* Wins: 1
* Losses: 1
* Total Score: 150
* Total Games: 2
* Win/Loss Ratio: 1.00
* Average Score per Game: 75.00

#### Nickname: cago

* Wins: 1
* Losses: 3
* Total Score: 162
* Total Games: 4
* Win/Loss Ratio: 0.33
* Average Score per Game: 40.50

#### Nickname: baldeniz

* Wins: 6
* Losses: 7
* Total Score: 561
* Total Games: 13
* Win/Loss Ratio: 0.86
* Average Score per Game: 43.15

#### Nickname: onur

* Wins: 2
* Losses: 5
* Total Score: 302
* Total Games: 7
* Win/Loss Ratio: 0.40
* Average Score per Game: 43.14

#### Nickname: baho

* Wins: 1
* Losses: 4
* Total Score: 270
* Total Games: 5
* Win/Loss Ratio: 0.25
* Average Score per Game: 54.00

#### Nickname: mengu

* Wins: 3
* Losses: 6
* Total Score: 508
* Total Games: 9
* Win/Loss Ratio: 0.50
* Average Score per Game: 56.44

#### Nickname: plutoc

* Wins: 14
* Losses: 13
* Total Score: 838
* Total Games: 27
* Win/Loss Ratio: 1.08
* Average Score per Game: 31.04

### 1.3 List of Existing Game Sessions

User named plutoc user have two saved game sessions, MainMenu -> Saved Games -> Select a game session to continue to the game.

## 2. Application Usage Information

### 2.1 Sign Up/Login Guide

1. **Open the Application:**
   * Launch the application from Runnable package -> Main to see the main login screen.
2. **Sign Up:**
   * Click on the "Sign Up" button.
   * Fill in the required fields: nickname, password, and password confirmation.
   * Click "Register" to create your account.
3. **Login:**
   * Enter your nickname and password.
   * Click the "Login" button to access your account.

### 2.2 User's Guide

1. **Navigating the Main Menu:**
   * After logging in, you'll see the main menu with options to start a new game, load a saved game, check the leaderboard, and view user statistics.
2. **Starting a New Game:**
   * Select the "Quick Play" option.
   * Choose the number of players and enter display name for user.
   * Click "OK" to begin the game.
3. **Loading an Existing Game:**
   * Select the “Saved Games” option.
   * Choose a game to continue.
   * Click Okay.

### 2.3 Game Session Gameplay

1. **Playing the Game:**
   * Players take turns according to the game rules.
   * The current player can play a card, draw a card, or perform special actions like saying "UNO."
2. **Game Interface:**
   * The game board displays player hands, the discard pile, and game controls.

### 2.4 Saving/Loading Game Sessions

During gameplay, the player can choose to save the current game state. This action will serialize the game data, including player hands, last played card, card left in the deck, current turn, and game state, to a file. A second game controller had been implemented for this aim.

### 2.5 Leaderboard

1. **Accessing the Leaderboard:**
   * From the main menu, click on "Leaderboard."
   * View the list of users with their names and scores besides. Additionally, click on a specific user to see details of their statistics.

### 2.6 How to Check Log Information

**Log Information:**

* Users can see the log information through the session, also saved the log information to the computer. Log information records game events such as card plays, actions taken, and game state changes.

## 3. Project Design Description

### 3.1 Class Relations

* **Game:** Manages the overall game state, players, and game logic.
* **Player:** Represents a general player with attributes like name, score, and cards. Implemented for the user.
* **CPUPlayer:** Extends Player, includes AI for making moves.
* **Card:** Abstract class representing a card with color, type, and value.
* **NumberCard, ActionCard, WildCard:** Concrete implementations of Card for different card types.
* **DealerShuffler:** Handles shuffling and dealing cards  
  .

### 3.2 Inheritances, Type Hierarchies, Interfaces, Abstract Classes

* **ViewCard (Gui)**
* **Card (Abstract Class)**
  + **NumberCard (Concrete Class)**
  + **ActionCard (Concrete Class)**
  + **WildCard (Concrete Class**
* **Constant** namedInterface including some game constants, and Game, Rules classes implements it for better readability. (Figure 3.2.1)

**A screen shot of a computer program

Description automatically generated**

Figure 3.2.1

### 3.3 GUI Components

* **MainFrame:** Main window containing all game components.
* **Session:** Panel displaying the game board and player information.
* **PlayerPanel, CPUPanel:** Panels displaying player and CPU information and cards.
* **TablePanel:** Displays the game table with the current top card and background.
* **InfoPanel:** Shows game status, errors, and remaining cards.

### 3.4 .txt File Processing Details

* **User Information Storage:** User credentials and game statistics are stored in user.txt.
* **File Operations:** Reading and writing operations are handled using BufferedReader2 and BufferedWriter.2

### 3.5 Game Session Loop Implementation

* **Game Loop:** Continuously checks the current player's turn, validates moves, updates the game state, and switches turns.

### 3.6 Computer Bot Implementation

* **CPUPlayer:** Extending from superclass named Player, this classimplements AI for making decisions, playing cards, and saying "UNO" based on predefined probabilities and conditions.

### *CPU Decision Making with Probability*

* **Saying "UNO"**: The CPUPlayer class uses probabilities to decide whether to say "UNO." For example, if the CPU has two cards, it might have a 80% chance of saying "UNO." If it forgets, it faces the penalty according to the game rules. (Figure 3.6.1). This is implemented to have a realistic UNO game, where players sometime forgot to say UNO, and penalized with two cards.
* **Choosing Wild Card Colours**: When a CPU plays a Wild Card, it analyses the colours of its remaining cards and chooses the colour it has the most of to maximize its chances of subsequent valid plays.
* **:) Kasparov BOTs:** Sometimes bots have bugs, and stop playing when their turns come, this problem couldn’t be figured out. However, if you say them “Are you Kasparov man, we are not playing chess”, and hit the table (basically playing with your cards, or with the card on the table), the bots will start playing, and the game will go on properly.

A screenshot of a computer

Description automatically generated

Figure 3.6.1

### 3.7 Additional Design Details

In the user profile page, regular expressions (regex) are extensively used for validating the user information. These regex patterns enforce specific constraints on user input, ensuring that the data entered adheres to predefined rules3. For instance, the isValidPassword method uses a regex pattern to verify that a new password contains at least one letter, one digit, and one special character, and has a minimum length of 8 characters. Regex checks are crucial for maintaining data integrity and security, offering a structured and reliable approach to input validation on the user profile page.

### 3.8 Additional Design Details

* **Card Matching Logic:** Ensures cards are played according to UNO rules, including colour and value matching, and handling special cards.
* **Session Management:** Allows saving and loading game sessions to maintain state across different plays.

## References

1. Java Swing Documentation: [Oracle Java Swing](https://docs.oracle.com/javase/tutorial/uiswing/)
2. File operations in Java: [BufferedReader](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/?ref=ml_lbp), [BufferedWriter](https://www.geeksforgeeks.org/io-bufferedwriter-class-methods-java/)
3. Regex: [Java Regex](https://www.w3schools.com/java/java_regex.asp)
4. Help: [Eclipse Help](•%09https:/help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.wb.doc.user%2Fhtml%2Findex.html)
5. Stack: [Stack Class](https://www.geeksforgeeks.org/stack-class-in-java/)