

Scrabble Game Using Test-Driven Development

Introduction

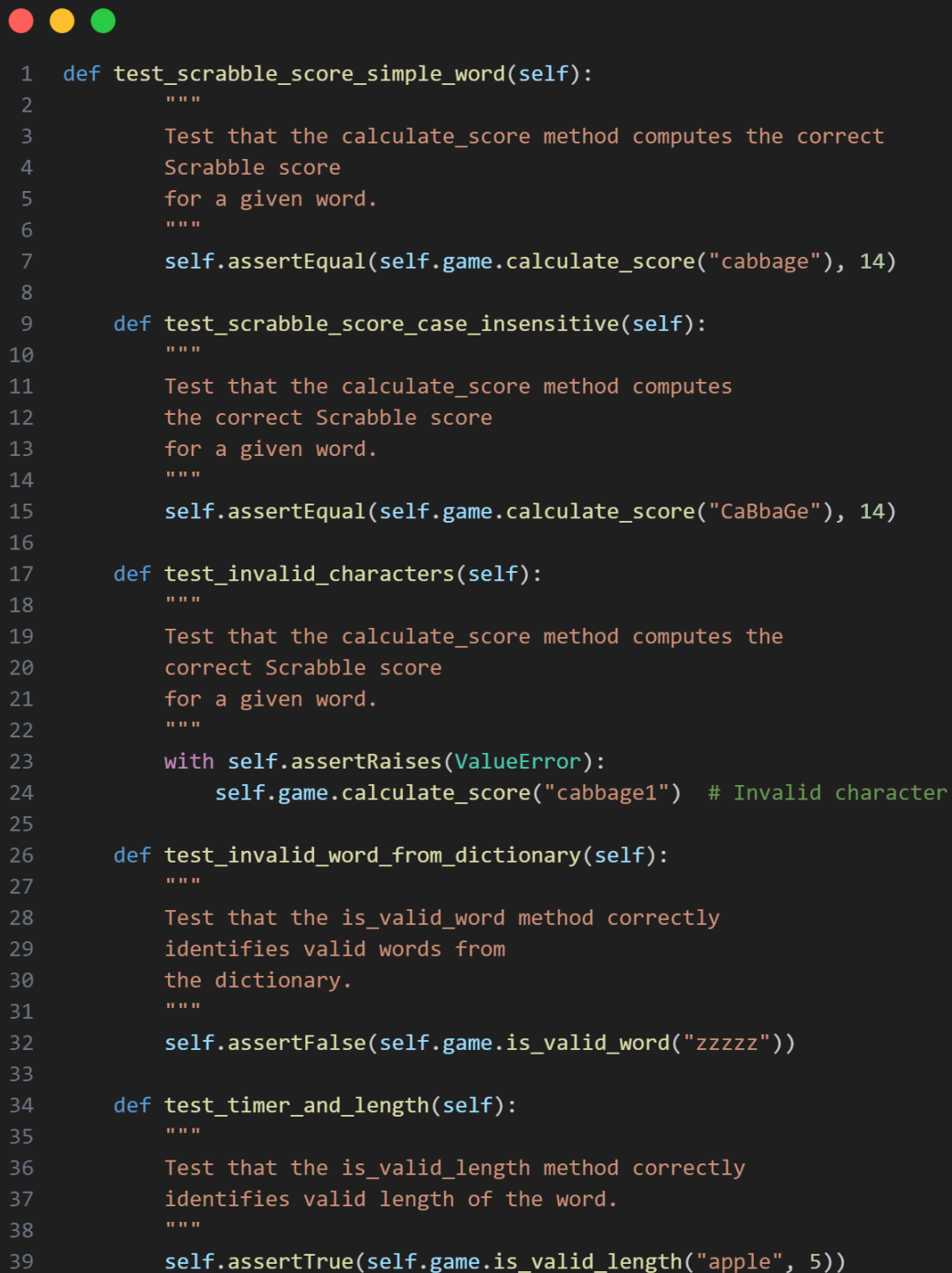
In this project, I worked on developing a Scrabble scoring game using Python. The main goal was to calculate the score for a word based on Scrabble rules, validate user input, and introduce a timer for word entry. To make sure the game worked correctly, I followed the principles of Test-Driven Development (TDD) throughout the project. I chose Python because of its simplicity, built-in support for unit testing, and ability to handle real-time operations like timers.

By leveraging the unittest module, I ensured that each function was working as intended before proceeding with the core logic. This approach gave me confidence that the code wouldn't break when introducing new changes or features.

Process

1. Test-Driven Development (TDD)

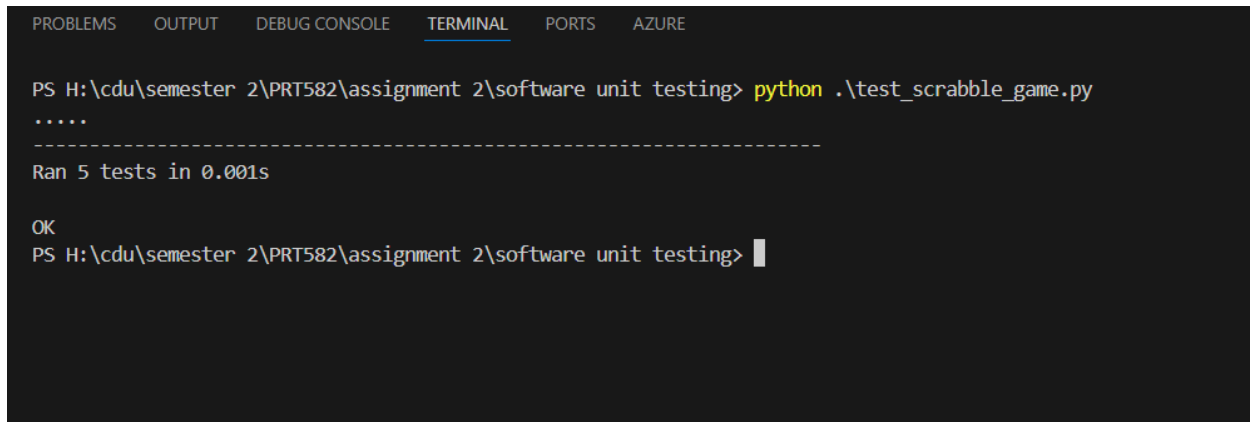
The first step was to clearly define what the program should do: calculate the Scrabble score, validate the length of the word, and ensure the word existed in the dictionary. I applied TDD by writing the unit tests before actually writing the code itself.



```
1 def test_scrabble_score_simple_word(self):
2     """
3     Test that the calculate_score method computes the correct
4     Scrabble score
5     for a given word.
6     """
7     self.assertEqual(self.game.calculate_score("cabbage"), 14)
8
9 def test_scrabble_score_case_insensitive(self):
10     """
11     Test that the calculate_score method computes
12     the correct Scrabble score
13     for a given word.
14     """
15     self.assertEqual(self.game.calculate_score("CaBbaGe"), 14)
16
17 def test_invalid_characters(self):
18     """
19     Test that the calculate_score method computes the
20     correct Scrabble score
21     for a given word.
22     """
23     with self.assertRaises(ValueError):
24         self.game.calculate_score("cabbage1") # Invalid character
25
26 def test_invalid_word_from_dictionary(self):
27     """
28     Test that the is_valid_word method correctly
29     identifies valid words from
30     the dictionary.
31     """
32     self.assertFalse(self.game.is_valid_word("zzzzz"))
33
34 def test_timer_and_length(self):
35     """
36     Test that the is_valid_length method correctly
37     identifies valid length of the word.
38     """
39     self.assertTrue(self.game.is_valid_length("apple", 5))
```

Figure 1: Test code snippet showing how the scoring function and input validation were tested.

Once the tests were in place, I wrote the code to make them pass. For example, I created a test to check if the word "cabbage" would return the correct score of 14 points. If a test failed, I revised the code until all tests passed successfully.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

PS H:\cdu\semester 2\PRT582\assignment 2\software unit testing> python .\test_scrabble_game.py
.....
-----
Ran 5 tests in 0.001s

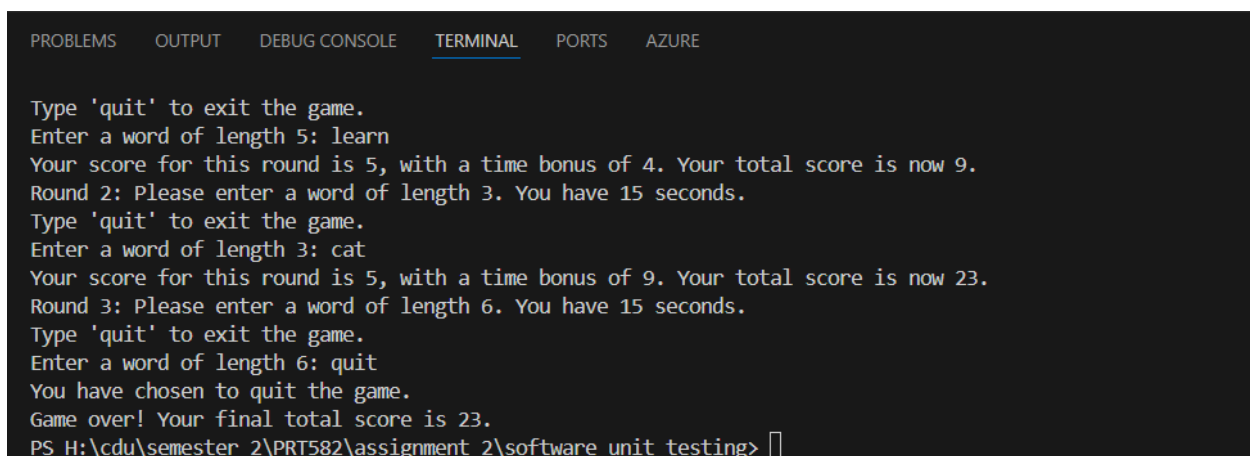
OK
PS H:\cdu\semester 2\PRT582\assignment 2\software unit testing> 
```

Figure 2: Test results in the console showing both passing and failing tests.

2. Automated Testing

To automate the testing process, I relied on Python's unittest module. After each significant change, I ran the tests to ensure that everything still worked correctly. This also helped in identifying edge cases early on, such as:

- Raising an error for non-alphabetic characters.
- Correctly calculating the score when a valid word is entered within the time limit.
- Ensuring case-insensitivity in word scoring.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

Type 'quit' to exit the game.
Enter a word of length 5: learn
Your score for this round is 5, with a time bonus of 4. Your total score is now 9.
Round 2: Please enter a word of length 3. You have 15 seconds.
Type 'quit' to exit the game.
Enter a word of length 3: cat
Your score for this round is 5, with a time bonus of 9. Your total score is now 23.
Round 3: Please enter a word of length 6. You have 15 seconds.
Type 'quit' to exit the game.
Enter a word of length 6: quit
You have chosen to quit the game.
Game over! Your final total score is 23.
PS H:\cdu\semester 2\PRT582\assignment 2\software unit testing> 
```

Figure 3: The Scrabble game running in the terminal, showing the user input and final score.

3. Core Features

Here's a breakdown of the main features I built into the game:

- **Scoring System:** The game uses the standard Scrabble scoring rules. It's also case-insensitive, meaning both "apple" and "Apple" will produce the same score.
- **Input Validation:** I made sure that only valid words from the dictionary are accepted and that only alphabetic characters can be used.
- **Timer and Bonus:** A 15-second timer runs for each round, and the player receives a bonus for entering their word quickly.
- **Game Continuation:** The game can be played for up to 10 rounds, with the option to quit earlier if desired.

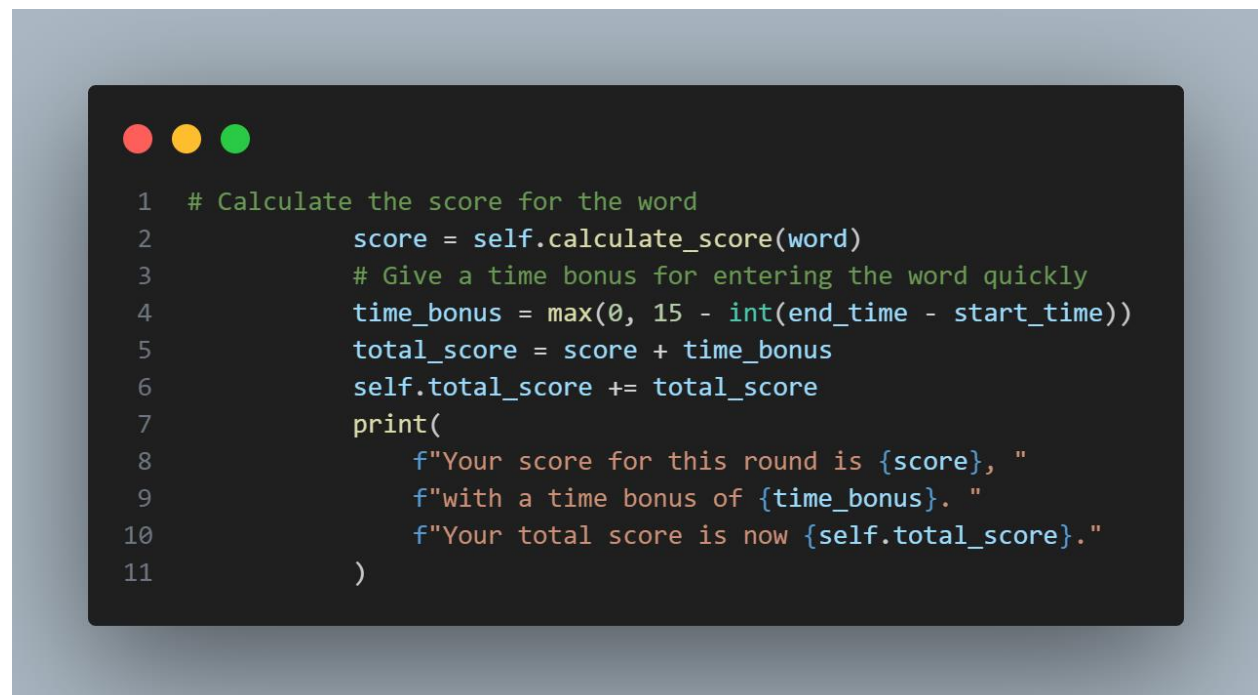


Figure 4: A screenshot showing how the timer and bonus system work during gameplay.

Conclusion

This project was a great exercise in applying TDD principles to a real-world scenario. I learned how valuable it is to plan and write tests before jumping into the actual coding. By doing so, I saved time and reduced potential bugs in the final product. Python's unittest module was extremely helpful in catching issues early, making the process smoother.

Looking back, there are a few areas that could be improved:

- **Improvement 1:** Strengthening input validation to handle more edge cases.
- **Improvement 2:** Optimizing the timer to work better across different input systems.

In conclusion, this project reinforced the importance of testing before coding, as it saves both time and effort in the long run.

GitHub Link: <https://github.com/ahsansaeed878/scrabble-game>