Nick Petrilli

Professor Arias

CMPT 220_203

24 April 2020

Project 2 Milestone

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 will be made possible to play against a computer player written in Java code.

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 the work I've already done, 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.

Detailed System Description

The complete system allows for a smooth game of crazy eights against one computer player. Once the game begins, both players will be given a hand of randomly selected cards from the deck. Another random card chosen from the deck will be the first card in the waste pile, where it then turns to the players. The user will look at their hand and select a valid card to place: a valid card is a card with the matching suit or face value as the previously placed card. A wild card, a card with a face value of 8, can be used at any time (even if a different number and different suit is the card placed before). The game is very easy to play because each time a player places a card it will be displayed in a message now stating who's turn it is. Additionally, the program checks if an invalid card is selected and in the event the card won't be able to be used resulting in the user selecting a different card or drawing from the deck to find a valid card. After each round (when all players place down their cards) an updated hand will be printed out, removing the previously placed card. The game ends when one player has no cards left in their hand, and the score of the game is printed out as well as the winner. The goal is to have the least amount of points because points are added based on how many cards still remain in the hand once the game finishes. 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.

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. 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. None of these type CardArray objects would be

functional without the Card class because the Deck, Hand and Piles all consist of card objects. There is also a Game class which creates and maintains the state of the game until it is completed. This is necessary because it is the base code that uses all of the classes together to make the functional game. Start is the last class which runs the main method, including a brief introduction and calls to the other classes.

Requirements

At this stage in the project it is tough to tell which classes aren't running correctly because all classes have to be written prior to writing the Game class which uses every class together. Once the Game class is created it'll be easier to tell which classes have the problems. The game is being made for entertainment purposes, not to solve a specific real life problem.

Literature Survey

Some of the work that is still needed to be done includes the Start class which is just the main function, the actual Game class which maintains the state of the game, and the computer class. The computer class will be very similar to the user class as they are both subclasses of Player. Instead of giving the option to choose what card to place, once each card is tested against the previously placed card, it will then be placed on the pile.

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. After going through and playing a round its very easy to catch on and continue the game.

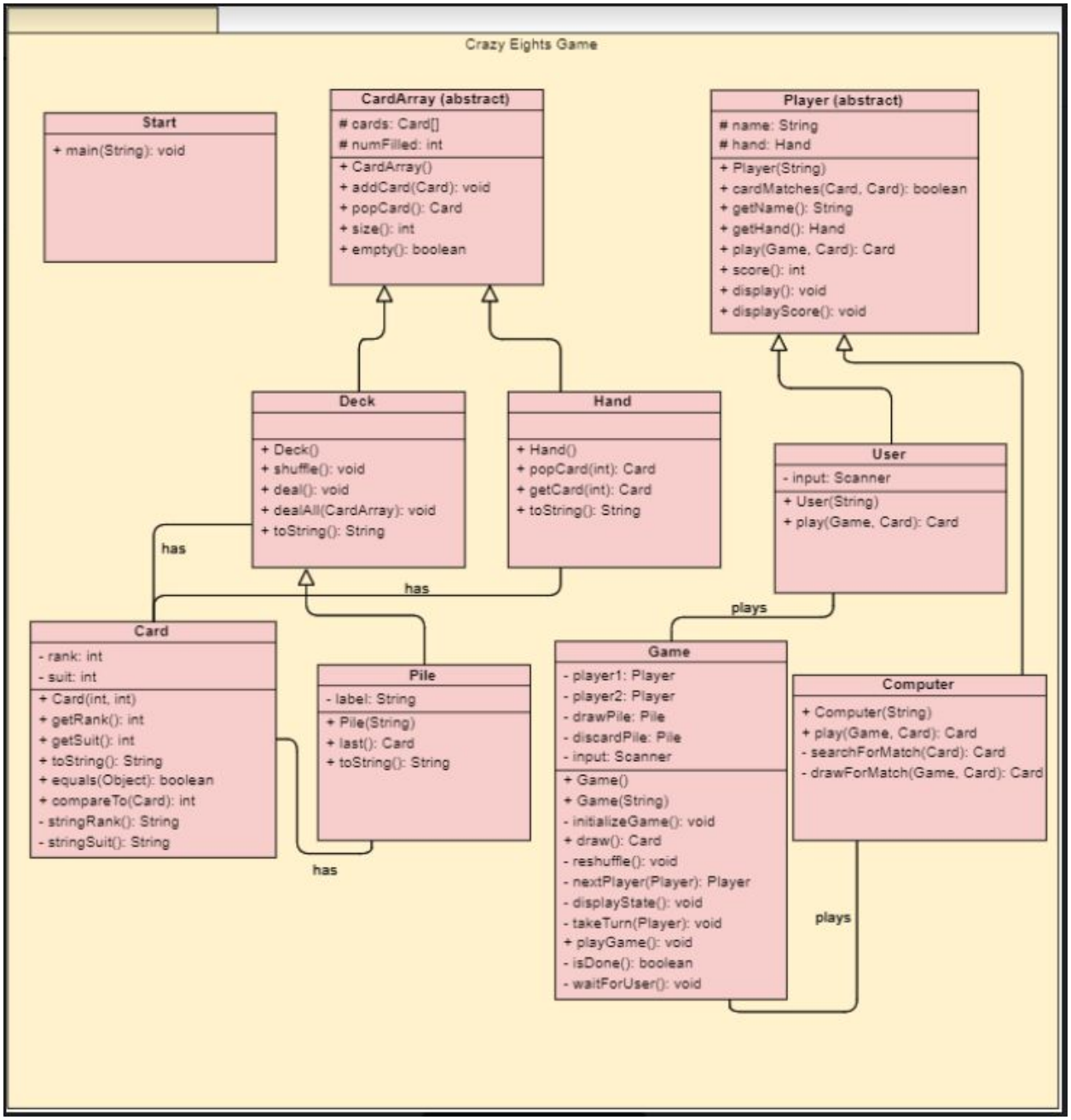
Conclusion

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. Every game is different, so users can play and have fun while beating the computer.

References

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

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