CMPE 135

Ron Mak

Spring 2019

Assignment #4 Report

Intro

We built upon the previous RPS game by injecting some machine learning and adding a chooser factory. The chooser factory allows the user to pick if they would like the computer to pick their hand randomly or to pick using machine learning.

Implementation of ChooserFactory

Our code can seamlessly swap between the random and simple machine learning algorithms with minimal code change thanks to our use of a factory design pattern. We could have passed a command-line argument, -r for random and -m for machine learning, that would pick the respective chooser. However, we simply prompt the user at the beginning of the game. Each algorithm has a subclass built around them, called a chooser. The chooser class that is used during the program is determined by the chooser factory. Because we are closely adhering to the principle of Polymorphism and the Liskov Substitution Principle, the ML chooser subclass can be replaced with the random chooser subclass in any instance within the program, and vice versa. We will want to implement additional algorithms in the future that would make the program a more formidable rock paper scissors opponent. Thanks to the flexibility afforded by the interfaces of our chooser classes we can implement those algorithms without needing to make any changes to the main code.

Implementation of the machine learning

The machine learning aspect of this assignment had us write down patterns to a text file. The patterns were recorded in the Game Instance Class. We added a member variable called 'pattern' that would record the pattern up to the length that we specified as 5. The pattern length can be changed to another size if necessary. When the pattern reaches the specified length it will invoke a Computer Player Class method called 'toTextFile()' which will save the pattern to a text file with the fstream c++ class. 'toTextFile()' checks the text file to see if the pattern exists. If it exists, we update the frequency, otherwise we put in a new entry.

The machine learning algorithm was a little tricky. For all intents and purposes, we will say that the pattern length is 5. The pattern is defined as, user's choice, CPU's choice, user's choice, CPU's choice, etc. Once the pattern length is equal to 4, the CPU makes a prediction about what the player will pick next based on the occurrence of the patterns that contain those 4 and what the 5th hand will most likely be. The Computer Player Class implements a new member function called 'makePrediction().' This function takes in the pattern, which is of length 4, and checks the text file for any matches. If there is a match, the CPU saves the prediction in a new member variable called 'prediction.' This means that on every third round, the CPU uses its prediction against the human player.

Cohesiveness

Adding on the machine learning algorithm did not disrupt the cohesiveness of each class. With the addition of the ML algorithm, the two classes that had modifications were GameInstance class and the ComputerPlayer class. In addition to those changes, there is a brand

new ChooserFactory class that allows the user to choose to play against an RNG computer or the simple ML computer.

Loose-Coupling

With the implementation of the the ML algorithm, the ComputerPlayer class grabs data from the "recordHand" function in the GameInstance class and makes a prediction accordingly. If the conditions between the GameInstance and ComputerPlayer hand are not met, then computer will only play random hands (relying only on itself).

Law of Demeter

With the addition of the ML algorithm, the ComputerPlayer class gained an important private value: the "prediction" char. The GameInstance class does not know about such a value and runs its functions normally. Once the GameInstance calls for the ComputerPlayer to set its hand, the predicted hand will automatically be set and returned back to the GameInstance without it ever knowing what happened in the the ComputerPlayer class.

Conclusion

This assignment was another way for us to practice our Object-Oriented-Design Principles. We compiled from the command line with: `g++ *.cpp -o output` followed by: `./output`. If you encounter any problems while compiling/running, please contact us or check out the github: https://github.com/lastjediluke/RPSLS_135.