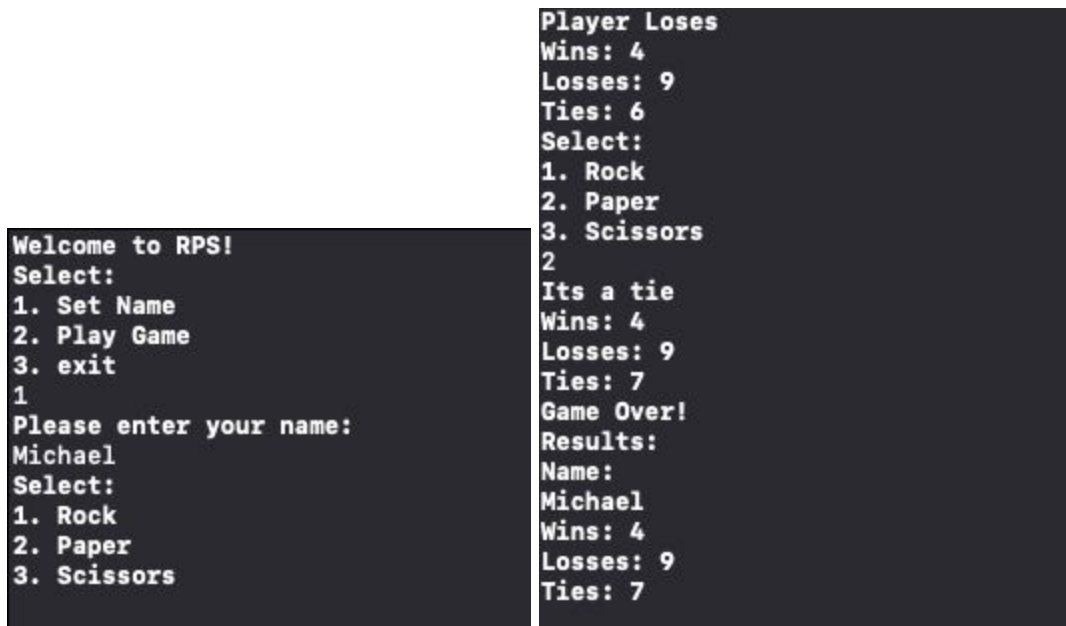


Thinh Le
Alex Escalante
Michael Ong
Sophia Yaksick
CMPE 135
Prof. Ron Mak
10/5/2018

Assignment 4: RPS with Machine Learning

- Your interface for the computer's choice algorithm.

The simple Rock, Paper, Scissors runs on a command line interface system. This means that the player is asked for their name and makes choices directly. The computer has been programmed to have a cache memory system that monitors a player's moves in a set of 5. If the computer has no moves already stored to rely on, then the system randomly chooses moves (i.e. `randomchoice()`) with the addition of recording the selected choices of the player during that game. As the reference data is incremented, the data is then used to supply the computer with a higher win rate based on the probability for future games. The machine learning algorithm (i.e. `educatedchoice()`) stores a player's previous moves in an external file where the computer is able to read and compare the player's moves.



```
Welcome to RPS!
Select:
1. Set Name
2. Play Game
3. exit
1
Please enter your name:
Michael
Select:
1. Rock
2. Paper
3. Scissors

Player Loses
Wins: 4
Losses: 9
Ties: 6
Select:
1. Rock
2. Paper
3. Scissors
2
It's a tie
Wins: 4
Losses: 9
Ties: 7
Game Over!
Results:
Name:
Michael
Wins: 4
Losses: 9
Ties: 7
```

Figure 1: Shows the terminal output results and what is typically seen during a game

- Your implementation of the simple ML algorithm.

The players moves stored in the external file are set groups of 5 followed by a count that monitors the frequency of occurrence done by the player. In this form, a players most likely choices are able to be read by the computer and counter measures are able to be taken. For example, if a player has chosen RRRRR: {3}, RPRPR: {10}, RRRRP: {2}, PPPPP: {1}, SPSPR {4}. The computer will decided that the best countermeasure for the next move the player does is play paper bc it has the highest probability {10} of being played. The computer does this by going into the external file, check the already stored data to see the player moves. If the 4 choices done by the player are already recorded and show high frequencies of occurring in the same pattern, then the computer takes that information to decide the players next move and most likely move that will lead to a win. As the data continues to grow, the computer will be able to have larger and larger reference patterns, and thus lead to a higher win rate .

- Are your classes cohesive? Loosely coupled?

The classes in our programs represent different primary responsibilities with unique, single word names that are simple to distinguish. They also consolidate functions and members that are of relation to the classes, keeping everything organized and cohesive. These classes create the control network of exchanging information along a data path to follow through the machine learning algorithm, which provides educated guesses for the game based on historical data. Therefore, the classes provide cohesion through their primary purpose within the program. The following names are the classes used in our program: Player, Computer, Human, GameInterface.

The GameInterface class is the highest level in the program since it directly communicates with the user who is playing the game along with the second player, which is the machine learning based computer. GameInterface features many ways of communicating messages to the user to guide them along the game which includes the functions welcome, menu, results, and game. These functions provide standard output to the terminal console of the user that indicate the start of a game, the progress along a game, and the end result of a game. Besides performing text display output functionality, the GamerInterface class also features a comparison function and game count tracker. The compare function is the program's handler for determining victories between the user and computer players.

The purpose of the Player class is combining the data members for tracking the score, wins, losses, and ties. The string "name" is an identification of the player in the game (i.e. the player's name). Lastly, the char "choice" will be the data member that holds the Player's decision between r for Rock, p for Paper, and s for Scissor. This class acts as a general medium to take input to assign to its members and pushes output to other classes that are accessing it. To set this choice, we include a set function called "setChoice" which will assign the characters "r", "p", or "s" for the later comparison between the computer and the player. Player behaves as a sort of transfer layer for communicating specific information from the child classes to the high level interface above it.

The purpose of the Human class acts as a carrier of information for the player class. Player contains a majority of the meta data that is needed to configure winners for the game. Human features a probability function which follows the subset rule stated in the assignment to determine how the Computer will be able to predict the next move. The pattern recognition for the machine learning will be used in the Human class. Although it is not implemented right now, the Human class will perform read and write operations to a .txt file which will extend the cache to provide a memory base for extracting patterns and improve decision making.

The purpose of the Computer class is to hold functions that help the computer decide on what choice to make. These functions include `educatedchoice`, `randomchoice`, and a private member that keeps track of the choices being made by the player `choicecount`. Since the Player class is the parent class, Computer holds the same private values as the Player class while having its own functions to make choices to combat the player. This keeps the Computer class and player class separate and loosely coupled.

- Do they obey the Law of Demeter?

The RPS game implements a class to display information that is the most relevant to the user without exposing the underlying intricacies of the program. The passage of information between class to class is prevalent within our program through accessor and mutator functions. For example, the `GameInterface` class has functions which pass the Human and Computer classes as cohesive objects to declare and assign values to. `GameInterface` only needs to know the choice that was made for the game function since that will pit the input of Human against the input of Computer. There is no need for `GameInterface` to know the name of the Human object or the amount of times it has tied. The focus of `GameInterface` is centered on one specific attribute so the process is straightforward and reduces code complexity for modularity and swappable functions.

- Your ability to swap algorithms with minimal code change.

In the future, we intend to create instances where the computer will be able to switching amongst algorithms to increase its win rate through a centralized class that encapsulates all the algorithms accessible to the computer. However, we would need to add new algorithms that can increase the computers win rate in a smooth manner. We would also need to include some way to keep track of how well each algorithm will work against the player and use that as a way to switch between algorithms (ie. count of some kind). We would also consider taking advantage of the output file that we would create in order to better understand the way the player selects the choices of rock, paper, or scissors and use that data to create or choose a better algorithm to use. For example, if the player chooses rock frequently, we would choose an algorithm that is based on the frequency of something being chosen. Likewise if there is a pattern that the player is using, we would choose an algorithm that can read the pattern and so forth.