San José State University
Department of Computer Engineering

CMPE 135
Object-Oriented Analysis and Design

Fall 2018
Instructor: Ron Mak

Assignment #4

Assigned:	Tuesday, September 25
Due:	Friday, October 5 at 11:59 PM
	Team assignment, 100 points max

RPS Game with simple machine learning

Human players of the Rock Paper Scissors game try to develop **strategies** to beat their opponents. Therefore, humans generally do not make random choices. Instead, their choices exhibit **patterns** that a computer can discover and exploit using a simple **machine learning** algorithm.

The computer's ML choice algorithm

Continuously record the last *N* choices between the human and the computer player. Throw out the oldest choice in order to add a new one. For example, suppose *N* = 5 and during the game, the recorded choices are (the human's choices are underlined):

PSRSP

The last choice was made by the human, and it was paper.

For each recorded sequence that ends with the human's choice, the computer should store how many times that sequence has occurred (each sequence's frequency).

For example, for *N* = 5, some of the stored sequences and their frequencies may be (in no particular order, the human choices are underlined):

RSPSR:1, **SPRPP**:3, **RSPRS**:2, **RSPSS**:4, **RSPSP**:3, **SSRPP**:2

Now suppose during the game, the last four choices are **RSPS.** In other words, in the last round, the human chose paper and the computer chose scissors. The computer can predict that the human will most likely next choose scissors, since **RSPSS** appears more times (4) than **RSPSR** (1, predict rock) and **RSPSP** (3, predict paper) in the stored frequencies. Therefore, the computer should choose rock to beat the human's predicted choice of scissors. After the human makes a choice, update the appropriate frequency.

If a sequence is not in the store, then the computer can make a random choice.

As the computer plays more games and stores more frequency data, it becomes increasingly more capable of predicting the human's next choice. Over the long haul, the computer will become very difficult for humans to beat.

Design an interface

Update your Rock-Paper-Scissors game program from Assignment #3 by adding the simple machine learning (ML) algorithm for making the computer's choice in each round. (You did encapsulate how the computer makes its choice, right?)

Design an interface (abstract class) with pure virtual member functions to represent the computer's choice algorithm. Implement the interface with the random choice algorithm and with the simple ML choice algorithm. You should be able to swap between the two algorithms with minimal changes to the rest of the code.

Save the stored frequency data at the end of each game to be read by the next game. This way, the ML algorithm can accumulate data over multiple games.

Tip

Test your computer's ML choice algorithm by playing the game (as the human player) with an obvious pattern, such as always choosing rock, or with a fixed sequence, such as rock, paper, and scissors.

Written report

In a short (2- or 3-page) report, describe:

- The design of your interface for the computer's choice algorithm.
- How your code can swap between the random and the ML algorithms with minimal code change.

What to turn in

Make a zip file of all your C++ source files and your report. Include two text files containing output from playing two games, one with the computer's random choice algorithm and one with the ML algorithm.

Don't worry if the ML algorithm isn't winning more than losing:

- It takes a number of games before the ML algorithm becomes "smart" enough.
- You will get better results if you have unsuspecting friends play against your program. You know what your program is doing for the computer's choices, so you will have a major advantage.

Submit it into Canvas: **Assignment #4.** This is a team assignment. Each member of the team will receive the same score.

Rubric

Your program will be graded according to these criteria:

Criteria	Max points
As implemented by your program and described by your report: ○ Your interface for the computer's choice algorithm. ○ Your implementation of the simple ML algorithm. Are your classes cohesive? Loosely coupled? Do they obey the Law of Demeter? ○ Your ability to swap algorithms with minimal code change.	100 ○ 25 ○ 50 ○ 25

Thoughts to ponder

Would the ML algorithm be "smarter" if you used a larger value for *N*? What if you recorded several throw sequences simultaneous for various values of *N* (for example, *N* = 3, 4, 5, 6, 7, etc.) and then somehow made a prediction from among all of them?

Can you devise a better computer's choice algorithm? If we did a tournament that pitted each team's program against the other teams, would your program win more games?