Nick Petrilli

Professor Arias

CMPT220_203

14 May 2020

Project 2 Final Write-up

Abstract

Ranging from 2-7 players, crazy eights is a popular card game in which the goal is to get rid of all of your cards before the opponent(s). By using multiple classes to act as the necessary items to play this game such as Card and Deck, and abstract classes such as Player and CardArray this game was made possible to play against a computer player written in Java code. This finished system allows a user to play a game of crazy eights against the computer, or against another user (has to be on the same computer).

Introduction

After encountering so many games in my childhood I’ve always wondered what was behind the scenes to make the game actually work. Users just see the game itself, not how it's written and the immense detail needed for it. Card games have always been an entertaining way to pass time, so I figured I’d create the crazy eights game that I used to play a lot as a child. This paper will go into detail of how this project was created and how all of the classes and objects work together to make the final product: a fully functioning game of crazy eights. The user can watch as two computers play against each other, but actually playing against one computer yourself is a little more fun. This system allows people to have fun playing a card game without using actual cards!

Detailed System Description

The complete system allows for a smooth game of crazy eights against one computer player. To begin, the Start class is run which includes the main method for this project. Initially a welcome message is displayed, followed by instantiating a Game which maintains the state of the game between the user and the computer. The playGame method within the Game class consists of a while loop that executes while the game is still going on. This method constantly calls the displayState method which displays the essential facts of the game including the last played card, the total cards discarded, and the total cards still remaining in the draw pile. Additionally, each turn is dependent on the user pressing the enter button so the game is controlled and not all happening at once. Once either player's hand is empty, the game is over and the final scores are displayed along with the winner. Scoring for each player is based on how many cards still remain in their hand when the game ends. Face cards are worth 10 points, 8s are worth 20 points, and cards with numbers are given the value of the number on the card. The player with the least amount of points is declared the winner.

As for the actual playing of the game, each player starts with a randomly selected hand from the draw pile. Another random card is selected which starts the game as it is the first card in the waste pile. This is when it turns to the players to start getting rid of their cards to win the game. Given the randomly selected hand, the user will have to look at their hand and determine which cards are valid to place based on the previous placed turn. Based on the rules of crazy eights, a valid card to place is a card that has a matching suit or face value as the previously placed card, or a wild card. A wild card is a card with a face value of 8 (any suit) can be placed at any time during the game. Once the card is selected, it is then placed down and removed from the user's hand. This shifts all of the other cards in the hand up one position. If the user selects a card that doesn't fit the criteria for a valid card, a message is displayed notifying the user that they need to place a new card. In this case, it will not be removed from the user's hand. After each round (when all players have placed down their cards), an updated hand is printed out.

In order for this game to work, the classes I created have to be related to one another. Since Player is an abstract class, it has subclasses User and Computer, which are two types of players. The user represents the actual human playing the game, and the computer is the artificial intelligence player that they are competing against. The Computer class consists of methods that search through their hand in order to find a valid card to place down. This is done by comparing the previously placed card to the cards within the computer's hand. If there are no valid cards to place down, the computer will draw cards until it finds one. The User class is where the message is printed to either choose a card from your hand, or press "0" in order to draw a new card. The turn is not over once a card is drawn, so a player has to draw however many cards necessary until finding a valid one.

CardArray is another abstract class, which is the superclass to both Deck and Hand, and Deck has a subclass of Pile. These are all arrays of cards used differently during the game. Obviously, the Hand class represents the hands of the players in the game. Within the Hand class is the popCard method which is responsible for removing the cards from the hand as they are placed. The Deck class is responsible for shuffling the cards, and then dealing them randomly to each player in the game. As for the Pile class, it's a subclass of Deck and there are constantly two piles that allow the game to work. The first pile is the stock pile which allows players to draw cards from when they cannot place any of the cards in their hand. The second pile is the “discard” pie which is the pile of cards that each player places down. None of these type CardArray objects would be functional without the Card class because the Deck, Hand and Piles all consist of card objects. The Card class is where the comparisons are made for the user to determine which cards are valid to place down to keep the game going. It also consists of multiple methods including stringRank and stringSuit which use switch statements to give a numerical value for each card in order to classify them. When the game ends and each player is given points to determine the winner, the Card class is the one that checks each card and gives the player that has them in their hand the amount of points each card is worth according to the rules of crazy eights.

Requirements

Since this project is a game for entertainment purposes, it isn’t solving a real life problem. The requirements for this project were all of the classes I’ve explained in the previous section. Without just one of these classes, the system wouldn’t work correctly. It’s impossible to play a card game without cards, a deck and players. Without the computer class, the user wouldn’t be able to play against anyone. Also, the use of abstract classes being CardArray and Player were necessary because there are multiple types of card arrays and players that can’t be represented using only one class. Since they are all closely related, the abstract superclass lays out the methods that each subclass will have, but within each subclass they will have different code bodies for the methods, along with being able to add any extra methods specific to that class.

Literature Survey

Abstract classes are classes that can have subclasses, but cannot be instantiated. They include abstract methods in which each subclass has to implement. These two classes have protected instance fields instead of private. A private instance field only allows that specific class to access it, but since these are both abstract classes their variables are protected, meaning they can be accessed by other classes within the same package. The card class is a final class, meaning that it cannot be extended by any other classes. This is because there is only one general type of card, any subclasses of this class wouldn’t make sense. In order to tie all of the classes together, the game class has private instance fields of everything necessary for the game to play: the players and the piles of cards.

User Manual

This system is very easy to use, even if the user doesn’t know how to play the game crazy eights. The rules are laid out for the user in the beginning, and each round there will be updates on what exactly just happened so nothing is missed or looked over. Also, if an invalid card is chosen, the user will receive a message saying that the card was invalid and they need to choose another one. This allows the game to play smoothly without any mistakes that contradict the rules of the game. After going through and playing a round it's very easy to catch on and continue the game.

Conclusion

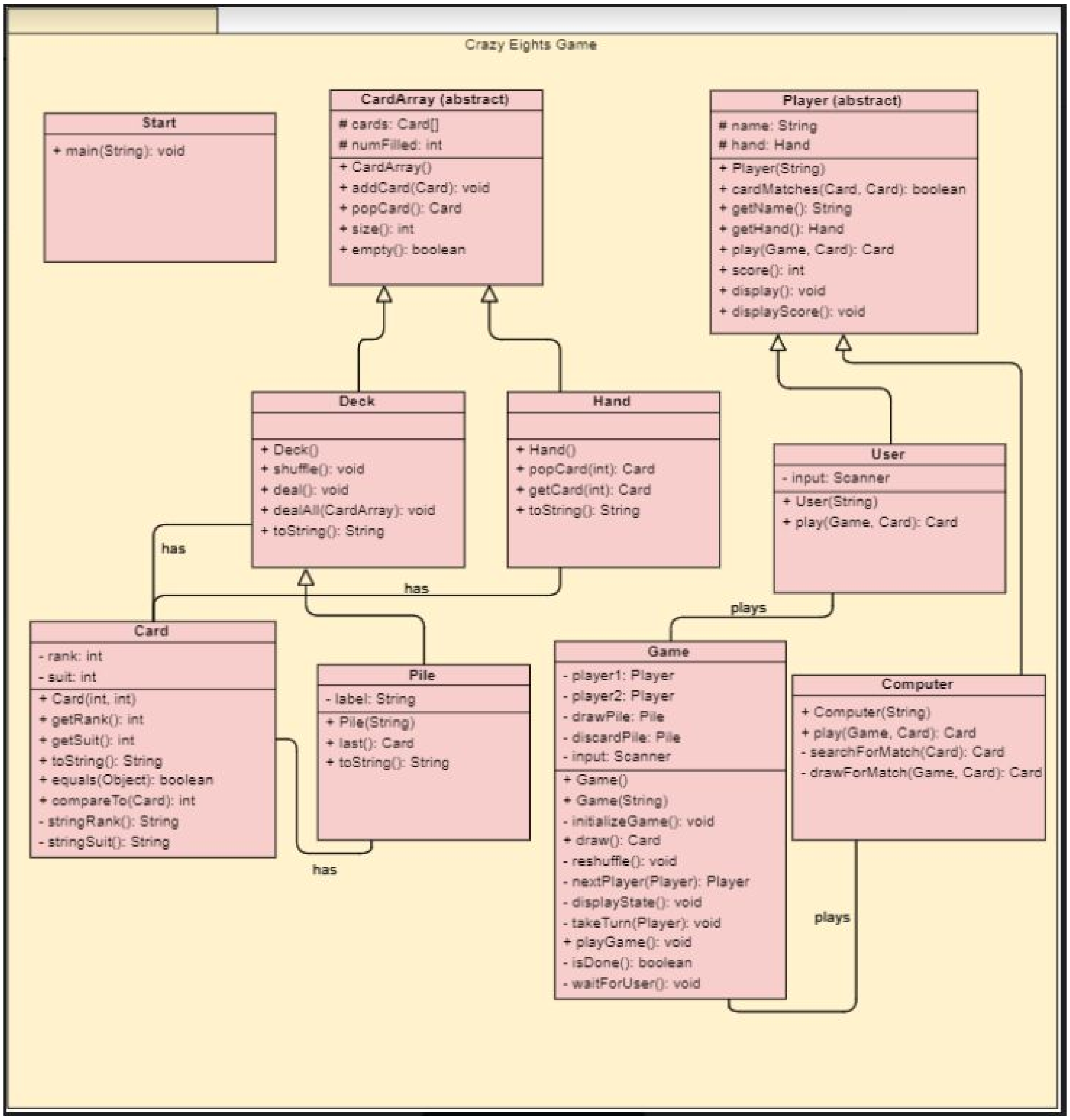
A fully functioning system allows a user to play in a game of crazy eights against the computer. Just like the normal game, the game can go on until there are no more cards left in the stock pile. The shuffling of the deck is always different so each game is different. Also since the game is mostly based on luck, the outcomes of each game will vary. Although some aspects change each game, one thing that will be the same is the fun you have while playing!

References

https://bicyclecards.com/how-to-play/crazy-eights/

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

UML Diagram


Crazy Eights Game
Start
+ main(String): void
CardArray (abstract)
# cards: Card[]
# numFilled: int
+ CardArray()
+ addCard(Card): void
+ popCard(): Card
+ size(): int
+ empty(): boolean
Player (abstract)
# name: String
# hand: Hand
+ Player(String)
+ cardMatches(Card, Card): boolean
+ getName(): String
+ getHand(): Hand
+ play(Game, Card): Card
+ score(): int
+ display(): void
+ displayScore(): void
Deck
+ Deck()
+ shuffle(): void
+ deal(): void
+ dealAll(CardArray): void
+ toString(): String
Hand
+ Hand()
+ popCard(int): Card
+ getCard(int): Card
+ toString(): String
User
- input: Scanner
+ User(String)
+ play(Game, Card): Card
has
has
plays
Card
- rank: int
- suit: int
+ Card(int, int)
+ getRank(): int
+ getSuit(): int
+ toString(): String
+ equals(Object): boolean
+ compareTo(Card): int
- stringRank(): String
- stringSuit(): String
Pile
- label: String
+ Pile(String)
+ last(): Card
+ toString(): String
has
Game
- player1: Player
- player2: Player
- drawPile: Pile
- discardPile: Pile
- input: Scanner
+ Game()
+ Game(String)
- initializeGame(): void
+ draw(): Card
- reshuffle(): void
- nextPlayer(Player): Player
- displayState(): void
- takeTurn(Player): void
+ playGame(): void
- isDone(): boolean
- waitForUser(): void
Computer
+ Computer(String)
+ play(Game, Card): Card
- searchForMatch(Card): Card
- drawForMatch(Game, Card): Card
plays