SOFTWARE UNIT TESTING REPORT

Writing Scissor Paper Rock game using Test Driven Development in Python

OCTOBER 14, 2022

CHARLES DARWIN UNIVERSITY
Student name: Gia An Vong. Student ID: s354803

Table of Contents

I. Introduction: 3

Objectives: 3

Requirements: 3

Automated unit test tools: 3

II. Process: 4

III. Process detail: 5

IV. Conclusion: 11

V. APPENDIX 12

1. Source code on git hub: 12

2. Game playing and screenshot for each requirement: 13

Requirement i: The computer randomly picks one of the options of scissor, paper and rock. ... 13

Requirement ii: Player is then given the option to pick/type one of the options of scissor, paper and rock. 13

Requirement iii: One point is given to the winner 13

Requirement iv: 14

Requirement v: 15

Requirement vi: While playing, user can quit the game at any time by input 0 16

Requirement vii: The winning rules and point giving. 17

Figure list

Figure 1: Test-driven development - process flow 4
Figure 2: Unit Test Case Class with some first test case 5
Figure 3: The main program game class with blank methods 6
Figure 4: Unit Test Case failures that must be resolved in the main program 6
Figure 5: Implement the main program code to fulfill the test case 7
Figure 6: Re-run the test case to verify if the new code passed by the unit test. 7
Figure 7: Pylint helps to check coding convention 8
Figure 8: Flake8 helps to validate the coding convention 8
Figure 9: Flake8 plugin in VS Code gives instant feedback on program source code 9
Figure 10: The unit test case full-listed methods 10
Figure 11: Unit Test Case execution result 10
Figure 12: Find the winner and give the point to him during the entire match round 13
Figure 13: The user chose an option of Rock, Paper, Scissor 13
Figure 14: Try several rounds to check the winning rules and score, adding to the winner 13
Figure 15: Current round of the match is shown in the game 14
Figure 16: The game winner is determined by whom got 5 points first 14
Figure 17: Ask the user to restart the game once the winner is found 15
Figure 18: Play a new match after the winner is determined. 16
Figure 19: The user does not start a new game. 16
Figure 20: The user decides to quit the game. 17
Figure 21: The player won the game by hitting 5 points first 17
Figure 22: Winning rules displayed at the beginning of the game 17
Figure 23: The player won the game. 18
Figure 24: The computer won the game 18

I. Introduction:

In our youth, many of us spent hours playing the game known as "Scissors, Rock, and Paper." At the absolute least, every programmer has spent at least one session playing or running a game or event they developed using their skills. Using the concept of Test-Driven Development, we will construct this little game once again for this project; however, we will do so in a different way.

Objectives:

In this assignment, we will focus on two main objectives below:

1. A mini-game written in python named Scissor Rock Paper game allows the user to compete with the computer.
2. We are using Test-Driven Development approach for developing test cases and the game.

Requirements:

The basic requirements of the game are defined as follows:

i. The computer randomly picks one of the options of scissors, paper, and rock.
ii. The player is then given the option to pick/type one of the scissors, paper, and rock options.
iii. One point is given to the winner.
iv. The first to get five points wins the game. The total number of rounds played in total will also be displayed.
v. Once the winner is determined, the player is asked to quit or restart the game
vi. Players can also quit the game at any time.
vii. The winning rules are as follows:
    - rock vs paper -> paper wins
    - rock vs scissor -> rock wins
    - paper vs. scissor -> scissor wins.

Automated unit test tools:

PyUnit will be utilized in this project so automated unit testing can be carried out. It is a framework for unit testing that was modeled after JUnit. This Python testing framework is contained within the Python package and is installed automatically.

II. Process:

Test-driven development, sometimes known as TDD, is a process for developing software that combines refactoring with test-first development. To put it another way, we start by writing a test and then proceed to develop just enough code to pass the test.

In most cases, the following steps are taken:

1. Add a test: Create a test case that completely describes the function.
   The developer must first understand the features and requirements to create test cases using user stories and use cases.
2. Perform the test cases and ensure that the new test case fails.
3. Make code of the program that passes the test case.
4. Execute the test cases
5. Refactoring code to eliminate code duplication.
6. Repeat the preceding steps until no more new program code, and all test cases are passed.

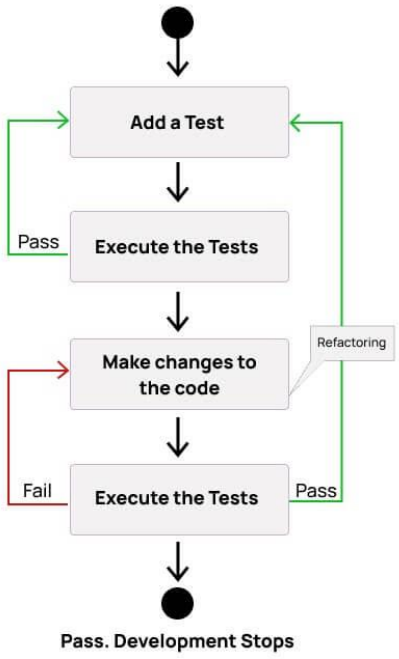


Figure 1: Test-driven development - process flow

Source: UnitTesting in Python, PRT582 Software Engineering Process and Tools

Benefits of TDD:

- Unit testing gives continual function feedback.
- The design quality improves, which facilitates proper maintenance.
- Test-driven development protects against bugs.
- TDD ensures that your application satisfies the predetermined requirements.
- TDD has a very short development cycle.

III. Process detail:

We create two python source files, rock_paper_scissor_game.py, and test_rock_paper_scissor_game.py, for the game and unit test case classes.

- test_rock_paper_scissor_game.py contains all the methods based on the user story that may have in a game.
- rock_paper_scissor_game.py contains the game class and only the method name without any code.

Step 1: We create some basic test cases for the unit test class inside the unit test case file, as shown in the screenshot below:

- Test method of RockPaperScissor.get_random_choice:
  - Requirement: the game class can generate a random choice from a list of [rock, paper, scissor]
  - The unit test case will compare one of the returning results in the list of [rock, paper, scissor] to judge whether it passed or failed.
- Test method of RockPaperScissor.get_choice_name:
  - Requirement: the game class should return the choice name base on input numbers such as 1 is rock, 2 is paper, and 3 is scissor.

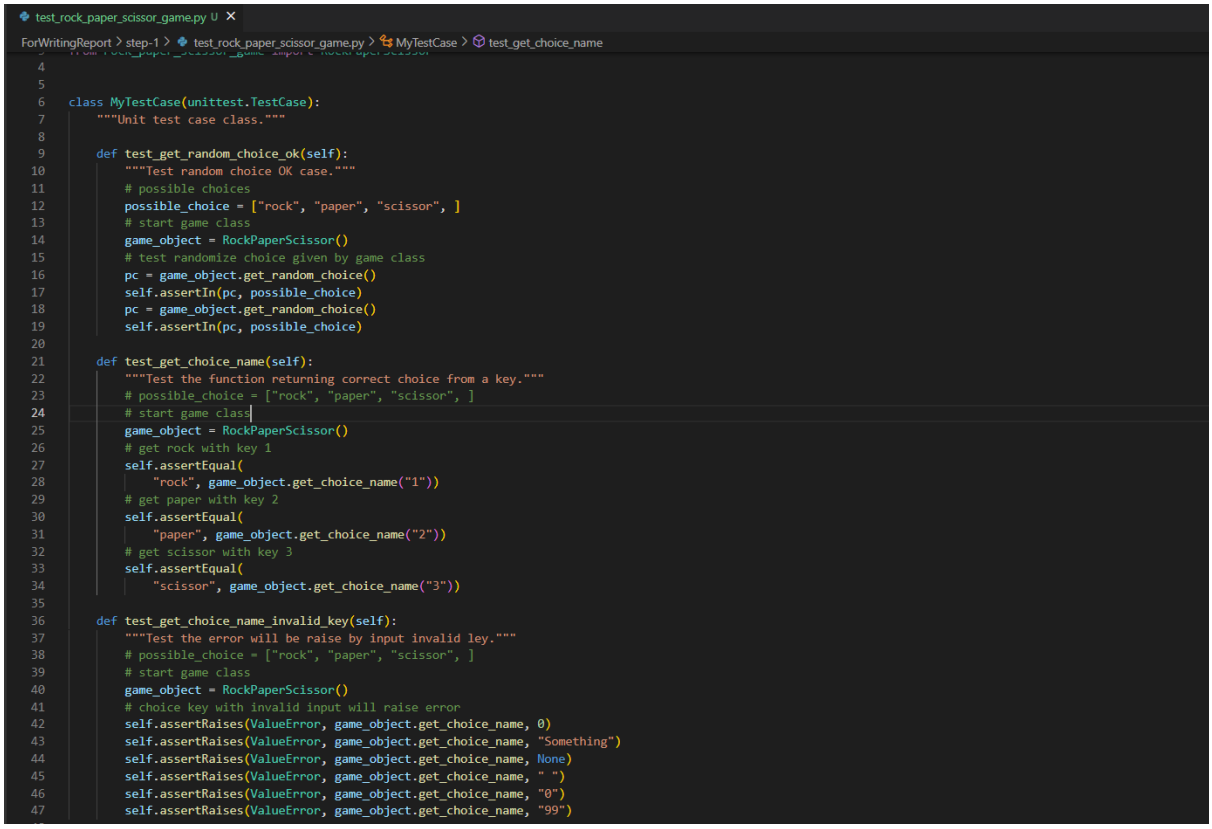


Figure 2: Unit Test Case Class with some first test case

```python
"""The mini game Rock Paper Scissor."""


class RockPaperScissor():
    """The game object of Rock Paper Scissor."""

    possible_choice = ["rock", "paper", "scissor"]
    player_score = 0
    computer_score = 0
    match_round = 1
    max_round = 5
    winner_point = 5

    def __init__(self):
        """Class constructer."""

    def get_random_choice(self):
        """Get a random choice."""

    def get_choice_name(self, enter_key):
        """Get choice name from an input key."""


if __name__ == '__main__':

    # Print multiline instruction
    # performstring concatenation of string
    print("Welcome to the rock paper scissor game, "
          "the Winning Rules are as follows: \n"
          "rock vs paper->paper wins \n"
          "rock vs scissor->rock wins \n"
          "paper vs scissor->scissor wins\n")

    # start first game
    game = RockPaperScissor()

```

Figure 3: The main program game class with blank methods

Step 2: We run the test case to see that there should be three failures. Those failures mean there are defined methods in the program class, but its code has not returned any correct result.

```python
"""Unit test for Rock Paper Scissor game."""
import unittest
from RockPaperScissorGame import RockPaperScissor


class MyTestCase(unittest.TestCase):
    """Unit test case class."""

    def test_get_random_choice_ok(self):
```

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong> python -u "c:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong\ForWritingReport\step-1\TestRockPaperScissorGame.py"
FFF
======================================================================
FAIL: test_get_choice_name (__main__.MyTestCase)
Test the function returning correct choice from a key.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong\ForWritingReport\step-1\TestRockPaperScissorGame.py", line 27, in test_get_choice_name
    self.assertEqual(
AssertionError: 'rock' != None

======================================================================
FAIL: test_get_choice_name_invalid_key (__main__.MyTestCase)
Test the error will be raise by input invalid ley.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong\ForWritingReport\step-1\TestRockPaperScissorGame.py", line 42, in test_get_choice_name_invalid_key
    self.assertRaises(ValueError, game_object.get_choice_name, 0)
AssertionError: ValueError not raised by get_choice_name

======================================================================
FAIL: test_get_random_choice_ok (__main__.MyTestCase)
Test random choice OK case.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong\ForWritingReport\step-1\TestRockPaperScissorGame.py", line 17, in test_get_random_choice_ok
    self.assertIn(pc, possible_choice)
AssertionError: None not found in ['rock', 'paper', 'scissor']

----------------------------------------------------------------------
Ran 3 tests in 0.002s

FAILED (failures=3)
PS C:\CDU_coding\PRT582-Code\rpt582-assignment1-giaanvong>
```

Figure 4: Unit Test Case failures that must be resolved in the main program

Step 3: Make code of the program that passes the test case.

Implement the code to the main program class to ensure the test case are passed.

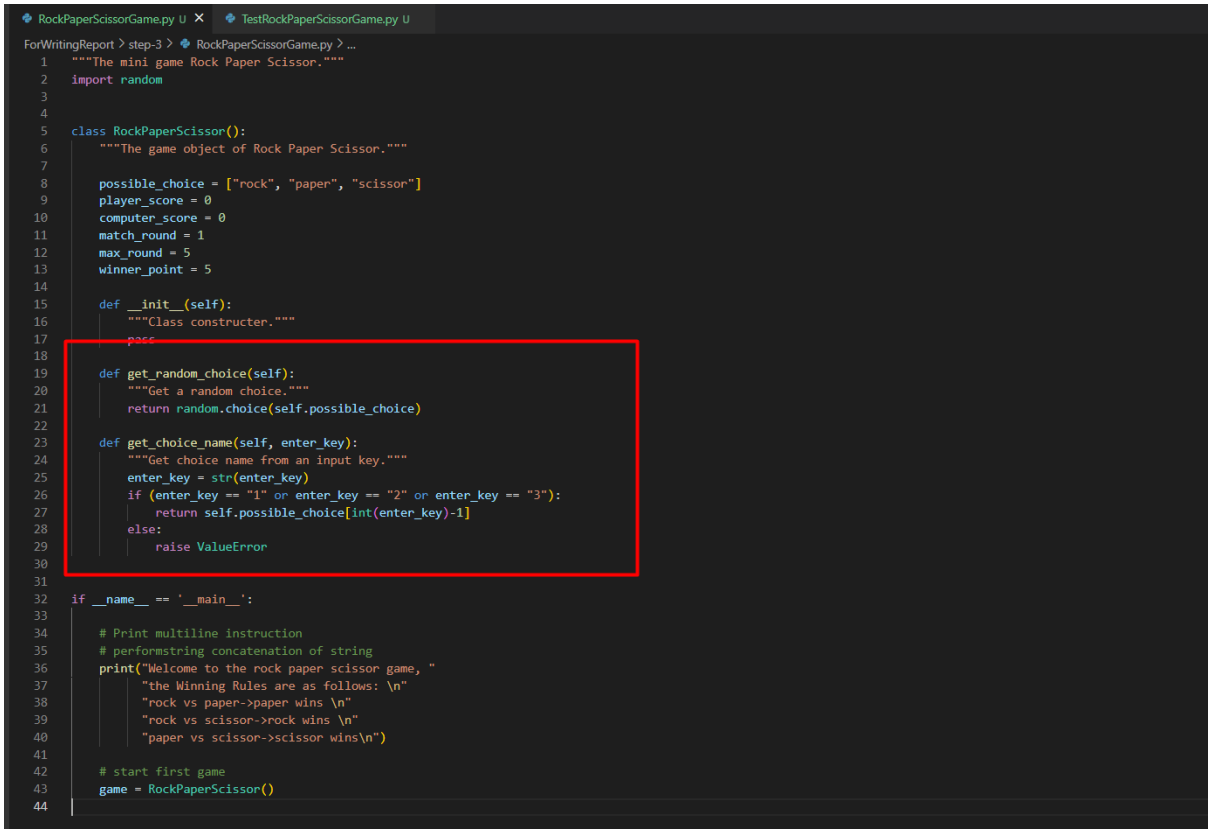


Figure 5: Implement the main program code to fulfill the test case

Step 4: Re-run the test case once again to check if the test case is OK.

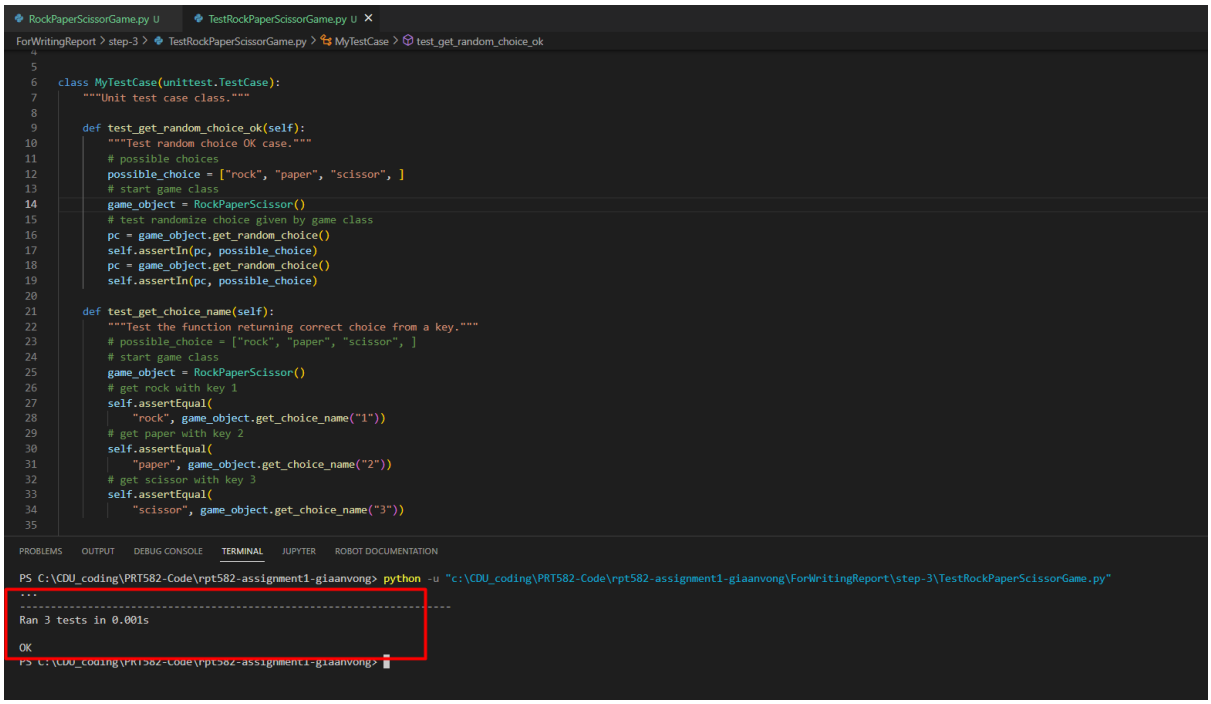


Figure 6: Re-run the test case to verify if the new code passed by the unit test.

Step 5: Refractory the source code or make the program code clean:

- Removing debug purpose code
- Put more comments to make the program clear
- Valid the source format with **pylint** and **flake8** to get the highest score as much as possible
- Using flake8 in Ms. Visual Studio Code while doing code also reduces the amount of reworking the code.

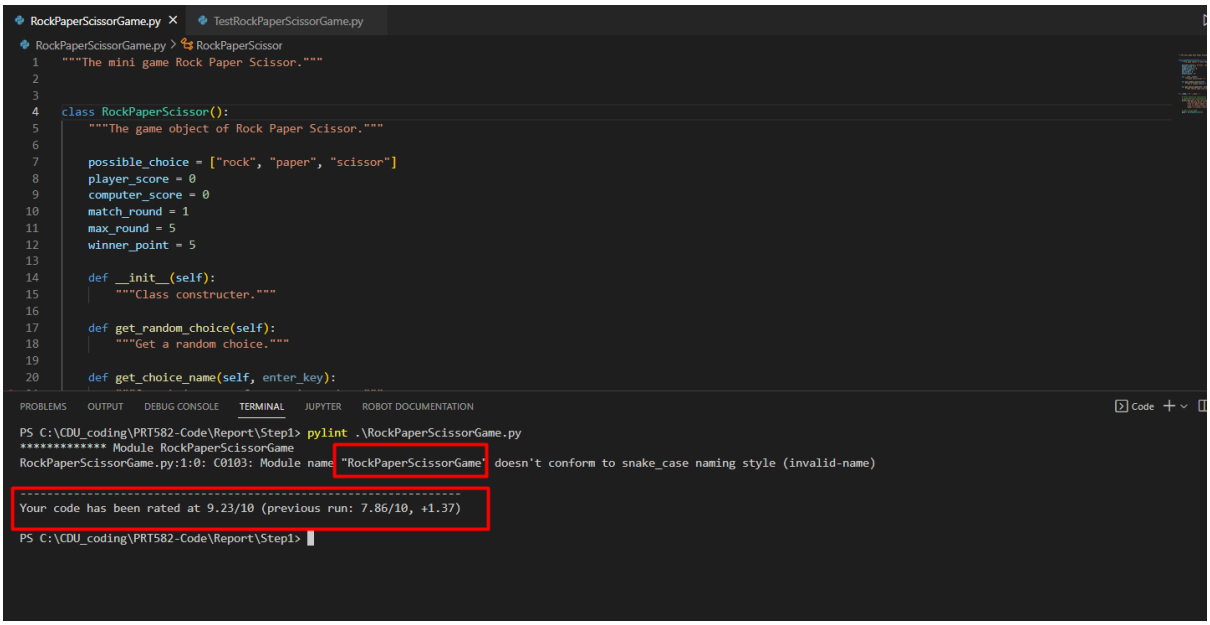


Figure 7: Pylint helps to check coding convention

Figure 8: Flake8 helps to validate the coding convention

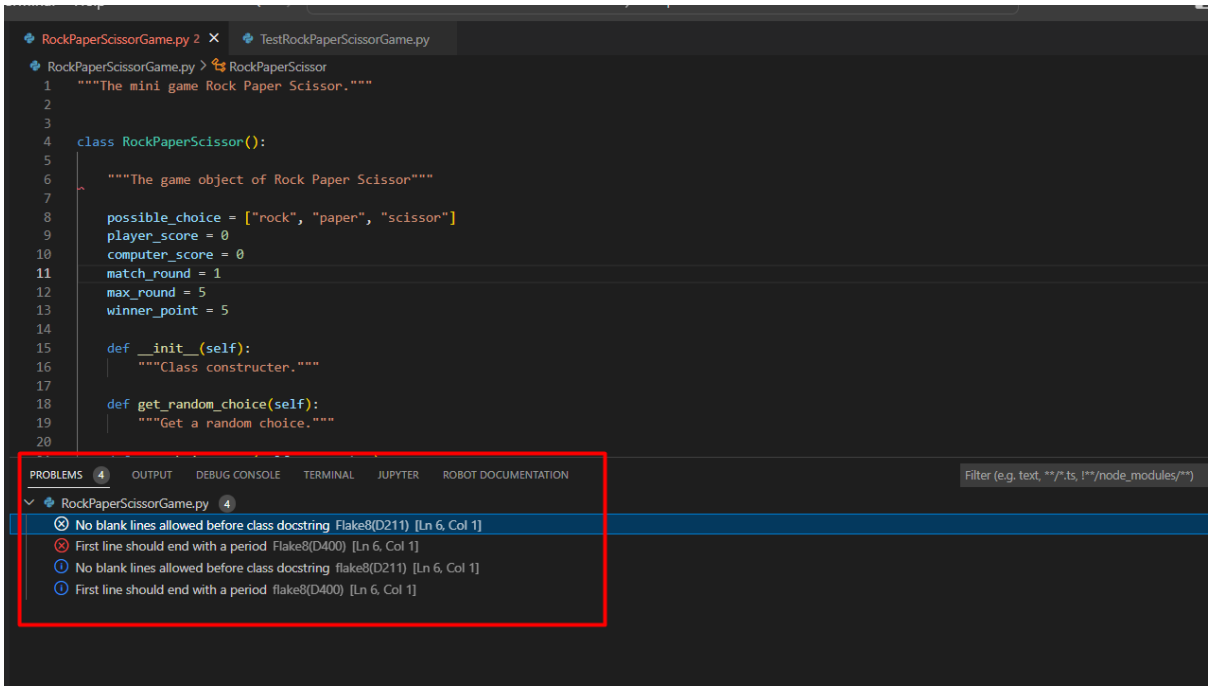


Figure 9: Flake8 plugin in VS Code gives instant feedback on program source code

Step 6: Repeat by adding a new test case into the Unit Test Case class file base on the requirements and new methods correspondingly into the game class source code.

The final test result:

The result, in the end, test case and programming, is all the test cases in the table below should be passed by Automated Unit Test.

No.	Unit test case	Description
1	test_find_winner_choice_player_win	Do the test player the winner
2	test_find_winner_choice_computer_win	Do test computer is the winner.
3	test_find_winner_choice_draw_result	Do test function find winner choice return a draw result.
4	test_find_winner_choice_invalid_input	Do test invalid input to find the winner's choice.
5	test_get_random_choice_ok	Test random choice OK case.
6	test_get_choice_name	Test the function returning the correct choice from a key.
7	test_get_choice_name_invalid_key	Test the error will be raised by inputting an invalid key
8	test_set_round_score	Test the function and set the round score.
9	test_set_round_score_raise_error	Test the function set_round_score raise Error.
10	test_get_game_winner	Test to get the game-winner.
11	test_print_round_result	Test print match round result is True

12	test_print_round_result_error	Test print match round result with error value is raised.

Table 1: TDD approach with all game methods must be passed by Unit Test.

Unit test case class and its test result:

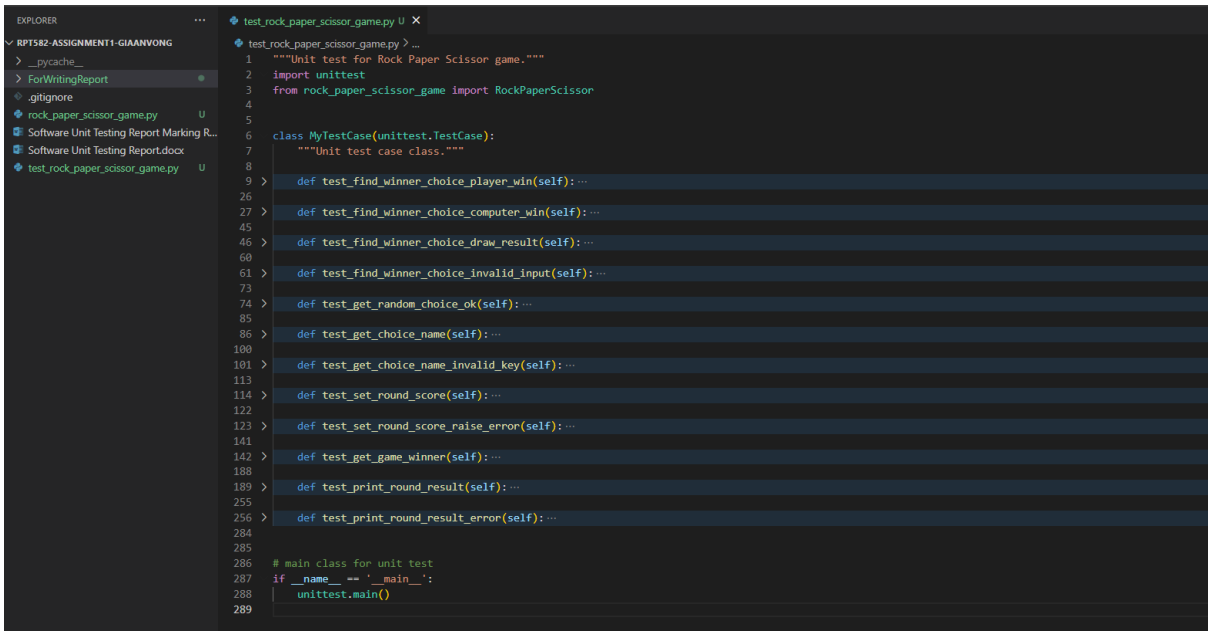


Figure 10: The unit test case full-listed methods

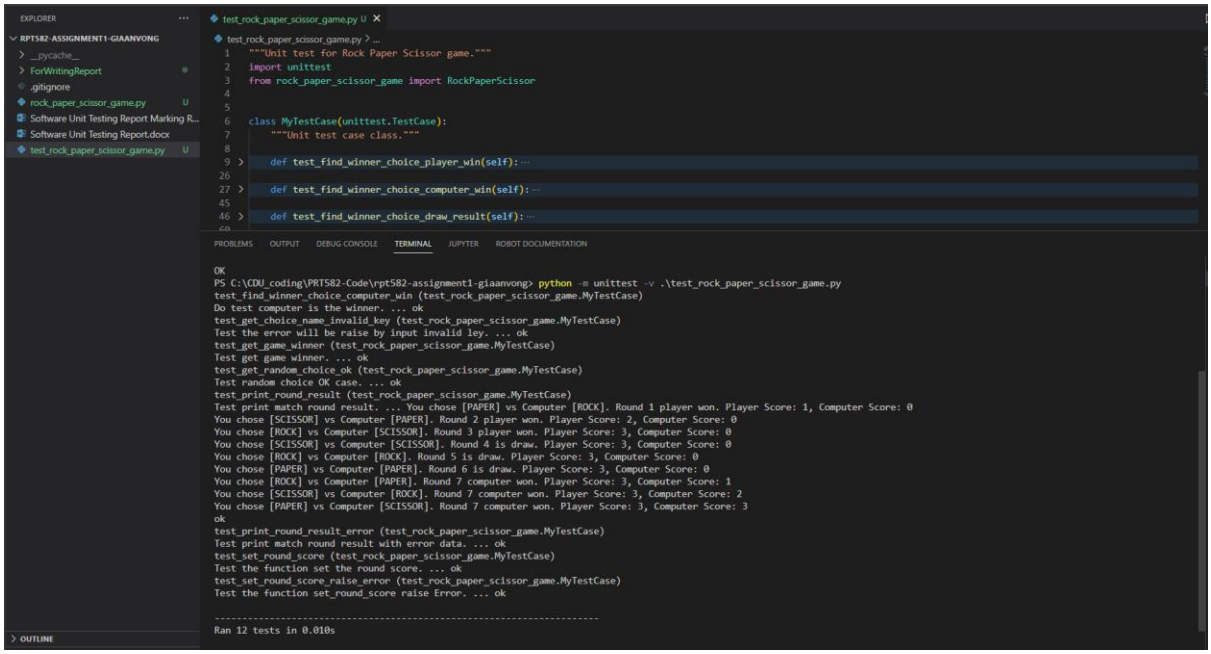


Figure 11: Unit Test Case execution result

IV. Conclusion:

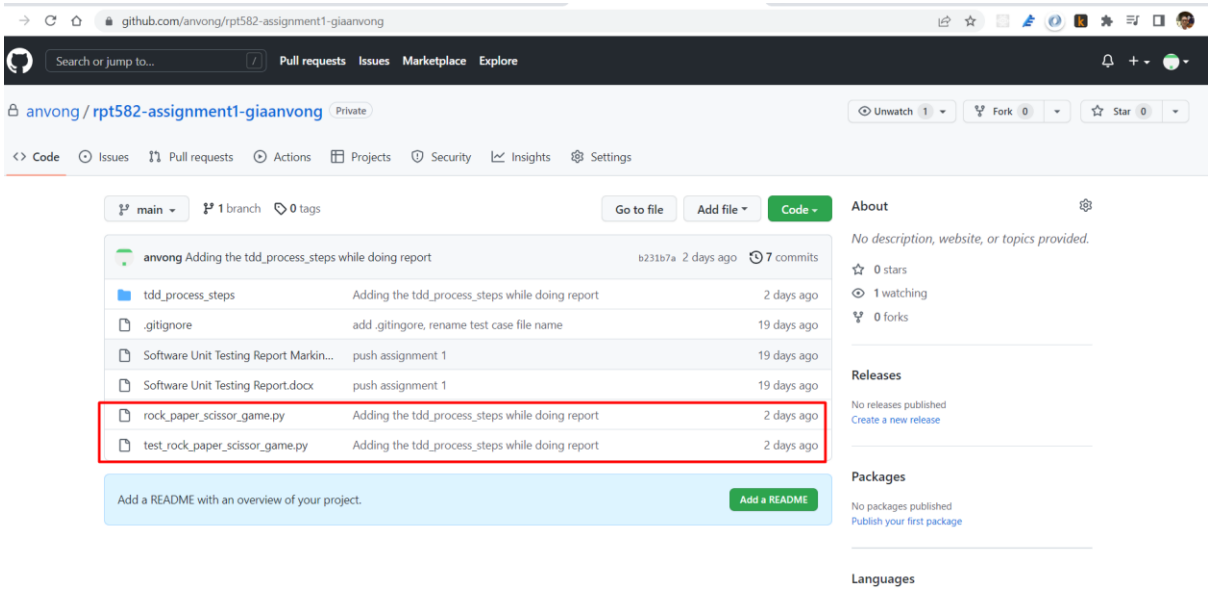
Programmers can utilize the development of a Rock Paper Scissors game as a realistic example of programming logic. However, test-driven development is a best practice inside the Agile methodologies paradigm, which accepts requirement modifications and produces less documentation. Developing unit test cases is the best way to define the requirements and verify that all objects and methods are tested. In the scope of this development and report, we concentrated on constructing test cases before implementing the game object's methods. However, some specific requirements, such as business logic, could not be performed by unit testing. This gap can be closed by human interaction with games to determine their correctness.

In conclusion, programming with the TTD methodology is complex, but it brings more benefits to the product outcomes. In addition, in a real-world development environment with several developers, an automated unit test framework can be coupled with other continuous integration/continuous delivery (CI/CD) processes to keep a product to a high-quality standard.

V. APPENDIX

1. Source code on git hub:

https://github.com/anvong/rpt582-assignment1-giaanvong



2. Game playing and screenshots for each requirement:

Requirement i: The computer randomly picks one of the options of scissor, paper, and rock.

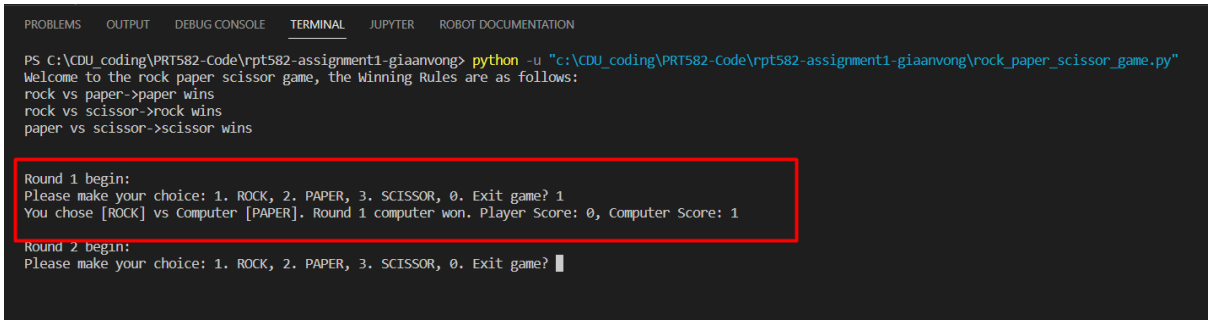


Figure 12: Find the winner and give the point to him during the entire match round

Requirement ii: The player is then given the option to pick/type one of the scissor, paper, and rock options.

Figure 13: The user chose an option of Rock, Paper, Scissor

Requirement iii: One point is given to the winner

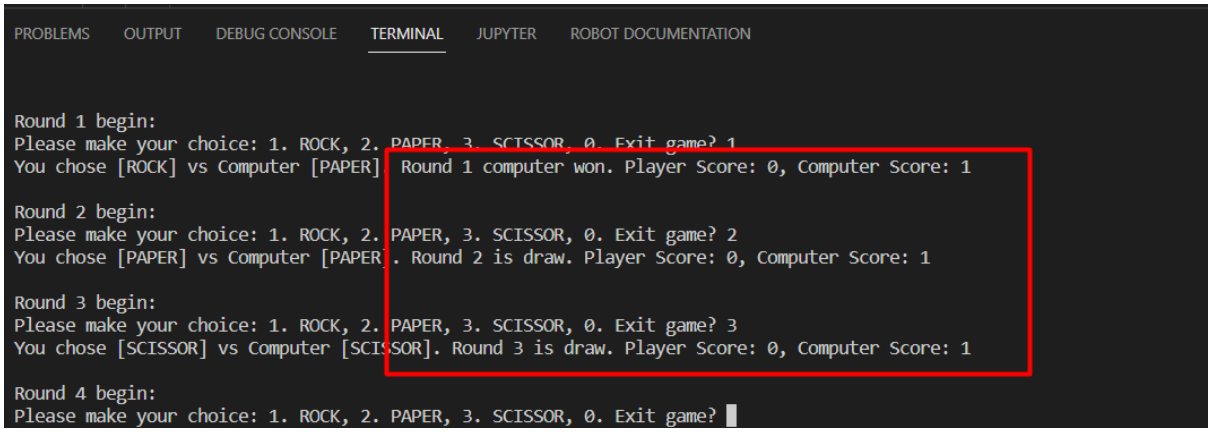


Figure 14: Try several rounds to check the winning rules and score, adding to the winner

Requirement iv:

- It also shows the current round of the match.

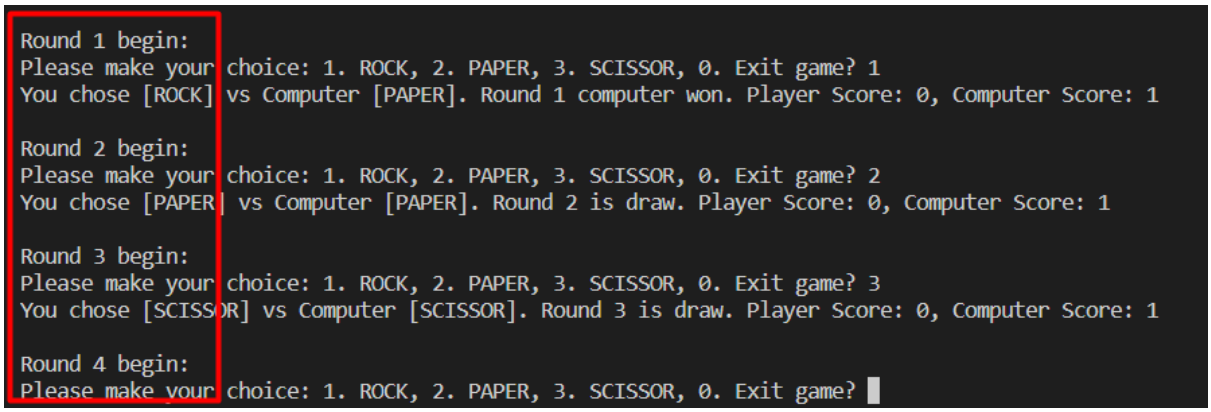


Figure 15: Current round of the match is shown in the game

- Who hits five points first will win the match. The total number of rounds played in total will also be displayed.

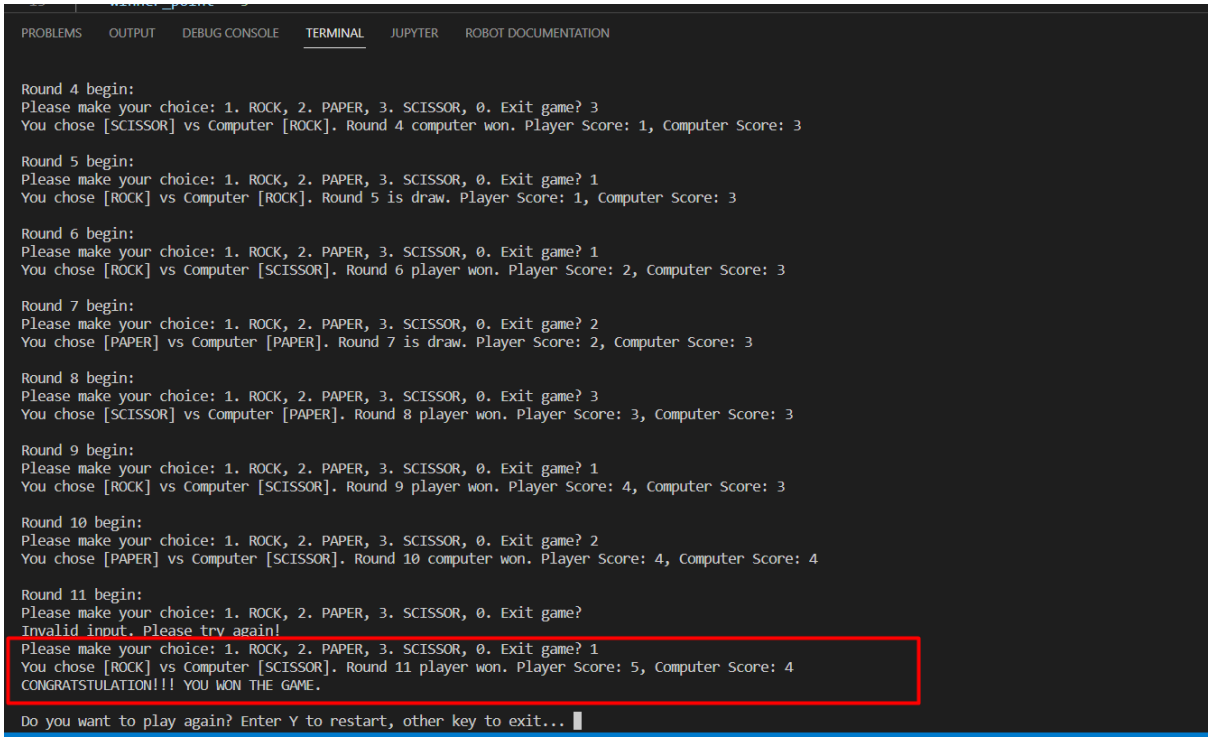


Figure 16: The game winner is determined by whom got 5 points first

Requirement v:

- The game match ends when the winner is determined, and the player is asked to quit or restart the game

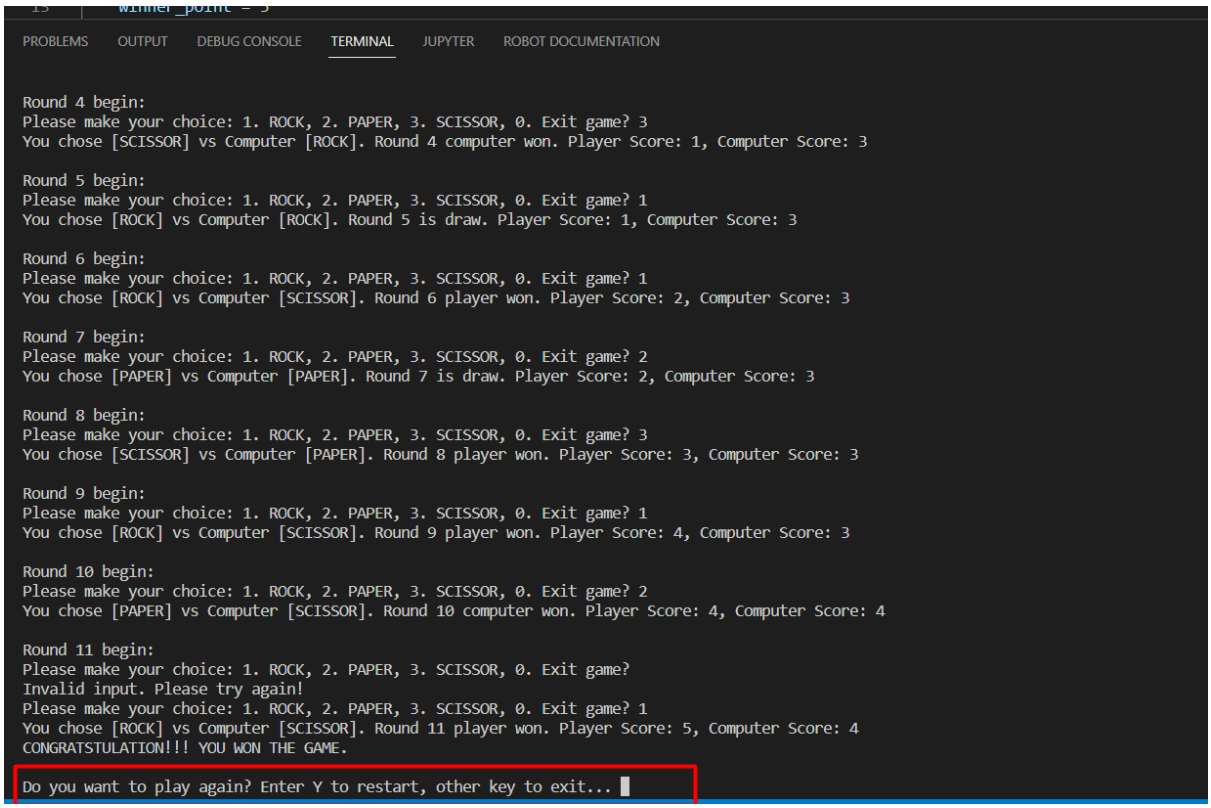


Figure 17: Ask the user to restart the game once the winner is found

- Restart the game if the user selects Y

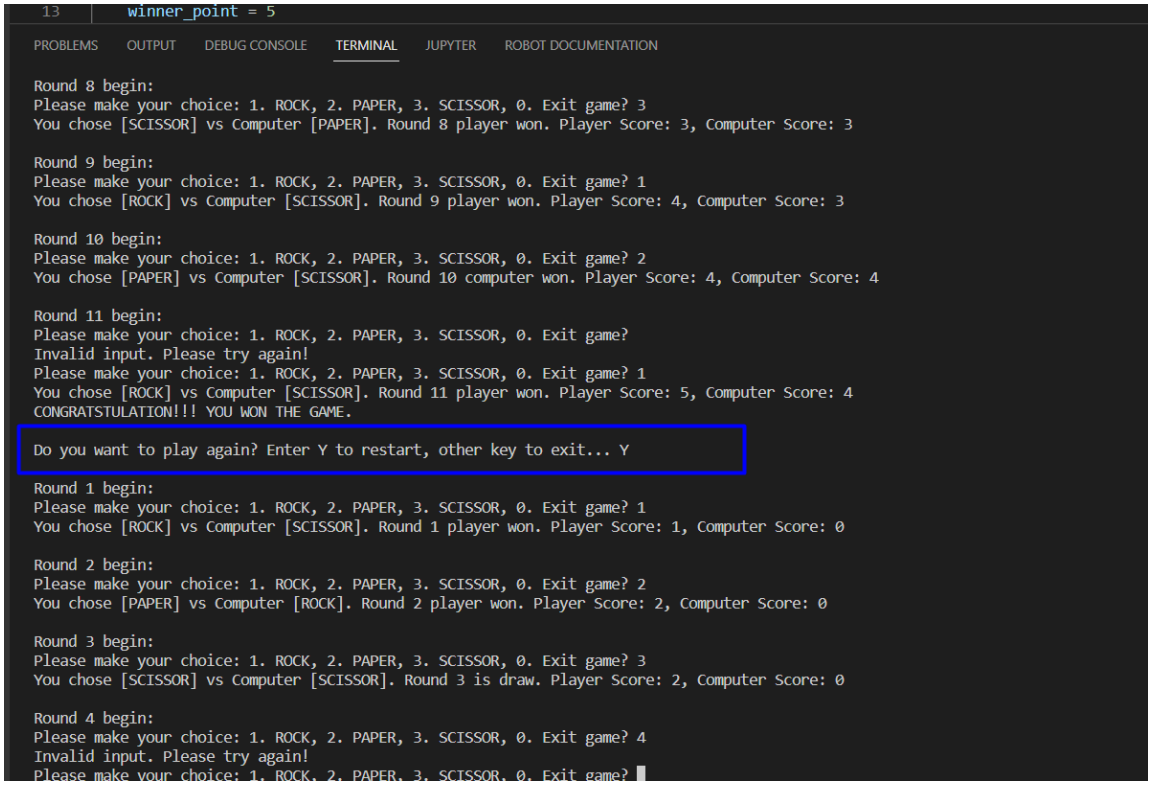


Figure 18: Play a new match after the winner is determined.

Or the user decides not to play a new game. Print out the greeting message.

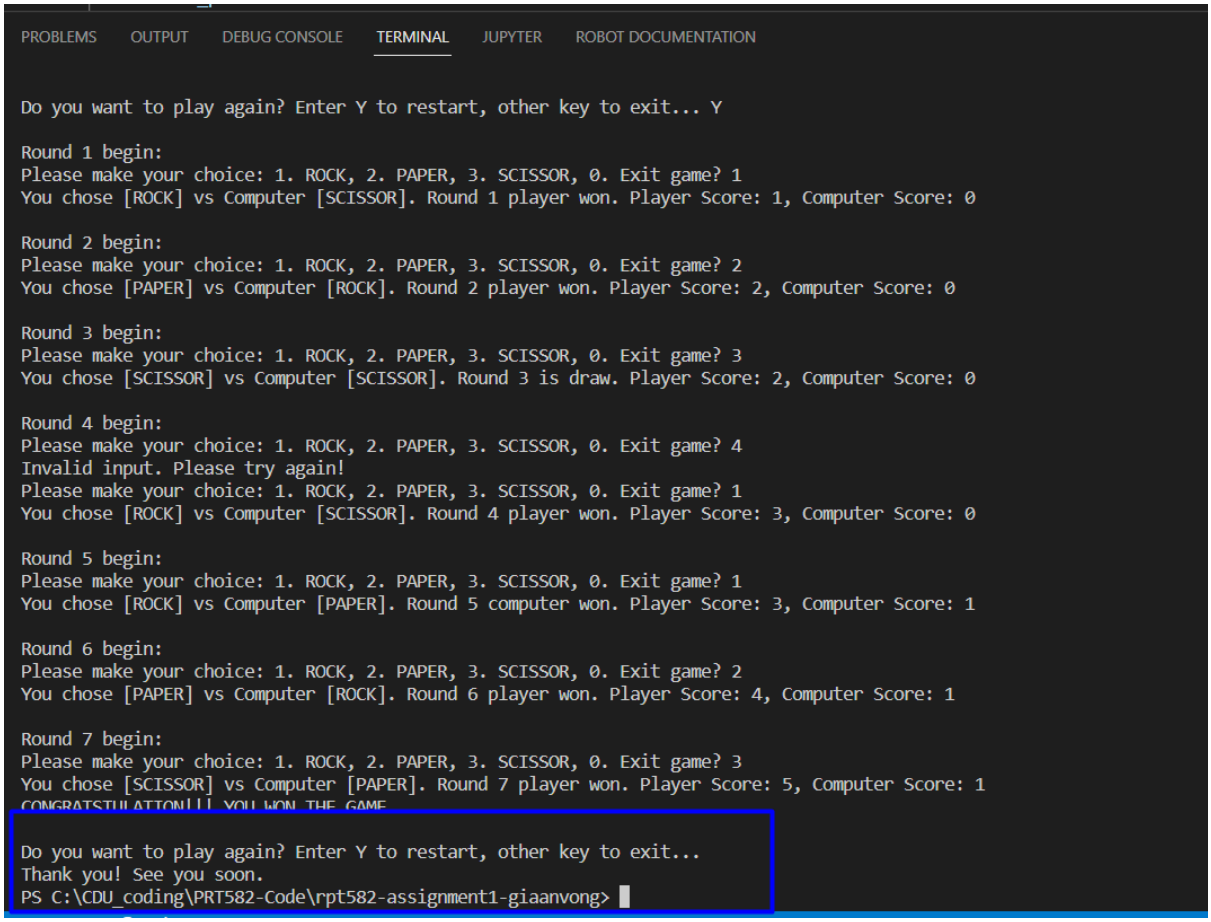


Figure 19: The user does not start a new game.

Requirement vi: While playing, the user can quit the game at any time by inputting 0

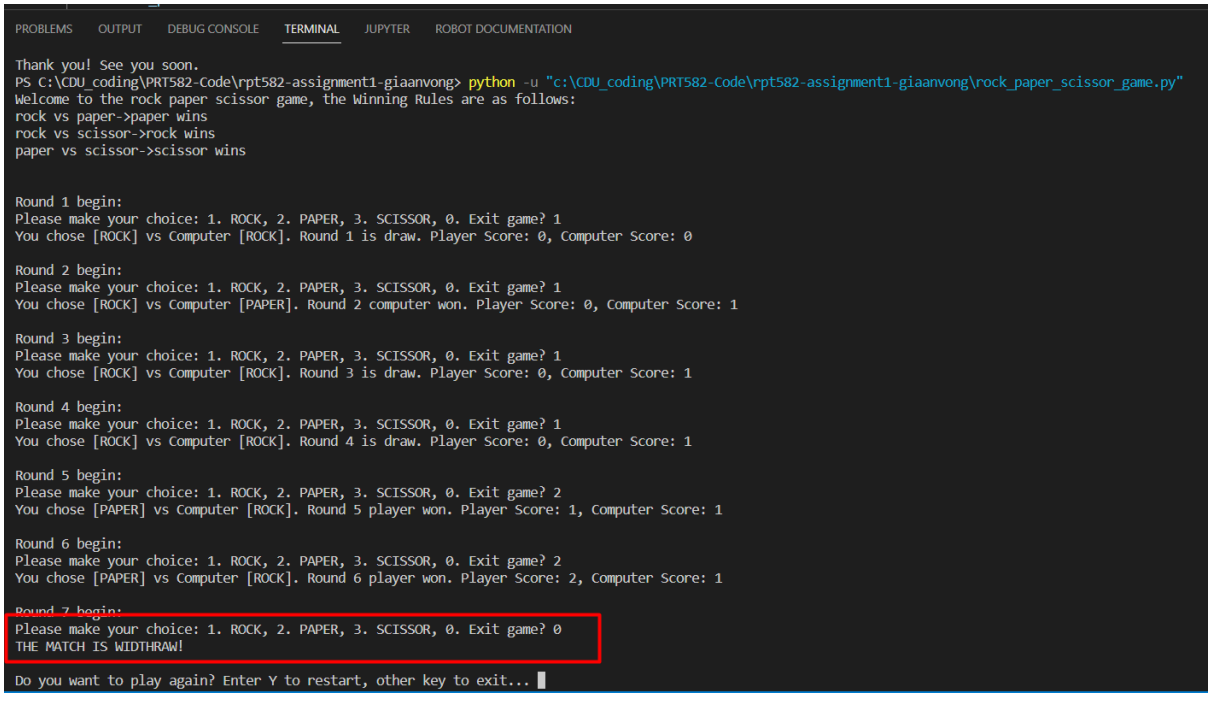


Figure 20: The user decides to quit the game.

The player won the game by hitting 5 points first.

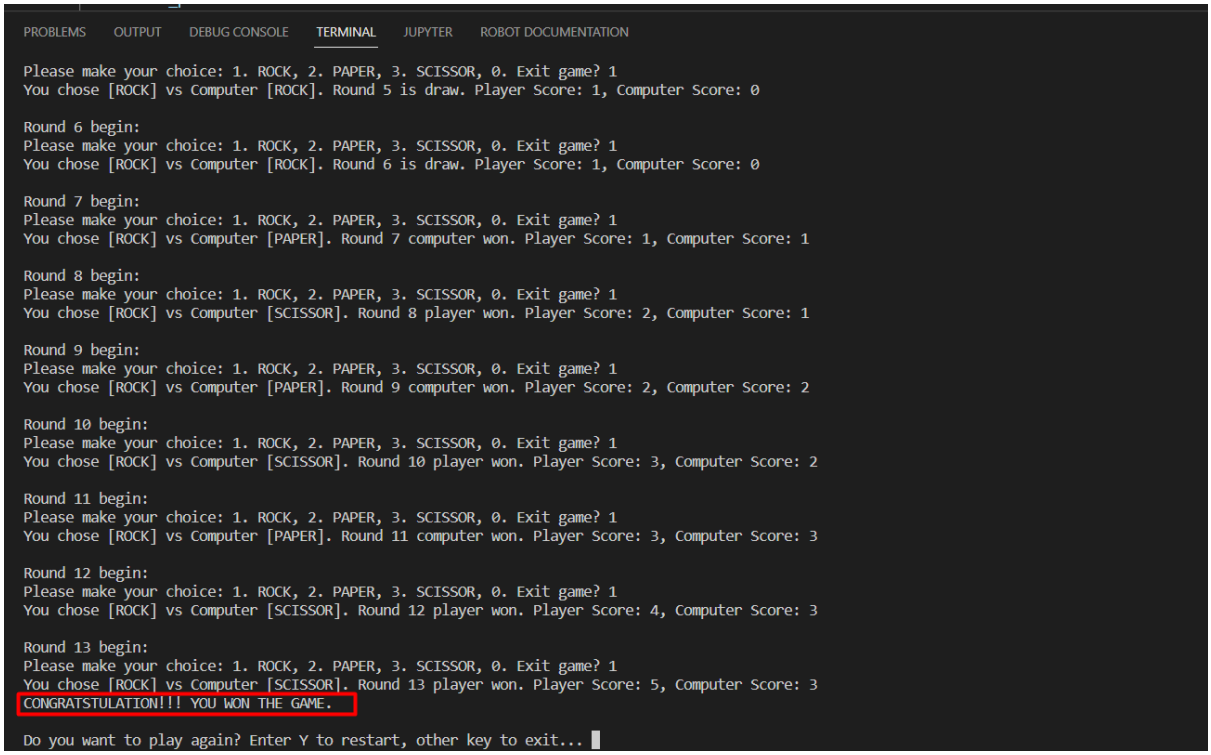


Figure 21: The player won the game by hitting 5 points first

Requirement vii: The winning rules and point giving.

Figure 22: Winning rules displayed at the beginning of the game

Figure 23: The player won the game.

The player lost the game since the computer hit 5 points first

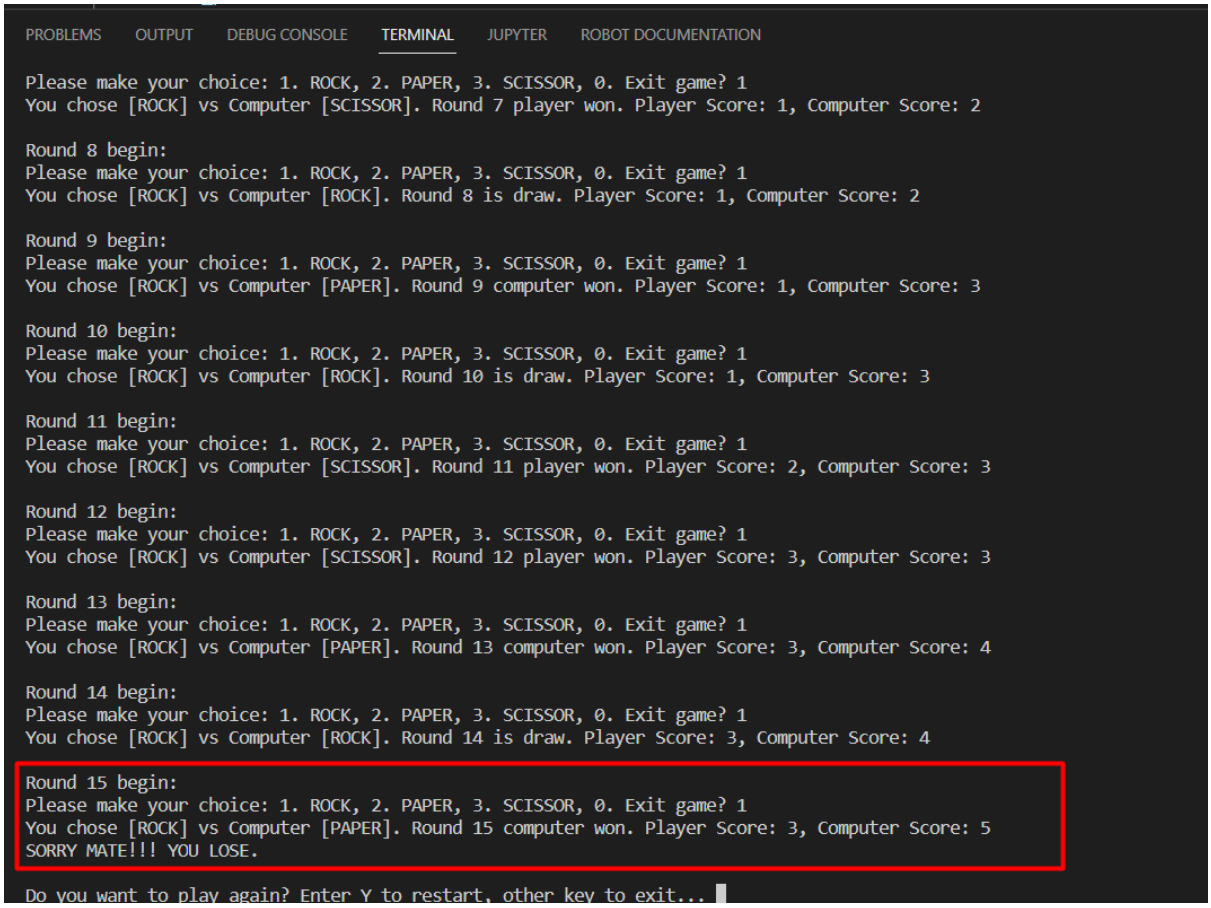


Figure 24: The computer won the game