“Rock-Paper-Scissors-Lizard-Spock”

Programming Lab 2

Concepts of Programming Languages
CSCI305, Spring 2013

Due: March 20, 2013 at 11:59 pm

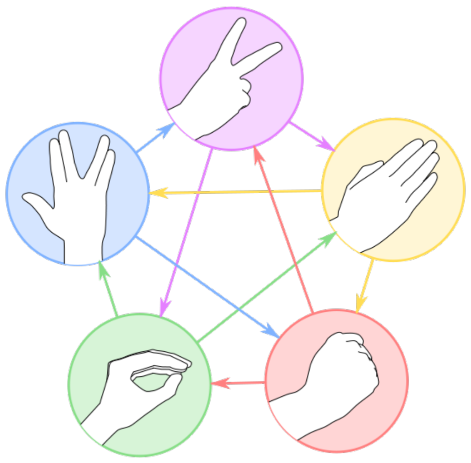
Python

For this lab you will use Python. Python is already installed on all machines in the lab. To install Python, visit `http://www.python.org/download/`.

Game Overview

Rock-paper-scissors-lizard-Spock is a five-gesture expansion of the classic game rock-paper-scissors.The game was invented by Sam Kass, but popularized in this clip from the TV Show “Big Bang Theory”: `https://www.youtube.com/watch?v=x5Q6-wMx-K8`. You may also find the wikipedia page useful: `http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock`.

This diagram explains the outcomes of the game:



For this Lab, you will implement this game, in Python, in an object-oriented way. Failure to implement the game using OO paradigms (described below) will result in a significant loss of points.

Element Class

First create a class named `Element`. This class will serve as the superclass to five subclasses: `Rock, Paper, Scissors, Lizard, Spock`.

The `Element` class has one instance variable `_name`, which will store the name of the Element (e.g., "Rock", "Lizard", etc). The constructor will take a name as a parameter and will save it to the instance variable `_name`. Next, create a getter method called `name()` that returns the instance variable `_name`. Lastly, create a method `compareTo` that takes an instance of the class `Element` as an argument. In the superclass `Element`, leave this method undefined, using the line `raise NotImplementedError("Not yet implemented")` to throw an exception. You will instantiate the method in the five concrete subclasses.

As the next step, create your five subclasses: `Rock, Paper, Scissors, Lizard, Spock`. For each you should only need to implement (override) the `compareTo` method. Python allows multiple return types. Your `compareTo` method will the compare the argument against its own `_name` and determine the outcome, returning two strings. The first is string describing one of the following outcomes:

- Scissors cut Paper
- Paper covers Rock
- Rock crushes Lizard
- Lizard poisons Spock
- Spock smashes Scissors
- Scissors decapitate Lizard
- Lizard eats Paper
- Paper disproves Spock
- Spock vaporizes Rock
- Rock crushes Scissors

For a tie, you can output a string such as "Rock equals Rock". The second return value will return one the following round outcomes:

- Win
- Lose
- Tie

Now create a concrete instance of each of the five Elements and store these in a global list named `moves`.

Self-Check

You can now test your Element classes. For example, the code:

```
rock = Rock('Rock')
paper = Paper('Paper')
print rock.compareTo(paper)
print paper.compareTo(rock)
print rock.compareTo(rock)
```

should yield the following output:

```
('Paper covers rock', 'Lose')
('Paper covers rock', 'Win')
('Rock equals rock', 'Tie')
```

Player Class

Next you will create a series of classes for the Players. Begin with a superclass named `Player`. This class has one instance variable `_name`. Create a getter method called `name()` that returns the variable `_name`, which is set via the constructor. Also create a method `play()`. In the superclass `Player`, leave this method undefined, using the line `raise NotImplementedError("Not yet implemented")` to throw an exception. You will instantiate the method in the concrete subclasses.

Now you are ready to create the concrete Player classes. Begin with a class named `StupidBot`. For this class, define the `play()` method to return the same move every time (e.g., `Rock`, `Paper`, etc). Just pick a single move and have your `StupidBot` play this move every time.

Next, create the class `RandomBot`. This Player will randomly pick and return one of the five possible moves from your `moves` list. Your next Player, `IterativeBot`, begins with one move and cycles through all the moves, one by one, repeating the sequence only after having played all five moves. Player `LastPlayBot` will always play the move that the opponent played on the previous move. You may implement this feature however you choose, but will explain in your Lab Report. For this Player's first move, you may arbitrary pick any move.

Next, you will define a `Human` class. This Player will ask the user to determine the move. For each turn, the `play()` method will print the options and request input from the user, as in Figure 1. Be sure to only accept valid moves from the user.

```
(1) : Rock
(2) : Paper
(3) : Scissors
(4) : Lizard
(5) : Spock
Enter your move: 6
Invalid move.  Please try again.
Enter your move: 2
```

Figure 1: Sample output of a move from the `Human` Player

Lastly, define a class `MyBot`. This Player can employ any strategy you determine that **differs** from the other Players described above. You will describe your strategy in your Lab Report.

Self-Check

You can now test your Player classes. For example, the code:

```
p1 = StupidBot('StupidBot')
p2 = RandomBot('RandomBot')
p1move = p1.play()
p2move = p2.play()
print p1move.compareTo(p2move)
```

might yield the following output:

```
('Rock crushes Scissors', 'Lose')
```

Main Class

You will define a main class that will simulate a game of Rock-Paper-Scissors-Lizard-Spock. Your game will play five rounds between two players, determining the winner (or a draw) at the conclusion of the game. A sample game output is shown in Figure 2.

Begin with a welcome message that also displays your name. Next have the user select Player1 and Player2 from a list. Again ensure that the user can only make valid selections of one of your six Players. Since all the Players implement the `play` method, the actual Player class instantiated will not be determined until runtime.

Now, using a loop structure, play five rounds of Rock-Paper-Scissors-Lizard-Spock. For each player, print out the move selected. Also print out the result description (e.g., Rock crushes Scissors) and determine the round winner. Your output should resemble the example output in Figure 2. You should be keeping score so that you can determine the game winner after the five rounds. Print out the game winner.

Troubleshooting

This lab requires an independent study of the Python language. You are encouraged to use any web tutorials and resources to learn Python. Given the size of the class, I will not be able to debug your code for you. Please do not send panicked emails requesting I fix your bug for you. Allow yourself plenty of time, and use patience, perseverance, and the internet to debug your code. I will gladly answer clarifying questions about the goals and instructions of the Lab assignment.

Lab Questions

Question 1: Describe your Player `LastPlayBot`. How did you implement this strategy?

Question 2: Describe your Player `MyBot`, explaining the strategy you employed and how you accomplished it.

Question 3: Using the course notes and any web resources of your choosing, explain the type system of Python, giving attention to the concepts of binding time, dynamic vs. static typing, strong vs. weak typing, and user-defined types (classes). Cite any sources you used other than class discussion or the textbook.

Question 4: Play a number of games, selecting your various players. Do you notice any trends? Are you, as the Human Player, able to beat any of the Bots on a consistent basis?

Question 5: Read the wikipedia entry on Normal Form Games (`http://en.wikipedia.org/wiki/Normal-form_game`). Also, review the wikipedia page `http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock`. Is it possible to design a Player strategy that is more likely succeed? Why or why not? Explain in a paragraph. Note that it is possible to answer this question even if you did not finish the Lab.

The following questions are for feedback and evaluation purposes. Points are awarded for any sincere answer.

Question 6: Name something you like about Python. Explain.

Question 7: Name something you dislike about Python. Explain.

Question 8: Did you enjoy this lab? Which aspects did you like and/or dislike?

Question 9: Approximately how many hours did you spend on this lab?

Question 10: Do you think you will use Python again? For which type(s) of project(s)?

```
Welcome to Rock, Paper, Scissors, Lizard, Spock, implemented by <Your Name Here>.

Please choose two players:
   (1) Human
   (2) StupidBot
   (3) RandomBot
   (4) IterativeBot
   (5) LastPlayBot
   (6) MyBot

Select player 1: 2
Select player 2: 3

StupidBot vs RandomBot. Go!

Round 1:
  Player 1 chose Scissors
  Player 2 chose Rock
  Rock crushes Scissors
  Player 2 won the round

Round 2:
  Player 1 chose Scissors
  Player 2 chose Spock
  Spock smashes Scissors
  Player 2 won the round

Round 3:
  Player 1 chose Scissors
  Player 2 chose Paper
  Scissors cut Paper
  Player 1 won the round

Round 4:
  Player 1 chose Scissors
  Player 2 chose Lizard
  Scissors decapitate Lizard
  Player 1 won the round

Round 5:
  Player 1 chose Scissors
  Player 2 chose Scissors
  Scissors equals Scissors
  Round was a tie

The score is 2 to 2.
Game was a draw.
```

Figure 2: Output of a sample game.

Submission

Each student will complete and submit this assignment individually. Do not consult with others. However, you are encouraged to use the internet to learn Python.

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

Save the final version of your program as `[lastname]_[firstname].lab2.py`. Type your lab questions in plain text as `[lastname]_[firstname].lab2.txt`. Include your name in the text file.

I must be able to run your program from the command line with no arguments. For instance, for my named file, I would run the file:

```
>> python Donnelly_Patrick.lab2.py
```

You will email these files to `msucsci305@gmail.com`. Use the subject line `[lastname]_[firstname] Lab2 Submission`. Do not archive your files but instead use two attachments. Email your two files before 11:59pm on the due date. Late submissions will not be accepted.