PRT582 Software Unit Testing Report

Mingxu Li S365503

Contents

1. Introduction

1.1 Game Objectives and Requirements

I created a "Guess the Number" game using Python. In this game, players are required to guess a randomly generated four-digit number. The game provides clues about the number, and it ends when the player correctly guesses the number or chooses to quit.

The basic requirements for the game are as follows:

Random Number Generation	The program should generate a random four-digit number, with each digit ranging from 0 to 9.
Player Guesses	The program should continuously prompt the player to guess the number until the player either correctly guesses the secret number or chooses to exit the game.
Accuracy Feedback	After each player input, the program should provide feedback in the form of "circles" and "x's": "Circles" indicate a correct number in the correct position. "X's" indicate a correct number in the wrong position.
Game Ending and Statistics	When the game ends, the program should display the number of attempts made by the player and ask if they want to exit the game or play again.
Free Exit	Players should have the option to exit the game at any time.

1.2 Automated Unit Testing Tool

Throughout the development and testing of this program, I employed Python's built-in testing framework, unittest. This automated unit testing tool allowed us to write and execute test cases to verify the correctness of the code. It proved to be a valuable tool for capturing and addressing potential issues, ensuring that the code functions as expected.

2. Process

2.1 Developing the ‘generate_random_number’ Function

In the first part of TDD testing, my focus was on developing the generate_random_number function. My goal was to ensure that this function could generate a random four-digit number that adheres to the game rules. I began by crafting a series of test cases that verified whether the randomly generated numbers had the correct length and if each digit fell within the range of 0 to 9. These test cases provided my development objectives and specifications, serving as benchmarks for evaluating code quality.

Subsequently, I incrementally refined the implementation of the generate_random_number function to ensure it met the requirements outlined by the test cases. This entailed logic for generating random numbers, checking their length, and ensuring the range of each digit. Through iterative testing, I continuously validated the correctness of the function until all test cases passed. This process helped us establish a reliable random number generator, a critical foundation for the game, and ensured the legality of the secret number.

main.py | TDD unittest 1 | TDD unittest 2 | TDD unittest 3

TDD unittest 1 > ...

```
import unittest
import random

# Import the function to be tested
from main import generate_random_number

class TestGenerateRandomNumber(unittest.TestCase):
    def test_length_of_generated_list(self):
        # Write a test case to check the length of the generated list
        result = generate_random_number()
        self.assertEqual(len(result), 4)  # Expect the generated list to have a length of 4

    def test_range_of_generated_numbers(self):
        # Write a test case to check the range of generated numbers
        result = generate_random_number()
        for num in result:
            self.assertTrue(0 <= num <= 9)  # Check if each number is within the range of 0 to 9

if __name__ == '__main__':
    unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python

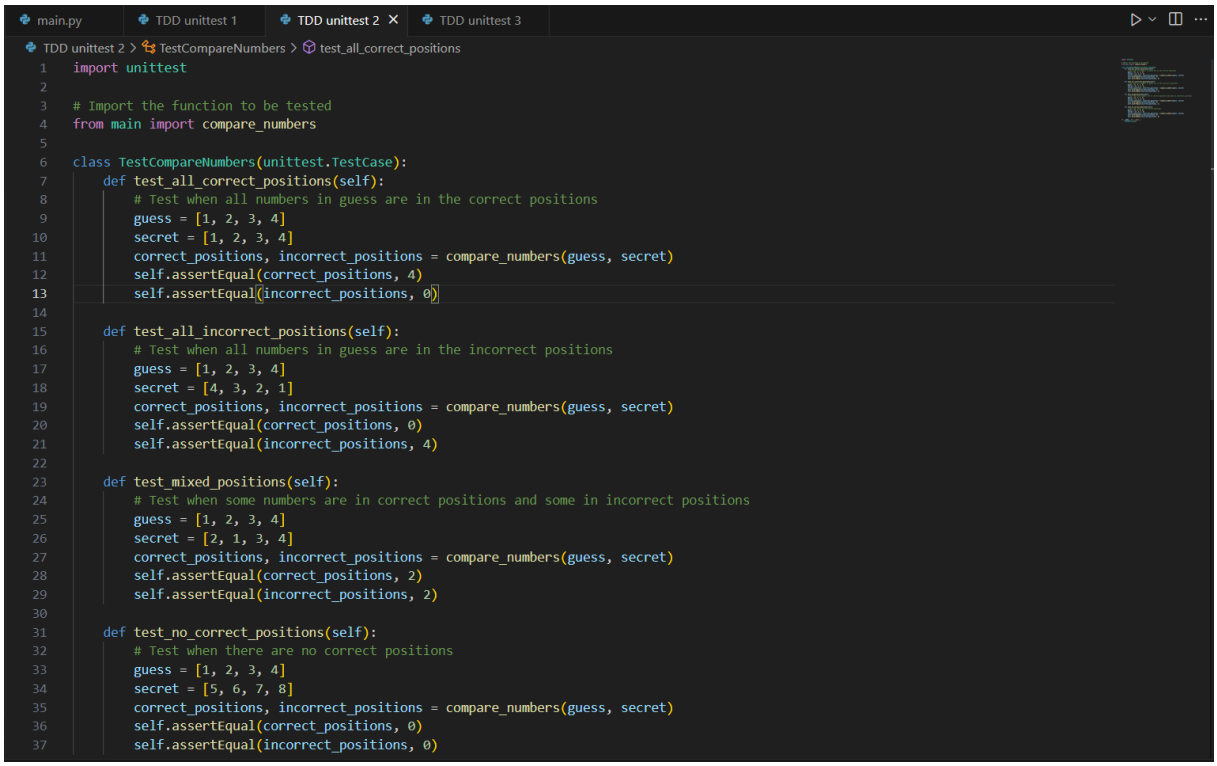
```
PS C:\Users\25458\Desktop\PY> & C:/Users/25458/AppData/Local/Programs/Python/Python38/python.exe "c:/Users/25458/Desktop/PY/TDD unittest 1"
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\25458\Desktop\PY>
```

2.2 Developing the ‘compare_numbers’ Function

In the second part of TDD testing, I focus shifted to the development of the ‘compare_numbers’ function. The goal of this function is to compare the player's guess with the generated secret number and then return the correct comparison results. To achieve this objective, I authored a series of detailed test cases that covered various scenarios, including all correct guesses, all incorrect guesses, and mixed cases. These test cases served as the specifications for the function's behavior and also as the basis for validating code correctness.

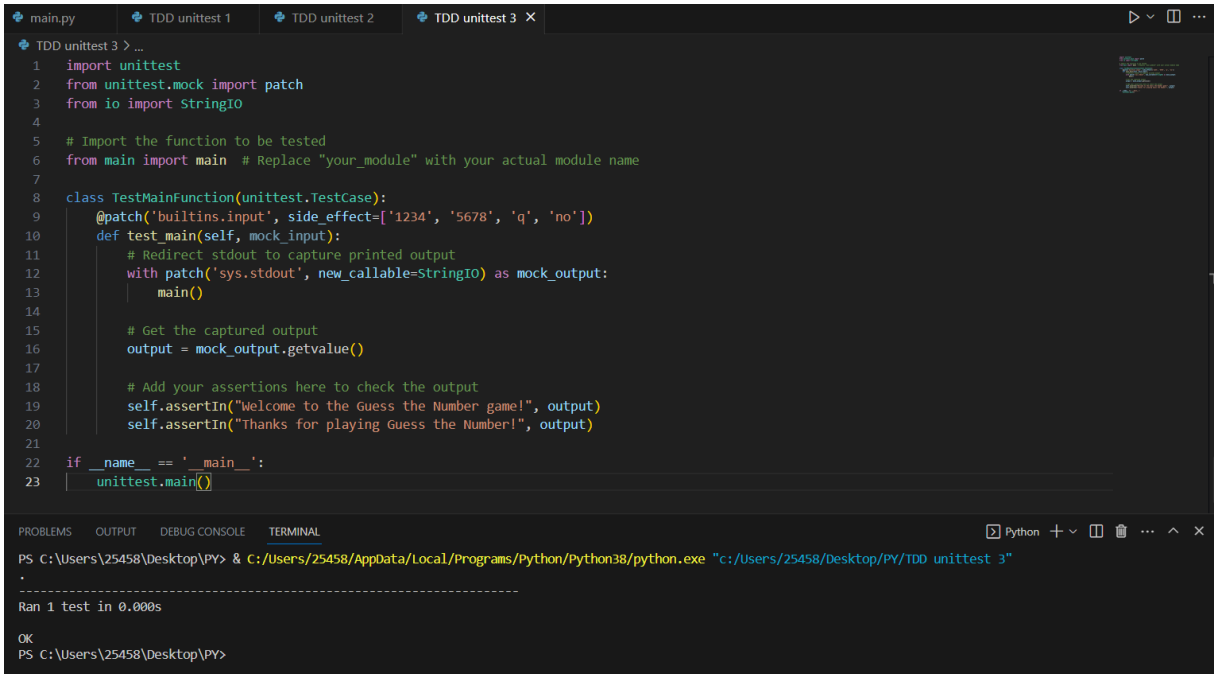
Next, I step-by-step implemented the logic of the compare_numbers function, ensuring that it correctly compared guesses and the secret number and returned accurate comparison results. This process included comparisons for each digit's position and checks for the presence of digits within the secret number. By repeatedly running the tests, I ensured that the function worked correctly according to the game rules. Testing this function validated the core logic of the game, as it assessed the accuracy of player guesses.



2.3 Developing the ‘main’ Function

In the third part of TDD testing, I focused on the main logic of the game, which resides in the main function. This function involves user input, output, and game interactions and serves as the core of the entire game. While not altering the structure of the function, I employed unittest.mock to simulate user input and capture output for testing purposes, thereby simulating game interactions.

I composed test cases to validate various aspects of the main function, including correct guesses, game exits, and input errors. These test cases ensured that the main function operated according to game rules in different user interaction scenarios and provided feedback about the game progress. Through this series of tests, I verified the overall functionality and user experience of the game. This section of testing ensured the correctness and stability of the game in terms of user interactions.



3. Conclusion

3.1 Lessons Learned

Significance of TDD: Adopting the TDD approach aided in the early detection of issues, reducing debugging time and ensuring the code worked as intended.

Value of Automated Unit Testing Tools: Using tools like unittest helped enhance code quality, reduce the effort required for manual testing, and provide rapid feedback.

Modular Design: Breaking code into small, testable modules makes code writing and maintenance more manageable and enhances code reusability.

Timely Feedback: Running test cases after each code modification allowed us to promptly identify and address issues, maintaining code stability.

3.2 Summary

In my TDD testing process, I methodically validated the "Guess the Number" game codebase in three phases. Firstly, I rigorously tested the 'generate_random_number' function, ensuring it reliably generated compliant random four-digit numbers. Next, I focused on the 'compare_numbers' function, evaluating its ability to accurately assess player guesses against the secret number. Lastly, I thoroughly examined the ‘main’ function, which handles user interactions. My testing encompassed various scenarios, including correct guesses, game exits, and input errors.

Collectively, this TDD approach ensured code accuracy, bolstered maintainability, and validated adherence to requirements, resulting in a robust game. This experience will help me better tackle future development challenges, ensuring that our code remains reliable and stable. I look forward to applying these methods to further enhance my programming skills.

3.3 GitHub link

2545866719/prt582 (github.com)