

**UNIT NAME: SOFTWARE ENGINEERING: PROCESS AND TOOLS (PRT582)**

**PROF: Charles Yeo**

**Assessment 1**

**SUBMITTED BY:**

**STUDENT NAME: Sahil Sharma**

**STUDENT NUMBER: S362216**

**STUDY LOCATION: Casuarina Campus**

Scissor Paper Rock using TDD and Unit Testing

|  |  |  |
| --- | --- | --- |
| Student Name: Sahil Sharma |  | Student ID: S362216 |

# Introduction

Rock, Paper, Scissors is a game in which two players simultaneously select a move rock, paper or scissors. This is then compared against the computer’s selection and determine who the winner is. Test Driven Development approach was used to develop the project. The programming language used is Python and the testing framework used for TDD and Unit Testing is Pytest and flake8.

The winning rules are mentioned in the following :

|  |  |  |  |
| --- | --- | --- | --- |
| sno | Player | Computer | Outcome |
| 1 | Rock | Rock | Draw |
| 2 | Rock | Paper | Player Lose |
| 3 | Rock | Scissors | Player win |
| 4 | Paper | Rock | Player win |
| 5 | Paper | Paper | Draw |
| 6 | Paper | Scissors | Player Lose |
| 7 | Scissors | Rock | Player Lose |
| 8 | Scissors | Paper | Player win |
| 9 | Scissors | Scissors | Draw |

Some of the main requirements are mentioned in the following:

* The computer randomly picks one of the options of scissor, paper and rock.
* Player is then given the option to pick/type one of the options of scissor, paper and rock.
* One point is given to the winner.
* The first to get five points wins the game. The total number of rounds played in total will also be displayed.
* Once the winner is determined, the player is asked to quit or restart the game.
* Player can also quit the game at any time.

**Test Driven Development (TDD)** is a software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.

**Flake8:** PyFlakes, PyCodeStyle, and Ned Batchelder's McCabe script are all wrapped up under the Python module Flake8. It is an excellent toolbox for inspecting your code base for programming problems like "library imported but unused" and "Undefined name" as well as coding style (PEP8) and cyclomatic complexity.

**Pytest:** With pytest , common tasks require less code and advanced tasks can be achieved through a variety of time-saving commands and plugins. It'll even run your existing tests out of the box, including those written with unittest .

# Process

Each unit's (function's) development process began with the creation of test cases specific to that unit. At first, the function would have no logic and be hardcoded to fail the test case (s). After that, the function is created by creating the necessary code for it, and it is examined using the test cases. Additional test cases are prepared (mainly to guarantee it works for special/corner cases to ensure it works for it) once the function passes all the test cases written for it to confirm its correctness. As a result, unit tests are utilised to verify accuracy and each unit is designed to pass the test cases.

3 functions were developed using TDD:

takeUserAndComputerInputs(self)

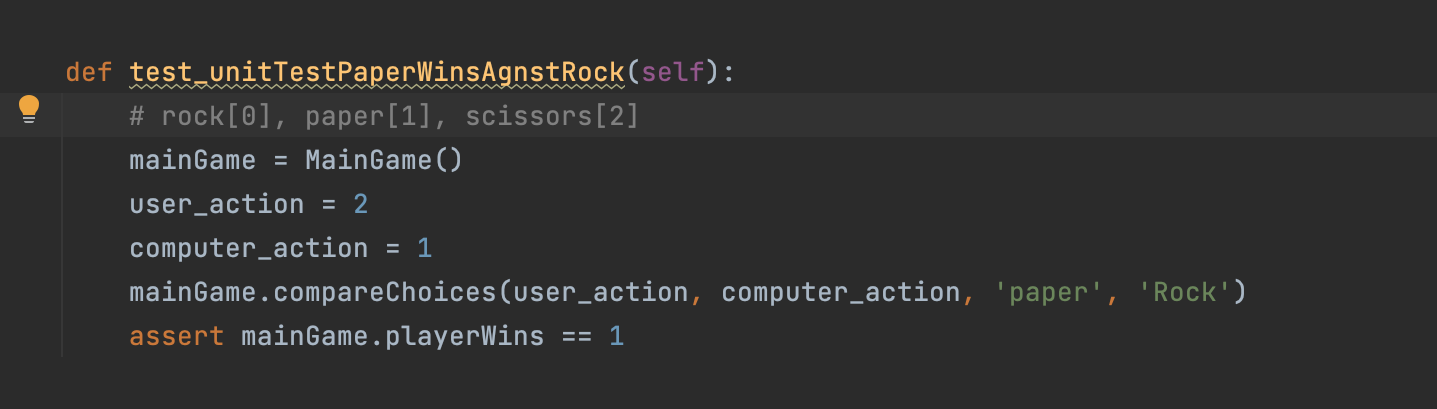
takeComputerInput(self)

CompareChoices(self , p ,c , playerchoicename , computerchoicename)

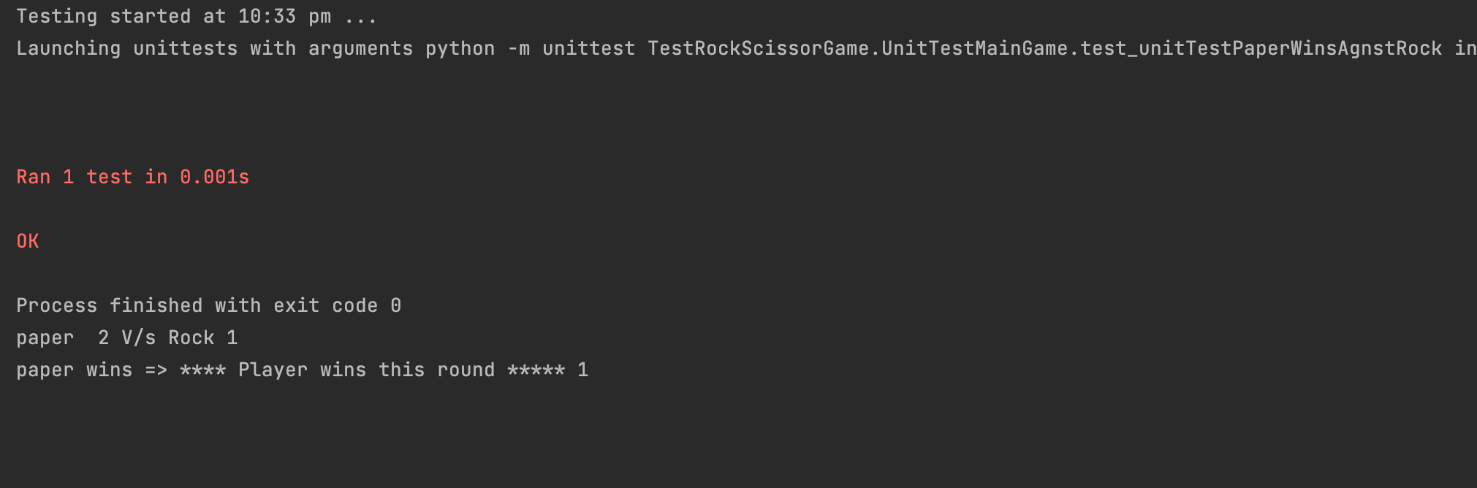
Playgame(self)

doAnywinTheGame(self)

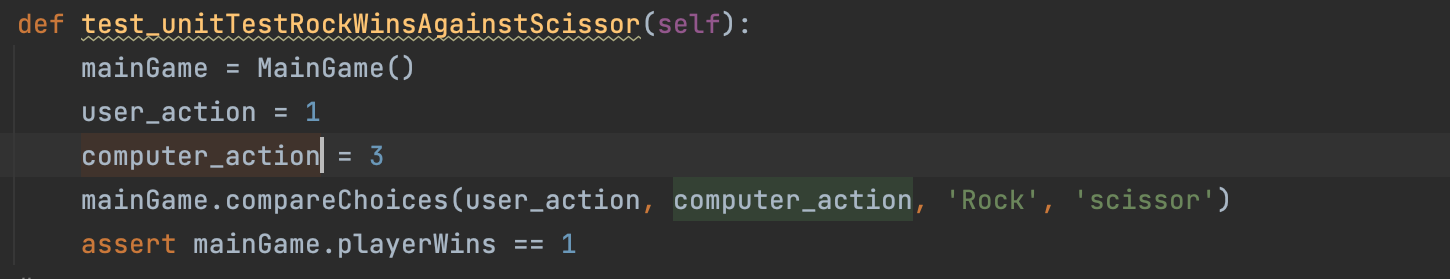
Test Case1:



Test Result:



Testcase 2:



Test Result:

Text

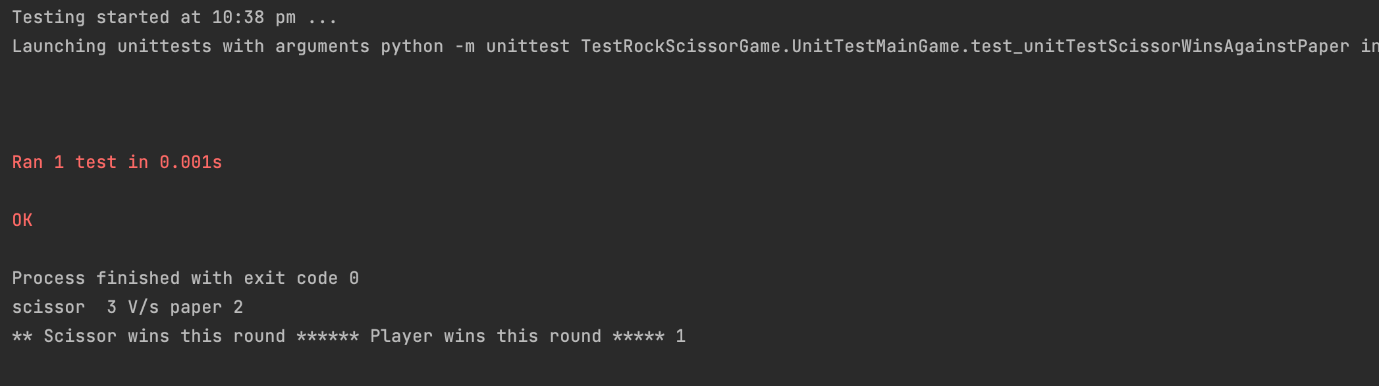
Description automatically generated

Test Case 3:

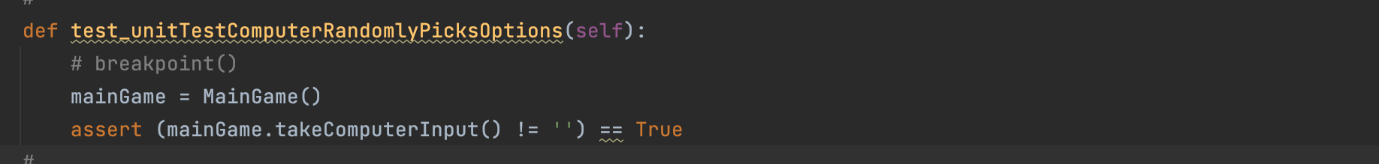
Graphical user interface, text, application

Description automatically generated

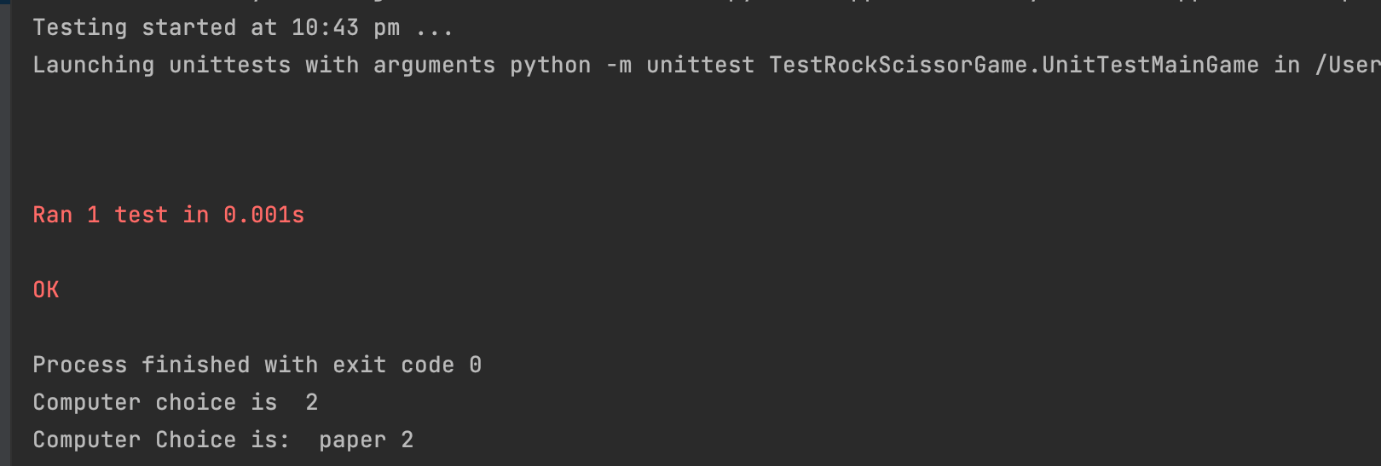
Test Result:



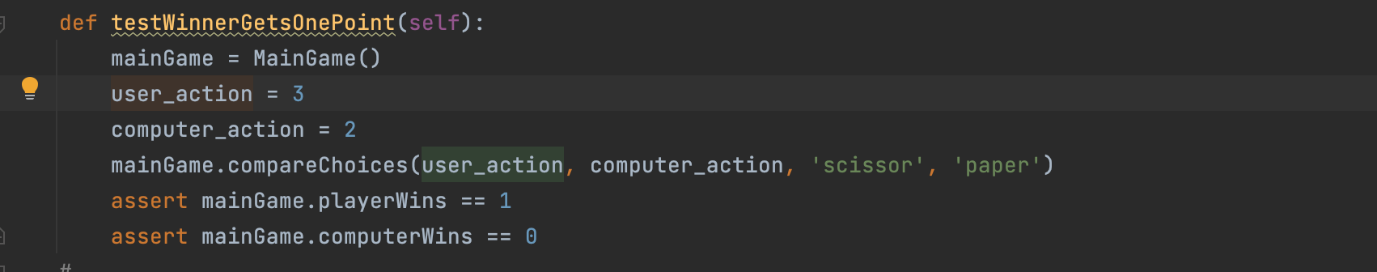
Test Case 4:



Test Result:



Test case 5:



Test Result:

Text

Description automatically generated

Test case 6:

Text

Description automatically generated

Test Result:

Text

Description automatically generated

## Conclusion

Test Driven Development was found to be beneficial in developing the code with different inputs and scenarios in mind. Writing test cases before writing code has led to developing code that is less prone to missing out on some edge cases. Also, writing test cases helped in finding bugs quickly and resolving them (as shown in one of the screenshots). It is learnt that writing good test cases is key for TDD to be effective. TDD’s ability in finding bugs quickly might lead to increasing the pace of development as bugs are found right away when the code is in nascent stages.

Overall, TDD helps in delivering a robust product and increases the pace of development if the test cases are designed smartly.

Github link: https://github.com/sahil64509/HIT582Assign1.git