

Introduction

The main goal of this project is to implement the Scrabble game using a test-driven development approach and validate its functionality using the automated unit testing tool. The Scrabble game will calculate the score for user-entered words based on Scrabble rules, validate word lengths, check if the word exists in a dictionary, and ensure proper input validation.

Programming Language: We use Python due to its simplicity, versatility, and wide range of libraries. Python is particularly effective for building the game's logic and the unit tests due to its clear syntax and the availability of testing frameworks such as pytest.

Automated Unit Testing Tool: The testing tool selected for this project is **pytest**. Pytest is a powerful testing tool that supports simple and complex testing needs in Python. It is easy to use, offers detailed output, and integrates well with continuous integration tools. pytest enables us to write test cases that automatically validate the correctness of each function and feature of the Scrabble game.

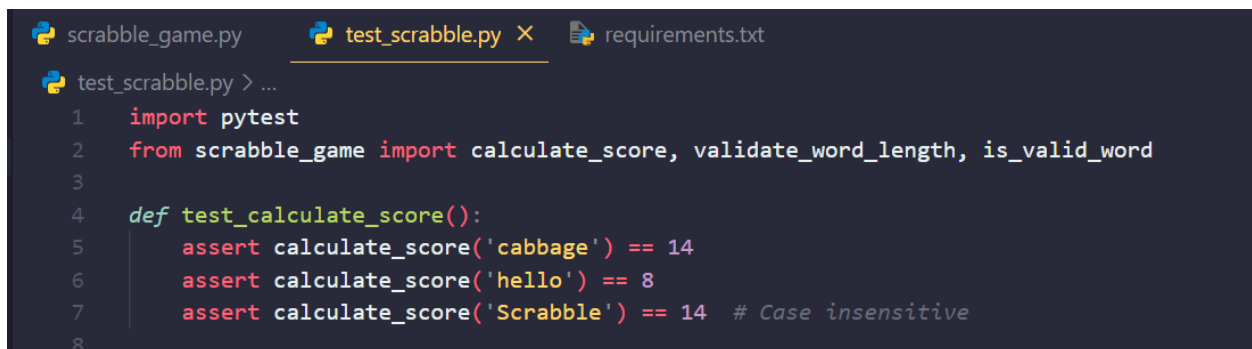
Process

In this section, we outline the process of developing the Scrabble game using TDD and illustrate how pytest has been used to ensure the code meets the requirements.

Requirement-1: The numbers are added up correctly for a given word.

Requirement-2: Upper- and lower-case letters should have the same value.

We first write a test to check if the game calculates the correct Scrabble score for a given word (e.g., "cabbage").

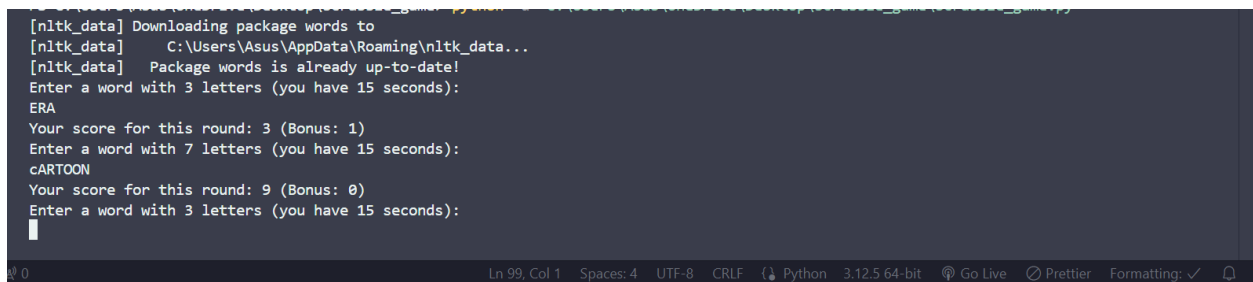


```
scrabble_game.py  test_scrabble.py X  requirements.txt
test_scrabble.py > ...
1  import pytest
2  from scrabble_game import calculate_score, validate_word_length, is_valid_word
3
4  def test_calculate_score():
5      assert calculate_score('cabbage') == 14
6      assert calculate_score('hello') == 8
7      assert calculate_score('Scrabble') == 14  # Case insensitive
8
```

We, then implement the function to calculate the score based on Scrabble letter values.



```
Go Run ... < -> scrabble_game
scrabble_game.py X test_scrabble.py requirements.txt
scrabble_game.py > ...
21
22 # Function to calculate score
23 def calculate_score(word):
24     if not word.isalpha():
25         raise ValueError("Invalid input! Please enter alphabets only.")
26     score = 0
27     for letter in word.upper():
28         score += LETTER_VALUES.get(letter, 0)
29     return score
30
```



```
[nltk_data] Downloading package words to
[nltk_data]   C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
Enter a word with 3 letters (you have 15 seconds):
ERA
Your score for this round: 3 (Bonus: 1)
Enter a word with 7 letters (you have 15 seconds):
cARTOON
Your score for this round: 9 (Bonus: 0)
Enter a word with 3 letters (you have 15 seconds):

```

Requirement-3: prompt user with the right feedback if user does not enter an alphabet.

When the user inputs numbers or a combination of numbers and alphabets, our program warns, "Invalid input! Please enter alphabets only". Below is a screenshot of some manually entered and automated input results.

Below is some manually entered inputs and automated tested input and their results from the code outcome.



```
26
27 # Function to calculate score
28 def calculate_score(word):
29     if not word.isalpha():
30         raise ValueError("Invalid input! Please enter alphabets only.")
31     score = 0
32     for letter in word.upper():
33         score += LETTER_VALUES.get(letter, 0)
34     return score
35
```

```

9 def test_invalid_input():
10     with pytest.raises(ValueError):
11         calculate_score('12345') #not an alphabet
12     with pytest.raises(ValueError):
13         calculate_score('cabbage123')
14     with pytest.raises(ValueError):
15         calculate_score('hello!')
16

```

```

Enter a word with 5 letters (you have 15 seconds):
2131
2131
Invalid input! Please enter alphabets only.
Enter a word with 7 letters (you have 15 seconds):
cinema12
Invalid input! Please enter alphabets only.
Enter a word with 4 letters (you have 15 seconds):
Time's up!

```

Requirement 4: A 15-seconds timer is shown. User is asked to input a word of a certain length. The number of alphabets required in the word is randomly generated. The program will check to ensure that the right length of word is entered before generating the score. Score will be higher if less time is used to enter the right length of word.

We need to validate that the user inputs a word with the required number of letters.

```

Enter a word with 7 letters (you have 15 seconds):
cinema12
Invalid input! Please enter alphabets only.
Enter a word with 4 letters (you have 15 seconds):
Time's up!

```

Fig: 15 sec timer

```

63
64 while round_count < max_rounds:
65     length_required = random.randint(3, 7) # Generate a random word length requirement
66     user_word, elapsed_time = get_user_word_with_timer(length_required)
67

```

Fig: code for randomly generated word length

```

83 # Calculate score based on how quickly the word was entered
84 score = calculate_score(user_word)
85 time_bonus = max(0, 5 - int(elapsed_time)) # Bonus for faster input
86 total_score += score + time_bonus
87
88 print(f"Your score for this round: {score} (Bonus: {time_bonus})")
89 round_count += 1
90

```

Fig: code for time bonus

Requirement 5: Ensure that user enters a valid word from a dictionary. The program will not tabulate the score if the word is not a proper word from a dictionary. Prompts will be given asking the user to enter a valid word if the user does not enter a valid word.

We write a test to verify if the entered word exists in a dictionary (using the nltk.words library).

```
20
21 def test_valid_word_check():
22     assert is_valid_word('hello') # Assume 'hello' is valid
23     assert not is_valid_word('asdfgh') # Assume 'asdfgh' is not valid
24
```

Fig: checking if entered word id valid

We implement the function is_valid_word() using the nltk library.

```
35 # Validate if word is in dictionary
36 def is_valid_word(word):
37     word_list = words.words() # Load dictionary from nltk
38     return word.lower() in word_list
39
40 # Timer-based input prompt
41 def get_user_word_with_time(length_required):
```

Requirement 6: The game will keep going:

- a. Until the player quits the game and display the total score of the player.
- b. After 10 rounds and compute the total score of the player.

The game should stop after 10 rounds and show the final score. To ensure this behavior, we manually test the game's flow after implementing the stopping logic and scoring mechanism.

```
59 def play_game():
60
61     continue
62
63     # Calculate score based on how quickly the word was entered
64     score = calculate_score(user_word)
65     time_bonus = max(0, 5 - int(elapsed_time)) # Bonus for faster input
66     total_score += score + time_bonus
67
68     print(f"Your score for this round: {score} (Bonus: {time_bonus})")
69     round_count += 1
70
71     # Game over after 10 rounds, print total score
72     print(f"Game Over! Your total score after {max_rounds} rounds: {total_score}")
73
```

Fig: code for calculating final score

```

Invalid word length! You need a word of 3 letters.
Enter a word with 6 letters (you have 15 seconds):
camera
Your score for this round: 10 (Bonus: 0)
Enter a word with 4 letters (you have 15 seconds):
from
Your score for this round: 9 (Bonus: 1)
Enter a word with 4 letters (you have 15 seconds):
frpm
'frpm' is not a valid word.
Enter a word with 7 letters (you have 15 seconds):
cabbage
Your score for this round: 14 (Bonus: 0)
Enter a word with 4 letters (you have 15 seconds):
from
Your score for this round: 9 (Bonus: 0)
Game Over! Your total score after 10 rounds: 85
PS C:\Users\Asus\OneDrive\Desktop\scrabble_game>

```

Fig: Final score

Finally, here is the screenshot of all the test cases and the scripts running successfully.

```

Game Over! Your total score after 10 rounds: 85
PS C:\Users\Asus\OneDrive\Desktop\scrabble_game> pytest
● ===== test session starts =====
platform win32 -- Python 3.12.5, pytest-8.3.2, pluggy-1.5.0
rootdir: C:\Users\Asus\OneDrive\Desktop\scrabble_game
collected 4 items

test_scrabble.py .... [100%]

===== 4 passed in 4.61s =====
PS C:\Users\Asus\OneDrive\Desktop\scrabble_game>

```

Conclusion

Using **Test-Driven Development (TDD)** and the automated tool **pytest**, we ensured that all features of the Scrabble game were thoroughly tested and worked as expected. TDD helped us focus on writing only the necessary code and ensured the functionality of each feature before moving to the next. The automated tests allowed for rapid feedback during development, improving the program's reliability. One thing that could be improved about the code is that we could add a prompt to quit the game in the midway. Where users decide, they don't want to continue and calculate their score.