Assignment 1 PRT581

# Introduction

In this assessment I will build a game of Rock, Paper, Scissors in Python. Using TDD the assessment requirements define which tests must be built and fail to define the features for the code. The test tools used in this assessment will be unittest part of the python vanilla library architecture.

The requirements for the assessment game are:

1. The computer picks a hand from a list of options randomly.

2. The player picks a hand from a list of options.

3. One point is assigned to the winner of the round.

4. The first player to win 5 points wins the game.

5. Once the winner is defined, the player is asked to quit or restart the game.

6. The human player can quit at any time during the game / round.

In this assessment I will write test for (4) four of the requirements listed. These will test I/O and functionality.

# Process

To begin the assessment, I will firstly define the (4) four tests chosen for this assessment.

1. The computer picks a random choice from a list of options.

2. Check for winner of round.

3. If the input from the user is valid.

4. The first player to a score of 5, wins the game.

The following details the steps and is supported with screen shots of the development below.

Test 1. - Test\_Random.py

Test 2. - Test\_winner.py

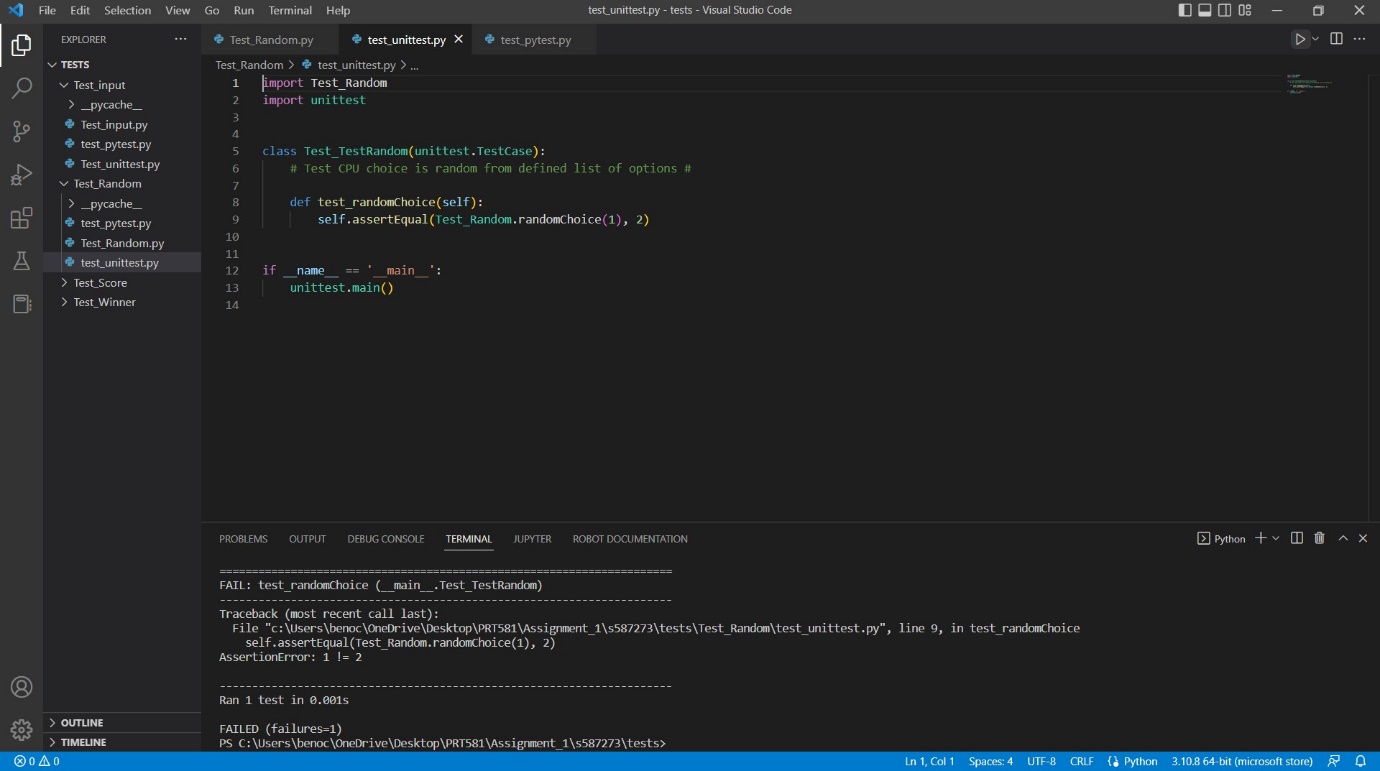
Test 3. - Test\_input.py

Test 4. - Test\_Score.py

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TestCase Id** | **Test Scenario** | **Test Case** | **Requirements** | **Test Steps** | **Test Data** | **Expected Results** | **Post Condition** | **Results** | **Pass/Fail** |
| Test\_Random | Test CPU randomly makes choice of options. | Possible Deals = [“Rock, Paper, Scissors’] | Choices: Rock, Paper, Scissors | Def PossibleDeals | Rock  Paper  Scissors | Return ‘player and playerscore’ or ‘CPU and CPU Score’ |  | Fail | Pass |
| Test\_Winner | Test winner of game round. | Define checkForWinner (playerHand, ComputerHand) | Return winner of round, Player or CPU | Def  CheckForWinner | You Lose, return ‘CPU’  You win return ‘Player’ | Return ‘you win’ or ‘you lose’ |  | Fail | Pass |
| Test\_Input | Test valid input. | While True statement of Possible Deals | Only valid input allowed | Def PlayerHand | PossibleDeals = Rock, Paper, Scissors | Rock, Paper,  Scissors |  | Fail | Pass |
| Test\_Score | Test player with score of 5 wins the game. | While (playerScore! = 5 and CpuScore! = 5) loop | Game end if playerScore = 5 | Def  PlayerScore | PlayerScore! =5 and CpuScore! =5 | PlayerScore +=1, CPU Score +=1 |  | Fail | Pass |

Test Cases

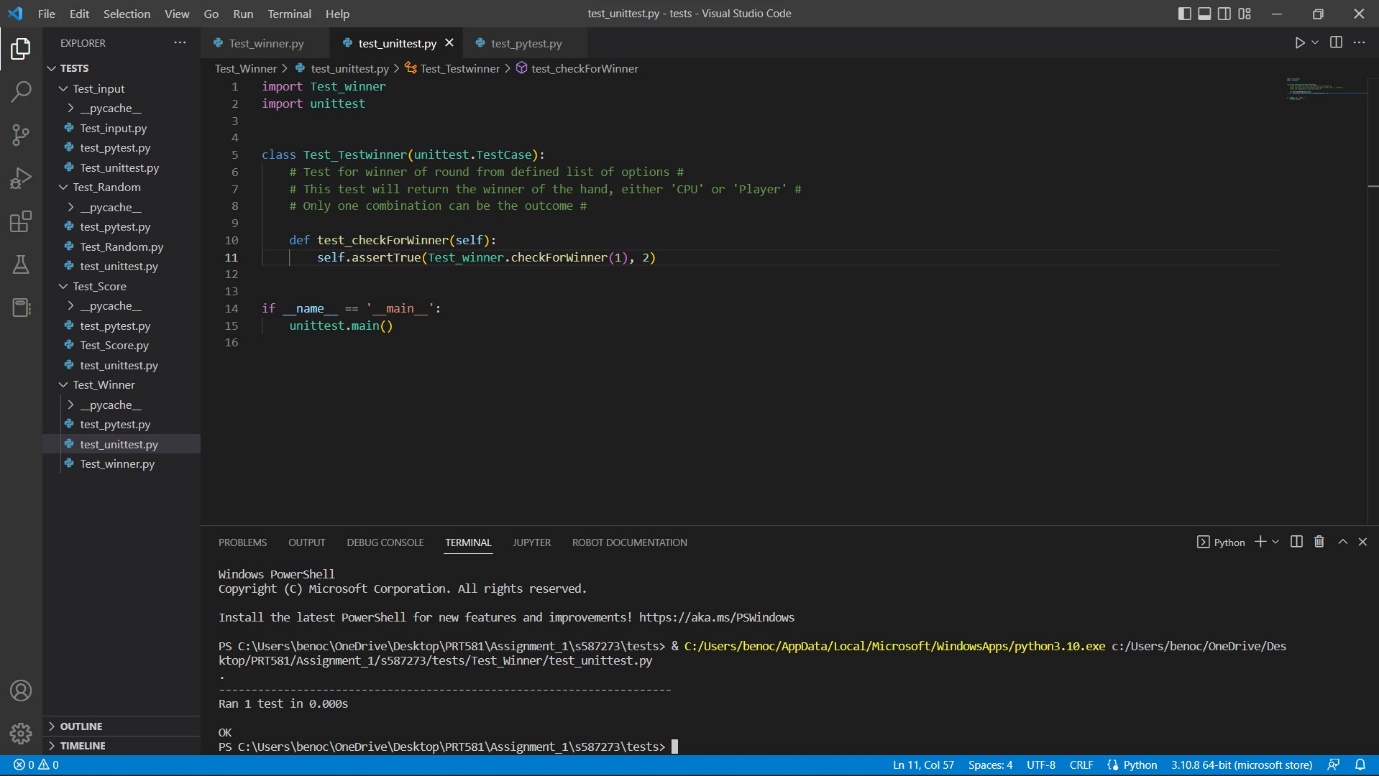
Test 1.



Test\_Random

Using the assertEqual tool the test ran and failed.

Test 2.



Test\_winner

Using the assertEqual tool the test ran and failed.

Test 3.

Text

Description automatically generated

Test\_input

Using the assertEqual tool the test ran and failed.

Test 4.

Text

Description automatically generated

Test\_Score

Using the assertEqual tool the test ran and failed.

# Conclusion

The conclusion of the tests is ‘assertEqual’ tool is not adequate for all forms of testing required in the game requirements. Using Unittest framework saves lines of code and time with the many different library’s available as part of vanilla Python code. Using VSC Editor also allows for fast debugging syntax errors that maybe overlooked or not understood.

The completed and commented code included supports the decisions made with the final features of the project. The full code runs without errors and is a simple example of A.I. included within the Python language.

The follow link is to my personal GitHub account where the full code and all requirements are located. I have included both the report and code in a submission via Learn line.

<https://github.com/S587273/PRT582>