CHARLES DARWIN UNIVERSITY AUSTRALIA

PRT582 SOFTWARE ENGINEERING

PROCESS AND TOOLS

ASSIGNMENT 2

SOFTWARE UNIT TESTING REPORT

Submitted by

Quoc Chien Kieu: s374001

CHARLES DARWIN UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

September 2024

Table of Contents

1. Introduction

1.1. Objectives

The goal of this project is to create a Python Scrabble Game software. The traditional Scrabble letter values are used by the computer to calculate the score for a particular word. A timed word entry challenge, consistent handling of uppercase and lowercase letters, user input validation to ensure only acceptable words are accepted, and ongoing gaming support are just a few of the requirements that the application must achieve.

The purpose of this project is to follow best practices in software development to construct a robust, user-friendly program that complies with these standards.

1.2. Justification for python and Unit Testing tool

Python's readability, large standard library, and ease of use make it the perfect programming language for this project's rapid development and testing. The automated unit testing tool is the unittest module, which is an integrated Python unit testing framework.

The reason this tool was chosen is that it works well with Python, facilitates test automation, and offers an organized method of testing using fixtures, test cases, and test suites. By methodically testing each functionality with unittest, it is possible to make sure that all requirements are appropriately implemented and upheld throughout the development process.

1.3. Overview of Requirements

The key requirements for the Scrabble Game program are as follows:

1. The numbers are added up correctly for a given word
2. Upper- and lower-case letters should have the same value
3. Your program should prompt user with the right feedback if user does not enter an alphabet.
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.
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.
6. The game will keep going:
   - Until the player quits the game and display the total score of the player.
   - After 10 rounds and compute the total score of the player.total score being displayed at the end.

2. Process

2.1. Applying Test-Driven Development (TDD)

2.1.1. Initial Setup

The project was initiated by setting up the development environment. Two primary Python files were created:

- **scrabble_game.py**: This file contains the main program logic.
- **scrabble_game_unittest.py**: This file contains the test cases written using the unittest module.

The project adheres to the TDD methodology, which calls for creating tests for every requirement before putting the relevant code into action. With this method, you can be sure that every line of code is driven by clearly specified requirements and that every feature is extensively tested before it is deemed finished.

2.1.2. Implementing Requirements with TDD

Requirement 1: Correct score calculation

- **Test:** A test case was written to verify that the program calculates the Scrabble Game correctly for a word. The expected score for that word derived from the sum of individual letter values.
- **Code:** The calculate_score function was implemented. This function iterates through each letter in the input word, converts it to uppercase, and sums the corresponding values from the Scrabble letter values dictionary.

Requirement 2: Case insensitivity

- **Test:** A test was developed to ensure that the program returns the same score for words entered in different cases, such as "CABBAGE" and "cabbage".
- **Code:** The input word is converted to uppercase using the upper() method before score calculation. This ensures that the score is independent of whether the user enters the word in uppercase or lowercase.

```python
# Calculate Scrabble Game
def calculate_score(word):
    """
    Calculate the Scrabble Game of a given word based on letter values.
    """
    letter_values = {
        'A': 1, 'E': 1, 'I': 1, 'O': 1, 'U': 1,
        'L': 1, 'N': 1, 'R': 1, 'S': 1, 'T': 1,
        'D': 2, 'G': 2,
        'B': 3, 'C': 3, 'M': 3, 'P': 3,
        'F': 4, 'H': 4, 'V': 4, 'W': 4, 'Y': 4,
        'K': 5,
        'J': 8, 'X': 8,
        'Q': 10, 'Z': 10
    }
    return sum(letter_values.get(letter.upper(), 0) for letter in word)
```

Requirement 3: Input validation

- **Test:** To make sure the software recognizes and processes erroneous input such as digits, special characters, or empty strings correctly, tests were built. If invalid input is discovered, the application ought to prompt the user to supply a legitimate word.
- **Code:** The program includes an input validation function that checks whether the entered word consists only of alphabetic characters. If the validation fails, the user is prompted to re-enter the word.

```
def check_word(self):
        """
        Check the user's input word for validity
        and update the score accordingly.
        """
        if not self.timer_running:
            return
        user_input = self.entry.get().strip()
        if not user_input.isalpha():
            self.show_warning("Please enter only alphabetic characters.")
            return
```

Requirement 4: Timed word entry

- **Test:** To replicate the word entry procedure in a given amount of time (15 seconds), a test was constructed. The user must be prompted by the computer to input a word with a length that is randomly generated, and the score will be modified according to how quickly the user enters the right word length.
- **Code:** The program implements a countdown timer using the time module in Python. The time it takes to enter the right word affects how the score is calculated. A time-based bonus is added to the score if the word is typed within the allotted period.

```
def check_word(self):
        """
        Check the user's input word for validity
        and update the score accordingly.
        """
        if len(user_input) != self.required_length:
            self.show_warning(
              f"Word must be exactly {self.required_length} letters long.")
            return
```

```
    # Calculate the bonus based on time
    def calculate_time_bonus(elapsed_time):
        """
        Calculate the bonus score based on the time taken to input the
word.
        """
        if elapsed_time <= 5:
            return 20
        if elapsed_time <= 10:
            return 10
        if elapsed_time <= 15:
            return 5
        return 0
```

Requirement 5: Dictionary validation

- **Test:** A test was developed to ensure that only valid words are scored. The program must check whether the entered word exists in a predefined dictionary or use a library such as SpellChecker for validation.
- **Code:** The program uses a predefined list of valid words or a third-party library to check if the entered word is valid. If the word is not found in the dictionary, the user is prompted to enter a valid word.

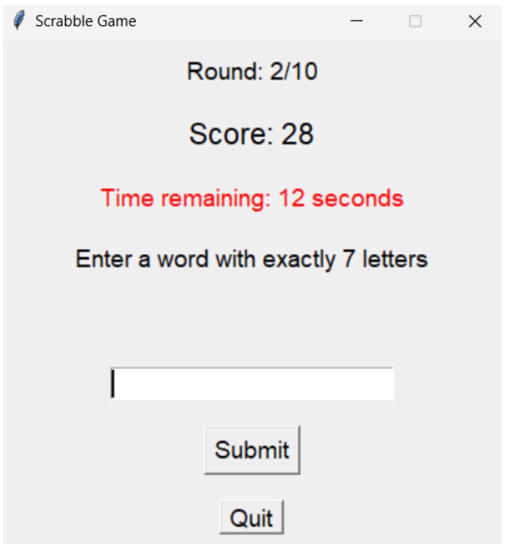
```
def check_word(self):
        """
        Check the user's input word for validity
        and update the score accordingly.
        """
        if not self.timer_running:
            return
        user_input = self.entry.get().strip()
        if not user_input.isalpha():
            self.show_warning("Please enter only alphabetic characters.")
            return
        if len(user_input) != self.required_length:
            self.show_warning(
                f"Word must be exactly {self.required_length} letters 
long.")
            return
        if not is_valid_word(user_input):
            self.show_warning("The word is not in the dictionary.")
            return
```

Requirement 6: Continuous gameplay

- **Test:** Tests were written to ensure the game can continue for multiple rounds, allowing the user to play up to 10 rounds or until they choose to quit. The total score must be correctly calculated and displayed at the end.
- **Code:** A loop was implemented to manage the game flow, keeping track of the total score across rounds and allowing the user to continue playing or quit at any time.

```
def start_round(self):
        """
        Start a new round of the Scrabble game.
        """
        if self.current_round < self.max_rounds:
            self.current_round += 1
            self.round_label.config(
                text=f"Round: {self.current_round}/{self.max_rounds}")
            self.required_length = random.randint(3, 7)
            self.entry.delete(0, tk.END)
            self.required_length_label.config(
                text=(
                    f"Enter a word with exactly {self.required_length} "
                    "letters"
                )
            )
            # Clear any previous warnings
            self.warning_label.config(text="")
            # Start the timer
            self.start_timer()
        else:
            # Show final screen after the last round
            self.show_end_game_screen()
```

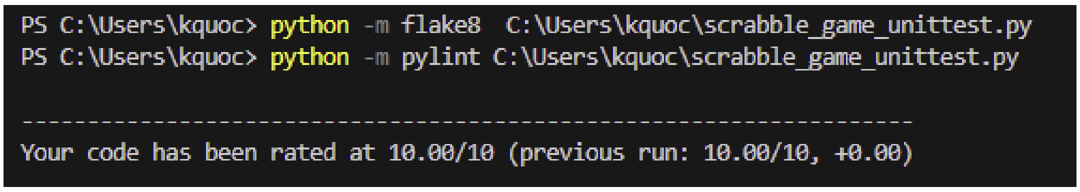
```
def quit_game(self):
        """
        Show the final score and quit the game after 2 seconds.
        """
        # Hide existing widgets
        for widget in self.root.winfo_children():
            widget.pack_forget()
        # Show final score
        final_score_label = tk.Label(
            self.root,
            text=f"Total Score: {self.total_score}",
            font=("Arial", 16))
        final_score_label.pack(pady=20)
        # Wait 2 seconds and then quit the game
        self.root.after(2000, self.root.quit)
```


Scrabble Game
Round: 2/10
Score: 28
Time remaining: 12 seconds
Enter a word with exactly 7 letters
Submit
Quit

The quit button was created to help exit the game.

2.2. Running tests and addressing issues

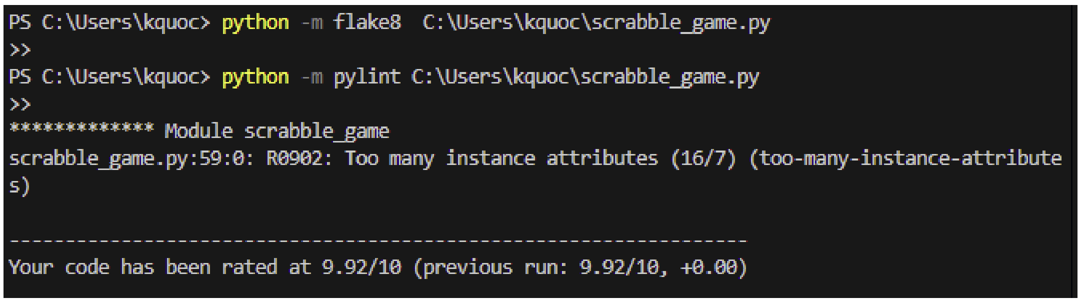
The unittest framework was used to thoroughly test every feature that was implemented. Multiple test cases were used to validate each requirement and make sure the application operates as intended in a variety of scenarios. Flake8 and Pylint were also used to verify that the code complied with Python's coding standards. After resolving all concerns found, the code was clear, well-documented, and manageable.



```
PS C:\Users\kquoc> python -m flake8  C:\Users\kquoc\scrabble_game_unittest.py
PS C:\Users\kquoc> python -m pylint C:\Users\kquoc\scrabble_game_unittest.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Pylint and Flake8 status report for **scrabble_game_unittest.py**



```
PS C:\Users\kquoc> python -m flake8  C:\Users\kquoc\scrabble_game.py
>>
PS C:\Users\kquoc> python -m pylint C:\Users\kquoc\scrabble_game.py
>>
************* Module scrabble_game
scrabble_game.py:59:0: R0902: Too many instance attributes (16/7) (too-many-instance-attribute
s)

-----------------------------------------------------------------
Your code has been rated at 9.92/10 (previous run: 9.92/10, +0.00)
```

Pylint and Flake8 status report for **scrabble_game_unittest.py**

Requirements	Test Case	Input	Output	Results
1. Calculate score for a given word	test_calculate_score	'apple', 'orange'	Score: 9 for 'apple', Score: 7 for 'orange'	Pass

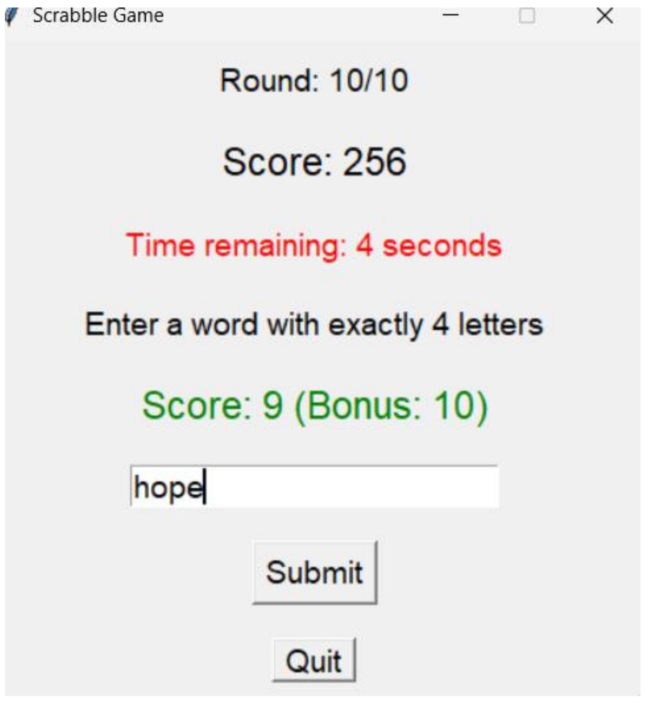
2. Handle case insensitivity	test_upper_lower_case	A', 'a', 'HELLO', 'hello', 'orange','oRANGe'	Same score for uppercase and lowercase inputs	Pass
3. Handle non-alphabetic input	test_non_alphabet_input_1	'123'	Warning: "Please enter only alphabetic characters."	Pass
	test_non_alphabet_input_2	'1a2b3'	Warning: "Please enter only alphabetic characters."	Pass
4. Validate input length	test_input_length_check_below	'hi'	Warning: "Word must be exactly 5 letters long."	Pass
	test_input_length_check_above	'orange'	Warning: "Word must be exactly 5 letters long."	Pass
5. Validate word against dictionary	test_valid_word_from_dictionary	'quoxyz'	Warning: "The word is not in the dictionary."	Pass
6. Timer-based scoring bonus	test_timer_and_score_bonus_5	'cabbage', "Time remaining: 4 seconds"	Score: 14, Bonus: 5	Pass
	test_timer_and_score_bonus_10	'cabbage', "Time remaining: 9 seconds"	Score: 14, Bonus: 10	Pass
	test_timer_and_score_bonus_20	'cabbage', "Time remaining: 16 seconds"	Score: 14, Bonus: 20	Pass

Detailed test case analysis for Scrabble Game Features

Testcases check manually the “Continuous gameplay” requirement.

Case 1: Show the final score of user after 10 rounds

- Step 1: Start game
- Step 2: Play game from round 1 to round 10



Scrabble Game

Round: 10/10

Score: 256

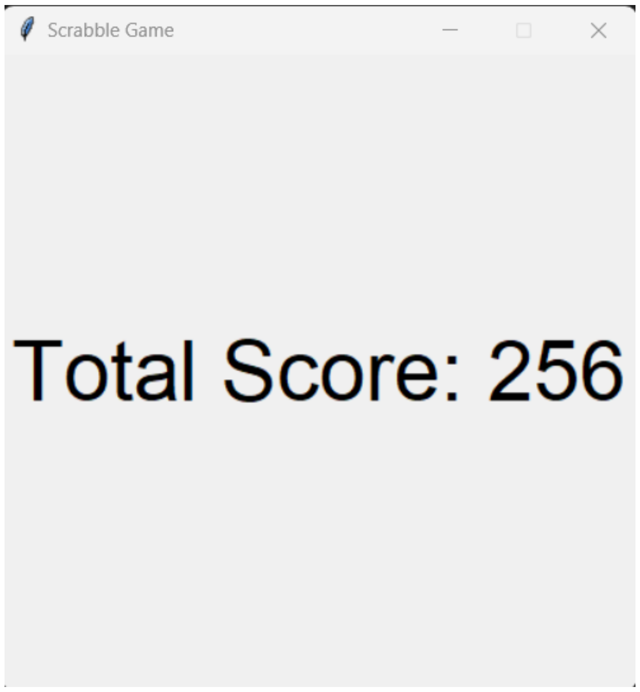
Time remaining: 4 seconds

Enter a word with exactly 4 letters

Score: 9 (Bonus: 10)

hope

- Step 3: Check the final score is shown



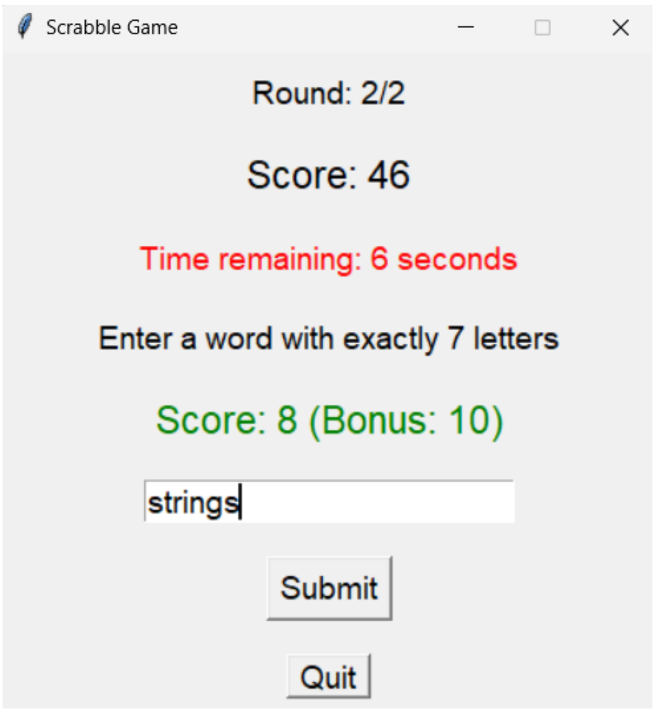
Scrabble Game

Total Score: 256

➔ **Result: Pass**

Case 2: Show the final score of user when Quit button is pressed

- Step 1: Start game
- Step 2: Play game 2 first rounds


Scrabble Game

Round: 2/2

Score: 46

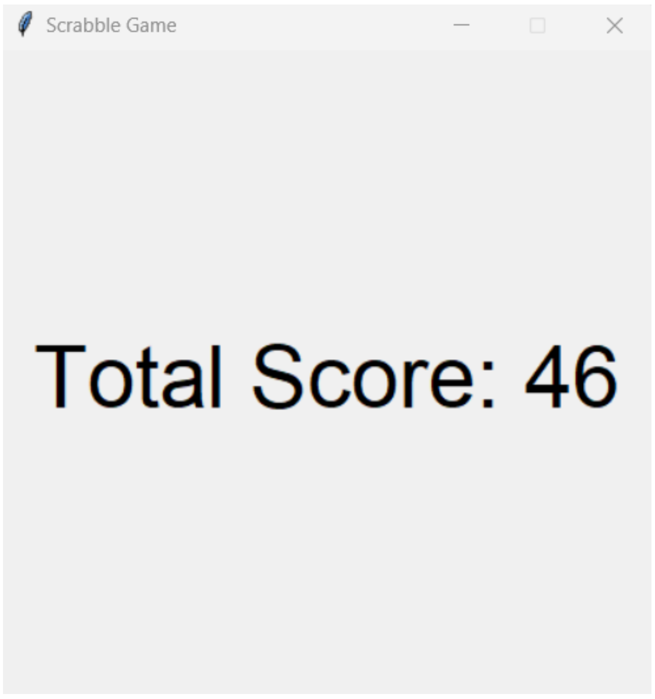
Time remaining: 6 seconds

Enter a word with exactly 7 letters

Score: 8 (Bonus: 10)

strings

- Step 3: Press Quit button
- Step 4: Check the total score is shown


Scrabble Game

Total Score: 46

➔ **Result: Pass**

3. Conclusion

3.1. Summary of achievements

All of the requirements accurate score calculation, case insensitivity, input validation, timed word entering, dictionary validation, and continuous gameplay are effectively met by the Scrabble Game application. By utilizing Test-Driven Development (TDD), a program's robustness and user-friendliness were enhanced by the meticulous planning, implementation, and testing of every feature. Python's use with the unittest framework made development go more smoothly and made it possible to test and validate each requirement quickly.

3.2. Lessons Learned

This project highlighted several key lessons:

- **Importance of TDD**: TDD played a critical role in identifying problems early on and guaranteeing that features fulfilled requirements. The goal of writing tests before coding was to stay focused on the intended results.
- **Effective input validation**: To handle a variety of user inputs and guarantee the robustness of the program, thorough input validation is necessary.
- **Time management in game**: A real-time timer was introduced, which increased complexity but enhanced the user experience. Learning how to balance responsiveness and accuracy in time-based features was a worthwhile experience.
- **Continuous integration**: Using tools like flake8 and pylint to fix linting errors and conduct tests on a regular basis helped maintain good code quality throughout the project.

3.3. Areas for Improvement

While the project was successful, there are areas where improvements could be made:

- **Enhanced input validation**: Improve checks for valid dictionary words and handle special characters more effectively.
- **Error handling**: Provide more specific and dynamic error messages for different input issues.
- **Timer accuracy**: Refine timer functionality for better real-time performance and clearer user feedback.
- **Scalability**: Optimize the game logic for multiple rounds and larger inputs.
- **UI/UX**: Improve the visual design and responsiveness of the user interface.
- **Testing coverage**: Expand automated tests to cover additional edge cases and performance scenarios.

3.4. Future Work

In the future, the project might need some more improvements, with an emphasis on enhancing functionality, boosting efficiency, and enhancing user experience.

- **Multiplayer Mode:** Adding multiplayer support could increase the game's replay value and competitiveness.
- **Adaptive Difficulty:** A new degree of difficulty could be added to the game by varying the word count or time limit according to the player's performance.

3.5. GitHub Repository

This report and the source code for the full project are available on GitHub.

GitHub Repository Link: Scrabble Game Project Repository