Team BatWomen- Bayan Berri, Alison Lee, Truc Dao
APCS2 Period 4
HW41 -- ProPro
2017-05-17

UNO

Description:

Our final project will mimic the card game UNO. The goal of the game is to be the first player to discard all of their cards. Two start the game, seven cards will be dealt to each player. The top card of the deck will begin the game. When it's someone's turn to play, they have to play a card with either the same color or the same number as the top card in the discard pile. If the player does not have a card in his or her hand that can be played, they must draw a card from their deck. In addition there are action cards including skip, reverse, wild, draw two, and draw four. Wild cards can be played at any time throughout the game.

There is also the option to play in a tournament. The tournament will be organized along a binary search tree. The binary tree will be used to record the progress of the ongoing tournament, with options to see each player's stats, which will be printed by searching the tree.

Usage of Topics Covered in Class:

- *Binary Trees*: We will be including an UNO tournament (see Features)
- *Double Linked Lists*: We will be using DLLs for the list of players. This is helpful because in UNO, it's possible that a player plays a reverse card, and DLLs make it easier to deal with that action card.
- *Random Queue*: each game will be initialized with a deck of 108 cards. This deck will be in the form of an RQueue. This is because we will have to shuffle the deck so that each game is slightly different. In a Queue you can only dequeue from the end of the deck, in this case it'll be useful because when playing a card game you typically can't access the cards in the middle.
- *ArrayList*: Array Lists (fall semester) will be helpful when dealing cards to players. The ability to expand the size of an array list is helpful because in UNO a hand varies in length from one game to another. ArrayLists also offer methods for adding and removing cards, which will help as the game progresses.
- *Stacks*: The discard pile will be in the form of a stack. The reason for this is that at any given moment, only the top card is visible to the players. Also, we would only be able to push cards onto the stack. When playing UNO, there's no reason to ever have to see the cards that have been discarded previously, except for the most recent card.

- *Recursion:* use to continue the game until there is one player left (who will be the last at the leaderboard)
- *Sorts*: We will be using either quick sort, heap sort, or merge sort in order to sort the leaderboard.

Features:

- Tournament, using a binary tree: All players will start at the bottom level of the tree, which will be of height $log_2(\#\ of\ players) + 1$. The players will be divided into groups of two (which is why a binary tree is used). The parent of that pair will take on the value of the winner of the pair. At the end, the root of the tree should hold the winner of all players. Each node of the tree contains two variables: String _name and int _score.
- We plan on coding an AI. The way this will work is by looping through the hand and checking to see if any of the cards match.
- We will also try to implement calling UNO! If someone has only one card left and they don't call UNO before ending their turn.
    - *Possibility:* If the player that has one card left does not call UNO before ending their turn, only the following player would be allowed to call UNO on them. After that, it'll be too late. If they forget and someone else calls UNO, they have to draw two cards.
- We can have different versions of UNO. One where the players have to keep drawing until they get a card that can be played (harder) and one where the players can just draw one card and if it's playable they can play it, otherwise end their turn. Many different versions are possible, we can have variety so the game doesn't get boring. These different versions will be finalized later on, still under construction.

Extra:

- We plan on adding visuals by using processing. This will be a simple set up that displays the player's hand, the top card of the discard pile, and the deck.
- We can add a timer, if a player takes too long we just move on to the next player.
- And more…

Class Details/Structure:

- Upon running the game we will prompt the users to ask them if they want to play in tournament mode, or just one classic round of the game. We will then ask them for their names, and the order that they enter their names will be the same order they get added to the DLL of players. We will also ask them what version they want to play. This can be through a series of questions or just one question with a set of rules under each possibility that explains the choices and rules. Then we will deal cards (seven per person), flip a card to start the discard pile, and play the game by following the rules of UNO based on their version.

- The UNO class will have instance variables for a deck of type Card in the form of an RQueue. It will also have a stack of type Card.
    - If we go on to have different versions of UNO we will include subclasses that follow the specifications of that version.
- We will have a class for players. Players will have an instance variable of type int that keeps track of how many games they have won and possibly a boolean of whether they won or lost. They will also have an instance variable for their hand and
- We will also have a card class. This class will be abstract because there are different types of cards (action cards/wild cards/ number cards)
    - We will include subclasses for the different types of cards. Each subclass will have to include the details of the cards. (Wild Cards can extend Action cards)

UMLs for MVP:

-

ClassicUno
- Deck: LinkedList<Card> - Players: DLL<Player> - Discard: Stack<Card>
+ chooseVersion() int + newGame() void + sortRank(): String + deal(): void + matches(Card) Boolean

-

Player
- int gamesWon - int gamesPlayed - int _score - Boolean won - ArrayList<Card> hand
+ drawCard() + playCard(): Card + callUno(): String + toString()

Interface Card
+ getColor()

-

NumberCard extends Card
- int number - String color
+ NumberCard() + getColor()

-

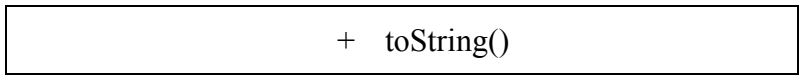
ActionCard extends Card
- String action - String color
+ ActionCard() + Ability() + getColor()

-

WildCard extends ActionCard
- String _ability
+ WildCard() + Ability() + getColor() + setColor()

-

Tournament
- Tree(Player) _leaderboard
+ play() + findPlayer(Player x)


+ toString()