**Software Unit Testing Report**

**Chao Xie s361499**

**Introduction**:

Outline the objectives and requirements of the game and automated unit testing tool you will be using.

pytest is a unit testing framework for python, similar to the unittest testing framework that comes with python, but simpler and more efficient to use than the unittest framework. According to pytest's official website, it has the following features: very easy to get started, simple to use, rich documentation, and many examples in the documentation.

The game I completed followed the requirements of the topic to the letter. The goal of my test was to check that the game's logic for primarily determining the winner of each round was correct

The main design of the game still follows the requirements of the topic, where the player can choose between scissors, paper and rock. This is then compared with the computer's choice and a determination is made as to who the winner is.

The rules for winning are as follows.

- Rock and Paper -> Paper wins

- Rock vs Scissors -> Rock wins

- Paper vs Scissors -> Scissors wins.

The basic requirements of the game are

i. The computer randomly selects one of scissors, paper and rock.

ii. The player can then select/key one of the scissors, paper and rock options.

iii. The winner scores one point.

iv. The first person to get five points wins the game. The total number of rounds will also be displayed.

v. Once a winner has been determined, the player will be asked to quit or restart the game

vi. Players can also quit the game at any time.

The addScore function, which I wrote, is used as a record of the increase in score by returning a value of 1 for a score, and a value of 0 for no score, i.e. a tie or a loss, corresponding to the occurrence of the two strings on the left for the result of the duel

The requirement was 3 \* 3 = 9 all cases as expected

**Process**:

You should clearly explain how TDD and automated unit testing tool have been used to create your program. Support it with relevant screenshots for each of the requirements.

Writing pytest test samples is very simple and only requires the following rules to be followed.

Test files start with test\_ (ending with \_test is fine)

Test classes start with Test and cannot have an init method

Test functions start with test\_

Assertions are made using the basic assert

The first step is to import the function module in the py file



Then it's time to use the test function's assert to check that the function runs as we expect it to

Anyone who has used the unittest framework knows that unittest encapsulates a lot of assertion methods, including assertEqua, assertNotEqual and dozens of other assertion methods.

assert xx: determine that xx is true

assert not xx: determine that xx is not true

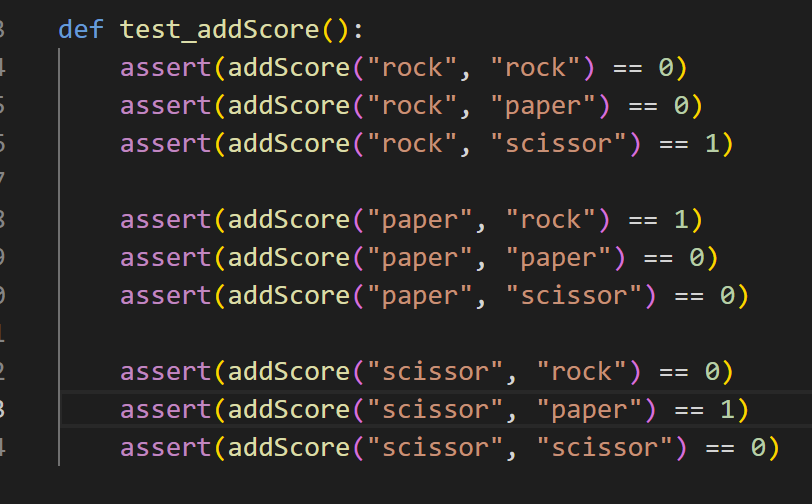
assert a in b: to determine whether b contains a

assert a == b: determine if a is equal to b

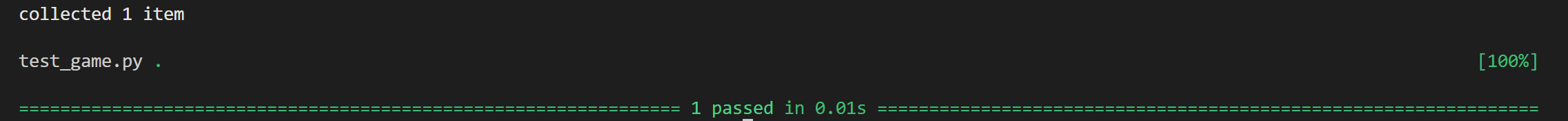
assert a ! =b: determine that a is not equal to b

What judgement the assert is to make can be defined by yourself. It is also possible to follow assert with a description of what happens when the assertion fails: the

assert a>b, 'Assertion failed, actual result is a<b'



Finally look at the results of the pytest test



**Conclusion**:

conclude the report with lessons learnt and your GitHub link.

Using unit tests motivates me to structure my code better, if the code is written in a messy and haphazardly coupled way, it can make it difficult to unit test.

My Github link is:

https://github.com/bawangbense/PRT582-Software-Unit-Testing-Report.git