Software Unit Testing Report

Name: Mingzhen Dong
StudentID: s371932
Date: 06/Sep/2024

Table of Content

1. Introduction

The report presents the development of a python game designed to calculate scrabble scores for input words. This program was developed using Test-Driven Development(TDD) and tested with the unittest framework.

1.1. Objective

The project is to implement a python program that calculate scrabble score of a word, validates user input, and provides a clear interface for playing a scrabble game with some boundaries like time limits and word length. The game handles edge cases like case sensitivity, invalid inputs and valid words.

1.2. Overview

The main features include:

Score Calculation: computes the score based on word letter values.

User Input Handling: only valid words are accepted.

Time-Based Score: adjusts scores based on the time left to input.
Random Length: randomly selects a valid word length for each round.
Game: allow multiple rounds until the user quits or all rounds ends.
Dictionary Validation: validates the words by checking against a dictionary.

1.3. Library Selection

For word validation, I researched pyenchant, NLTK, and online verification.
Pyenchant requires the installation of the Enchant library on the system. On Windows, it needs a precompiled wheel to be installed. On my Mac, I encountered issues with Enchant not being found, even after installing it via Homebrew.
NLTK does not have the dependency issues associated with precompiled libraries; it only requires downloading the necessary dictionary libraries as needed.
Online verification requires real-time internet access, which is not ideal for a Scrabble game.
Therefore, I chose to use **NLTK**.
For user interface management, **curses** is a standard library already integrated with Python on UNIX systems. For Windows, just install `windows-curses` through **pip**, which makes it convenient for terminal-based game interaction.
For testing, I selected the **unittest** module, which is built-in and suitable for TDD development. Mocking functionalities also allow for simulation of user inputs.

2. Development Process

This part outlines the steps to implement the game using TDD.

2.1. Test-Driven Development (TDD) Approach

The development process followed the TDD cycle:

1. Write a Test
2. Run the Test
3. Implement the Code
4. Refactor
5. Repeat

2.2. Initial Tests and Implementing Code

The initial tests focused on functionalities below:

Score Calculation

```python
class ScoreCalculatorTestCase(unittest.TestCase):

    def setUp(self):
        """Setup for ScoreCalculator tests."""
        self.calculator = ScoreCalculator()

    def test_score_calculation(self):
        """Test the score calculation for a word."""
        self.assertEqual(self.calculator.calc("cabbage"), 14)
        self.assertNotEqual(self.calculator.calc("scrabble", timeleft=10), 14)
```

```
(v3.11) Open file in editor (cmd + click) ster) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_score_calculation
Traceba
  File "/Users/randall/workspace/CDU/PRT582/assignment2/scrabble_score_tdd/test_ScrabbleScore.py", line 3, in <module>
    from ScrabbleScore import LETTERS
  File "/Users/randall/workspace/CDU/PRT582/assignment2/scrabble_score_tdd/ScrabbleScore.py", line 151, in <module>
    class ScrabbleScore:
  File "/Users/randall/workspace/CDU/PRT582/assignment2/scrabble_score_tdd/ScrabbleScore.py", line 152, in ScrabbleScore
    scoreCalc = ScoreCalculator()
                ^^^^^^^^^^^^^^^
NameError: name 'ScoreCalculator' is not defined
(v3.11) → scrabble_score_tdd git:(master) ✗ python test_ScrabbleScore.py ScoreCalculatorTestCase.test_score_calculation
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Case Insensitivity

```python
def test_score_case_insensitivity(self):
    """Test score calculation is case insensitive."""
    self.assertEqual(
        self.calculator.calc("Scrabble"), self.calculator.calc("scrabble")
    )
```

```
(v3.11) → scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_score_case_insensitivity
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

2.3. Edge Cases and Input Validation

Empty input: ensure the program returns 0 score for empty strings.

```python
def test_empty_word(self):
    """Test calculation for an empty word."""
    self.assertEqual(self.calculator.calc(""), 0)
```

```
(v3.11) ➜  scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_empty_word
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Non-alphabetic letter: inputs containing numbers or symbols returns 0 score.

```python
def test_non_alphabetic_characters(self):
    """Test score calculation for non-alphabetic input."""
    self.assertEqual(self.calculator.calc("1234"), 0)
    self.assertEqual(self.calculator.calc("!@#$"), 0)
    self.assertEqual(
        self.calculator.calc("hello123"), 8
    )  # Only letters should count
```

```
(v3.11) ➜  scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_non_alphabetic_characters
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(v3.11) ➜  scrabble_score_tdd git:(master)
```

Long input: checked performance and correctness for very long inputs.

```python
def test_extremely_long_word(self):
    """Test calculation for a long word."""
    long_word = "a" * 1000
    self.assertEqual(self.calculator.calc(long_word), 1000)
```

```
(v3.11) ➜  scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_extremely_long_word
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Timing: verified behavior when timeleft is 0 or negative.

```python
def test_timing_edge_cases(self):
    """Test timing edge cases."""
    self.assertEqual(self.calculator.calc("scrabble", timeleft=0), 0)
    self.assertEqual(self.calculator.calc("scrabble", timeleft=-5), 0)
```

```
(v3.11) → scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_timing_edge_cases
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(v3.11) → scrabble_score_tdd git:(master)
```

Invalid words: words not found in the dictionary are well handled.

```python
def test_check_spelling(self):
    """Test word validation against the dictionary."""
    self.assertTrue(self.game.check_spelling("apple"))
    self.assertFalse(self.game.check_spelling("xyzabc"))
    self.assertFalse(self.game.check_spelling(""))
```

```
(v3.11) → scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScrabbleScoreTestCase.test_check_spelling
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

2.4. Integrating Timer and Word Length Features

Countdown Timer: adjust scores based on how quick the user input.

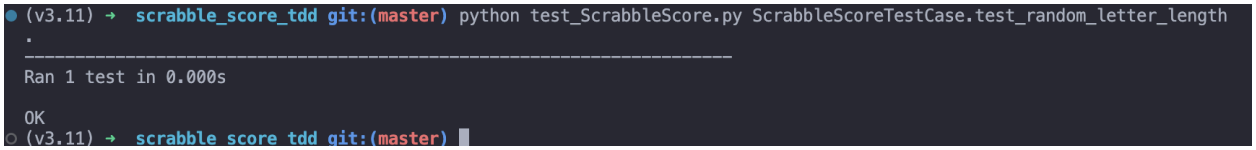
```python
def test_score_calculation(self):
    """Test the score calculation for a word."""
    self.assertEqual(self.calculator.calc("cabbage"), 14)
    self.assertNotEqual(self.calculator.calc("scrabble", timeleft=10), 14)
```

```
(v3.11) → scrabble_score_tdd git:(master) python test_ScrabbleScore.py ScoreCalculatorTestCase.test_score_calculation
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Random Word Length: the function random_letter_length was tested to ensure it generates reasonable length.

```python
def test_random_letter_length(self):
    """Test random letter length generation."""
    length = self.game.random_letter_length()
    self.assertIsInstance(length, int)
    self.assertGreater(length, 0)
    # self.assertLessEqual(length, 15)
```



3. Test Cases

Test Case ID	Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC_SCORE_01	Verify correct score calculation for a valid word	Enter the word “cabbage”		Word: “cabbage”	Score: 14	As Expected	Pass
TC_SCORE_02	Verify score calculation is case-insensitive	Enter the word “Scrabble” in cases: “Scrabble”, “scrabble”		Words: “Scrabble”, “scrabble”	Scores are equal	As Expected	Pass
TC_INPUT_01	Validate handling of numbers or symbols	Enter a string with numbers and symbols like “1234”, “!@#$”		Input: “1234”, “!@#$”	Score: 0	As Expected	Pass
TC_INPUT_02	Verify an empty string	Enter empty string “”		Input: “”	Score: 0	As Expected	Pass
TC_EDGE_01	Verify calculation for a long word	Enter a long word (“a” * 1000)		Word: “a” * 1000	Score: 1000	As Expected	Pass
TC_TIMER_01	Verify score by	Set different		Time values: 0,	Scores based on	As Expected	Pass

	different time values	time values and calculate scores		-5, 99999	time values		
TC_DICT_01	Validate word against dictionary	Enter a valid and invalid word (“apple”, “xyzabc”)		Words: “apple”, “xyzabc”	“apple” is valid, “xyzabc” is invalid	As Expected	Pass
TC_RANDOM_01	Verify random word length	Run the length generator multiple times		N/A	Random length is an integer and > 0	As Expected	Pass

4. Challenges and Solutions

4.1. Difficulties in Implementing

The challenges included handling complex user input scenarios and integrating real-time components like a timer.

I used the curses library to manage user interaction scenarios, allowing the timer to be displayed in real-time. However, unit testing with curses is difficult to simulate, so I extracted part of the business logic for separate testing. To implement the real-time timer, I used multithreading to handle different rounds and timing functions.

4.2. Issues During Unit Testing

Testing user input handling required using mock techniques to simulate user behavior. I addressed this issue by using the mock module from unittest.

5. Conclusion

5.1. Thoughts

The project successfully met all requirements, and the use of TDD and unittest ensured high-quality, maintainable code.

5.2. GitHub Repository Link

The complete project code, tests are on GitHub at:
https://github.com/ddrandy/scrabble_score_tdd/

6. Reference

NLTK Documentation: https://www.nltk.org/
Curses Documentation: https://docs.python.org/3/library/curses.html
Python unittest Documentation: https://docs.python.org/3/library/unittest.html