CS111 Fall 2013 Project: Milestone 2

Due November 18th (40 pts total)

The Project

In this milestone, you will implement the basic strategy of your UNO player. We will be testing your code through a framework for running Uno games. It simulates shuffling a deck, dealing hands to players, drawing an initial up card, enforcing all of the rules of Uno (see Milestone 1 description), declaring a winner, and calculating scores. The only thing it does not do is actually choose a card to play from a hand (or call a color if a wild is played). You will hand in your netid_UnoPlayer.java only.

For this milestone, every student in the class will be writing their own code to play a card from a hand ~~(for this milestone, you do not need to worry about wild cards)~~. The Uno framework will run the game, and then at the appropriate points, call your method(s) to do pick a card. In this way, your program will be able to "play" Uno against your classmates in a tournament. Whoever has the best algorithm for playing a card should win. Games will be simulated thousands of times to minimize the impact of luck in the games.

Please read this ENTIRE DOCUMENT before asking any questions. It is a lengthy document, but most of your questions will be answered by reading it.

Getting Started

You will find several java files attached that provide helpful information for your implementation.

- Card.java (DO NOT MODIFY)
  - This defines a card object within the game.
  - The relevant instance methods that you'll need are:
    - Color getColor() //returns the color of this card. Color.NONE if wild
    - int getNumber() //returns the numeric value of this card -1 if a special card
    - Rank getRank() //returns the type of card
- GameState.java (DO NOT MODIFY)
  - This class contains information pertaining to the current state of the game that the player has vision of.
- UnoPlayer.java (DO NOT MODIFY)
  - This defines an UnoPlayer and what methods are needed to implement one. You will not write your implementation in this file.
- netid_UnoPlayer.java (MODIFY and hand in)
  - This class will define your implementation of an UnoPlayer. All of your code should go in this file. For this milestone, you will implement play() and chooseColor().
  - **Don't forget to replace netid with your netid!**
- TestCaseProcessor.java (DO NOT MODIFY)

    - A program to help you test your code by running it through some simulations. Note: this will tell you nothing about how good of an algorithm you have developed, only if it follows all of the rules of Uno.

UnoPlayer

The UnoPlayer.java file contains what is called a "Java interface". You do not need to understand anything about how works it right now except for what is in this description. What *is* important about this file is the two lines that begin with "public enum."

There are two "enumerated types" to represent the colors and the ranks of the cards in the program. Essentially, what this does is add to the basic list of Java data types (int, double, etc.) Now, in addition to having a variable of type "int" or type "double," you can have a variable of type "Color" or "Rank."

The way you specify a value of one of these types is to prefix one of the capitalized words with "Color." or "Rank." For instance, here is some legal code:

```
int x = 5;
double y = 3.14;
Color myFavoriteColor = Color.BLUE;
Rank aPowerfulRank = Rank.WILD_D4;
```

There's really nothing more to it than that. Just be aware that the way you say "green" in the program is "Color.GREEN", and you'll be fine.

netid_UnoPlayer

The netid_UnoPlayer.java file has detailed comments describing the **play()** and **callColor()** method you are to write code for. ***Reading*** **these detailed comments is an excellent and praiseworthy idea.** Note that the play() method takes four parameters. Here is its method signature:

```
public int play(List<Card> hand, Card upCard, Color calledColor,
    GameState state);
```

30 pts for implementation

Collectively, these arguments tell your method:

1. What cards are in your hand
2. What card is the "up card"
3. What color was called (this argument only has relevance if the "up card" is a wild)

4. A way to find out other miscellaneous things about the state of the game, for your use in building a sophisticated strategy. (Ignore this parameter for this milestone. We will provide more details in Milestone 3)

Your job is to write code that returns the integer index of the card you wish to play. In the event that you cannot play any card, you should return -1 from this method. **(Note that returning a -1 for a hand in which you *can* legally play is an error).**

The callColor() method is simpler. It takes only one parameter, telling you what's in your hand:

```
public Color callColor(List<Card> hand)  10 pts for implementation
```

My code will call this method of yours when you have just played a wild and I need to know what color you want to call. **It must return one of the four valid Color values (*not* Color.NONE.)**

Testing

For your convenience, we have included a testing program for simulating some games. We encourage you to use this to test your code for bugs and errors. To run the test program simply call:

```
javac Card.java UnoPlayer.java netid_UnoPlayer.java TestCaseProcessor.java
java TestCaseProcessor netid
```

If your program does not pass this test, you will receive 0 credit for this Milestone.