**Introduction**

The aim of this project is to create a scoring tool using Test driven Development (TDD) in Python programming language that calculates the score of a word according to the usual letter points system used in Scrabble games for both capital and lowercase letters while also checking and guiding users during word input, within a specific time limit and verifying that the entered word is valid based on a set dictionary of acceptable words. The game was created to be played over rounds with the total score tallied either at the conclusion of the game or, after completing 10 rounds.

We decided to use Python for this project because of its ease of use and readability well as the wide range of libraries available that support Test Driven Development (TDD) and Graphical User Interface (GUI) development needs. Particularly the unittest framework for automated testing and the tkinter library, for building interactive user interfaces.

Automated Testing Tool Choice: We opted for the Python framework, for automated unit testing because of its integrated features that help in orchestrating and executing tests effectively while also producing detailed test reports. This framework empowers developers to create test cases that run automatically to verify the expected functionality of every aspect of the program.

**Process**

**Test-Driven Development (TDD) Approach:**

Step 1: The writing of the Test It involves the creation of the set tests before writing the actual code of the program. It identifies what the program should or should not do, by providing tests meant to check on the expected behavior of the program. For this project, tests were created to ensure the following requirements: For this project, tests were created to ensure the following requirements:

• Any word is then given the correct Scrabble score.

• Both big and a small case letter are treated equally in the program.

• The program is also able to detect that some of the input made are invalid (for example, entry of non-alphabet characters).

• The timer feature is also checked so that it counts down the time and pauses the game when the time is up.

• The program checks whether the given word is present in a lexicon of words taken from a specific document.

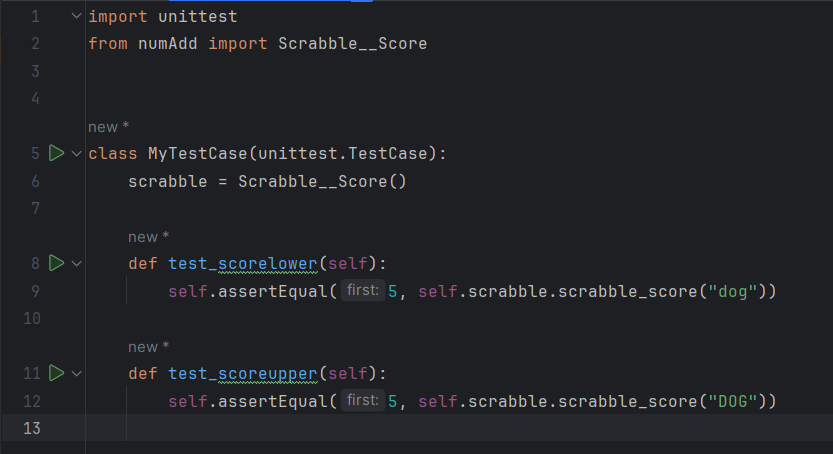
Step 2: Writing the Code After then writing tests, the program code was written to ensure it can meet all the tests. This involved creating functions for calculating scrabble scores, check the validity of the texts entered and controlling the GUI components including the timer. Its designing was done according to the incremental approach: At first, only the most minimum need of the program was addressed, which is to calculate the Scrabble score, and with each iteration, new capability was incorporated (handling different cases, input validation etc. ).

Step 3: Refactoring After running test, the code was rewritten for better quality and efficiency. The changes included the optimising of the function that calculates the scrabble score on the form as well as improving on the GUI to provide the users with a near perfect interface.

**Automated Unit Testing**

**Test Case 1: Score Calculation** A test was written to verify that the Scrabble score for a given word, such as "dog" is calculated correctly. The test checks if the score is computed as 5 points.

**Screenshot:**

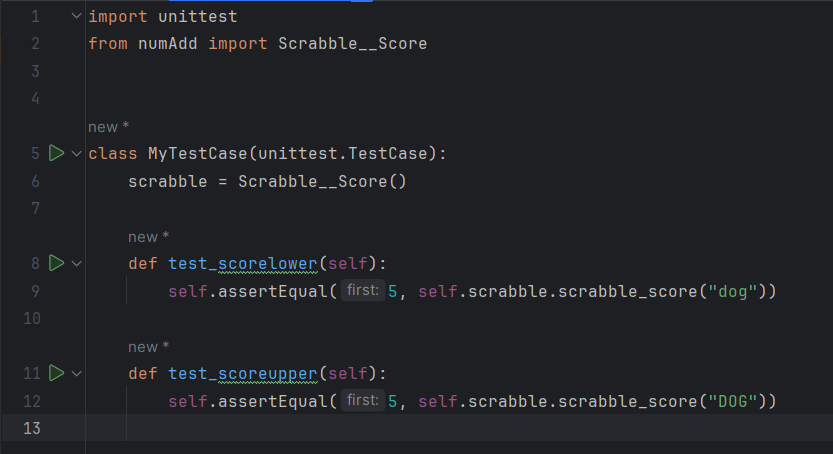


A screen shot of a computer program

Description automatically generated

**Test Case 2: Case Insensitivity** This test ensures that the program treats uppercase and lowercase letters equally.

**Screenshot:**



A screen shot of a computer program

Description automatically generated

**Test Case 3: Input Validation** Tests were implemented to ensure that the program correctly identifies and rejects non-alphabetical characters and invalid words.

**Screenshot:**

A screen shot of a computer

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**Test Case 4: Timer Functionality** A test was created to verify that the countdown timer works as expected, stopping the game when time runs out.

**Screenshot:**

A screenshot of a computer program

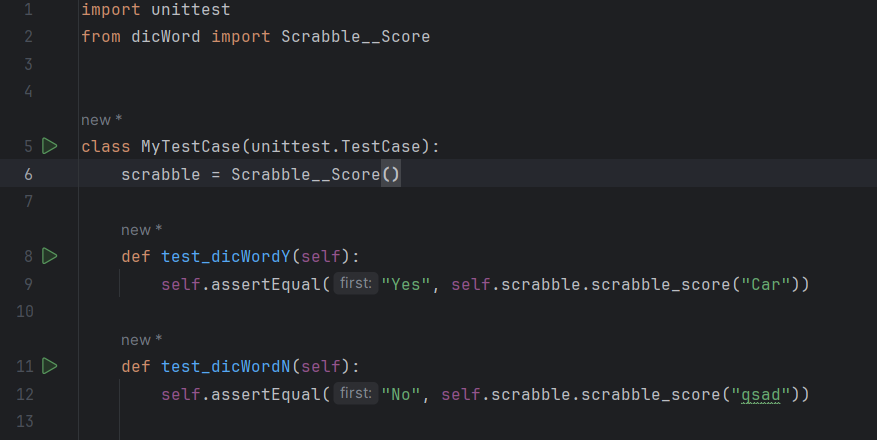
Description automatically generated

A screen shot of a computer program

Description automatically generated

**Test Case 5: Dictionary Word** A test was created to verify that the word is from dictionary or not.

**Screenshot:**



A screen shot of a computer program

Description automatically generated

**Test Case 6: Check Word length** A test was created to verify that the input word limit is same as asked or not.

**Screenshot:**

**A screen shot of a computer program

Description automatically generated**

A computer screen shot of a program

Description automatically generated

**GUI Implementation**

After passing all tests, a graphical user interface was created using tkinter. The GUI allows users to input words, see the remaining time, and receive feedback if their word is valid. The game also terminates if the user fails to submit a word within the allocated time.

**Screenshot:**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Conclusion**

**Lessons Learned:**

* **What Went Well:**
  + **TDD Effectiveness:** Implementing TDD allowed structured approach to development, ensuring that each feature fully tested before being finalized. It minimized bugs and led to a more reliable final product.
  + **Automated Testing with unittest:** It is valuable in automating the testing process, making it easy to rerun tests after each code change. This helped maintain a high level of code quality throughout the project.
* **Areas for Improvement:**
  + **Timer Integration Challenges:** Initially, integrating the countdown timer with the GUI presented challenges, particularly in preventing the GUI from freezing. This was mitigated by using tkinter's after() method, but more efficient methods could be explored in the future.
* **How to Improve:**
  + **Explore Advanced Testing Tools:** Future projects could benefit from exploring other testing frameworks like pytest, which offers more advanced features such as fixtures, parameterized tests, and better reporting capabilities.
  + **Enhance GUI Development Skills:** Learning more about advanced GUI frameworks would allow for the creation of more complex and responsive interfaces, improving the overall user experience.

Overall, this project demonstrated how TDD can be effectively used to create reliable software, and how automated testing tools in Python can simplify the process of ensuring program correctness.

Github Link (Includes Report and python files): <https://github.com/meetsavaliya135520/SUTR>