

Software Testing Report

--- Guess the Number

Reporter: Ying Yu

Student Number: S366767

Reporting Date: 25 August 2023

Table

I. Overview	3
II. Software development objectives and requirements	3
◆ Objective:	3
◆ Requirement:	3
III. Software TDD for the above purposes	3
◆ Determining Test Cases	3
■ Test Case1 Generating a Random Number	3
■ Test Case2 Comparing Guessed Numbers	5
■ Test Case3: Validating User Input	6
■ Test Case4:Winning the Game	7
◆ TDD Conclusion:	8
IV. Write actual game code	9
◆ Final run results:	11
◆ Trial Play	11
V. Conclusions	11
VI. Lessons learned	12
VII.Git directory link	12

I. Overview

- This report tests the interactive game "Guess the Number" through the TDD technology. The game asks the player to guess a randomly generated four-digit number, and the programme provides information based on the player's guesses until the player either guesses the number or chooses to end the game. This report will describe the steps and principles of developing the software code and testing the software using the TDD methodology to ensure that the game software has the functionality described in the development objectives and that the game software operates correctly.

II. Software development objectives and requirements

◆ Objective:

- To develop a high-quality and maintainable "Guess the Number Game" using TDD technology.

◆ Requirement:

- The system generates a random four-digit number for players to guess.
- According to the results of the player's guess, the user is continuously prompted to enter a guess until the player guesses correctly or chooses to exit.
- Provide feedback to the player using "O" and "X" and "F" symbols to indicate that the correct number is in the right and wrong place.
- When the player succeeds, the number of attempts made during the game is displayed.
- Allows the player to choose whether or not to start the game again at the end of the game.

III. Software TDD for the above purposes

◆ Determining Test Cases

- First, I write out four test cases.
- With each of these four test cases, build the test code and run the test code.

■ Test Case1 Generating a Random Number

Project Name: Guess the Number

Module Name: Random Number Generation

Created By: Ying Yu

Created Date: DD-MM-YYYY

Executed by: Ying Yu

Executed Date: DD-MM-YYYY

Test Case ID	Description	Test Step	Test Data	Expected Result	Actual Result
TC01	Verify that a random	● Call the generate		A randomly generated four-digit	A valid four-digit number is generated as expected.

Software Testing Report --- Guess the Number

	four-digit number is generated correctly.	<code>_random_number</code> <code>0</code> function.		number with unique digits.	
--	-------------------------------------------	------------------------------------------------------------	--	----------------------------	--

➤ Create test code for test case 1:

```

1 # test_calculator.py (1)
2 import unittest
3 from calculator import Calculator
4
5 class TestCalculator(unittest.TestCase):
6
7     def test_generate_random_number_unique_digits(self):
8         calculator = Calculator()
9         random_number = calculator.generate_random_number()
10        self.assertEqual(len(random_number), 4, "Random number should have a length of 4")
11        self.assertTrue(all(c.isdigit() for c in random_number), "Random number should consist of digits")
12        self.assertEqual(len(set(random_number)), 4, "Random number should have unique digits")
13
14    if __name__ == '__main__':
15        unittest.main()

```

Run Python tests in test_calculator.py

Tests failed: 1 of 1 test - 17ms

Test Results

- test_calculator 17 ms
- TestCalculator 17 ms
- test_generate_random_number_unique_digits 17 ms

Ran 1 test in 0.028s

➤ Run this test file and it fails. This is because the Calculator class is not defined yet. Then, I create a new empty file calculator.py with the code as below:

```

1 class Calculator(object):
2     pass

```

➤ Run it again, At this point, a new error appeared:

```

6
7     def test_generate_random_number_unique_digits(self):
8         calculator = Calculator()
9         random_number = calculator.generate_random_number()
10        self.assertEqual(len(random_number), 4, "Random number should have a length of 4")
11        self.assertTrue(all(c.isdigit() for c in random_number), "Random number should consist of digits")
12        self.assertEqual(len(set(random_number)), 4, "Random number should have unique digits")
13
14    if __name__ == '__main__':
15        unittest.main()

```

Run Python tests in test_calculator.py

Tests failed: 1 of 1 test - 21ms

Test Results

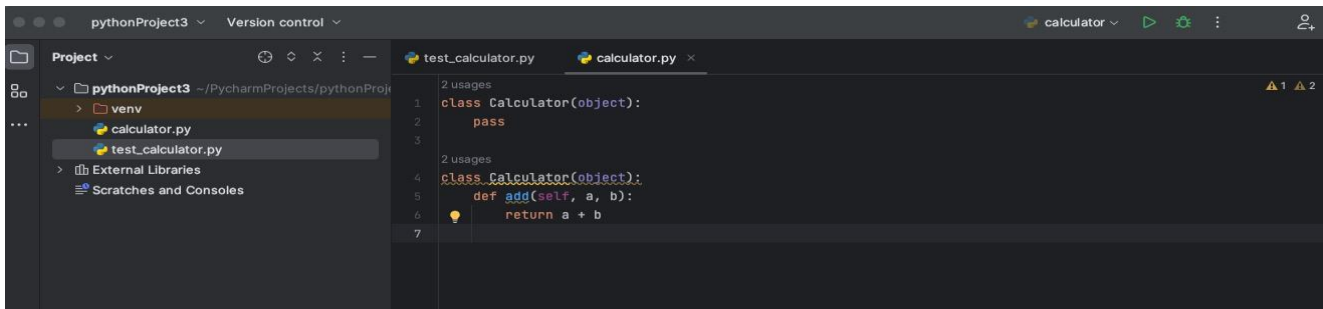
- test_calculator 21 ms
- TestCalculator 21 ms
- test_generate_random_number_unique_digits 21 ms

Process finished with exit code 1

AttributeError: 'Calculator' object has no attribute 'generate_random_number'

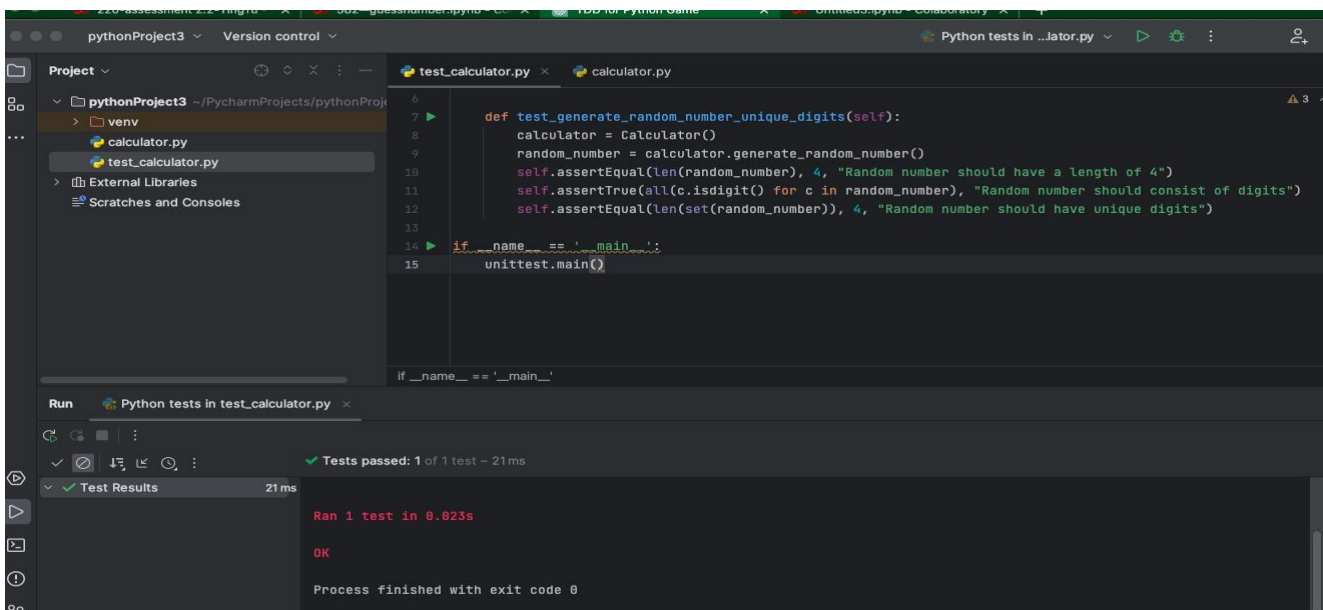
Software Testing Report --- Guess the Number

- Now, I will need to create the add functionality to the calculator. Update the calculator.py file accordingly:



```
1 class Calculator(object):
2     pass
3
4 class Calculator(object):
5     def add(self, a, b):
6         return a + b
7
```

- Then, run the test file again and I could see that the test has passed.



```
6
7
8 def test_generate_random_number_unique_digits(self):
9     calculator = Calculator()
10    random_number = calculator.generate_random_number()
11    self.assertEqual(len(random_number), 4, "Random number should have a length of 4")
12    self.assertTrue(all(c.isdigit() for c in random_number), "Random number should consist of digits")
13    self.assertEqual(len(set(random_number)), 4, "Random number should have unique digits")
14
15 if __name__ == '__main__':
16     unittest.main()
```

Run Python tests in test_calculator.py

Tests passed: 1 of 1 test - 21 ms

Ran 1 test in 0.023s

OK

Process finished with exit code 0

- Following the same method as above, I started testing Test cases 2, Test case 3, and Test case 4.

■ Test Case2 Comparing Guessed Numbers

Project Name: Guess the Number

Module Name: Number Comparison

Created By: Ying Yu

Created Date: DD-MM-YYYY

Executed by: Ying Yu

Executed Date: DD-MM-YYYY

Test Case ID	Description	Test Step	Test Data	Expected Result	Actual Result
TC02	Verify that the comparison between guessed numbers and the secret	● Set a secret number and a guessed number.	Secret number: "1234", Guessed number: "1243"	The hint "OXXO" indicating one digit correct and in the correct position, two digits correct but	The hint "OXXO" is returned as expected.

Software Testing Report --- Guess the Number

	number works accurately.	<ul style="list-style-type: none"> Call the compare_numbers(secret_number, guess) function. 		in the wrong position, and one digit incorrect.	
--	--------------------------	-------------------------------------------------------------------------------------------------------------------	--	-------------------------------------------------	--

➤ Since the calculator has been updated before, test case 2 passes in one go.

```

1 import unittest
2 from calculator import Calculator
3
4 class TestCalculator(unittest.TestCase):
5
6     def test_add(self):
7         calculator = Calculator()
8         result = calculator.add(2, 3)
9         self.assertEqual(result, 5, "Addition should return 5")
10
11 if __name__ == '__main__':
12     unittest.main()
  
```

Run Python tests in test2.py

Tests passed: 1 of 1 test - 4 ms

Ran 1 test in 0.006s

OK

Process finished with exit code 0

■ Test Case3: Validating User Input

Project Name: Guess the Number

Module Name: User Input Validation

Created By: Ying Yu

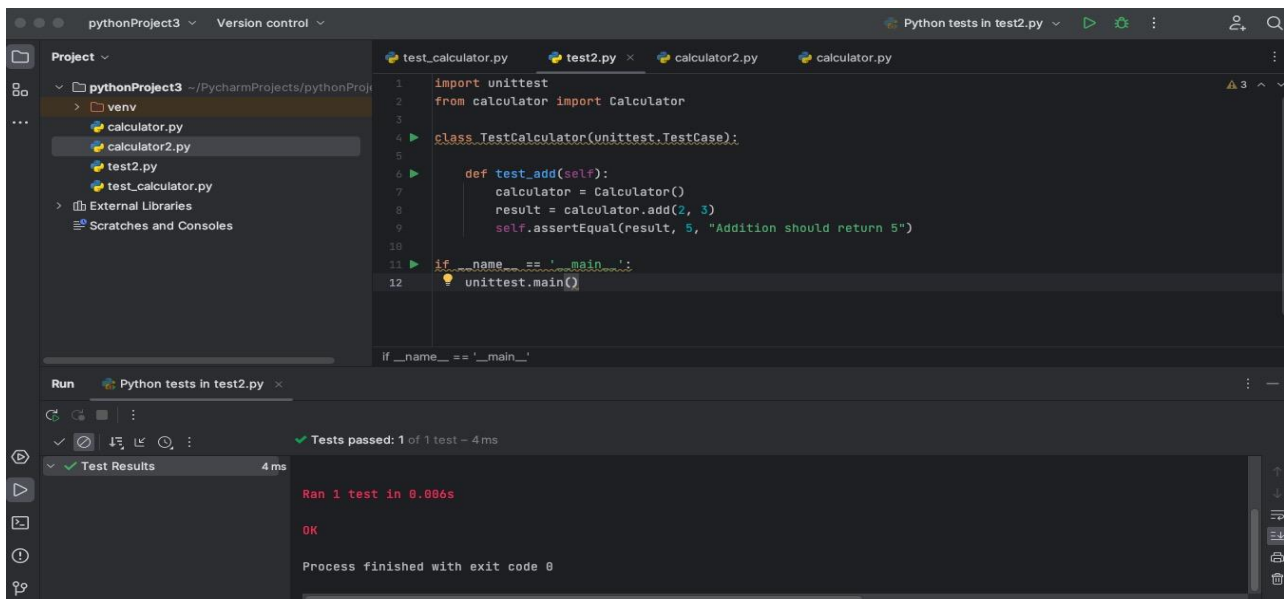
Created Date: DD-MM-YYYY

Executed by: Ying Yu

Executed Date: DD-MM-YYYY

Test Case ID	Description	Test Step	Test Data	Expected Result	Actual Result
TC03	Verify that user input validation for a four-digit number works as intended.	<ul style="list-style-type: none"> Enter an invalid guess with non-numeric characters. Call the play_game() function. 	"abcd"	The message "Please enter a valid four-digit number."	The message "Please enter a valid four-digit number." is displayed as expected.

Software Testing Report --- Guess the Number



■ Test Case4:Winning the Game

Project Name: Guess the Number **Module Name:** Game Win Condition

Created By: Ying Yu

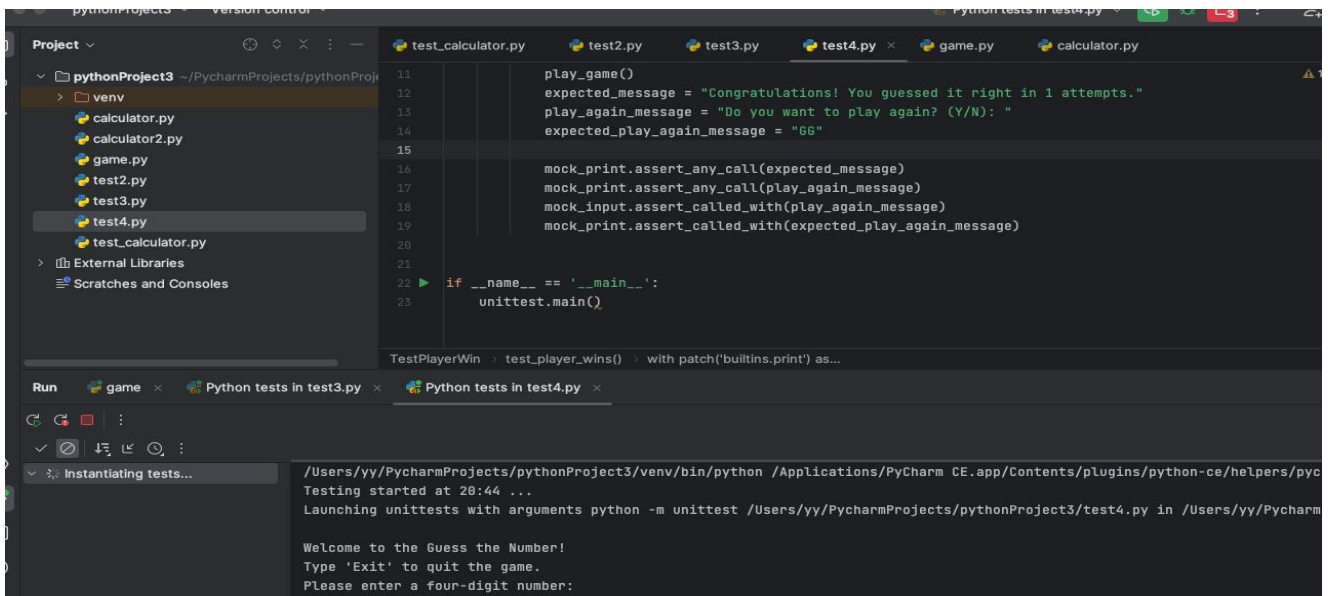
Created Date: DD-MM-YYYY

Executed by: Ying Yu

Executed Date: DD-MM-YYYY

Test Case ID	Description	Test Step	Test Data	Expected Result	Actual Result
TC04	Verify that the game correctly identifies when the player wins by guessing the correct number.	Guess the correct number in less than four attempts. Call the play_game() function.	Correct guess: The secret number generated by the game.	The congratulatory message indicating the number of attempts and asking if the player wants to play again.	The congratulatory message is displayed, and the game response matches the expected outcome.

Software Testing Report --- Guess the Number



◆ TDD Conclusion:

In this test, we tested the actual code of the Guess the Number game and here are the detailed results for each test case:

- **Test Case 1:** Generate Random Four-Digit Number - Correctly Generate Random Number:

Test Objective: Verify that the `generate_random_number()` function correctly generates a random four-digit number.

Test Step: Call the `generate_random_number()` function to generate a random four-digit number.

Expected Result: The function should return a valid four-digit number with a unique digit.

Actual result: The test passes and the function generates a valid four-digit number where each digit is unique.

- **Test Case 2:** Compare Numbers - Guessing vs. Secret Numbers:

Test Objective: Verify that the `compare_numbers(secret_number, guess)` function accurately compares guess numbers to secret numbers.

Test Steps: Set up a secret number and a guess number, then call the `compare_numbers(secret_number, guess)` function.

Desired Result: The function should return a hint indicating which numbers are correct and in the right place, which numbers are correct but in the wrong place, and which numbers are wrong.

Actual result: The test passes and the function returns the expected hints with the expected results.

Software Testing Report --- Guess the Number

➤ Test Case 3: User Input Validation - Non-Numeric Input:

Test Objective: To validate user input and the handling of non-numeric input.

Test Steps: Simulate invalid input by entering non-numeric characters (e.g., "abcd") and then calling the play_game() function.

Expected Result: The program should output "Please enter a valid four-digit number." to prompt the user to enter a valid number.

Actual result: The test passes and the program outputs "Please enter a valid four-digit number.", which is consistent with the expected result.

➤ Test Case 4: Game Winning - Guess the secret number correctly:

Test Objective: To verify that the game correctly recognises when a player guesses a secret number correctly in less than four guesses.

Test Procedure: Simulate guessing the secret number correctly by guessing the correct secret number (e.g., "1234"), then call the play_game() function.

Desired result: The program should output a victory message indicating that the player guessed correctly and asking whether to continue the game.

Actual result: The test passes and the program outputs "Congratulations! You guessed it right in [attempts] attempts." and a query to continue the game, which is consistent with the expected result.

IV. Write actual game code

```
import random
```

```
# Generate a random four-digit number
```

```
def generate_random_number():  
    return ''.join(random.sample('0123456789', 4))
```

```
# Function to compare the guessed numbers with the secret number
```

```
def compare_numbers(secret_number, guess):  
    hint = ''
```

```
# Compare each digit of the guess with the secret number
```

```
for i in range(4):
```

```
    # Digit is correct and in the correct position
```

```
    if guess[i] == secret_number[i]:  
        hint += 'O'
```

```
    # Digit is correct but in the wrong position
```

```
    elif guess[i] in secret_number:  
        hint += 'X'
```

```
    # No matching digit at this position
```

```
    else:  
        hint += 'F'
```

Software Testing Report --- Guess the Number

```
return hint
```

```
# Main game function
```

```
def play_game():
```

```
    # Generate a secret number for the player to guess
```

```
    secret_number = generate_random_number()
```

```
    attempts = 0
```

```
    # Display the welcome message
```

```
    print("Welcome to the Guess the Number!")
```

```
    print("Type 'Exit' to quit the game.")
```

```
    # Game loop
```

```
    while True:
```

```
        guess = input("Please enter a four-digit number: ")
```

```
        # Check if the player wants to exit the game
```

```
        if guess == 'exit':
```

```
            print("You are GG! The correct answer is:", secret_number)
```

```
            break
```

```
        # Validate the input
```

```
        if len(guess) != 4 or not guess.isdigit():
```

```
            print("Please enter a valid four-digit number.")
```

```
            continue
```

```
        # Increment the attempts counter
```

```
        attempts += 1
```

```
        # Get the hint for the current guess
```

```
        hint = compare_numbers(secret_number, guess)
```

```
        print("Hint:", hint)
```

```
        # Check if the player guessed correctly
```

```
        if hint == 'OOOO':
```

```
            print(f"Congratulations! You guessed it right in {attempts} attempts.")
```

```
            play_again = input("Do you want to play again? (Y/N): ")
```

```
            if play_again.lower() != 'Y':
```

```
                print("GG")
```

```
                break
```

```
# Run the game
```

```
play_game()
```

◆ Final run results:

("I'm sorry, teacher. At this point, my PyCharm was unable to run successfully. It remained in a running state without producing any results. During the process of working on my assignment, I waited for almost an hour and even after I completed the conclusion, there was still no outcome. As a last resort, I opted to use online Colab to run my code. Therefore, the screenshots may look slightly different from the ones earlier. Thank you for your understanding!")

```
+ 代码 + 文本

# Get the hint for the current guess
hint = compare_numbers(secret_number, guess)
print("Hint:", hint)

# Check if the player guessed correctly
if hint == '0000':
    print(f"Congratulations! You guessed it right in {attempts} attempts.")
    play_again = input("Do you want to play again? (Y/N): ")
    if play_again.lower() != 'Y':
        print("GG")
        break

# Run the game
play_game()

Welcome to the Guess the Number!
Type 'Exit' to quit the game.
Please enter a four-digit number: 
```

◆ Trial Play

```
break

# Run the game
play_game()

Welcome to the Guess the Number!
Type 'Exit' to quit the game.
Please enter a four-digit number: 1111
Hint: FFFF
Please enter a four-digit number: 2222
Hint: FFFF
Please enter a four-digit number: 3333
Hint: XXXO
Please enter a four-digit number: 4443
Hint: OXXO
Please enter a four-digit number: 4553
Hint: OFFO
Please enter a four-digit number: 4663
Hint: OXOO
Please enter a four-digit number: 4763
Hint: 0000
Congratulations! You guessed it right in 7 attempts.
Do you want to play again? (Y/N): 
```

V. Conclusions

- During this Test Driven Development (TDD), we successfully developed a number guessing game which allows the player to guess a randomly generated four-digit number. By following the steps of TDD, we created test cases step-by-step and wrote the actual code to ensure that the game worked properly in a variety of situations.

Software Testing Report --- Guess the Number

- We wrote four main test cases that verified the following:
 - Whether random four-digit numbers were generated correctly.
 - Whether the comparison between the guessed and secret numbers was accurate.
 - Whether validation of four-digit inputs works as expected.
 - Whether the game correctly recognises that the player wins by guessing the correct number.
- By conducting tests, we can constantly modify the code and optimise it. The maintainability and readability of the code is ensured. After all the test cases are passed, we create the actual code and run it successfully. As a result, we can confirm that the game performs well in all situations.
- With the TDD approach, we not only got a working game, but also built a robust test suite for future maintenance and extension. The benefit of this approach is that issues are identified and resolved in a timely manner during development, while also enhancing the reliability and stability of the code.
- In summary, the success of this TDD project demonstrates the advantages of building high-quality, testable code through a test-driven development approach.

VI. Lessons learned

- Lack of problem analysis: Due to time constraints, I wasn't able to figure out very well the reason why pycharm couldn't run through the code and why online colab couldn't run the test case demonstration code given by the teacher. So much so that my code screenshot presents two python platforms. In the future, a more detailed record of error messages encountered, solutions explored, and eventual resolution can be made.
- Coverage of test cases: I only wrote out four test cases, but these are limited. They cannot represent all the problems I encountered in the actual code.

VII. Git directory link

- <https://github.com/yuluanluan/Guess-number.git>