San José State University
Engineering Extended Studies

CMPE 202
Software Systems Engineering

Section 47

Spring 2026
Instructor: Ron Mak

Assignment #6

Assigned: Wednesday, March 18
Due: Friday, March 27 at 11:59 PM
Team assignment, 200 points max

Class design and the Rock-Paper-Scissors game

This assignment will give you a chance to demonstrate iterative development and good class design using design principles and design patterns.

Write an application that implements a Rock-Paper-Scissors game with two players. One player is human, and the other player is the computer. The computer can play using one of two strategies — one where it simply makes random choices, and the other where it uses some basic machine learning to make more optimal choices.

Your application should demonstrate the use of appropriate design principles and design patterns that we covered in class. Use an iterative approach during development.

Basic game rules

Rock beats scissors, scissors beat paper, and paper beats rock. See https://www.wikihow.com/Play-Rock,-Paper,-Scissors

Your application should play a game of 20 rounds. During each round, the human player makes a choice by typing **R**, **P**, or **S** in response to a prompt to choose rock, paper, or scissors, respectively. The computer makes its choice (without "seeing" the human player's choice, of course). Display whether the human won, or the computer won, or it was a tie if both players made the same choice. Rock beats scissors, scissors beat paper, and paper beats rock. At the end of the game, display the final score.

The computer can play using either of the two algorithms to make its choices. Before the start of each game, use a command-line option or a prompt to determine which algorithm the computer will use.

Create a "game engine" to control the game (prompt the human player each round for a choice, obtain the computer's choice, keep track of the score, etc.). The engine should not need to know which algorithm the computer is using.

The random algorithm

For each round, the computer simply chooses rock, paper, or scissors randomly. You can use the C++ random number generator function **`rand()`**.

The smart algorithm

Human players of Rock-Paper-Scissors try to develop personal algorithms to beat their opponents. Therefore, human players generally do not make random choices. Instead, their choices exhibit patterns that a computer can discover and exploit using a machine learning (ML) algorithm.

Here's how to implement a very simple ML algorithm that enables a computer to predict with increasing accuracy what the human player's choice will be for each round.

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 player'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, and the human player's 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 player chose paper and the computer chose scissors. The computer can predict that the human player will most likely next choose scissors, since **RSPSS** appears more times (4) than **RSPSR** (1 time, predict rock) and **RSPSP** (3 times, predict paper) in the stored frequencies. Therefore, the computer should choose rock to beat the human player's predicted choice of scissors. Each time after the human player makes a choice, use the sequence to update the appropriate stored frequency.

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

At the end of each game of 20 rounds, write the frequencies to a disk file. At the start of each game, read the frequencies. As the computer plays more games and obtains more frequency data, it becomes increasingly better at predicting the human player's next choice. Over the long haul, the computer will become very difficult for human players to beat, especially those who don't know the computer's smart algorithm.

Instead of a Large Language Model (LLM) for a chatbot, is this a Large Game Model (LGM) for Rock-Paper-Scissors?

What to turn in

Each team designs and implements the Rock-Paper-Scissors application. The application should include both the random and the smart algorithms to play against a human player. Determine which algorithm to use before a game starts either by a command-line option or a prompt. Your game engine should not depend on which algorithm is in use.

Submit a zip file named after your team that contains your C++ source files and a short report that describes:

- Your iterative development. What were your major iterations and what was accomplished during each one? Was there any significant backtracking?
- A UML class diagram at an appropriate level of detail of your primary classes and their relationships.
- Which appropriate design principles you used and how did you used them (cohesion, loose coupling, hidden implementations, coding to the interface, encapsulate change, etc.)
- You should be able to use either the Template Method Design Pattern or the Strategy Design Pattern, or both. Describe how you used one or both design patterns. Full credit if you managed (reasonably) to use both design patterns.

Rubric

Your submission will be graded according to these criteria:

Criteria	Max points
Iterative development • What was completed during each iteration and any significant backtracking.	**20**
Well-designed classes • UML class diagrams. • Good class relationships (dependency, aggregation, inheritance). • Well-chosen names of classes, member functions, and member variables.	**60** • 25 • 25 • 10
Descriptions of design principles used by the application.	**30**
Descriptions of design pattern(s) used by the application.	**30**
Working application • Rock-Paper-Scissors game, human player vs. computer. • Random algorithm for the computer. • Smart algorithm for the computer. • Game engine doesn't depend on which algorithm is used.	**60** • 25 • 5 • 25 • 5

Thoughts to ponder

Would the ML algorithm be “smarter” if you used a larger value for *N*? What if you recorded several sequences simultaneously 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?