CSci 1133, Spring 2019
Homework Assignment 5
100 points

Due: 1 May 2019 (before 11:59pm)

Unlike the computer lab exercises, this is ***not*** a collaborative assignment, though you may work in groups of up to two students (or alone). You must design and implement the solution to each problem on your own without consulting anybody other than your one partner, nor can you employ solutions obtained from any source other than those provided in class. Obtaining or providing solutions to any homework problems for this class is considered academic misconduct.

There are 2 programming problems worth 70 points total. **Construct, implement and test a separate Python script for each problem**. Name each script using the following format:

*<X.500 username>*_*<assignment number><problem letter>*.py

where *<X.500 username>* is your UMN user id, *<assignment number>* is the number of the homework assignment (listed above), and *<problem letter>* is the letter designation for the specific problem. Note the underscore following your user id. Here's an example:

`smit0341_5A.py`

for pair submissions, concatenate the names with an underscore:

`smit0341_xang0008_5A.py`

You must name your python files *exactly* as detailed above including all punctuation. Failure to do so will result in loss of points. Instructions for submitting your homework assignment are provided on the class webpage.

On a similar note, your solutions to homework problems should be contained within functions that match the name of the function used in the example *precisely* – the order of inputs and the capitalization of letters in the function name need to be copied exactly. We also expect your functions to properly return in the format we give. Simply printing your result to the interpreter will be seen as an incorrect answer from the script. Failure to comply with these standards will result in loss of points.

Zip all files into a single archive called HW5.zip

A. (*50 points*) Rock-Paper-Scissors AI

In this problem you'll construct a class named `RPSai` that learns to play Rock-Paper-Scissors against a human (or another AI). The problem is broken down into several sub-problems implementing various portions of the AI.

1) *(1 point)* Write a constructor that creates a member variable of type list called `history`, which will store the series of moves played so far. The constructor can also create other member variables as you see fit to produce better AI.

2) *(4 points)* Write a mutator method named `opponentMove()` that takes one parameter, the opponent's move, and stores that move in the object's `history` list. Valid moves are the string "R", "P" or "S". If a non-valid move is given, print "ERROR".

```
>>> p1 = RPSai()
>>> p1.opponentMove("R")
>>> p1.opponentMove("R")
>>> p1.opponentMove("S")
>>> p1.history
['R', 'R', 'S']
```

3) *(5 points)* Write a method called `beatMove()` that takes one parameter, the expected opponent's move, and returns the move that will beat it. The rules of RPS are that: "R" beats "S", "S" beats "P", and "P" beats "R".

```
>>> p1 = RPSai()
>>> p1.beatMove("P")
'S'
>>> p1.beatMove("R")
'P'
```

4) *(15 points)* Write a method called `predictMove()` that returns a prediction of the most likely move the opponent will make. For this problem, your prediction should be whichever move the opponent made most frequently. If there are multiple moves with the same play rate, you can return any move.

```
>>> p1 = RPSai()
>>> p1.opponentMove("R")
>>> p1.predictMove()
'R'
>>> p1.opponentMove("S")
>>> p1.opponentMove("S")
>>> p1.predictMove()
'S'
```

4) *(5 points)* Write a method called `playMove()` that returns the play which will beat the expected move the opponent will make (as determined by `predictMove`). If the opponent has not made a move yet, you may play any move.

```
>>> p1 = RPSai()
>>> p1.opponentMove("R")
>>> p1.playMove()
'P'
```

5) *(20 points)* Write a method called `playMovePro()` that returns the play which returns the best move according to an algorithm of your design. The better your algorithm performs the more points you will get.

5a. *15 points* – There will be several secret test cases to beat. For each test case that you get more than a 60% winrate on, you will get points. These tests will include simple patterns such as "RRRRRR", "PRPRPR…", as well as more complex AI opponents. Your AI will be evaluated over at least 200 turns for each opponent.

Here is some example test code.

```
def winner(p1,p2):
    if p1+p2 in ["RS","SP","PR"]:
        return 1
    if p1==p2:
        return 0
    else:
        return -1

comp = RPSai()
wins = 0
games = 0
while games < 200:
    move = "P"
    if games%2 == 1:
        move = "S"
    cpuMove = comp.playMovePro()
    comp.opponentMove(move)
    w = winner(cpuMove,move)
    #print(cpuMove, move, w)
    if w == 1: #we win
        wins += 1
    if w == 0: #tie
        wins += .5
    games += 1
print("Win rate: %.3f"%(wins/games))
```

5b. *5+ points* – We'll hold an automated tournament. If your program conforms to specifications you get 5 points. Programs in the top half of performance will receive extra points as follows (non-cumulative).
50-70 percentile: 1 extra point
71-80 percentile: 2 extra points
80+ percentile: 3 extra points
3rd place: 4 points
2nd place: 7 points
1st place: 10 points

T. Tournament Entry
Submit *one file per partnership* to Canvas with your tournament-ready code. This should be named as follows: **`name01_name02_5T.py`** for example:

`smit0341_john0125_5T.py` if you don't have a partner name the file as you normally would, e.g.: **`smit0341_5T.py`**

This file should be submitted to the tournament submission link on Canvas. Submit the file directly, do not zip it. Again: only one submission per partnership. You may not use a late day on the tournament entry!

B. (*20 points*) Custom Markup Language

For this problem, you are to implement a custom markup language for annotating how text is displayed. You will implement this language by translating from this custom language to HTML.

Write a function called `markup2HTML()` which takes in a single parameter that is the name of a file. Your code should write a new file, with the last three characters replaced with html. So if passed in *test.txt*, your code should write out a file named *test.html*.

The markup language is described as follows:

New lines should appear as new lines (using a single
 tag)

Text between two *'s should be **bold** (using   tags)

Text between two _'s should be underlined (using <u> </u> tags)

Text between two /'s should be *italicized* (using   tags)

Text between the pair of characters -- and matching -- should written in ~~strikethrough~~ (using <strike> </strike> tags)

Text between an starting #RGBCOL and a # should be written in that color (using   tags)

The html file you output should always start with the following text:

```
<html><head> <title> CSCI1133 Markup </title> </head><body>
```

and should always end with:

```
</body></html>
```

Here is my suggested strategy:

-First, write code that replaces all newlines (\n) with the html tag for newlines

-Then, expand your code to replace the first * with a the next with a and keep alternating through the text

-Following the strategy you used to turn * to or , also turn _ to <u> or </u> and / to or

-Next think about how to extend your code to pairs of characters like -- to turn the text between them to strikethrough using <strike> and </strike>. I had to completely restructure my code turning my for-loop into a while-loop.

-Finally, we want to turn #ff0000 red text # into red text . This is a fancier version of what you had to do to get strikethrough working.

A sample markup file as the associated HTML output is available on the course webpage.