PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS
Assignment 2

Submitted by:

Student Name:	Student ID:
Shahil Jha	S368427

CONTENTS

Introduction 3
Programming Language and Unit Testing Tool 3
Process 3
1. Random Number Generation Function 4
2. Get Hint Function 7
3. Number comparision Function 10
4. Quit or Replay Option Function 14
5. Integretion Testing 17
Outcome of the Program 18
Conclusion 19
References 20

INTRODUCTION

This report provides a detailed explanation of the development of the number guessing program using Test Driven Development. The main intention of this program is to generate a score based on the number of times it takes the user to guess the randomly generated four-digit number. The program will then give some clues about the number. Once the player guesses the number correctly, the program will show the number of attempts taken.

The requirements of the program are:

- Randomly generate a four-digit number.
- The program will keep asking the user to guess the number until the player guesses. it correctly or has quit.
- When the number is entered, the program will respond with hints using 'circle' and 'x' to show how accurate the guess was.
    - A 'circle' indicates that one digit is correct and is in the right spot.
    - A 'x' indicates that one digit is correct but in the wrong spot.
- Once the game is finished,
    - The number of attempts taken will be displayed.
    - The player will be asked to quit or to play again.
- Players can quit the game anytime.

PROGRAMMING LANGUAGE AND UNIT TESTING TOOL

Python is a potent programming language that's simple to comprehend and easy on the eyes. It showcases many traits shared by numerous other languages and proves its practicality in real-life scenarios. Moreover, it is freely available, comes with a standardized version, and enjoys the support of a sizable and affable community of passionate developers.

PyUnit (unittest): The foundation of the unittest unit testing framework drew its initial inspiration from JUnit and carried a comparable essence to prominent testing frameworks in various programming languages. It enables automated testing, sharing setup and teardown code for tests, grouping tests into collections, and ensuring that tests remain unaffected by the reporting system used (Python, 2023). The fact that this is a built-in module of python makes it more dependable.

PROCESS

Test-driven development (TDD) represents an agile approach to software development. Like other agile methodologies, TDD constructs software in incremental steps, commencing with creating an automated test, succeeded by a small segment of code tailored to fulfill that test's requirements. Subsequently, the code is refined to enhance its performance (Anwer et al., 2017). TDD starkly contrasts conventional software development practices that follow a sequence of design, coding, and testing. Instead, TDD inverts this sequence, initiating testing before coding even begins. The primary aim of TDD is to curtail the occurrence of defects and elevate the overall quality of code.

The phases of the number guessing program are as follows:

1. Test Cases and Initial Failing Test
2. Development to Pass the Test
3. Passing the Test

Unlike in other development approaches, in TDD the tests for any unit function are written first before the development of the business logic itself. Similarly, the test cases for the functionality of the number guessing game were written first and was expected to fail first.

1. RANDOM NUMBER GENERATION FUNCTION

The random number generation function is responsible for generating the random four-digit number that the player must guess.

```python
import unittest
from unittest.mock import patch
from guess_number_game import guess_number_game

class TestGuessNumberGame(unittest.TestCase):
    def setUp(self):
        self.game = guess_number_game(1234)

    def test_generate_random_number_randomness(self):
        random_number1 = self.game.generate_random_number()
        random_number2 = self.game.generate_random_number()
        random_number3 = self.game.generate_random_number()
        random_number4 = self.game.generate_random_number()
        self.assertNotEqual(random_number1, random_number2)
        self.assertNotEqual(random_number1, random_number3)
        self.assertNotEqual(random_number1, random_number4)

    def test_generate_random_number_data_type(self):
        random_number = self.game.generate_random_number()
        result = isinstance(random_number, int)
        self.assertTrue(result)
        result = isinstance(random_number, float)
        self.assertFalse(result)
        result = isinstance(random_number, str)
        self.assertFalse(result)
```

FIGURE 1: TEST CASES FOR RANDOM NUMBER GENERATION

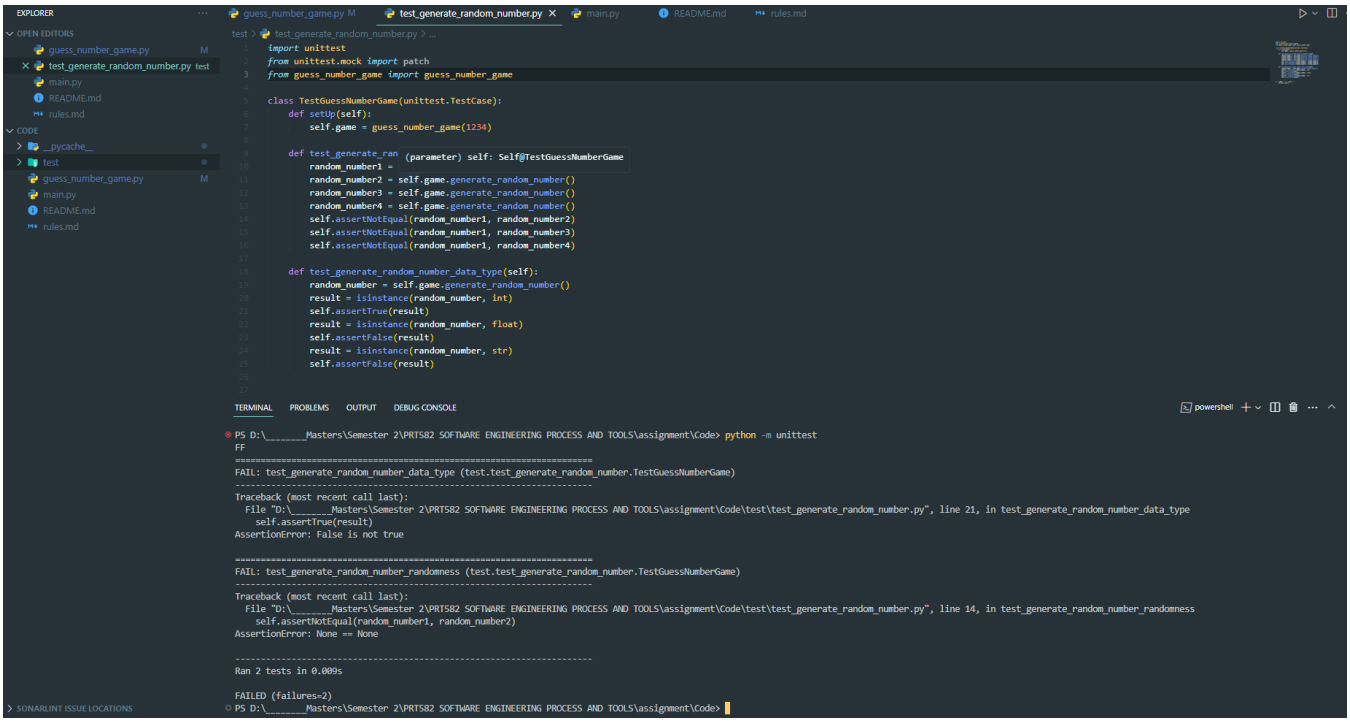

FIGURE 2: FAILED TEST FOR RANDOM NUMBER GENERATION

```python
# Generate a random four-digit number
def generate_random_number(self):
    digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    # Shuffle the digits
    random.shuffle(digits)
    # Take the first 4 digits
    generated_value = int(''.join(map(str, digits[:4])))
    return generated_value
```

FIGURE 3: FUNCTION DEVELOPMENT OF RANDOM NUMBER GENERATION

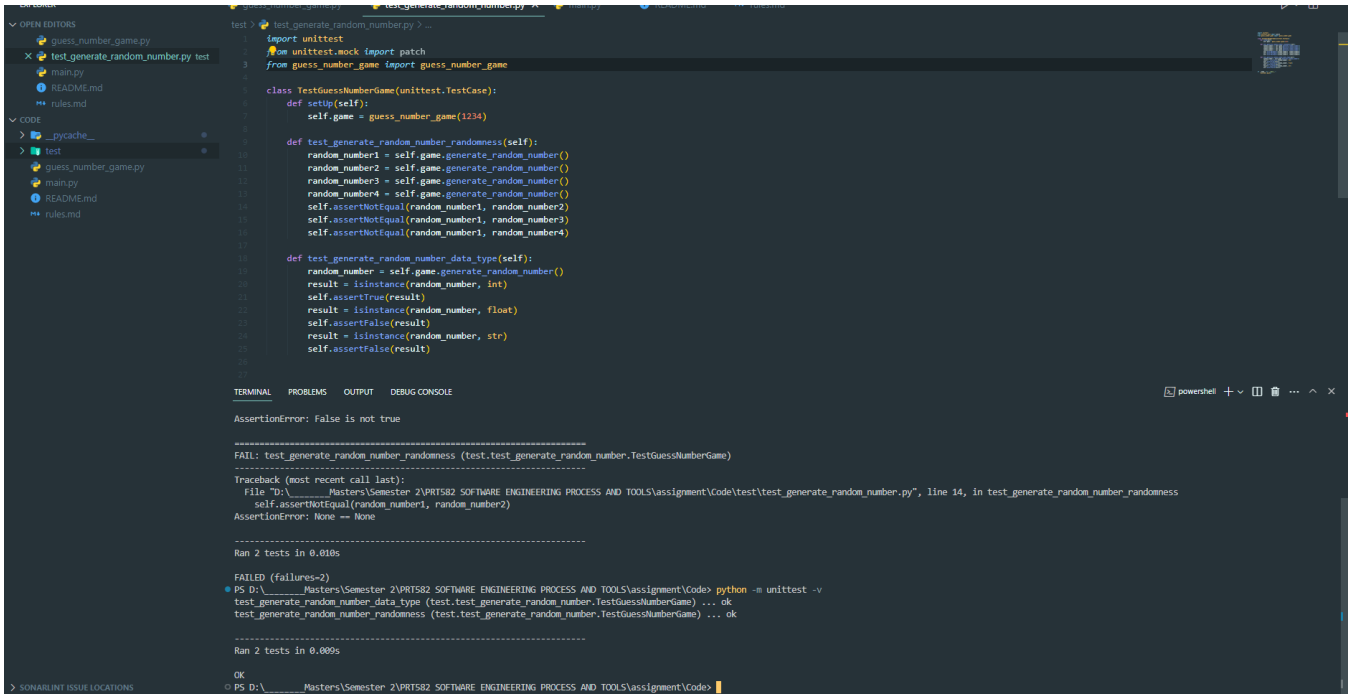

FIGURE 4: PASSED TEST FOR RANDOM NUMBER GENERATION

2. GET HINT FUNCTION

This function is responsible for hint generation for the input number using circles and crosses.

```python
import unittest
from unittest.mock import patch
from guess_number_game import guess_number_game

class TestGuessNumberGame(unittest.TestCase):
    def setUp(self):
        self.game = guess_number_game(1234)

    def test_get_hint_different_numbers(self):
        str_num1 = "1234"
        str_num2 = "9876"
        result = self.game.get_hint(str_num1, str_num2)
        compare_data = {
            'O': "",
            'X': ""
        }
        self.assertEqual(result, compare_data)

    def test_get_hint_different_places(self):
        str_num1 = "1234"
        str_num2 = "4871"
        result = self.game.get_hint(str_num1, str_num2)
        compare_data = {
            'O': "",
            'X': "XX"
        }
        self.assertEqual(result, compare_data)

    def test_get_hint_some_correct(self):
        str_num1 = "1234"
        str_num2 = "1874"
        result = self.game.get_hint(str_num1, str_num2)
        compare_data = {
            'O': "OO",
            'X': ""
        }
        self.assertEqual(result, compare_data)

    def test_get_hint_some_correct_some_different(self):
        str_num1 = "1234"
        str_num2 = "1324"
        result = self.game.get_hint(str_num1, str_num2)
        compare_data = {
            'O': "OO",
            'X': "XX"
        }
        self.assertEqual(result, compare_data)
```

FIGURE 5: TEST CASES FOR GET HINT FUNCTION

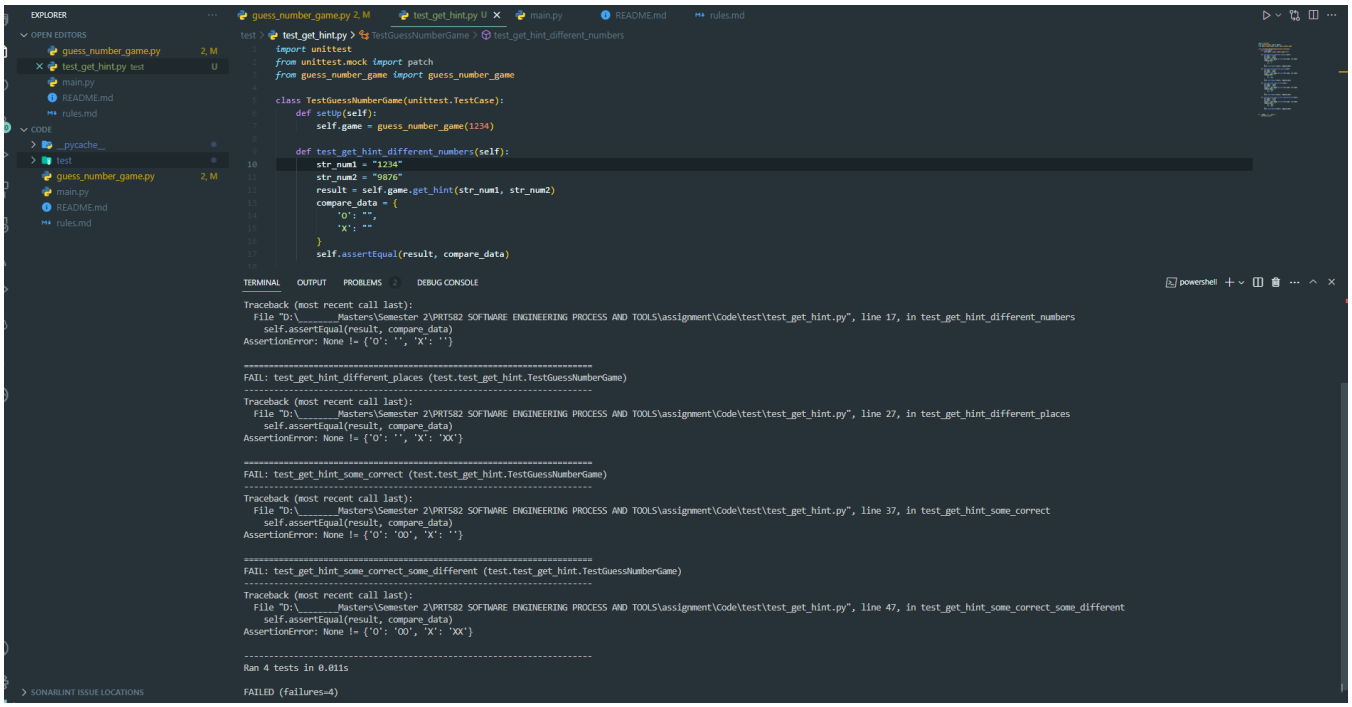

FIGURE 6: FAILED TESTS FOR GET HINT FUNCTION

```python
# method to compare numbers and generate hint
def get_hint(self, str_num1, str_num2):
    crosses = ""
    circles = ""
    # iterate the string casted numver generate hints
    for i in range(4):
        if str_num1[i] == str_num2[i]:
            circles += "O"
        elif str_num1[i] in str_num2:
            crosses += "X"
    data = {
        'O': circles,
        'X': crosses
    }
    return data
```

FIGURE 7: FUNCTION DEVELOPMENT OF GET HINT FUNCTION

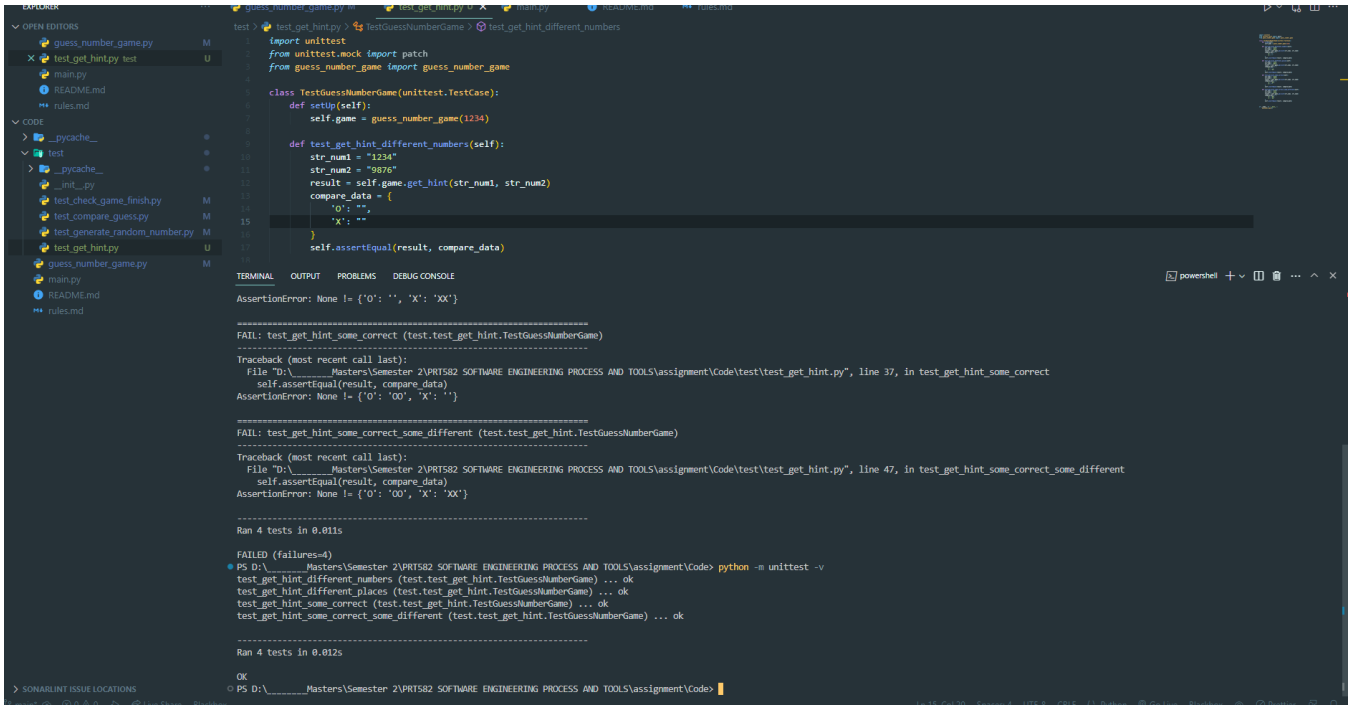

FIGURE 8: PASSES TESTS FOR GET HINT FUNCTION

3. NUMBER COMPARISION FUNCTION

This function is responsible for comparing the randomly generated four-digit number and the input number by the user.

```python
import unittest
from unittest.mock import patch
from guess_number_game import guess_number_game

class TestGuessNumberGame(unittest.TestCase):
    def setUp(self):
        self.game = guess_number_game(1234)

    def test_compare_guess_correct(self):
        self.assertTrue(self.game.compare_guess(1234))

    def test_compare_guess_incorrect(self):
        self.assertFalse(self.game.compare_guess(5678))

    def test_compare_guess_input_exception(self):
        with self.assertRaises(ValueError) as exception_context:
            self.game.compare_guess(guess_number="1234")
        self.assertEqual(
            str(exception_context.exception),
            "Both inputs must be integers"
        )

    def test_compare_guess_generated_number_exception(self):
        self.game.generated_number = "1234"
        with self.assertRaises(ValueError) as exception_context:
            self.game.compare_guess(guess_number=1234)
        self.assertEqual(
            str(exception_context.exception),
            "Both inputs must be integers"
        )

    def test_compare_guess_input_value_range_exception(self):
        with self.assertRaises(ValueError) as exception_context:
            self.game.compare_guess(guess_number=12345)
        self.assertEqual(
            str(exception_context.exception),
            "Both numbers must be 4-digit numbers"
        )

    def test_compare_guess_generated_number_value_range_exception(self):
        self.game.generated_number = 12345
        with self.assertRaises(ValueError) as exception_context:
            self.game.compare_guess(guess_number=1234)
        self.assertEqual(
            str(exception_context.exception),
            "Both numbers must be 4-digit numbers"
        )
```

FIGURE 9: TEST CASES FOR NUMBER COMPARISION FUNCTION

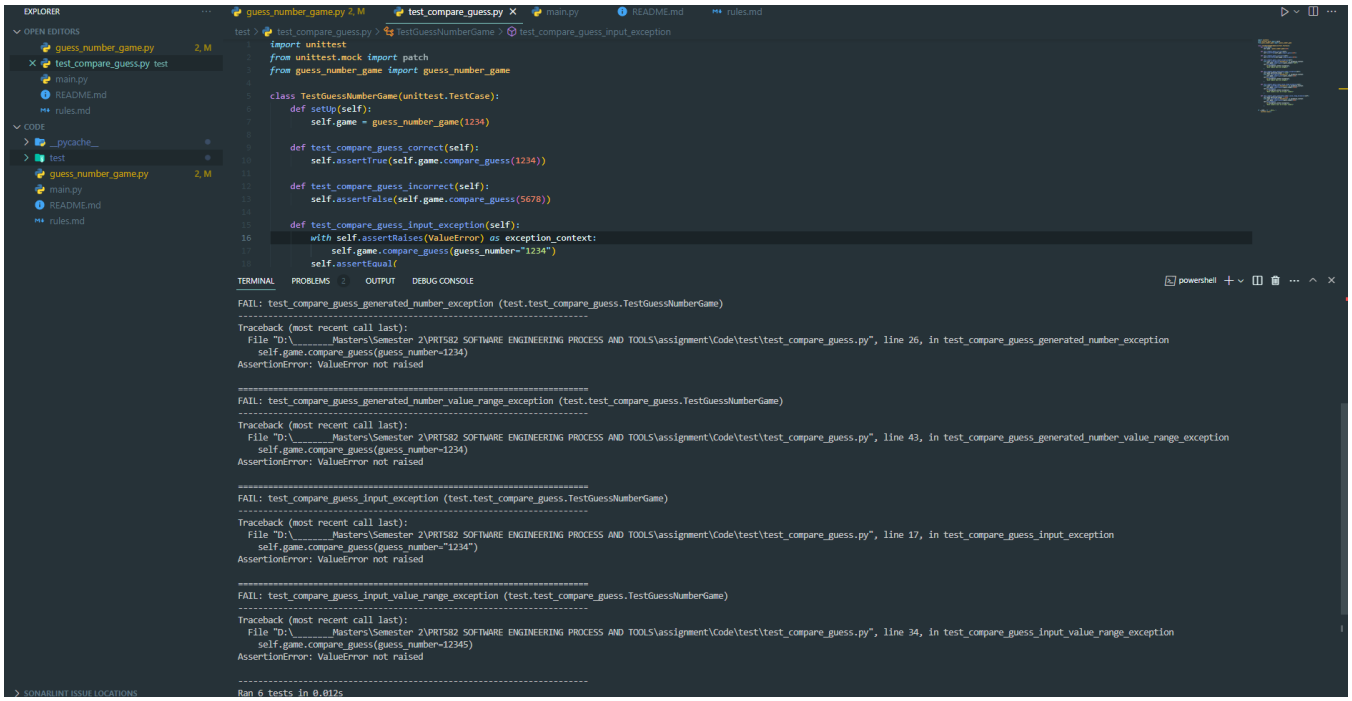

FIGURE 10: FAILED TEST NUMBER COMPARISION FUNCTION

```python
   # Compare the player's guess with the generated number
    def compare_guess(self, guess_number):
        # Check input validity
        if not (
            isinstance(self.generated_number, int) and isinstance(guess_number, int)
        ):
            raise ValueError("Both inputs must be integers")

         # Convert numbers to strings for comparison
        temp_str_number1 = str(self.generated_number) if self.generated_number != 0 else
"0000"
        temp_str_number2 = str(guess_number) if guess_number != 0 else "0000"

        # resolve length issue if the front of any number is 0
        str_number1 = temp_str_number1 if len(temp_str_number1)==4 else ('0' +
temp_str_number1)
        str_number2 = temp_str_number2 if len(temp_str_number2)==4 else ('0' +
temp_str_number2)

        # check the numbers are in range and are of 4 digits
        if not (len(str_number1) == 4 and len(str_number2) == 4):
            raise ValueError("Both numbers must be 4-digit numbers")


        # Increment the number of attempts
        self.attempts += 1
        if str_number1 == str_number2:
            return True

        # get hint
        hint = self.get_hint(str_number1, str_number2)

        # add the hint to table
        self.table.add_row(temp_str_number2, hint['X'], hint['O'])
        return False
```

FIGURE 11: FUNCTION DEVELOPMENT OF NUMBER COMPARISION FUNCTION

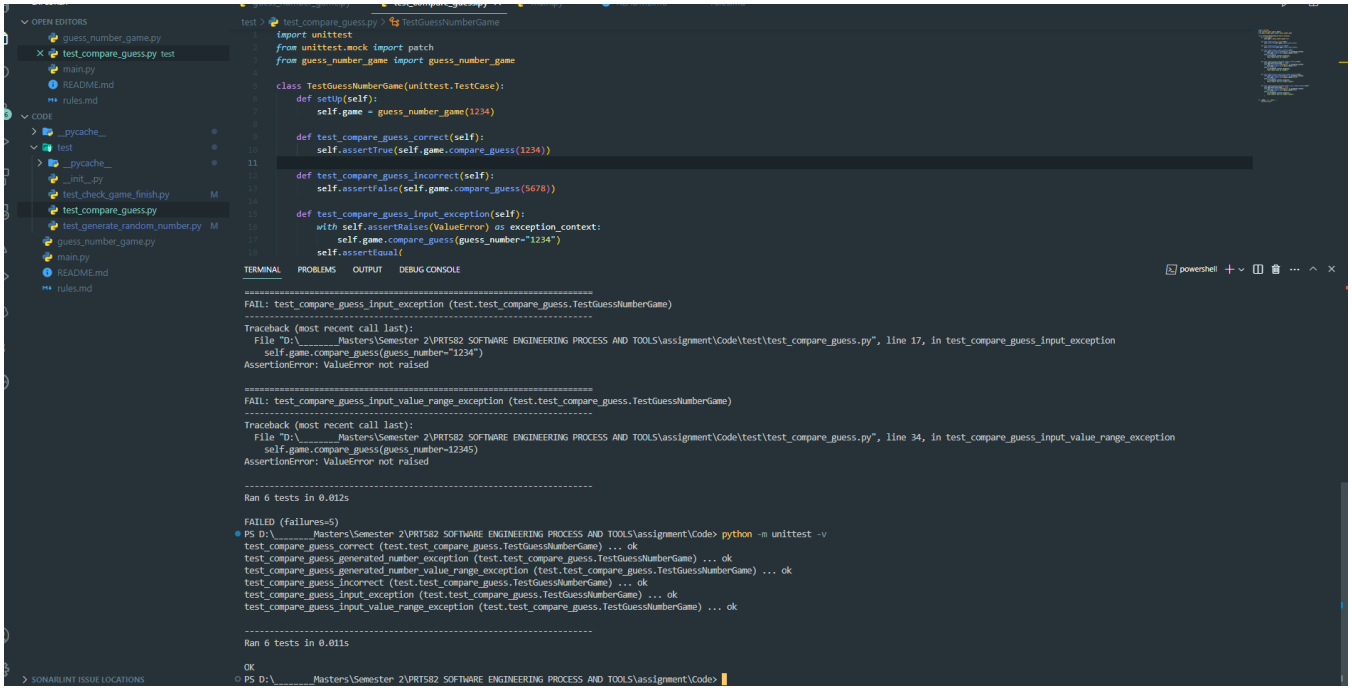

FIGURE 12: PASSED TEST NUMBER COMPARISION FUNCTION

4. QUIT OR REPLAY OPTION FUNCTION

```python
import unittest
from unittest.mock import patch
from guess_number_game import guess_number_game

class TestGuessNumberGame(unittest.TestCase):
    def setUp(self):
        self.game = guess_number_game(1234)

    def test_check_game_finish_input_option_quit(self):
        self.assertFalse(self.game.check_game_finish_input_option("q"))


    def test_check_game_finish_input_option_replay(self):
        self.assertTrue(self.game.check_game_finish_input_option("r"))


    def test_check_game_finish_input_option_quit_case_insensitive(self):
        self.assertFalse(self.game.check_game_finish_input_option("Q"))


    def test_check_game_finish_input_option_replay_case_insensitive(self):
        self.assertTrue(self.game.check_game_finish_input_option("R"))


    def test_check_game_finish_input_option_invalid_then_quit(self):
        with self.assertRaises(ValueError) as exception_context:
            self.game.check_game_finish_input_option(input='a')
        self.assertEqual(
            str(exception_context.exception),
            "Value should be either Q or R (in lowercase or uppercase)."
        )
```

FIGURE 13: TEST CASES FOR QUIT OR REPLAY OPTION FUNCTION

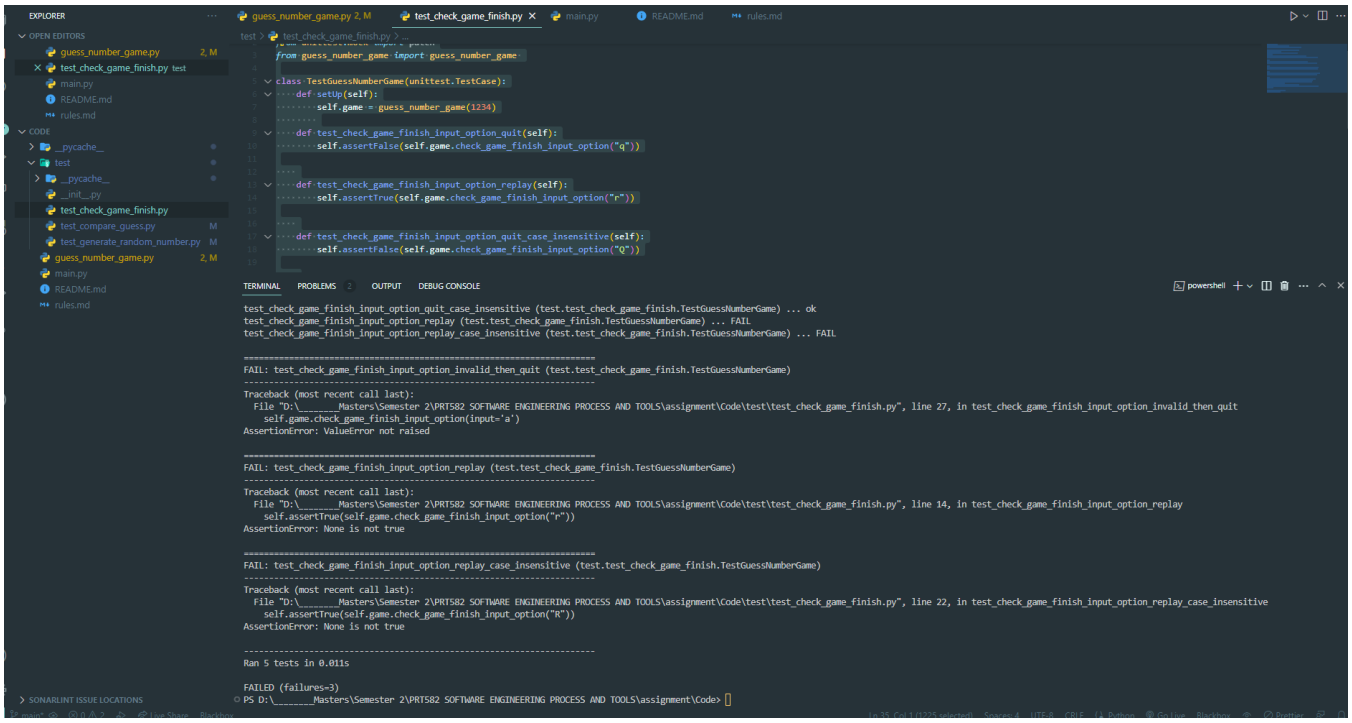

FIGURE 14: FAILED TEST FOR QUIT OR REPLAY OPTION FUNCTION

```python
# Check if the player wants to quit or replay
def check_game_finish_input_option(self, input):
    if input not in ["q", "Q", "r", "R"]:
        raise ValueError("Value should be either Q or R (in lowercase or uppercase).")

    # return false if to quit the game
    if input.lower() == "q":
        return False
    # return true to replay the game
    else:
        return True
```

FIGURE 15: FUNTION DEVELOPMENT FOR QUIT OR REPLAY OPTION FUNCTION

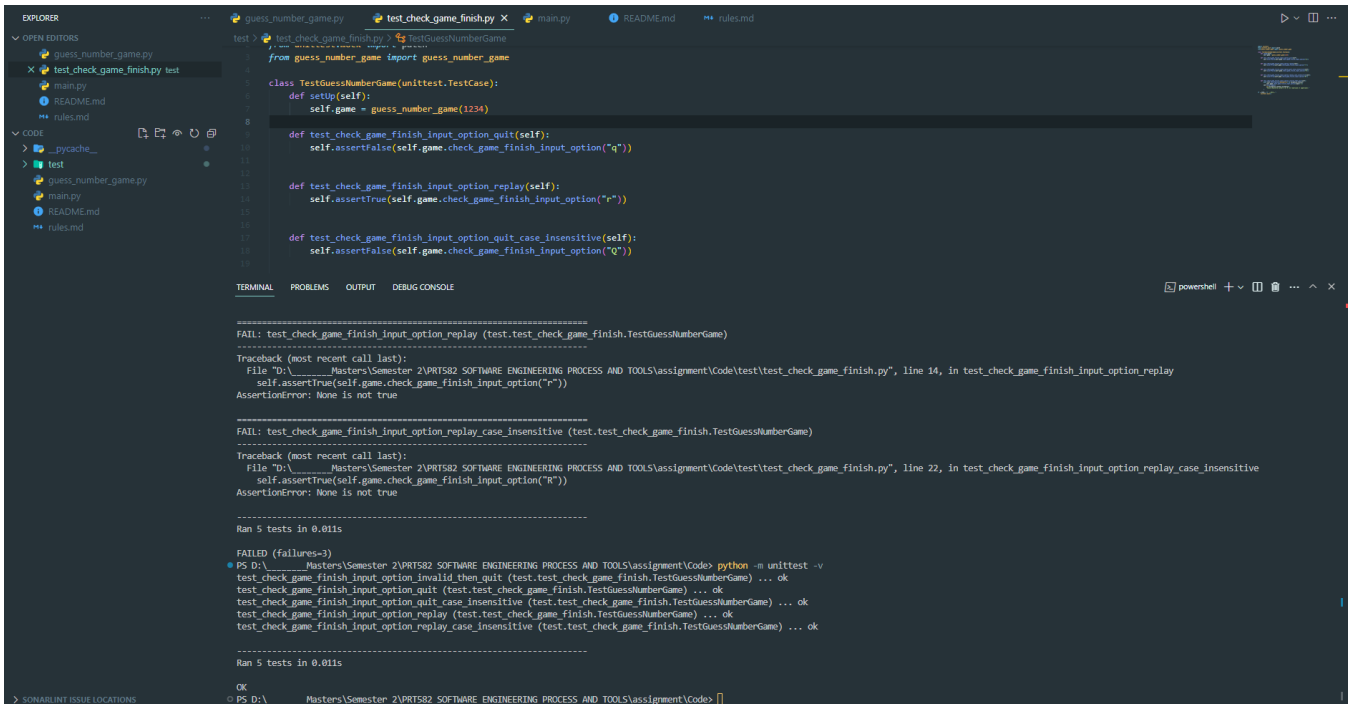

FIGURE 16: PASSED TEST FOR QUIT OR REPLAY OPTION FUNCTION

5. INTEGRETION TESTING

There are a total of 4 functions for unit testing and a total of 17 test cases.

Performing all tests together.

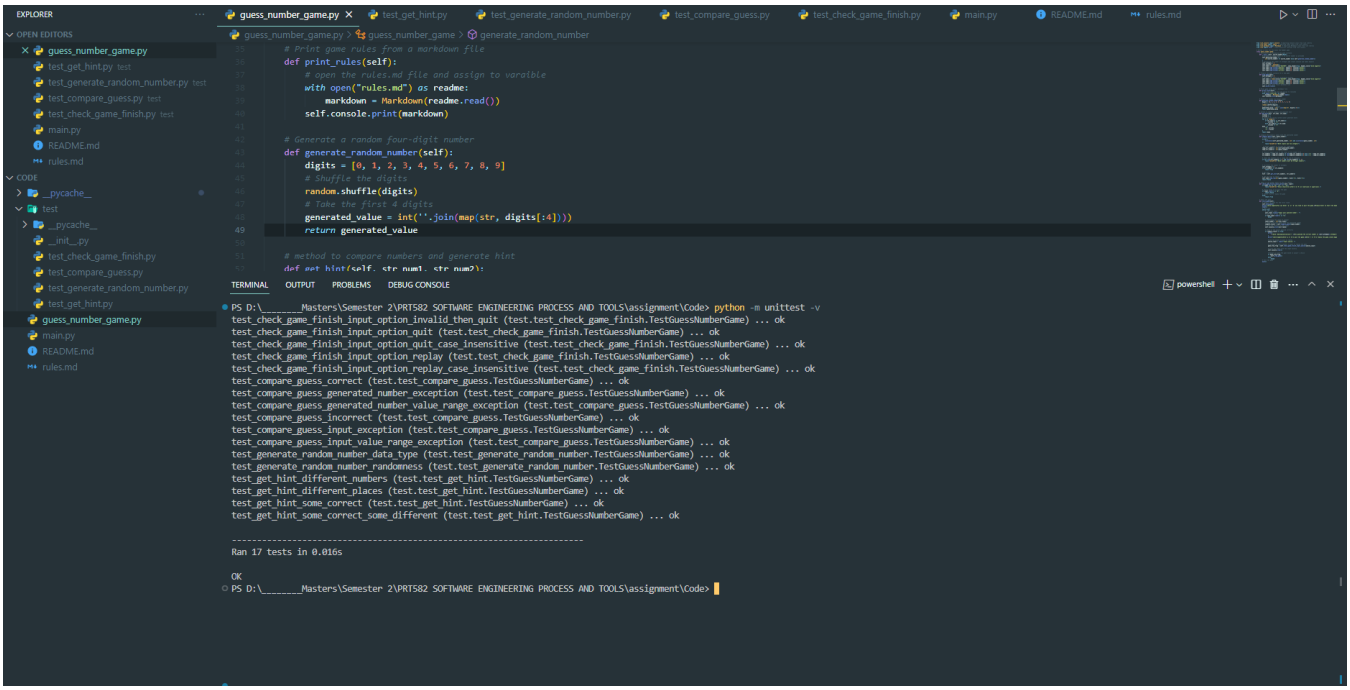

FIGURE 17: PERFORMING ALL UNIT TESTS

OUTCOME OF THE PROGRAM

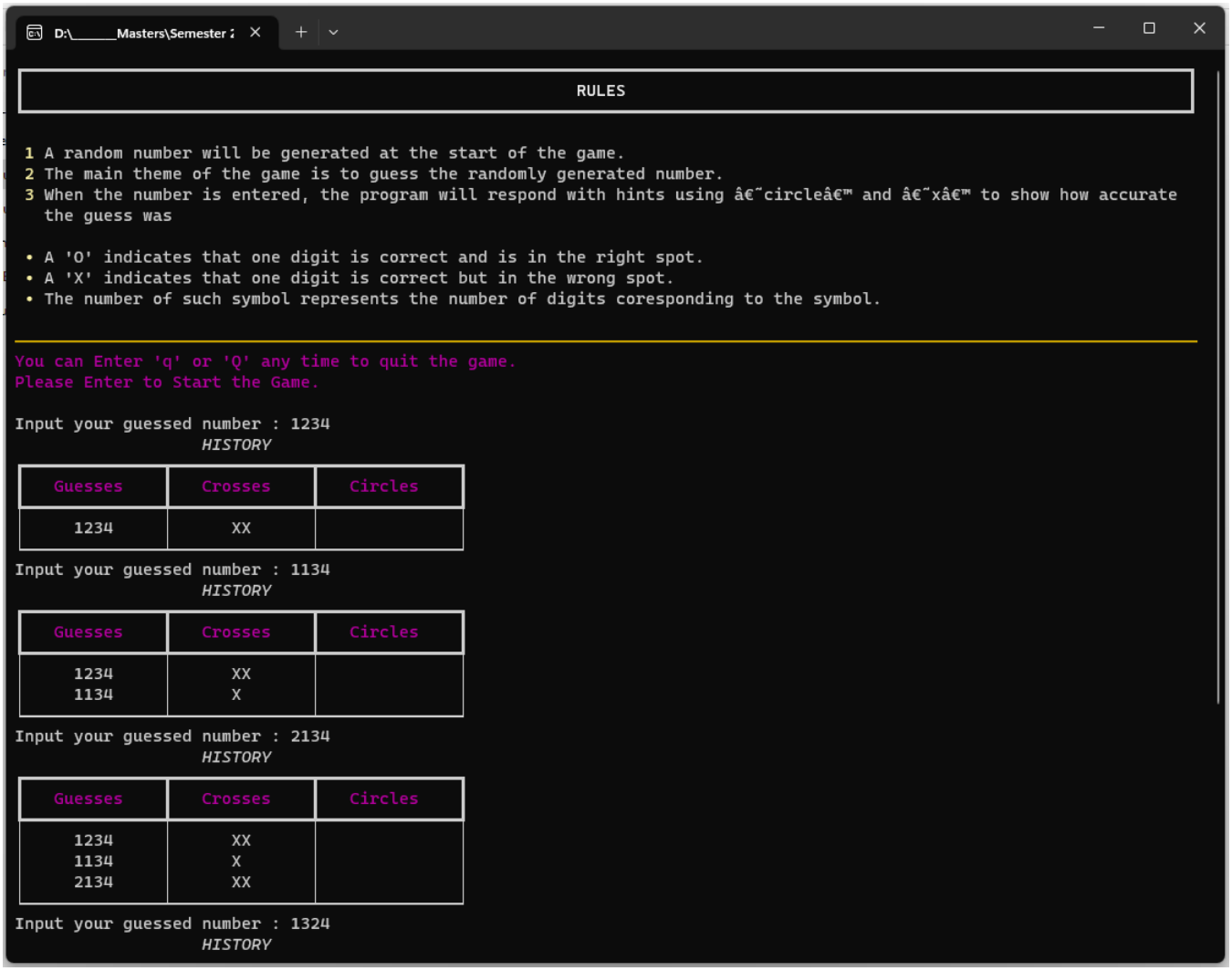

FIGURE 18: EXECUTABLE PROGRAM OF THE GAME

CONCLUSION

This report details how Test-Driven Development can be used to develop a Number Guessing Game. There were many requirements for the applications, such as random four-digit number generation, the input of user-guessed numbers until a correct number is guessed or the program is quit, using circles and crosses to show hints and the ability for the player to quit the game at any time. Overall, developing the program was relatively easy, but it was not easy as well. Python was used to create this program because it makes it easier to understand and maintain code, is fast, and, most importantly, comes pre-installed with the testing tools. In the beginning, it took time to write test cases and develop the application. Later, the familiarity of the tool increased, making it easier to implement. Most of the testing and debugging was done during the development phase, making the program less prone to bugs and issues. Therefore, this is the first time I have developed a program by writing tests first and then coding, and this assignment taught me a new way to do programming.

Link to the GitHub repository: https://github.com/ShahilJha/s368427-PRT582-Assignment-2.2

Executable Program File: *guess_number_game.exe* in the same file directory.

REFERENCES

Anwer, F., Aftab, S., Waheed, U., & Muhammad, S. S. (2017). Agile software development models tdd, fdd, dsdm, and crystal methods: A survey. International journal of multidisciplinary sciences and engineering, 8(2), 1-10.

Python, W. (2021). Python. Python Releases for Windows, 24.

Python. (2023, August 24). unittest — Unit testing framework. In Python Documentation. https://docs.python.org/3/library/unittest.html