Scissor Paper Rock Game Report

INTRODUCTION

Rock, paper, and scissors is a hand game usually played between a minimum of two players where each player makes one of three shapes using their hand. The main objective of the task is to write the Rock, paper, and scissors game using Test Driven Development (TDD) in python and to outline how the automated unit testing has been used.
The main requirement of the game are listed below:

- The computer should make a random pick among rock, paper, and scissors.
- The player should be asked to make a choice among the three options.
- The winner of each round should be with a single point.
- The total of each win or point should be counted and displayed and the first to get a total of 5 points will win the game.
- After a winner is decided there should be an option to quit or restart the game.
- At any time the player should be able to restart the game.

The unit testing framework that has been used is Unittest, which is python's built-in testing framework. Unittest is an automated test tool that verifies small pieces of code known as units. These units can be a function or method of a class. Since it runs in an isolated manner the execution time of the unit tests is very fast. Since unit tests check each small piece of code precisely, they ensure that the program runs properly. Unit tests are efficient as they help to reduce the bugs in the code. The code analysis tools used are pylint and flake8. They are used to check the quality of the code.

PROCESS

TDD is the process of software development where we write the test cases first before writing the code first. In this process, Unittest has been used to write the test cases. The detailed process of using TDD and unit tests is mentioned below.
Before starting to write the unit test we need to create a file for the test and import unittest library to let python know it's a unit test. We also need to import the class though for now, we don't have it. We need to assume we have a class from which we are going to import the functions for testing its functionality. We are making two files one for the unit test and one for the main program.

```
import unittest
# Importing the main file
from the_rps_game import TheRPSGame
```

The next step will be to create a test case which is done by defining a new class UnitTestRPSGame that will inherit the test case from unittest.

```
class UnitTestRPSGame(unittest.TestCase):
```

Then we define a method showing the case where paper wins the rock and to test this method we add a method call as

```
def test_paper_wins_against_rock(self):
```

In this process, we create a new instance of the class TheRPSGame. Then the user_action is assigned to 1, which is Paper and computer_action is assigned to 0, which is rock. Then the assert section is used to check whether the code returns the expected value. In this case, the winner is the player who selects the paper and hence the user win count will be 1.

assert the_rps_game.user_win_count == 1

```
def test_paper_wins_against_rock(self):
       # rock[0], paper[1], scissors[2]
       the_rps_game = TheRPSGame()
       user_action = 1
       computer_action = 0
       the_rps_game.check_winning_player(user_action, computer_action)
       assert the_rps_game.user_win_count == 1
```

Similarly, several other test cases are written in a similar format and executed for testing the code. We have a test case for the computer to choose the random value. So first we have written a case as follows:

```
def test_computer_randomly_picks_options(self):
       # breakpoint()
       the_rps_game = TheRPSGame()
       assert (the_rps_game.get_computer_selection() != None) == True
```

So as this is a TDD, I have written a method regarding this test case:

```
   def get_computer_selection(self):
       selection = random.randint(0, len(Action) - 1)
       action = Action(selection)
       return action
```

Here Action means a separate class:

```
class Action(IntEnum):
   Rock = 0
   Paper = 1
   Scissors = 2
```

Test case for winner's getting point:

```
def test_winner_gets_one_point(self):
       the_rps_game = TheRPSGame()
       user_action = 2
       computer_action = 1
       the_rps_game.check_winning_player(user_action, computer_action)
       assert the_rps_game.user_win_count == 1
       assert the_rps_game.computer_win_count == 0
```

Now, the program written by acknowledging above test case, we have:

```
   def check_winning_player(self, user_action, computer_action):
       if user_action == computer_action:
           for i in range(2):
               self.next_line()
            print(f"Both players selected {user_action.name}. It's a tie!
\n")
       elif user_action == Action.Rock:
           if computer_action == Action.Scissors:
               for i in range(2):
                   self.next_line()
               self.user_win_count += 1
               print(f"Rock smashes scissors! You win this round!")
           else:
               for i in range(2):
                   self.next_line()
               self.computer_win_count += 1
               print(f"Paper covers rock! You lose.")
       elif user_action == Action.Paper:
           if computer_action == Action.Rock:
               for i in range(2):
                   self.next_line()
               self.user_win_count += 1
               print(f"Paper covers rock! You win!",  self.user_win_count)
           else:
               for i in range(2):
                   self.next_line()
               self.computer_win_count += 1
               print(f"Scissors cuts paper! You lose.")
       elif user_action == Action.Scissors:
           if computer_action == Action.Paper:
               for i in range(2):
                   self.next_line()
               self.user_win_count += 1
                               print(f"Scissors  cuts  paper!  You  win!",
self.user_win_count)
           else:
               for i in range(2):
                   self.next_line()
               self.computer_win_count += 1
               print(f"Rock smashes scissors! You lose.")
```

After finding the combination, as per the requirement, we need to count five points to declare the winner. So the test for this requirement:

```
def test_game_is_won_with_five_points(self):
        the_rps_game = TheRPSGame()
        the_rps_game.user_win_count = 4
        the_rps_game.computer_win_count = 0
        assert the_rps_game.is_game_won() != True

        the_rps_game.user_win_count = 5
        assert the_rps_game.is_game_won() == True
```

The program after the test case:

```
def is_game_won(self):
        if self.user_win_count == self.WIN_COUNT:
            for i in range(2):
                self.next_line()
                print(f"You won")
                self.next_line()
                return True
        elif self.computer_win_count == self.WIN_COUNT:
            for i in range(2):
                self.next_line()
                print(f"Computer won")
                self.next_line()
                return False
        else:
            None
```

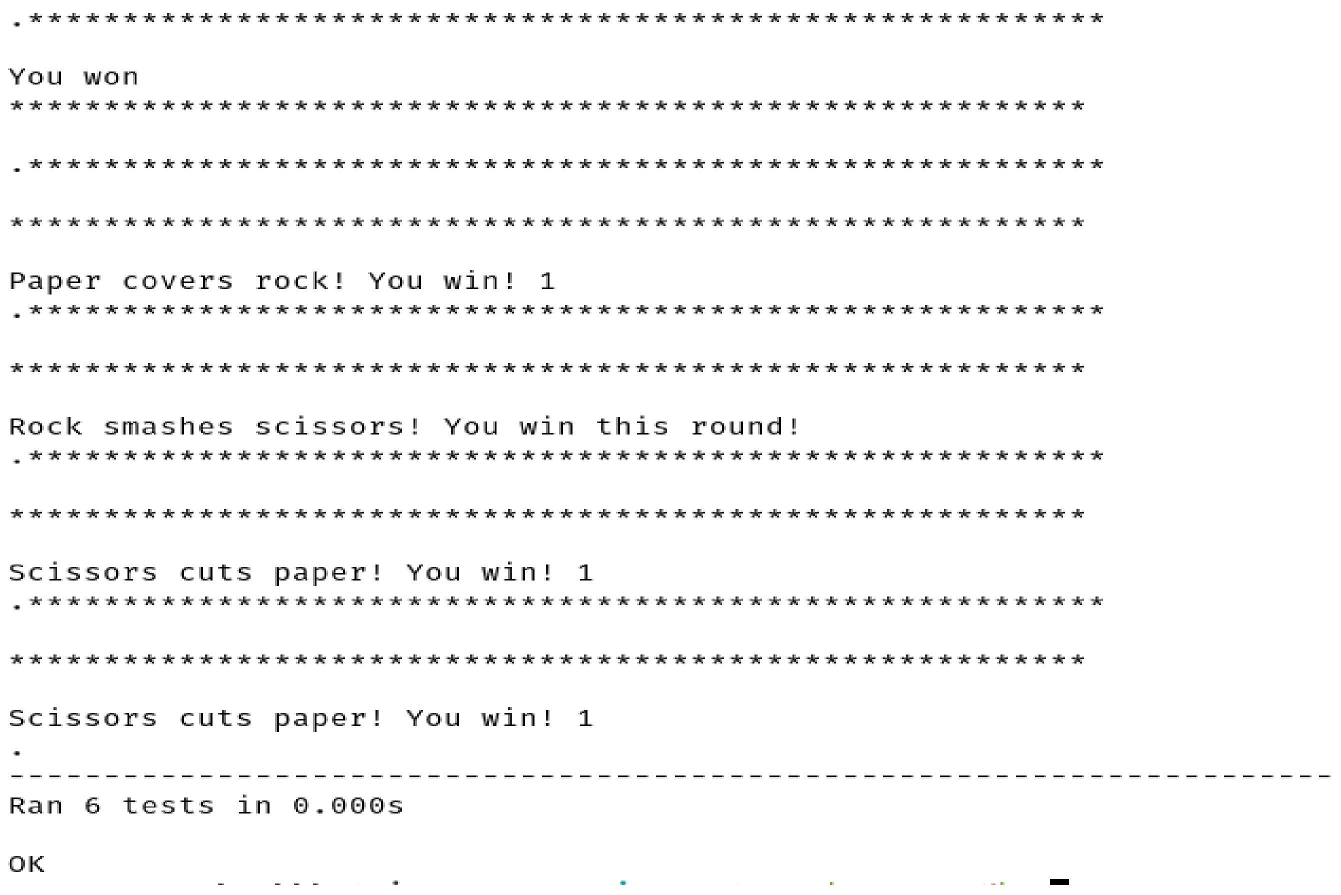
Before running the test, we call the main() function of the unittest module as follows:

```
if __name__ == "__main__":
   unittest.main()
```

After this we use the terminal to run the test by navigating to the folder and executing the command below:

```
python3 test_rps_game.py
```

I have added the screenshot of the output, that we get after the complete test.



```
.*********************************************************

You won
*********************************************************

.*********************************************************

*********************************************************

Paper covers rock! You win! 1
.*********************************************************

*********************************************************

Rock smashes scissors! You win this round!
.*********************************************************

*********************************************************

Scissors cuts paper! You win! 1
.*********************************************************

*********************************************************

Scissors cuts paper! You win! 1
.
----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
```

CONCLUSION

From this rock, paper, scissors game which was written using TDD, we can see that there are fewer chances of errors in the code. Writing the test cases at the beginning will bring improvement in the codes and that will help to run the program effectively. We also learned that using TDD makes it easier to detect bug and reduce it. When writing a large program this approach will help to save a lot of time and increase the efficiency of the program. After writing the test cases it was much easier to write the main code without any errors. This approach helps to understand the main scope of the project.

Github_link: Rock Paper Scissors game by Subash Khatri

Name: Subash Khatri
Student Number: S360909