**A white and blue cover with black text

Description automatically generated**

**Scrabble Score Calculator Report**

**Introduction**

This project aims to use Test-Driven Development (TDD) and an automated unit testing tool to develop Python software that computes the Scrabble score for a given word. The application complies with several specifications, including case insensitivity, input validation, proper letter scoring, and a game structure. The automated testing tool provides strong validation and continuous integration of updates.

**Programming language:** Python is used for this project because of its many testing frameworks, including 'unittest', it’s readability, and easy access. The unittest module is perfect for beginners and logic and functionality enthusiasts because it offers a comprehensive analysis of game logic.

**Automated unit testing tool:** Python's built-in unittest framework is an automated tool that offers robust functionality for code validation and efficient Python application development.

**Process**:

Requirement 1: Accurate Score Calculation

Every letter in this game has a value, and every word receives a score based on the preset letter values. The program adds these points correctly for each valid word and shows the final score.

* A, E, I, O, U, L, N, R, S, T = 1 point
* D, G = 2 points
* B, C, M, P = 3 points
* F, H, V, W, Y = 4 points
* K = 5 points
* J, X = 8 points
* Q, Z = 10 points

TDD Approach:

* First, a test was designed to determine whether the system can accurately calculate the score for simple words such as "Python."
* The software was then used to pass this test by adding up the values of each letter.
* Finally, additional tests were added to ensure that exceptional cases, such as inputs with upper and lower cases, were handled appropriately.

Automated Unit Test :

A screenshot of a computer program

Description automatically generated

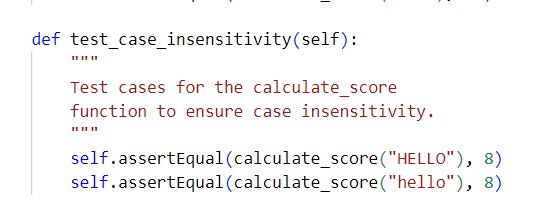
Requirement 2: Case Insensitivity

The input is converted to lowercase before scoring, considering upper- and lowercase letters equally.

TDD Approach:

* A unit test was written to verify that "CABBAGE" and "cabbage" provide the same score; a unit test was developed.
* A lowercase version of every input word is now the default implementation.

Automated Unit Test :



Requirement 3: Alphabet Validation

The game asks players when they type an incorrect alphabetic word. Requests a new input until a valid word is input till that happens.

TDD Approach:

* A test was written to verify non-alphabetic inputs such as numbers or symbols.
* The implementation was improved by alerting users to insert valid words if their input is invalid.

Automated Unit Test :

A computer code with colorful text

Description automatically generated with medium confidence

A screen shot of a computer code

Description automatically generated

Requirement 4: Time-Based Scoring

The user is prompted to enter a word generated at random by a timer that counts down to fifteen seconds. The user receives a message if the input is longer, or the time runs out..

TDD Approach:

* Tests were written to ensure that the timer is correctly displayed and that scoring is affected by the time taken.

Automated Unit Test :

A screenshot of a computer program

Description automatically generated

Requirement 5: Dictionary Validation

A list of dictionary words is compared with the entered word to determine validity. If the term is invalid, it asks the user to enter something valid.

TDD Approach:

* A test was written to check if the word exists in the dictionary.
* The program was implemented to use Python’s PyDictionary module (or similar) to verify word validity.

Automated Unit Test :

A screen shot of a computer code

Description automatically generated

Requirement 6: Game Flow

The user can end the game anytime between 10 rounds. After each game, the player receives feedback on how they performed.

TDD Approach:

* Tests were written to ensure that the game flow continues as expected for multiple rounds and exits gracefully when the user chooses to quit.

Automated Unit Test:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

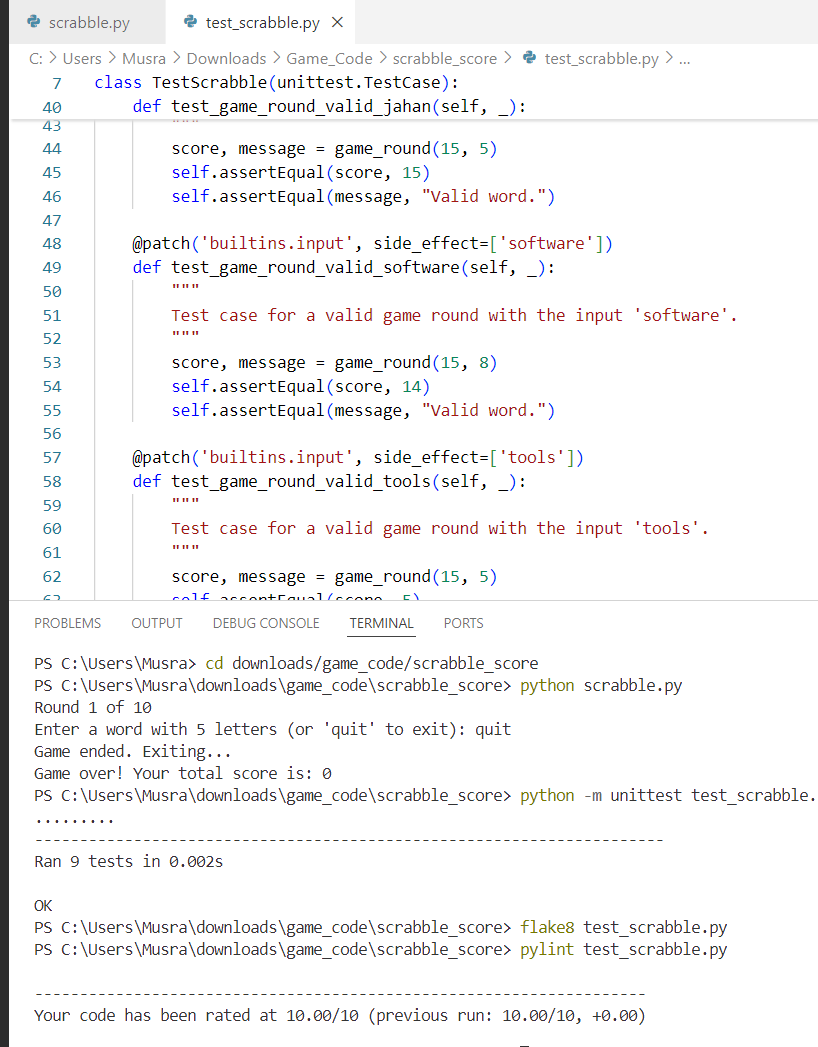
Description automatically generated

**Scrabble Score Code : Flake8 and pylint**

A screenshot of a computer

Description automatically generated

**Test Scrabble Score Code : Flake8 and pylint**

****

**Conclusion**

In conclusion, the scrabble scoring game features dictionary checks, countdown timer, and input validation. Players create words and score based on letter values. The game lasts ten rounds or until players give up. The overall score is displayed at the end. The Python unittest framework was used for automated unit testing, ensuring all requirements were met and the code was correct. Test-driven development and modular design allowed for rapid development and reduced testing times, resulting in an effective and accurate Scrabble scoring game.

**Lessons learned**

* TDD provided for early error detection and design guidance.
* Boundary situations, wrong inputs, and other exceptions in program logic call for prudence.
* The development process improved, and assurance was provided that no new defects would be introduced by changes in the future through automation.

**GitHub Link**

[Musrat-Jahan/PRT582-Software-Unit-Testing-Report (github.com)](https://github.com/Musrat-Jahan/PRT582-Software-Unit-Testing-Report)