

PRT582

SOFTWARE ENGINEERING:

PROCESS AND TOOLS

Assignment 02

Howard Mai S371832
2 September 2024

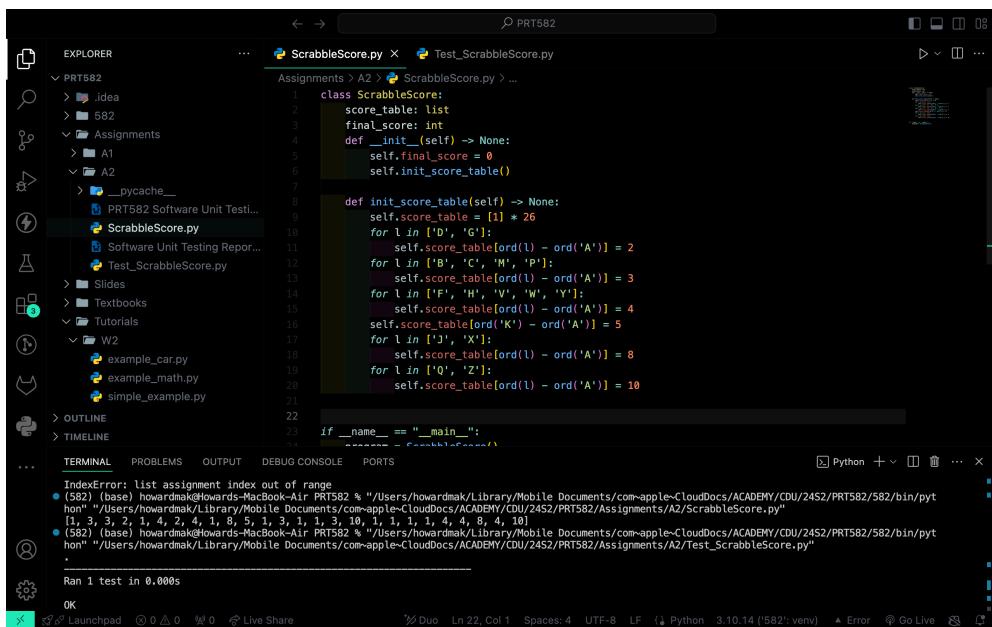
1. INTRODUCTION

To complete the given task and requirements, I use Python to design the game and the TDD method to write the program. Additionally, I also use `unittest` to test all the test cases. Finally, `Flake8` and `PyLint` are utilised to check whether the program is well organised.

2. PROCESS

2.1. Initialisation

To make sure the numbers can be added up correctly, it is important to initialise a score table corresponding to the score of every letter.



```
class ScrabbleScore:
    score_table: list
    final_score: int
    def __init__(self) -> None:
        self.final_score = 0
        self.init_score_table()

    def init_score_table(self) -> None:
        self.score_table = [1] * 26
        for l in ['D', 'G']:
            self.score_table[ord(l) - ord('A')] = 2
        for l in ['B', 'C', 'M', 'P']:
            self.score_table[ord(l) - ord('A')] = 3
        for l in ['F', 'H', 'V', 'W', 'Y']:
            self.score_table[ord(l) - ord('A')] = 4
        self.score_table[ord('K') - ord('A')] = 5
        for l in ['J', 'X']:
            self.score_table[ord(l) - ord('A')] = 8
        for l in ['Q', 'Z']:
            self.score_table[ord(l) - ord('A')] = 10

if __name__ == "__main__":
    pass
```

Running through the test case, it can be seen that the program's score table matches the given condition:

```

1 import unittest
2 from ScrabbleScore import ScrabbleScore
3
4 class ScrabbleScoreTestCase(unittest.TestCase):
5     program = ScrabbleScore()
6
7     def test_initialisation(self):
8         self.assertEqual(
9             [1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10],
10            self.program.score_table
11        )
12
13    if __name__ == "__main__":
14        unittest.main()
15
16

```

IndexError: list assignment index out of range
• (base) howardmak@Howards-MacBook-Air PRT582 % "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/582/bin/python" "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/Assignments/A2/ScrabbleScore.py"
[1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10]
• (582) (base) howardmak@Howards-MacBook-Air PRT582 % "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/582/bin/python" "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/Assignments/A2/Test_ScrabbleScore.py"
.
Ran 1 test in 0.000s
OK

2.2. Adding Validation

In this step, I wrote a function called `add_up` to add up scores of every letter.

```

1 class ScrabbleScore:
2     def __init__(self):
3         self.score_table = {}
4         for l in "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z":
5             self.score_table[ord(l) - ord('A')] = 1
6             self.score_table[ord('K') - ord('A')] = 5
7             for l in ['J', 'X']:
8                 self.score_table[ord(l) - ord('A')] = 8
9             for l in ['Q', 'Z']:
10                self.score_table[ord(l) - ord('A')] = 10
11
12    def add_up(self, s: str):
13        for l in s:
14            if "A" <= l <= "Z":
15                self.final_score += self.score_table[ord(l) - ord('A')]
16            elif " ":
17                parameter self: Self@ScrabbleScore
18        return self.final_score
19
20

```

Ran 2 tests in 0.000s
OK
• (582) (base) howardmak@Howards-MacBook-Air PRT582 % "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/582/bin/python" "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/Assignments/A2/test_ScrabbleScore.py"
.
Ran 1 test in 0.000s
OK
• (582) (base) howardmak@Howards-MacBook-Air PRT582 %

In this test case, the input variable is “cabbage”, and the returned score should be 14. It can be that the second test is passed:

```

import unittest
from ScrabbleScore import ScrabbleScore

class ScrabbleScoreTestCase(unittest.TestCase):
    program = ScrabbleScore()

    def test_initialisation(self):
        self.assertEqual(
            [1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10],
            self.program.score_table
        )

    def test_add_up(self):
        self.assertEqual(
            14, self.program.add_up("cabbage")
        )

if __name__ == "__main__":
    unittest.main()

```

Ran 1 test in 0.000s
OK
Ran 2 tests in 0.000s
OK

2.3. Letter Score Validation

Given the condition that upper- and lower-case letters should have the same value, I wrote another function called `get_value` to complete this and replace some codes in the function `add_up` with this function:

```

class ScrabbleScore:
    def __init__(self):
        self.score_table = {chr(i): chr(i) - ord('A') + 1
                           for i in range(65, 91)}
        for l in ['Q', 'Z']:
            self.score_table[ord(l)] = 10

    def add_up(self, s: str):
        for l in s:
            self.final_score += self.get_value(l)
        return self.final_score

    def get_value(self, l: chr):
        if "A" <= l <= "Z":
            return self.score_table[ord(l) - ord('A')]
        elif "a" <= l <= "z":
            return self.score_table[ord(l) - ord('a')]
        else:
            return 0

```

Ran 1 test in 0.000s
OK
Ran 2 tests in 0.000s
OK

In this test case, the inputs are “A”, “D”, “H”, “K”, “Q”, and they should have the same value with their corresponding lower-case letters. It can be observed that the third test can be also passed:

```

class ScrabbleScoreTestCase(unittest.TestCase):
    def test_letter_values(self):
        self.assertEqual(
            self.program.get_value("A"),
            self.program.get_value("a")
        )
        self.assertEqual(
            self.program.get_value("D"),
            self.program.get_value("d")
        )
        self.assertEqual(
            self.program.get_value("H"),
            self.program.get_value("h")
        )
        self.assertEqual(
            self.program.get_value("K"),
            self.program.get_value("k")
        )
        self.assertEqual(
            self.program.get_value("Q"),
            self.program.get_value("q")
        )

```

Ran 3 tests in 0.000s

FAILED (failures=1)

(582) (base) howardmai@Howards-MacBook-Air PRT582 % "/Users/howardmai/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/582/bin/python" "/Users/howardmai/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/Assignments/A2/Test_ScrabbleScore.py"

Ran 3 tests in 0.000s

OK

2.4. Input Validation

In this stage, a function called `validate_input` is created to check whether the user has input any alphabets, which will raise a `RuntimeError` when there are no alphabets or return the number of letters.

```

class ScrabbleScore:
    def get_value(self, l: chr) -> int:
        if "A" <= l <= "Z":
            return self.score_table[ord(l) - ord('A')]
        elif "a" <= l <= "z":
            return self.score_table[ord(l) - ord('a')]
        else:
            return 0

    def validate_input(self, s:str):
        letter_cnt = 0
        for l in s:
            if ("A" <= l <= "Z") or ("a" <= l <= "z"):
                letter_cnt += 1
        if letter_cnt == 0:
            raise RuntimeError("You did not enter any alphabet!")

        return letter_cnt

```

Ran 3 tests in 0.000s

FAILED (failures=1)

(582) (base) howardmai@Howards-MacBook-Air PRT582 % "/Users/howardmai/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/582/bin/python" "/Users/howardmai/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/24S2/PRT582/Assignments/A2/Test_ScrabbleScore.py"

Ran 3 tests in 0.000s

OK

In this test case, there are two inputs, which are null string and "abc". For the first input, the program should raise a `RuntimeError`, while it should return 3 for the second input. It can be seen that this test can be passed:

```

class ScrabbleScoreTestCase(unittest.TestCase):
    def test_letter_values(self):
        self.assertEqual("K", self.program.get_value("K"))
        self.assertEqual("Q", self.program.get_value("Q"))

    def test_input_validation(self):
        with self.assertRaises(Exception):
            self.program.validate_input("")

        self.assertEqual(3, self.program.validate_input("abc"))

if __name__ == "__main__":
    unittest.main()

```

Ran 4 tests in 0.000s

OK

2.5. Single Round

To implement the given condition, three functions are created, which are `countdown`, `get_user_input`, and `play_for_one_round`. Firstly, in the `countdown` function, `time` is imported to set the timer and count down. Secondly, in the `get_user_input`, users will be noticed to input a word, where the number of alphabets is randomly generated. If the word input by the user cannot be validated, it should return 0 scores. Lastly, in the `play_for_one_round`, threading is utilised to implement the timer and user input at the same time, and return the score the user get and the time used, making sure that the score will be higher if less time is used to enter the right length of word.

```

class ScrabbleScore:
    def countdown(self, seconds, stop_event):
        while seconds and not stop_event.is_set():
            # mins, secs = divmod(seconds, 60)
            # timer = f'{mins:02d}:{secs:02d}'
            # print(timer, end="\r")
            time.sleep(1)
            seconds -= 1

        if not stop_event.is_set():
            print("\nTime's up!")

    def get_user_input(self, stop_event):
        required_letter_cnt = random.randint(0, 10)
        user_input = input(f"Please input a word with {required_letter_cnt} words in 15 seconds: ")
        stop_event.set()
        if self.validate_input(user_input) != required_letter_cnt:
            print("You did not enter the word as required, you get 0 points!")
            return 0
        else:
            return self.add_up(user_input)

    def add_up(self, user_input):
        pass

```

Time's up!

You did not enter the word as required, 0 points!

You got: 0 scores, used: 12.376019954613965

```

class ScrabbleScore:
    def get_user_input(self, stop_event):
        print("You did not enter the word as required, you get 0 points!")
        return 0
    else:
        return self.add_up(user_input)

    def play_for_one_round(self):
        stop_event = threading.Event()
        timer_thread = threading.Thread(target=self.countdown, args=(15, stop_event))
        timer_thread.start()
        begin_time = time.time()
        scores = self.get_user_input(stop_event)
        used_time = time.time() - begin_time
        timer_thread.join()

        print(f"You got: {scores} scores, used: {used_time: 0.2}s")
        return [scores, min(15.0, used_time)]

```

TERMINAL

```

(base) howardmai@Howards-MacBook-Air PRT582 %
(base) howardmai@Howards-MacBook-Air PRT582 % pip install pyenchant
Collecting pyenchant
  Downloading pyenchant-3.2.2-py3-none-any.whl (55 kB)
  55.7/55.7 kB 636.8 kB/s eta 0:00:00
Installing collected packages: pyenchant
Successfully installed pyenchant-3.2.2

[notice] A new release of pip available: 22.3.1 > 24.2
[notice] To update, run: pip install --upgrade pip
(base) howardmai@Howards-MacBook-Air PRT582 %

```

2.6. Word Validation

To implement the function that checks whether the word is from a dictionary, a dictionary is initialised when the program is initialised and a function called `validate_word`, which will return true when the word is in the dictionary. In this test case, two inputs, which are “apple” and “boook” are used to process the test. It can be seen that the test case is passed.

```

class ScrabbleScoreTestCase(unittest.TestCase):
    def test_input_validation(self):
        self.assertEqual(
            3,
            self.program.validate_input("abc")
        )
    def test_word_validation(self):
        self.assertTrue(True, self.program.validate_word("apple"))
        self.assertFalse(False, self.program.validate_word("boook"))

if __name__ == "__main__":
    unittest.main()

```

TERMINAL

```

FAILED (failures=1)
(base) howardmai@Howards-MacBook-Air PRT582 % /Users/howardmai/anaconda3/bin/python "/Users/howardmai/Library/Mobile Documents/com~apple~CloudDocs/ACADEMY/LDU/245Z/PRT582/Assignments/A2/test_ScrabbleScore.py"
.....
Ran 5 tests in 0.000s
OK
(base) howardmai@Howards-MacBook-Air PRT582 %

```

2.7. Keep the Game Going

To meet the last requirement, a function `play` is created, where there will be 10 rounds. In each round, the user can choose to continue or exit. If any exception is raised,

the user should re-enter the option or re-enter the word. Finally, after all rounds, the game will count the final total score and used time.

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists a project folder 'PRT582' containing 'Assignments', 'Slides', 'Textbooks', 'Tutorials', and 'W2'. Inside 'W2', there are files like 'example_car.py', 'example_math.py', 'simple_example.py', 'T2.1 Challenges in Agile Dev...', and 'T2.2 Linting.docx'. The 'TERMINAL' tab at the bottom shows command-line output for running tests:

```

FAILED (failures=1)
● (base) howardmak@Howards-MacBook-Air PRT582 % /Users/howardmak/anaconda3/bin/python "/Users/howardmak/Library/Mobile Documents/com-apple-CloudDocs/ACADEMY/CDU/2452/PRT582/Assignments/A2/Test_ScrabbleScore.py"
.....
Ran 5 tests in 0.000s
OK
○ (base) howardmak@Howards-MacBook-Air PRT582 %

```

2.8. Check the program with PyLint and Flake8

It can be observed that it passes the PyLint and Flake8 test.

The terminal window displays the results of running PyLint and Flake8 on the 'scrabble_score.py' file:

```

Your code has been rated at 9.89/10 (previous run: 9.89/10, +0.00)

● (base) howardmak@Howards-MacBook-Air PRT582 % pylint Assignments/A2/scrabble_score.py
-----
Your code has been rated at 10.00/10 (previous run: 9.89/10, +0.11)

● (base) howardmak@Howards-MacBook-Air PRT582 % flake8 Assignments/A2/scrabble_score.py
○ (base) howardmak@Howards-MacBook-Air PRT582 %

```

3. CONCLUSION

Through the lesson, I learnt that though TDD development, a complex program can be built and tested step by step, which is very convenient to discover issues in every components.

Github link: <https://github.com/HowardMak/PRT582.git>, the program file and test cases are in the folder `A2_unit_test`.