Assignment #4
Group: Benedict + 3

Rock-Paper-Scissors Game with Simple Machine Learning

Interface Design for the Computer's Choice Algorithm

In order to allow for different algorithm choices for future projects, we implemented an abstract class for the computer algorithm which provides function declarations for the most essential functions that can translate over to different types of Rock, Paper, Scissors implementations. Our interface for the computer algorithm choice, chooseAi.h, declares a play function, print function, and export function, along with a deconstructor. We believe from an interface stand point, these three functions are the most necessary functions in order to allow for scalability of multiple choice algorithms that could be implemented for future iterations of this game. The play function is the most necessary function within this interface as every single algorithm will need to implement it's own version in order to make its weapon choice. The print and export functions are important, however those are more for a programming point of view. The print function was included, since giving the player and programmer an easy to read table of their choices throughout the game along with the statistics, is helpful for any type of implementation for the computer choice algorithm. Following along with this idea, exporting this table will allow for a database of player moves, which can help develop other algorithms, or make more sophisticated machine learning algorithms.

Following the explanation behind the interface developed for the computer's choice algorithm, two classes were created based off of this interface. The first class is randomAI, which is intended to make the computer's weapon choice based off of a random choice between Rock, Paper, or Scissors. In this implementation, there is no need for the importing and printing of the player choice table, as this recorded data will provide no improvements for the computers weapon choosing. The second class is the machineLearningAI, which is intended to make the weapon choice of the computer based off of the players choice pattern. Since the data of the player's moves are needed for the machine learning algorithm, the print and export functions are defined. This will allow for the player's choices to be exported into a database to improve the machine learning of the program and allow the programmers an easier way to see if the algorithm is working as expected.

Along with the classes which alter how the computer makes it's weapon choices, there is another class which follows the factory method for choosing the algorithm used by the computer. This class is called the factoryChooseAI, which generates a new computer object that follows a specific algorithm in weapon choosing. In the case of this current iteration, there are only two objects created, the randomAI and the machineLearningAI. These two objects are chosen based on a difficulty variable. For our case, we labeled our difficulty as “hard”. If it is true, it will keep the machine learning AI, however if it is false it will randomize the AI. This would help keep track of the machine learning AI, and give the user some control of how the machine learning would be regularly used once it fit the conditions. However, this is not limited to these two options and can be expanded upon to include other algorithms in future iterations of the project, which will be easy to add and will require no alterations in other classes.

Swapping the Game Types from Random to ML with Minimal Code Changes

Adding in the simple machine learning algorithm for the computer to try and predict the move of the player by keeping track of the past moves and patterns was the largest change to the code that we had to make. In our Rock-Paper-Scissors game, we made the machine learning algorithm by creating several new classes that would help to encapsulate the code and prevent major changes to the already working program. After finishing the assignment, the program is now able to swap between either the machine learning algorithm or a random choice game, with the option to easily add more algorithms. In order for this to happen, the user must use a console command to swap between the two game difficulties. We made it so that when the user enters in the “-r” command, the CPU will make random moves and not save any of the user’s moves, thus making it unable to learn from past rounds. Alternatively, the user can enter in the “-m” argument that will make the CPU change from the default, or “-r” random, play style to machine learning. This causes the CPU to remember past rounds and try and beat the player by recognizing familiar choices and choosing the most probable weapon to win the round.

We were able to keep most of the code the same after adding in the machine learning algorithm, however in order to not have useless info on the console window during the game, we needed to add a few more lines that made sure the output text was relevant to the current game type. Therefore, we saved the game type as a boolean variable that would allow if statements and some loops to either be operated or ignored entirely. This was one of the ways we were able to reuse our code and avoid problems if there were any contentions within the code. For example, with the output matrix that shows the past move, it is only used when the CPU is using the machine learning algorithm, but at first we had it printing out after each game no matter the game type. But by using the boolean variable, we able able to have the RPSControl class only print out the matrix when the CPU is using ML, or when the boolean value is true.