Introduction

Objective: Make a Rock-Paper-Scissors game using Test Driven Development in Python.

Requirements:

1. Player can pick one of the options from rock, paper and scissors.
2. The computer should randomly choose one of the options from rock, paper and scissors.
3. Winner for each round is determined and one point needs to be given to the winner. In case of draw no one gets any points and the game is continued to next round.
4. The game is finished when anyone either player or computer reaches to five points.
5. Once the game is finished, the player can quit or restart the game.
6. The current number of the round and the score card should be displayed in every round.
7. Player can quit the game at any time.

Process

The Rock-Paper-Scissors game is developed using Test driven development (TDD) as follows:

For every requirement, automated unit testing is done. Two files will be discussed in the upcoming report. One is actual game code and other is unit testing file. So, the game file is Game.py and unit testing file is Test.py file. Both file's screenshots after each step is shared simultaneously. The purpose of unit testing file is to test the implemented functionality alongside the development.

The starting code for both files is as follows:

game.py file:

```
class Game:
    def play_game(self):
        """ Main game code """
        pass

if __name__ == '__main__':
    game = Game()
    game.play_game()
```

test.py file:

```
import unittest
from game import Game


class Test(unittest.TestCase):
    game = Game()
```

Step 1:

Requirement to fulfil is: Player can pick any one of the options from rock/paper/scissors.

If player input wrong choice our code should say player choose wrong choice. Player can input the choice in various ways.

The choice for picking 'Rock' can be any from following:

'r', 'R', 'rock', 'ROCK', 'rock' word with combination of any number of capital and lower-case letters (Here consider that '(quote) marks are just for differencing the values in actual input they are not allowed) are allowed to be input from the player for choosing 'Rock'.

Similarly, to the 'Rock', player can pick his/her choice as 'Paper' or 'Scissors'. Internally, these values are considered as numbers. So, 'Rock' is stored as 1, 'Paper' is 2 and 'Scissors' is 3.

The code after implementing this functionality look like following (Only this requirement's code is shown below consider that other existing code is there and this will be continued in next steps):

game.py file:

```
class Game:
    """ Main game class"""

    def __init__(self):
        self.player_input = None
        self.player_choice = False

    def verify_player_input(self):
        """ Verify player has input correct choice from Rock/Paper/Scissors """
        lower_case_player_input = self.player_input.lower()
        if lower_case_player_input in ('r', 'rock'):
            self.player_choice = 1
        elif lower_case_player_input in ('p', 'paper'):
            self.player_choice = 2
        elif lower_case_player_input in ('s', 'scissors'):
            self.player_choice = 3
        else:
            self.player_choice = False
        return bool(self.player_choice)

    def get_player_input(self):
        """ Get player's input from Rock/Paper/Scissors options """
        self.player_input = input(" Choose your choice"
                                  "(Rock/Paper/Scissors) ---> ")

    def play_game(self):
        """ Main game code """
        self.get_player_input()
        if self.verify_player_input():
            print(True)
        else:
            print("Wrong input!!")
```

test.py file:

```
class Test(unittest.TestCase):
    """ Main unit testing class"""
    game = Game()

    def test_verify_player_input(self):
        """ Test the functionality of the verify_player_input method"""
        # Assume player has input 'r' for Rock
        self.game.player_input = 'r'
        self.assertEqual(True, self.game.verify_player_input())

        # Assume player has input 'paper' for Paper
        self.game.player_input = 'paper'
        self.assertEqual(True, self.game.verify_player_input())

        # Assume player has input 'SciSSors' for Scissors
        self.game.player_input = 'SciSSors'
        self.assertEqual(True, self.game.verify_player_input())

        # Assume player has input wrong word. It could be any string
        self.game.player_input = 'any'
        self.assertEqual(False, self.game.verify_player_input())
```

Step 2:

Requirement to fulfil is: Computer should choose its choice randomly from rock/paper/scissors. Since these choices are considered as an integer internally, the computer needs to randomly pick its choices from 1 to 3.

The code after implementing this functionality look like following:

game.py file:

```
def make_computer_choice(self):
    """ Computer choose randomly from Rock(1)/Paper(2)/Scissors(3) """
    self.computer_choice = random.randint(0, 2) + 1
```

test.py file:

```
def test_verify_computer_choice(self):
    """ Test the functionality of the make_computer_choice method"""
    # Computer make random choice from 0-2.
    self.game.make_computer_choice()
    self.assertEqual(True, self.game.computer_choice in [1, 2, 3])
```

Step 3:

Requirement to fulfil is: Winner of the round needs to be found.

Print the winner for now. In case of draw print draw for now

This is main critical step where we are determining the winner from player and computer. Following is the table for finding the result:

Player's Choice	Computer's Choice	Winner
Rock (1)	Rock (1)	Draw
Rock	Paper (2)	Computer
Rock	Scissors (3)	Player
Paper (2)	Rock (1)	Player
Paper	Paper (2)	Draw
Paper	Scissors (3)	Computer
Scissors (3)	Rock (1)	Computer
Scissors	Paper (2)	Player
Scissors	Scissors (3)	Draw

The logic of finding the winner is following:

First check if both choices are similar then the result is draw. In other cases, if player has chosen the choice one higher than the computer in scale of 3 then player will win the round. It can easily be seen in following code:

game.py file:

```
def evaluate_result(self):
    """ Find the winner based on player and computer's choices """
    if self.player_choice == self.computer_choice:
        return "draw"
    if (self.computer_choice + 1) % 3 == (self.player_choice % 3):
        return "player"
    return "computer"
```

```
def play_game(self):
    """ Main game code """
    self.get_player_input()
    if self.verify_player_input():
        self.make_computer_choice()
        print("Winner : ", self.evaluate_result())
    else:
        print("Wrong input!!")
```

test.py file:

```
def test_verify_winner(self):
    """ Test the functionality of evaluate_result method by verifying all
        possible combination of the player and computer's choices """
    # Player's choice is Rock(1)
    self.game.player_choice = 1
    self.game.computer_choice = 1
    self.assertEqual("draw", self.game.evaluate_result())
    self.game.computer_choice = 2
    self.assertEqual("computer", self.game.evaluate_result())
    self.game.computer_choice = 3
    self.assertEqual("player", self.game.evaluate_result())

    # Player's choice is Paper(2)
    self.game.player_choice = 2
    self.game.computer_choice = 1
    self.assertEqual("player", self.game.evaluate_result())
    self.game.computer_choice = 2
    self.assertEqual("draw", self.game.evaluate_result())
    self.game.computer_choice = 3
    self.assertEqual("computer", self.game.evaluate_result())

    # Player's choice is Scissors(3)
    self.game.player_choice = 3
    self.game.computer_choice = 1
    self.assertEqual("computer", self.game.evaluate_result())
    self.game.computer_choice = 2
    self.assertEqual("player", self.game.evaluate_result())
    self.game.computer_choice = 3
    self.assertEqual("draw", self.game.evaluate_result())
```

Step 4:

Requirement to fulfil is: Add scoring system in the game.

One point is given to the winner of the round. In case of draw, no one gets any point and the game continues to the next round. Whoever reaches to the 5 points earlier wins the game. For now, game is finished when someone reaches 5 points.

The code of this score system is following:

game.py file:

```
def evaluate_result_and_update_score(self):
    """ Find the winner based on player and computer's choices and update
        the score of the winner. In case of draw, no one gets point. """
    if self.player_choice == self.computer_choice:
        return "draw"
    if (self.computer_choice + 1) % 3 == (self.player_choice % 3):
        self.player_score += 1
```

```
            return "player"
        self.computer_score += 1
        return "computer"

    def is_game_over(self):
        """ Check if the game is finished. The game is finished when either
            player or computer reaches to 5 points. """
        if self.player_score == 5 or self.computer_score == 5:
            return True
        return False

    def play_game(self):
        """ Main game code """
        while not self.is_game_over():
            self.get_player_input()
            if self.verify_player_input():
                self.make_computer_choice()
                winner = self.evaluate_result_and_update_score()
                print("The winner of this round is : ", winner)

        if self.player_score == 5:
            print("Congratulations!! You win the game!!")
        else:
            print("Oops!! You lose the game!!")
```

test.py file:

```
    def test_verify_score_management(self):
        """ Test the score management functionality """
        # Initial score for player and computer are 0
        self.assertEqual(0, self.game.player_score)
        self.assertEqual(0, self.game.computer_score)

        # Check computer's score updates after computer wins
        self.game.player_choice = 1
        self.game.computer_choice = 2
        self.assertEqual("computer",
                         self.game.evaluate_result_and_update_score())
        self.assertEqual(1, self.game.computer_score)
        self.assertEqual(0, self.game.player_score)

        # Check player's score updates after player wins
        self.game.player_choice = 3
        self.game.computer_choice = 2
        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual(1, self.game.computer_score)
        self.assertEqual(1, self.game.player_score)

        # Check the draw case
        self.game.player_choice = 1
        self.game.computer_choice = 1
        self.assertEqual("draw", self.game.evaluate_result_and_update_score())
        self.assertEqual(1, self.game.computer_score)
        self.assertEqual(1, self.game.player_score)

        # Try to reach score of Player : Computer = 2 : 3 and test
        self.game.player_choice = 3
        self.game.computer_choice = 1
        self.assertEqual("computer",
                         self.game.evaluate_result_and_update_score())
        self.game.computer_choice = 2
```

```
        self.assertEqual("player",self.game.evaluate_result_and_update_score())

        self.game.player_choice = 2
        self.game.computer_choice = 3
        self.assertEqual("computer",
                         self.game.evaluate_result_and_update_score())

        self.assertEqual(3, self.game.computer_score)
        self.assertEqual(2, self.game.player_score)

    def test_verify_game_winner(self):
        """ Test the game over functionality. Player wins the game when player
            reaches to 5 points. """
        self.assertEqual(0, self.game.computer_score)
        self.assertEqual(0, self.game.player_score)
        self.assertEqual(False, self.game.is_game_over())

        self.game.player_choice = 3
        self.game.computer_choice = 2
        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual(1, self.game.player_score)
        self.assertEqual(False, self.game.is_game_over())

        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual(False, self.game.is_game_over())

        self.assertEqual("player",self.game.evaluate_result_and_update_score())
        self.assertEqual(0, self.game.computer_score)
        self.assertEqual(5, self.game.player_score)
        self.assertEqual(True, self.game.is_game_over())
```

Step 5:

Requirements to fulfil is: Player can restart or quit the game after game is finished.

Player inputs the choice for quit the game or restart after game finishes.

Player can input the ‘Quit’ multiple ways same as choosing any choice from rock, paper or scissors. ‘Q’, ‘q’, ‘quit’, ‘QUIT’ or ‘quit’ word with combination of any number of capital and lower-case letters are allowed to be input from the player for quitting the game.

For ease, any other input is considered as restarting of the game.

The code for restart functionality is following:

game.py file:

```
def reset_the_game(self):
    """ Reset the game when player wants to restart the game """
    self.player_score = 0
    self.computer_score = 0

def restart_game_choice(self):
    """ Option to restart or quit the game after game is over. """
    restart_choice = input(" Choose your choice (Restart/Quit) ---> ")
    lower_case_restart_choice = restart_choice.lower()
    if lower_case_restart_choice in ('q', 'quit'):
        return "quit"
    self.reset_the_game()
```

```
    return "restart"

def play_game(self):
    """ Main game code """
    while True:
        while not self.is_game_over():
            self.get_player_input()
            if self.verify_player_input():
                self.make_computer_choice()
                winner = self.evaluate_result_and_update_score()
                print("The winner of this round is : ", winner)
            else:
                print("Wrong input!!")

        if self.player_score == 5:
            print("Congratulations!! You win the game!!")
        else:
            print("Oops!! You lose the game!!")

        if self.restart_game_choice() == "quit":
            break
```

Step 6:

Requirement to fulfil is: Current number of round and score is shown in each round.

This step is pure aesthetic so no need of testing. We are also printing some lines to make our application user friendly and eye appealing.

The game has its starting banner, winning screen, losing screen, scorecard banner, restart/quit screen after game is finished and better user input screen. These all leads to the better user experience when player starts playing the game.

Since they are very long in lines, I do not add them these changes in the report. However, you can find the code from GitHub link shared in the conclusion.

Step 7:

Requirement to fulfil is: Player can quit the game at any time.

This requirement is straight forward. Whenever player can input their choices, we are allowing player to enter the 'Quit' choice similar as when we ask player for restarting or quitting the game.

game.py file:

```
def verify_player_input(self):
    """ Verify player has input correct choice from Rock/Paper/Scissors.
"""
    lower_case_player_input = self.player_input.lower()
    if lower_case_player_input in ('r', 'rock'):
        self.player_choice = 1
    elif lower_case_player_input in ('p', 'paper'):
        self.player_choice = 2
    elif lower_case_player_input in ('s', 'scissors'):
        self.player_choice = 3
    elif lower_case_player_input in ('q', 'quit'):
        return "quit"
    else:
```

```
            return False
        return True
    def play_game(self):
        """ Main game code """
        print_start_game_screen()
        while True:
            self.print_score_card()
            self.get_player_input()
            verify_player_input = self.verify_player_input()
            if verify_player_input == "quit":
                break
            if verify_player_input:
                self.make_computer_choice()
                self.evaluate_result_and_update_score()
                self.round_no += 1
            else:
                print_false_input_screen()

            if self.is_game_over():
                if self.player_score == 5:
                    print_winning_screen()
                else:
                    print_losing_screen()

                if self.restart_game_choice() == "quit":
                    break
```

Conclusion

The Test Driven Development (TDD) process is great way to create any application. Its step-by-step approach allows us to do work in small pieces. Adding new functionality in the existing application is very easy. Sometimes you might need to rethink the flow of application if the new feature is affecting the main flow.

All steps for creating rock-paper-scissors game are straight forward. Only the last step when player can quit the game anytime is little bit complex. Since this step affect the whole game flow, you need to be extra cautious. In whole TDD process only last step feels little bit hard as you need to change the game flow.

The GitHub link for this project is following:

https://github.com/sh-13/rock-paper-scissors-game