Student name: Rafsan Rahman
Student ID: 378025
Unit code: PRT582
Unit name: Software Engineering: Process and Tools
Lecturer: Thuseethan Selvarajah
Assignment title: Software Unit Testing Report

Submission date: 06/09/2024

Introduction

Objective

The goal of this project is to create an accurate Scrabble score calculator that uses the letter values from the game to determine a word's score.The program should handle functionalities including word validation, length checking, and score calculation. Users should be able to play the game, enter words within a time constraint, and receive feedback if the words are valid. The game should also track scores across multiple rounds and display the total score upon quitting or after a specified number of rounds.

Justification for Programming Language and Testing Tool

Python is used to implement the project because of its readability, simplicity, and wide library support, all of which speed up development and testing. The unittest framework for Python is used for automated unit testing because it works well with Python and offers a reliable, integrated tool for testing different parts of the program. This program runs pre-written test cases and confirms that the code works as intended, supporting the Test-Driven Development (TDD) technique.

Pylint and Flake8 were also utilized to evaluate the code's style and quality.Pylint offers thorough linting, enforces a coding standard, and checks for potential issues and code flaws. To make sure that the code complies with PEP 8 requirements, flake8 is used for style guide enforcement.Both tools help maintain high-quality, readable, and consistent code.

Process

Use of TDD and Automated Unit Testing

The Scrabble score calculator was made using Test-Driven Development (TDD), which makes sure that every part of the program is tested before it is put into use.

The development process was as follows:

1) **Test Case Design**: Initially, test cases were designed for each functionality based on the requirements. These test cases included checks for score calculation, word length validation, and dictionary validation.
2) **Writing Tests for Implemented Features**: Tests were written using the `unittest` framework. For each requirement, corresponding tests were defined to validate the functionality. Tests were created to check if the following functions correctly computes the score for various inputs, including edge cases.

a) `calculate_score():` Calculates the Scrabble score for a given word and returns it.

A class named `TestScrabbleScore` was created to test all the possible score calculation tests. It checks if the score calculation is accurate for:

I. Single letter
II. Word with lower case only
III. Word with mixed cases

It also checks whether it catches the error if non alphabetic element is given.

```
class TestScrabbleScore(unittest.TestCase):
    """
    Tests for the Scrabble score calculation functions.
    """

    def test_single_letter(self):
        """
        Test the score calculation for a single letter.
        """
        self.assertEqual(calculate_score('A'), 1)

    def test_word_lowercase(self):
        """
        Test the score calculation for a word in lowercase.
        """
        self.assertEqual(calculate_score('cabbage'), 14)

    def test_word_uppercase(self):
        """
        Test the score calculation for a word in uppercase.
        """
        self.assertEqual(calculate_score('CABbAge'), 14)

    def test_invalid_input(self):
        """
        Test the score calculation for invalid(non-alphabetic) inputs.
        """
        with self.assertRaises(ValueError):
            calculate_score('cab123')
```

b) `validate_word_length():` Validate if the word length matches the required length.

A class named TestScrabbleWordLength was created to test the word length match.

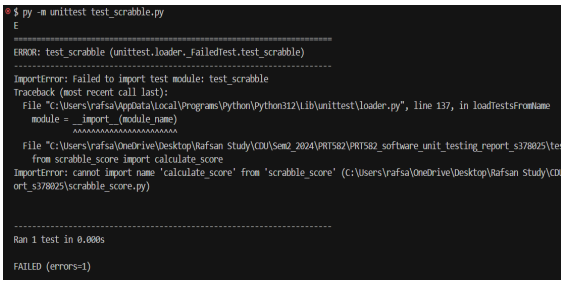
```python
class TestScrabbleWordLength(unittest.TestCase):
    """
    Tests for validating word length.
    """

    def test_valid_length(self):
        """
        Test validation for a word with the correct length.
        """
        self.assertTrue(validate_word_length('hello', 5))

    def test_invalid_length(self):
        """
        Test validation for a word with an incorrect length.
        """
        self.assertFalse(validate_word_length('hello', 6))
```

c) `is_valid_word()`: Check if a word is valid by confirming it exists in the dictionary.

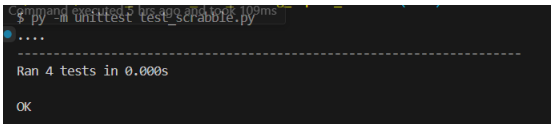
A class named TestScrabbleDictionary was created to test whether the input is a valid word or not.

```python
class TestScrabbleDictionary(unittest.TestCase):
    """
    Tests for checking word validity against the dictionary.
    """

    def test_valid_word(self):
        """
        Test if a word is considered valid.
        """
        self.assertTrue(is_valid_word('hello'))

    def test_invalid_word(self):
        """
        Test if a word is considered invalid.
        """
        self.assertFalse(is_valid_word('xyzzy'))
```

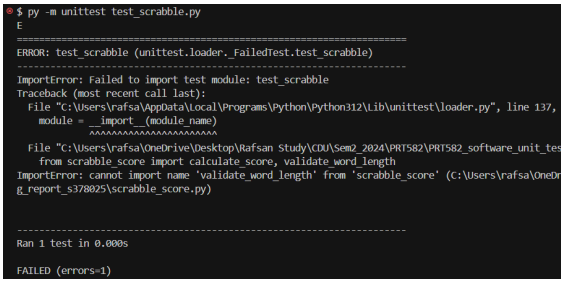
3) **Running Tests**: After implementing each feature, the tests were run to verify correctness. Any failing tests indicated issues that were addressed before moving on to the next feature.
Screenshot of test results of each implemented test cases:



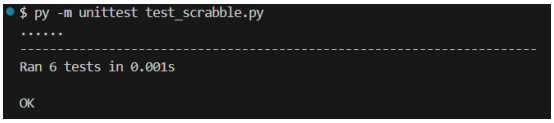
Before implementing calculate_score feature



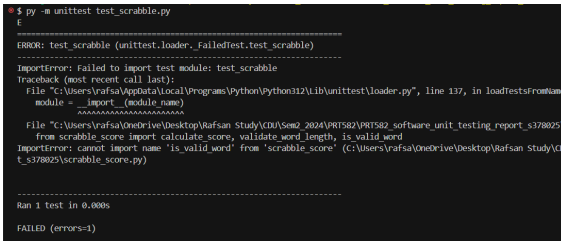
After Implementation



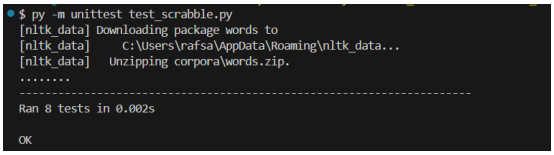
Before implementing validate_word_length feature



After Implementation



Before implementing is_valid_word feature



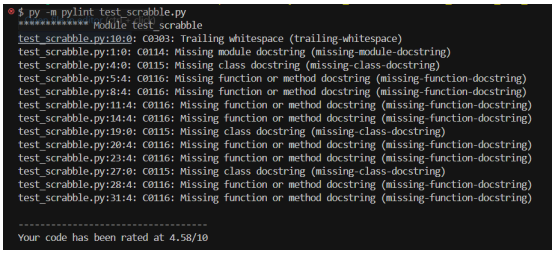
After Implementation

4) **Refactoring**: The code was refactored for readability and efficiency while maintaining test pass rates, taking into account feedback and test results.

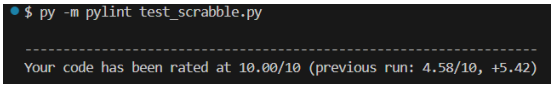
Code Quality and Style Checks

Pylint and flake8 were used to guarantee code quality and conformance to coding standards:

- **Pylint**:Used to carry out thorough code linting, spot possible mistakes, enforce coding standards, and offer advice on how to make the code better.
Screenshot of Pylint output:



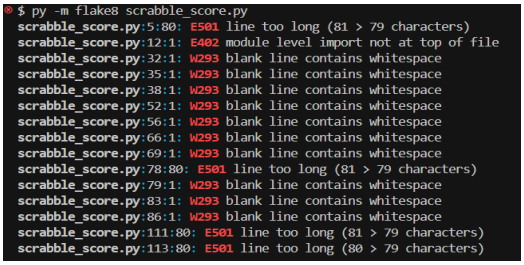
Before implementing pylint



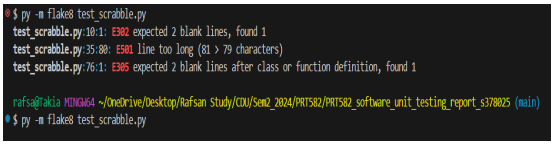
After implementing pylint

- **Flake8**: used to enforce PEP 8 style standards, guaranteeing that the code complies with recommended standards for consistency and readability.

Screenshot of Flake8 output:



Before implementing flake8



After implementing flake8

Conclusion

Throughout the development process, several lessons were learned:

- **What Went Well:** Used Pylint, flake8, unittest, and TDD for a solid development process. TDD ensured proper testing and requirement compliance, while Pylint and flake8 maintained coding standards.
- **Areas for Improvement:** Need to add more game rules, improve the UI, and enhance test coverage with additional edge cases and performance considerations.
- **Future Enhancements:** Introduce more Scrabble rules, features, and a GUI to enhance user experience.

GitHub Repository

The github repository can be visited at: [Scrabble Score Calculator].