|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LEARNING PROFILE FOR ASSIGNMENT#\_2\_\_\_\_\_ AND QUESTION#\_2\_\_\_\_\_\_\_** | | | | | |
| *Name* | *:* | *Steven Morrissey* | *Due Date* | *:* |  |
| *Student ID* | *:* | *3300222* | *Submission Date* | *:* |  |

**1. Problem Statement:**

Write a program that plays the Rock-Paper-Scissors-Lizard-Spock game. Refer to http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock for more information. Normally, one player is a human and the other is the computer program. However, in this exercise, the program will generate two players who play against each other. The play continues until either of the computer-generated players wins four consecutive times. In this game, two random integers are generated in the range of [1 to 5], one per player. 1 refers to Rock, 2 refers to Paper, 3 refers to Scissors, 4 refers to Lizard, and 5 refers to Spock. For example, if the computer randomly generates integers 2 and 5 in the first iteration, 2 is for the first player and 5 is for the second player. Based on Rule 8 in the following 10 rules, Paper (2) disproves Spock (5), so Player 1 wins. Repeat it to generate one more pair and determine who wins that iteration. Continue the iterations until one player wins four consecutive times.

**2. Description of the Code:**

Most of the game logic is done in the play() method. From the api/requirements I thought it looked like thats where it was wanted. The play()method then uses all the other methods defined in the class to make the game work.

First we check for consecutive wins to make sure the win condition is not met, then inside the loop the dice are rolled for the 2 players. Then the outcome is evaluated and the winner is decided. If the current winner has either won already or consecutive wins are 0 we increment consecutive wins to keep track. Then the loop restarts and we again evaluate consecutive wins.

**3. Errors and Warnings:**

Table 1: List of Errors and Warnings Encountered in the Program

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
|  | No errors |  |  |
|  |  |  |  |

**4. Sample Input and Output:**

**The “happy path” test cases are basically all the conditions listed in the play() method. You could also unit test all the outputs from the convert method, given an input.**

**Other test cases aren’t exactly necessary as there is no human interaction with the program other than starting it up. The computer controls all of the inputs… the random generator uses the already tested Math.random function… and because what I understood from the api and requirements is that most of the game logic is in the play method, AND there is no overloaded constructor to be able to inject mocks, it’s not very possible to unit test most of the program.**

**5. Discussion:**

The part I struggled with was following the api… I would’ve liked to split a little more logic out to make it more concise and also more testable. As I didn’t want to deviate from it, I just wrote everything in play().

I also saw that a static, final PLAYER1 and PLAYER2 were defined, so I took the opportunity to declare them right away and then put both in the convert() method so I could also use it to get String literals when I needed them for the players.