219116/117 Programming I

Extended Lab 1

Magic: The Lambda-ing

In this extended lab, you will be implementing a card game called Magic: The Lambda-ing (created for a course at UC Berkley, USA). Download the zip file that comes with this extended lab to get started.

This game uses several different files.

- Code that you have to complete in this lab can be found in classes.py.
- Some utility for the game can be found in cardgame.py, but you won't need to open or read this file. This file doesn't actually mutate any instances directly - instead, it calls methods of the different classes, maintaining a strict abstraction barrier.
- If you want to modify your game later to add your own custom cards and decks, you can look in cards.py to see all the standard cards and the default deck; here, you can add more cards and change what decks you and your opponent use.
- A sample outcome for this lab is given in the file sample_session_result.txt.

Rules of the Game:

There are two players. Each player has a hand of cards and a deck, and at the start of each round, each player draws a random card from their deck. If a player's deck is empty when they try to draw, they will automatically lose the game. Cards have a name, an attack statistic, and a defense statistic. Each round, each player chooses one card to play from their own hands. The card with the higher power wins the round. Each played card's power value is calculated as follows:

(player card's attack) - (opponent card's defense) / 2

For example, let's say Player 1 plays a card with 2000 attack/1000 defense and Player 2 plays a card with 1500 attack/3000 defense. Their cards' powers are calculated as:

P1: 2000 - 3000/2 = 2000 - 1500 = 500

P2: 1500 - 1000/2 = 1500 - 500 = 1000

For example, let's say Player 1 plays a card with 2000 attack/1000 defense and Player 2 plays a card with 1500 attack/3000 defense. Their cards' powers are calculated as:

So Player 2 would win this round.

The first player to win 8 rounds wins the match!

Cards are split into Tutor, TA, and Professor types, and each type has a different effect when they're played. All effects are applied before power is calculated during that round:

- A Tutor card will cause the opponent to discard and re-draw the first 3 cards in their hand.
- A TA card will swap the opponent card's attack and defense.
- A Professor card will add the opponent card's attack and defense to all cards in their deck and then remove all cards in the opponent's deck that share its attack or defense!

These are a lot of rules to remember, so refer back here if you need to review them.

1. Making Cards

*For this question, you must complete the part marked *** YOUR CODE HERE *** in the Card class in the file classes.py*

To play a card game, we're going to need to have cards, so let's make some! We're gonna implement the basics of the Card class first.

First, implement the Card class constructor in classes.py. This constructor takes three arguments:

- the name of the card, a string
- the attack stat of the card, an integer
- the defense stat of the card, an integer

Each Card instance should keep track of these values using instance attributes called name, attack, and defense.

You should also implement the power method in Card, which takes in another card as an input and calculates the current card's power. Check the Rules section if you want a refresher on how power is calculated.

2. Making a Player

*For this question, you must complete the part marked *** YOUR CODE HERE *** in the Player class in the file classes.py*

Now that we have cards, we can make a deck, but we still need players to actually use them. We'll now fill in the implementation of the Player class.

A Player instance has three instance attributes:

- name is the player's name. When you play the game, you can enter your name, which will be converted into a string to be passed to the constructor.
- deck is an instance of the Deck class. You can draw from it using its .draw() method.
- hand is a list of Card instances. Each player should start with 5 cards in their hand, drawn from their deck. Each card in the hand can be selected by its index in the list during the game. When a player draws a new card from the deck, it is added to the end of this list.

Complete the implementation of the constructor for Player so that self.hand is set to a list of 5 cards drawn from the player's deck.

Next, implement the draw and play methods in the Player class. The draw method draws a card from the deck and adds it to the player's hand. The play method removes and returns a card from the player's hand at the given index.

After you complete this problem, you'll be able to play a working version of the game! Type

```
python3 cardgame.py
```

The following code-writing questions will all be in classes.py.

For the following sections, do not overwrite any lines already provided in the code. Additionally, make sure to uncomment any calls to print once you have implemented each method. These are used to display information to the user, and changing them may cause you to fail tests that you would otherwise pass.

3. Tutors: Flummox

*For this question, you must complete the part marked *** YOUR CODE HERE *** in the TutorCard class in the file classes.py*

Implement the effect method for Tutors, which causes the opponent to discard the first 3 cards in their hand and then draw 3 new cards. Assume there at least 3 cards in the opponent's hand and at least 3 cards in the opponent's deck.

Remember to uncomment the call to print once you're done!

4. TAs: Shift

*For this question, you must complete the part marked *** YOUR CODE HERE *** in the TACard class in the file classes.py*

Let's add an effect for TAs now! Implement the effect method for TAs, which swaps the attack and defense of the opponent's card.

5. The Professor Arrives

*For this question, you must complete the part marked *** YOUR CODE HERE *** in the ProfessorCard class in the file classes.py*

A new challenger has appeared! Implement the effect method for the Professor, who adds the opponent card's attack and defense to all cards in the player's deck and then removes all cards in the opponent's deck that have the same attack or defense as the opponent's card.

After you complete this problem, we'll have a fully functional game of Magic: The Lambda-ing!

See also a sample outcome for this lab in the file sample_session_result.txt.

Submission:

- ***Create StudentID_Firstname_lab_ext1 folder, where StudentID is your KU ID and Firstname is your given name***
- ***Put the files to submit - classes.py, cardgame.py, and cards.py - into this folder***
- ***Zip the folder and submit the zip file to the course's Google Classroom before the due date***