Crazy Eights Game - Operating System Project

Daniel A. Marin

Security is a critical aspect of software development, and it is essential to ensure that the code we write is secure and free from vulnerabilities. In this report, we will discuss the Crazy Eights game, a popular card game, and we implemented it securely in Java.

The Crazy Eights game is a fun and engaging card game that can be played by two or more players. The objective of the game is to be the first player to get rid of all their cards. Players take turns playing cards from their hand, matching the rank or suit of the top card on the discard pile (or using any 8 card). If a player cannot play a card, they must draw from the deck, play or pass their turn. The game continues until one player has no cards left.

I. Project Objectives

The main objectives of this project were to:

- Understand, compare and contrast the different data storage techniques
- Construct a program module for an operating system
- Become proficient system programmers
- Understand basics of system security

II. Implementation

The project was developed using the Java programming language, which is known for its platform independence and strong security features. The Crazy Eights game was implemented using object-oriented programming principles, allowing for easy maintenance and extensibility.

III. Compile and Run Instructions

To compile and run the Crazy Eights game, first ensure that you have Java Development Kit (JDK) installed on your system. You can download the latest version of JDK from the official Oracle website.

Once you have JDK installed, compile the Java files using the following command:

```
cd src
javac -d bin *.java
```

After compiling the Java files, you can run the Crazy Eights game using the following command:

```
java -cp bin CrazyEights --init --game [game_name]
java -cp bin CrazyEights --add-user [user_name] --
    game [game_name]
java -cp bin CrazyEights --remove-user [user_name]
    --game [game_name]
java -cp bin CrazyEights --start --game [game_name]
java -cp bin CrazyEights --order --user [user_name]
    --game [game_name]
java -cp bin CrazyEights --play [card] --user [
    user_name] --game [game_name]
java -cp bin CrazyEights --draw --user [user_name]
    --game [game_name]
java -cp bin CrazyEights --pass --user [user_name]
    --game [game_name]
java -cp bin CrazyEights --cards [user_x] --user [
    user_y] --game [game_name]
```

The above commands allow you to initialize a new game, add or remove users, start the game, check the order of players, play a card, draw a card, pass your turn, and check the cards of one user. Each command comes with security checks to ensure that the game is played fairly and securely.

Alternatively, you can run the game using the provided shell script. To do this, navigate to the root directory of the project and run the following command for Windows:

```
cd src
./Script.bat
```

For MacOS and Linux, use the following command:

```
cd src
chmod +x Script.sh
./Script.sh
```

These scripts will provide an interactive command-line interface for the Crazy Eights game, allowing you to easily navigate through the game commands and options while not having to type them manually each time. Also, the scripts provide a loop-back mechanism to ensure that the game continues until the user decides to exit.

IV. Class Diagram

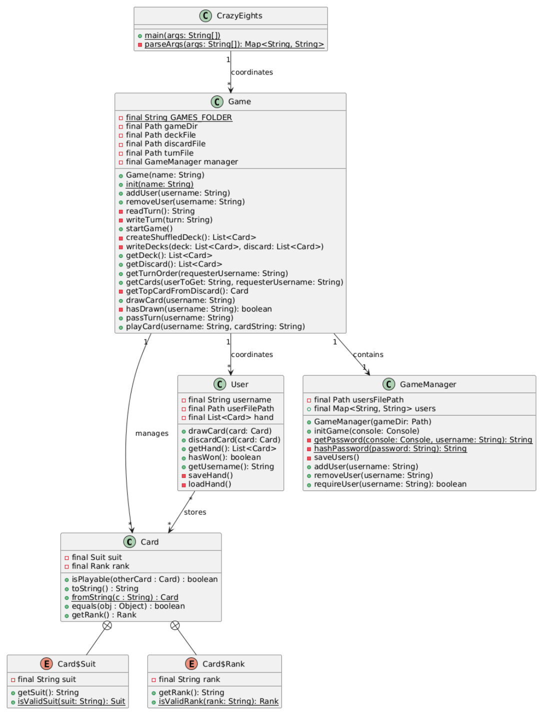
CrazyEights
main(args: String[])
parseArgs(args: String[]): Map<String, String>
1
coordinates
*
Game
final String GAMES_FOLDER
final Path gameDir
final Path deckFile
final Path discardFile
final Path turnFile
final GameManager manager
Game(name: String)
init(name: String)
addUser(username: String)
removeUser(username: String)
readTurn(): String
writeTurn(turn: String)
startGame()
createShuffledDeck(): List<Card>
writeDecks(deck: List<Card>, discard: List<Card>)
getDeck(): List<Card>
getDiscard(): List<Card>
getTurnOrder(requesterUsername: String)
getCards(userToGet: String, requesterUsername: String)
getTopCardFromDiscard(): Card
drawCard(username: String)
hasDrawn(username: String): boolean
passTurn(username: String)
playCard(username: String, cardString: String)
1
manages
*
1
coordinates
*
1
contains
1
User
final String username
final Path userFilePath
final List<Card> hand
drawCard(card: Card)
discardCard(card: Card)
getHand(): List<Card>
hasWon(): boolean
getUsername(): String
saveHand()
loadHand()
GameManager
final Path usersFilePath
final Map<String, String> users
GameManager(gameDir: Path)
initGame(console: Console)
getPassword(console: Console, username: String): String
hashPassword(password: String): String
saveUsers()
addUser(username: String)
removeUser(username: String)
requireUser(username: String): boolean
*
stores
*
Card
final Suit suit
final Rank rank
isPlayable(otherCard : Card) : boolean
toString() : String
fromString(c : String) : Card
equals(obj : Object) : boolean
getRank() : Rank
Card$Suit
final String suit
getSuit(): String
isValidSuit(suit: String): Suit
Card$Rank
final String rank
getRank(): String
isValidRank(rank: String): Rank

Fig. 1: Class Diagram of the Crazy Eights Game

In figure 1, we can see the class diagram of the Crazy Eights game. The diagram illustrates the relationships between different classes, including the Game, User, Card, and Deck classes. Each class has its own attributes and methods, which are used to implement the game logic.

A. *CrazyEights Class*

The CrazyEights class is the main entry point of the program. It handles command-line arguments to coordinate the game flow. The class is responsible for validating user input, and invoking methods from other classes to perform specific actions.

B. *Game Class*

The Game class represents the game itself. It manages the game state, including the players, deck of cards, and the discard pile. The class provides methods to add or remove players, start the game, and handle player actions such as playing a card or drawing from the deck. In essence it contains the game logic and rules, thus acting as the brain; managing the game state with the turn file, the players file, and the decks files. The class also ensures that the game is played fairly by enforcing the rules of Crazy Eights.

C. *User Class*

The User class represents a player in the game. It contains information about the player's name, hand of cards, and their current status in the game. The class provides methods to add or remove cards from the player's hand, check if the player has won, and display the player's hand. In addition, this class is responsible for managing the player's actions, such as playing a card or drawing from the deck; and demonstrating those actions on the actual files for each respective player.

D. *Card Class*

The Card class represents a playing card in the game. It contains information about the card's rank and suit. The class provides methods to compare cards, check if a card is playable, and display the card's information. The class also ensures that the cards are represented correctly and can be easily manipulated during the game.

E. *GameManager Class*

The GameManager class is responsible for managing the security aspects of the game. It handles user authentication, authorization, and input validation. The class ensures that only authorized users can perform certain actions in the game, such as starting a new game or playing a card. This class is crucial for maintaining the integrity and security of the game.

V. Run-time Demo

VI. AI Disclosure Segment

The project was developed with the help of GitHub copilot, which is an AI-powered code completion tool. The tool was used to help generate the class diagram, and performing certain tedious actions. Overall, the AI tool was helpful in speeding up the development but was not relied upon for the logic of this project instead it was used to help with the boilerplate code and repetitive tasks. The final implementation was thoroughly reviewed and tested to ensure that it met the project objectives and security requirements.