Introduction

We implemented BUno strictly using JavaFX and CSS while following a Model-View-Controller design pattern. The gameplay happens through the model, with listeners attached to the GUI elements on the screen. For example, drawing a card will remove the Card object from the Deck object, and add it to the Hand object. As events happen, the view accesses the model's components and creates an interface that is easy for anyone to use in order to play.

Upon startup, the user is automatically able to complete his turn. To do so the user may click on one of the cards in their hand or on the draw deck. Clicking on an invalid card will result in the game to display a popup message, telling the user about the mistake. After the user makes a legal choice, the user is no longer able to click on either of those two objects. At this time the AI's will complete their turns, and gameplay will go back to the user.

Implementing our design to fit our exact goal was time-consuming and challenging. Lots of detail went into arranging our view in the cleanest way possible. For example, getting our hand to look just the way we wanted took a lot of trial and error using various JavaFX objects.

User stories

Complete:
As a player, I want to be able to hold a card in my hand
As a player, I want to be able to see the board
As a player, I want to know what the first card in the discard pile is at the beginning of the game
As a player, I want to always have a card to draw from
As a player, I want to be able to draw cards after the game begins
As a player, I want to be able to play a number card
As a player, I want to be able to start my turn
As a player, I want to be able to renege
As a player, I want to start a game with 7 random cards
As a player, I want the computer to make a basic play (AI)
As a player, I want to know who is going first
As a player, I want to score my hand once I have no cards in my hand
As a player, I want the first card played into the discard pile to tell me how gameplay begins

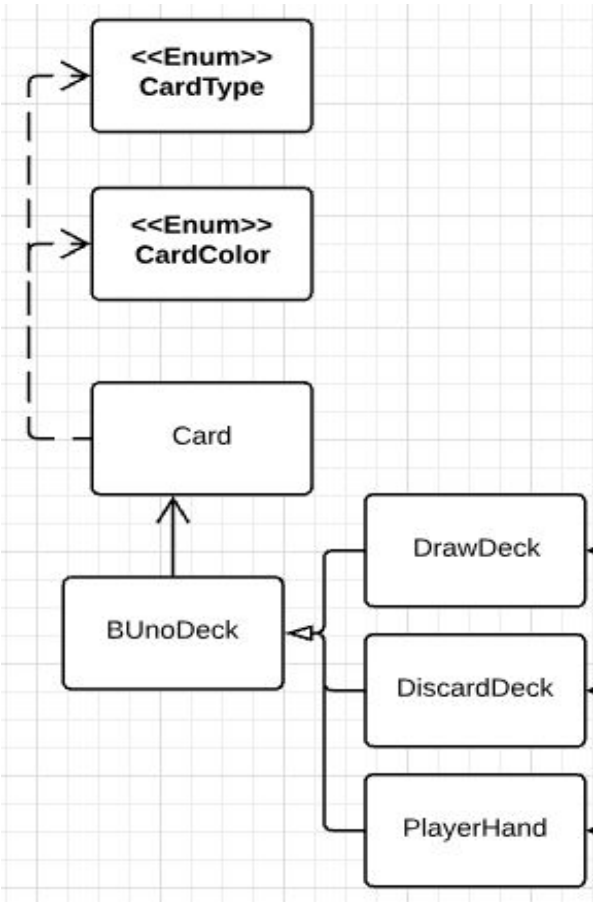
Finished coding but not tested/delivered:
As a player, I want to be able to play a draw 2 card
As a player in a TWO player game, I want to be able to play a draw 2 card
As a player, I want to be able to play a Reverse Card
As a player in a TWO player game, I want to be able to play a Reverse Card

As a player, I want to be able to play a skip card
As a player in a TWO player game, I want to be able to play a Skip Card
As a player, I want to be able to play a wildcard
As a player in a TWO player game, I want to be able to play a wildcard
As a player, I want to be able to play a Wild Draw 4 Card
As a player, I want to be able to Challenge a Wild Draw 4 Card
As a player in a TWO player game, I want to be able to play a Wild Draw 4 Card
As a player, I want to win the game with 500 points
As a player, I want to be able to yell UNO!
As a player, I want the computer to make an advanced play

Unfinished:
As a player, I want to see a lobby to join a game

OOD
We followed the object-oriented design pattern as we implemented BUno. Based on the user stories and designed some of the most important classes as follows:

1. After joining a game, A player needs four basic actions: Draw card, play card, callBUno, get score. We designed BUno can divided into three parts:
    a. BUnoDeck: Handles draw and play cards.



Class: Card
Responsibilities: color, type

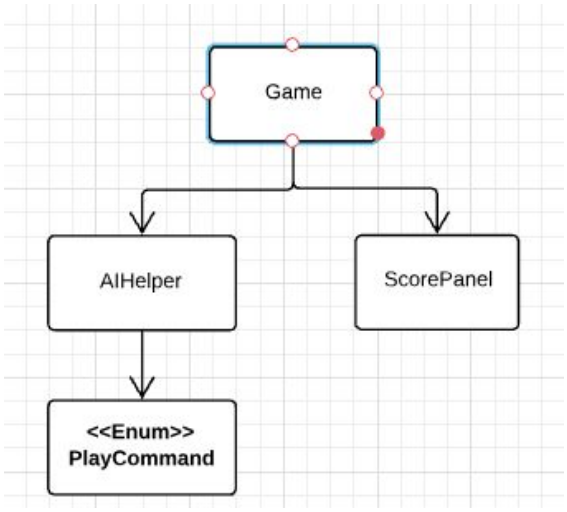
Class: BUnoDeck
Responsibilities: abstract class representing the deck
Collaborators: Card

Class: DrawDeck
Responsibilities: holds the cards that have not been played and are not in someone's hand, is a list of the cards
Collaborators: Card

Class: DiscardDeck
Responsibilities: holds the cards that have been already played
Collaborators: Card

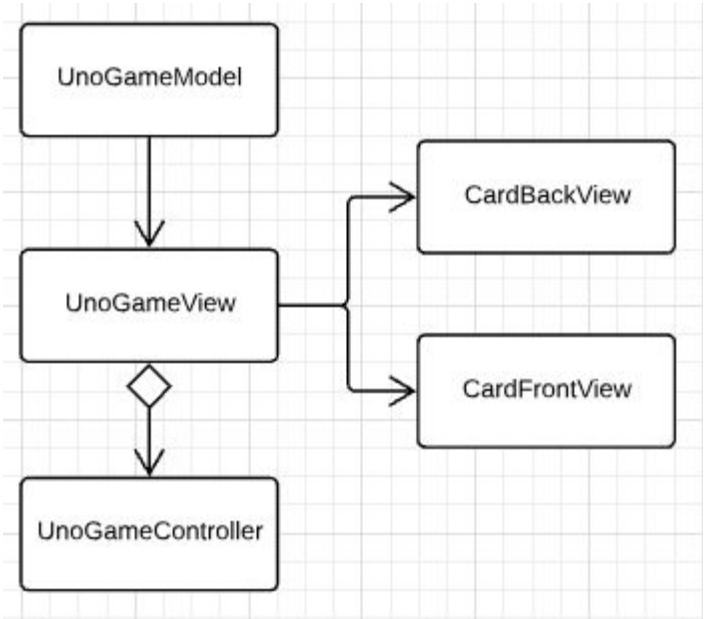
Class: PlayersHand
Responsibilities: holds the cards in a player's hand
Collaborators: Card

b. Game: Handles the BUno rules



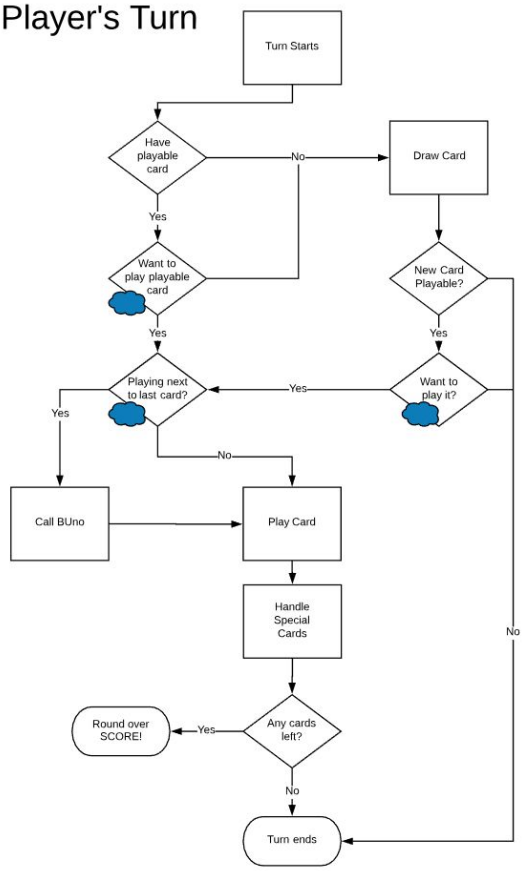
Class: Game
Responsibilities: Creates and manages an unoGame
Collaborators: Card, Deck, AI Intelligence level

c. Gui: Model, View, Controller



2. Analysis of player hand:
    a. Active player
    b. Inactive player
3. Enum classes: CardType, CardColor, PlayCommand

Active Player's Turn



Not Active Player's Turn