PRT582 – ASSIGNMENT

SOFTWARE UNIT TESTING REPORT

Github link: https://github.com/Moriom889/PRT582-Assignment1

Table of Contents

Introduction:

The Scissor Paper Rock game is the type of game in which a player gets to choose one of the options between scissors, paper, and rock. This is then compared against the computer's selection and determines who the winner is.

The winning rules are listed below as follows:

- rock vs paper - paper wins
- rock vs scissors - rock wins
- paper vs scissor - scissor wins.

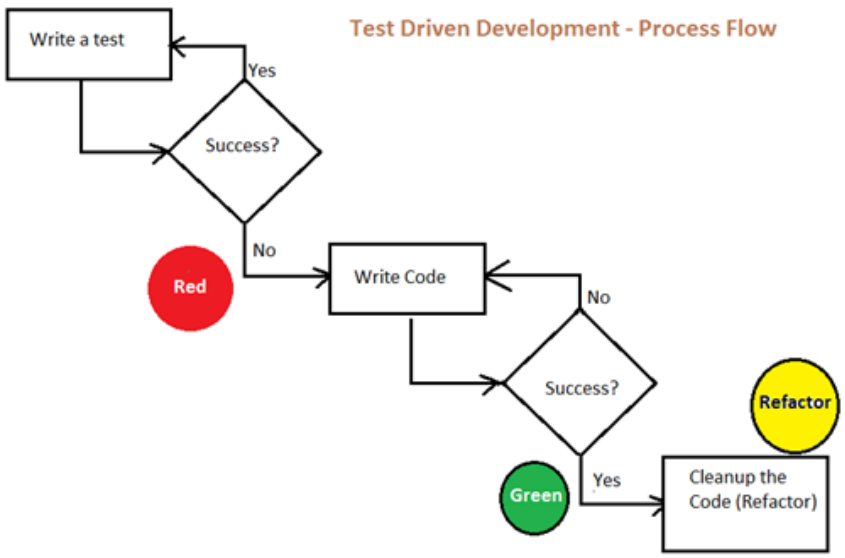
The game will have the following features, which are requirements:

1. The computer randomly picks one of the options of scissors, paper, and rock.
2. Player is then given the option to pick/type one of the options of scissors, paper, and rock.
3. One point is given to the winner.
4. The first to get five points wins the game. The total number of rounds played in total will also be displayed.
5. Once the winner is determined, the player is asked to quit or restart the game

For this sort of program, I decided to use Python to build all the functions of the application. Pytest will be used to test the code automatically and Test-Driven Development Process (TDD) will be applied to specify and validate what the code is expected to do.

Especially, the TDD process will be followed by the following steps:

1. Add a Test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.



Process:

The Pytest needs to be installed on the local machine before execution.
As we follow the TDD process, the test cases will be created for each specific requirement using the Pytest library.

Requirement One:

For the first requirement, the computer randomly picks one of the options of scissors, paper, and rock. Therefore, created the test cases for the requirement as shown :

The computer randomly picks one of the options of scissors, paper, and rock		
Test case ID	**Description**	**Expected result**
TC1	Computer select Paper	Output is “Paper”
TC2	Computer select Scissors	Output is “Scissors”
TC3	Computer select Rock	Output is “Rock”

```python
# Test-cases for function one

def test_case1():
    var = computer_selection("com_action")
    print(f"test_case1 result: {var}")
    assert var in selection_list

def test_case2():
    var = computer_selection("com_action")
    print(f"test_case1 result: {var}")
    assert var in selection_list

def test_case3():
    var = computer_selection("com_action")
    print(f"test_case1 result: {var}")
    assert var in selection_list
```

These Test cases will test if “computer_action” will randomly select either Rock, Paper, or Scissors respectively.

In the next step, the function was built based on the test cases and the given 1st requirement:

```python
import random
import pytest

# Requirement One

selection_list = ("Scissors","Rock", "Paper", )
def computer_selection(t):

        com_action = random.choice(selection_list)
        print(f"\n computer choses {com_action}.\n")

        return com_action
```

I used the Random inbuilt python function to select the preferred string randomly for the computer from selection_list

The output is printed after the loop finishes.
I ran the tests after finishing building the function. Here are the test results of the test cases:

```
============================= test session starts ==============================
collecting ... collected 3 items

function4.py::test_case1 PASSED                                      [ 33%]
 computer choses Paper.

test_case1 result: Paper

function4.py::test_case2 PASSED                                      [ 66%]
 computer choses Scissors.

test_case1 result: Scissors

function4.py::test_case3 PASSED                                      [100%]
 computer choses Rock.

test_case1 result: Rock


============================== 3 passed in 0.02s ===============================
```

Following the above result, all the test cases are passed so the function is built successfully.

Requirement Two:

The second requirement, the player picks one from any of the options of scissor, paper, and rock. I created the test cases for the requirement as shown below:

Player is then given the option to pick one of the options of scissors, paper, and rock.		
Test case ID	**Description**	**Expected result**
TC4	Player selects Scissors	Output is “Scissors”
TC5	Computer Select Paper	Output is “Paper”
TC6	Computer select Rock	Output is “Rock”

```python
import pytest
from function2 import player_selection
import mock
import builtins

# Test-cases

def test_case4():
    with mock.patch.object(builtins, 'input', lambda _: 'Scissors'):
        assert player_selection() == 'Scissors'


def test_case5():
    with mock.patch.object(builtins, 'input', lambda _: 'Paper'):
        assert player_selection() == 'Paper'


def test_case6():
    with mock.patch.object(builtins, 'input', lambda _: 'Rock'):
        assert player_selection() == 'Rock'
```

Created a function to accept user input.

Ran the tests after finishing building the function. Here are the test results of the test cases:

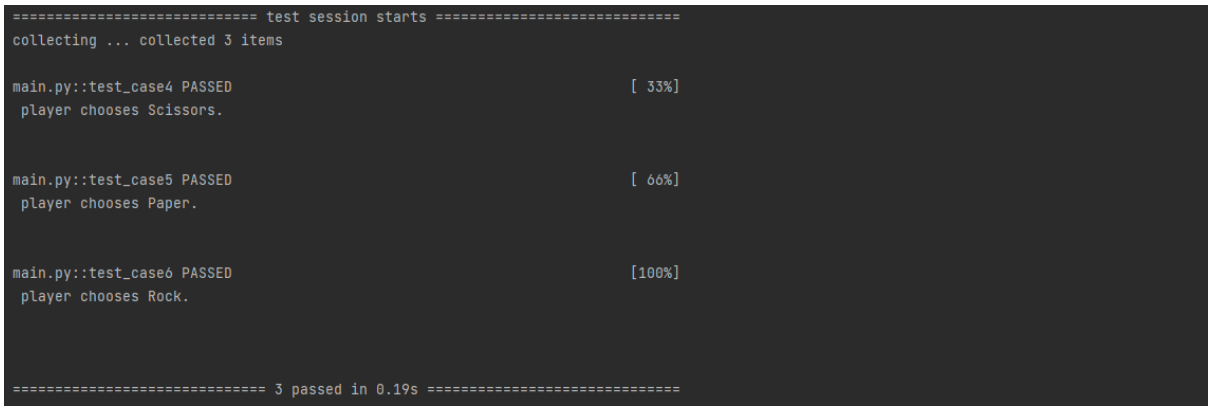
```python
# Requirement  Two

def player_selection():

        user_action = input("Enter a choice (rock,scissors,paper): ")
        print(f"\n player chooses {user_action}.\n")

        return user_action
```

The tests were run automatically, and the results are shown below:



Following the above result, all the test cases are passed so the function is built successfully.

Requirement Three:
The third requirement, this requirement is used to determine who wins each round after a selection between the player and computer for all possible scenarios or result

The test cases are created to test the function:

Determining the winner of each round for all possible outcomes		
Test case ID	**Description**	**Expected result**
TC7	The computer wins this round	Computer win
TC8	A player wins this round	Player win
TC9	The computer wins this round	Computer win
TC10	A player wins this round	Player win
TC11	The computer wins this round	Computer win
TC12	A player wins this round	Player win

The test cases will confirm who wins each game round between the player or computer after the selection of rock, paper, or scissors.

```python
import pytest
from function3 import winner_each_round

def test_case7():
    var = winner_each_round('scissors', 'Rock')
    assert var == 2
def test_case8():
    var = winner_each_round('Paper', 'Scissors')
    assert var == 2
def test_case9():
    var = winner_each_round('Scissors', 'Paper')
    assert var == 1

def test_case10():
    var = winner_each_round('Rock', 'Paper')
    assert var == 2
def test_case11():
    var = winner_each_round('Paper', 'Rock')
    assert var == 1
def test_case12():
    var = winner_each_round('Rock', 'scissors')
    assert var == 1
```

The function is built to work with the test cases and to follow the requirement:

```python
#Requirement Three
selection_list = ("Scissors","Rock", "Paper")

def winner_each_round(user_action, com_action):

    if user_action.lower() == com_action.lower():
        print(f"Both players selected {user_action}. It's a tie!")
        return 0
    elif user_action.lower() == "rock":
        if com_action.lower() == "scissors":
            print("Rock smashes Scissors. You win!")
            return 1
        else:
            print("Player lose.")
            return 2
    elif user_action.lower() == "paper":
        if com_action.lower() == "rock":
            print("Paper wraps Rock. You win!")
            return 1
        else:
            print("Player lose.")
            return 2
    elif user_action.lower() == "scissors":
        if com_action.lower() == "paper":
            print("Scissors cuts paper! You win!")
            return 1
        else:
            print("Player lose.")
            return 2
    else:
        print("You entered the wrong input")
```

The variables user_win and comp_win were assigned an initial value of zero, which increases by a value of 1 whenever the player or computer wins the round.
The tests were run automatically, and the results are shown below:

```
============================= test session starts ==============================
collecting ... collected 6 items

main.py::test_case7 PASSED                                           [ 16%]Player lose.

main.py::test_case8 PASSED                                           [ 33%]Player lose.

main.py::test_case9 PASSED                                           [ 50%]Scissors cuts paper! You win!

main.py::test_case10 PASSED                                          [ 66%]Player lose.

main.py::test_case11 PASSED                                          [ 83%]Paper wraps Rock. You win!

main.py::test_case12 PASSED                                          [100%]Rock smashes Scissors. You win!


============================== 6 passed in 0.03s ===============================
```

Following the above result, all the test cases are passed so the function are built successfully.

Requirement Four:

Requirement four requires a conditional statement to check the number of win between the player and the computer, the first to get to five wins the game.

Following the TDD process, firstly, the test cases are created for this requirement:

The first to get five points wins the game.		
Test case ID	**Description**	**Expected result**
TC 13	The Player gets to five wins	Display output
TC 14	The Player gets to five wins	Display output
TC15	The computer gets to five wins	Display output
TC16	The computer gets to five wins	Display output

```python
import pytest
from function4 import is_get_enough_point

# testcase
def test_case13():
    var = is_get_enough_point(5, 5)
    print(f"test_case13 result: {var}")
    assert var == True

def test_case14():
    var = is_get_enough_point(4, 5)
    print(f"test_case14 result: {var}")
    assert var == False

def test_case15():
    var = is_get_enough_point(5, 5)
    print(f"test_case15 result: {var}")
    assert var == True


def test_case16():
    var = is_get_enough_point(3, 5)
    print(f"test_case16 result: {var}")
    assert var == False
```

The function is built to work with the test cases and to follow the requirement:

```python
import random

#requirement four

game_winner = 5



def is_get_enough_point(counter, game_winner):
    if counter >= game_winner:
        return True
    return False
```

The tests were run automatically, and the results are shown below:

```
============================= test session starts ==============================
collecting ... collected 4 items

main.py::test_case13 PASSED                                   [ 25%]test_case13 result: True

main.py::test_case14 PASSED                                   [ 50%]test_case14 result: False

main.py::test_case15 PASSED                                   [ 75%]test_case15 result: True

main.py::test_case16 PASSED                                   [100%]test_case16 result: False


============================== 4 passed in 0.03s ===============================
```

Programme:

After sorting out all the functions, combined them together, includeed some appropriate code to make them work together.

The full code of the program is shown below:

```python
import random
def main():
    user_win=0
    comp_win=0
    while True:
        user_action = input("Enter a choice (rock, paper, scissors): ")
        option_list = ["Rock", "Paper", "Scissors"]
        com_action = random.choice(option_list)
        print(f"\nYou chose {user_action}, computer chose {com_action}.\n")

        if user_action.lower() == com_action.lower():
```

```
            print(f"Both players selected {user_action}. It's a tie!")
        elif user_action.lower() == "rock":
            if com_action.lower() == "scissors":
                user_win=user_win + 1
                print("Rock smashes Scissors. You win!")
            else:
                comp_win = comp_win + 1
                print(" You lose.")
        elif user_action.lower() == "paper":
            if com_action.lower() == "rock":
                user_win=user_win + 1
                print("Paper wraps Rock. You win!")
            else:
                comp_win = comp_win + 1
                print(" You lose.")
        elif user_action.lower() == "scissors":
            if com_action.lower() == "paper":
                user_win=user_win + 1
                print("Scissors cuts paper! You win!")
            else:
                comp_win = comp_win + 1
                print("You lose.")
        else:
            print("Your input is wrong")
        if user_win >= 5 or comp_win >=5:
           if comp_win < user_win:
                    print("Player won this game")
           else:
                    print("Computer won this game")
           print("Final Scores:")
           print(f"Player:{user_win}")
           print(f"Comp:{comp_win}")
           play_again = input("Play again? (yes/no): ")
           if play_again.lower() != "yes":
            break


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

Finally, run the code to ensure it would work:

Case 1: Checking User and Computer input

```
Enter a choice (rock, paper, scissors): paper

You chose paper, computer chose Rock.

Paper wraps Rock. You win!
```

Case 2: All the conditions are passed, and the winner is shown

```
 You lose.
Computer won this game
Final Scores:
Player:3
Comp:5
Play again? (yes/no):
```

Conclusion:

The Test-Driven Development process is being utilized by many developers nowadays. It helps them to build the program properly. Furthermore, with the support of many automated testing tools such as Pytest, the developers can save time testing and debugging the code. I will be practicing more of these techniques and tools to improve my coding skill.

Github link: https://github.com/Moriom889/PRT582-Assignment1