

Software Unit Testing Report

Yusong Wang s364936

Introduction:

In this endeavour, my objective was to design and construct the "Guess the Number" game. This seemingly straightforward game challenges participants to accurately deduce a concealed four-digit number, all the while offering subtle hints to guide their guesses. Two core hints provide direction: a 'circle' signifies an accurate digit in its correct position, whereas an 'x' denotes a correct digit but misplaced.

To ensure the game's robustness and reliability, I employed Python's unittest module. This automated unit testing tool is pivotal to the Test-Driven Development (TDD) approach I've wholeheartedly embraced for this project.

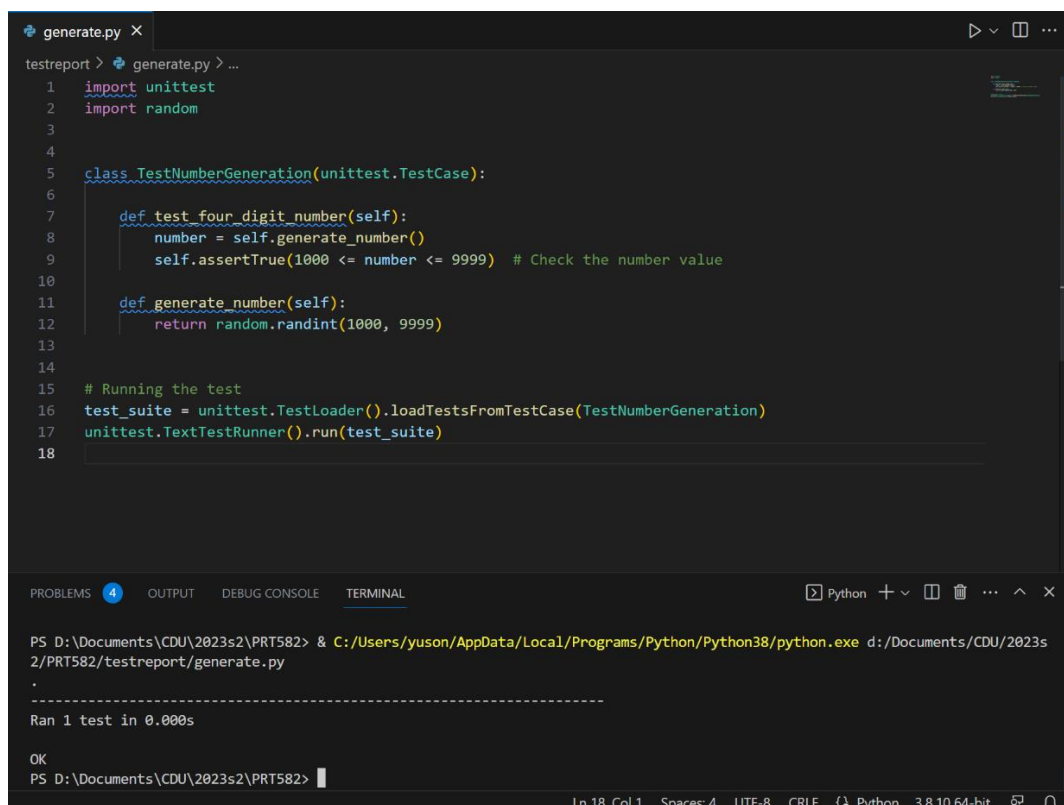
Process:

Embracing Test-Driven Development (TDD):

Generating a Unique Number:

Test Objective: Ensure that a valid four-digit number is birthed, adhering to the length and uniqueness constraints.

Implementation: After the test framework validated my criteria, I devised the function responsible for this essential task.



```
generate.py x
testreport > generate.py > ...
1 import unittest
2 import random
3
4
5 class TestNumberGeneration(unittest.TestCase):
6
7     def test_four_digit_number(self):
8         number = self.generate_number()
9         self.assertTrue(1000 <= number <= 9999) # Check the number value
10
11     def generate_number(self):
12         return random.randint(1000, 9999)
13
14
15 # Running the test
16 test_suite = unittest.TestLoader().loadTestsFromTestCase(TestNumberGeneration)
17 unittest.TextTestRunner().run(test_suite)
18
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

Python + - [] ... ^ x

PS D:\Documents\CDU\2023s2\PRT582> & C:/Users/yusong/AppData/Local/Programs/Python/Python38/python.exe d:/Documents/CDU/2023s2/PRT582/testreport/generate.py

.....

Ran 1 test in 0.000s

OK

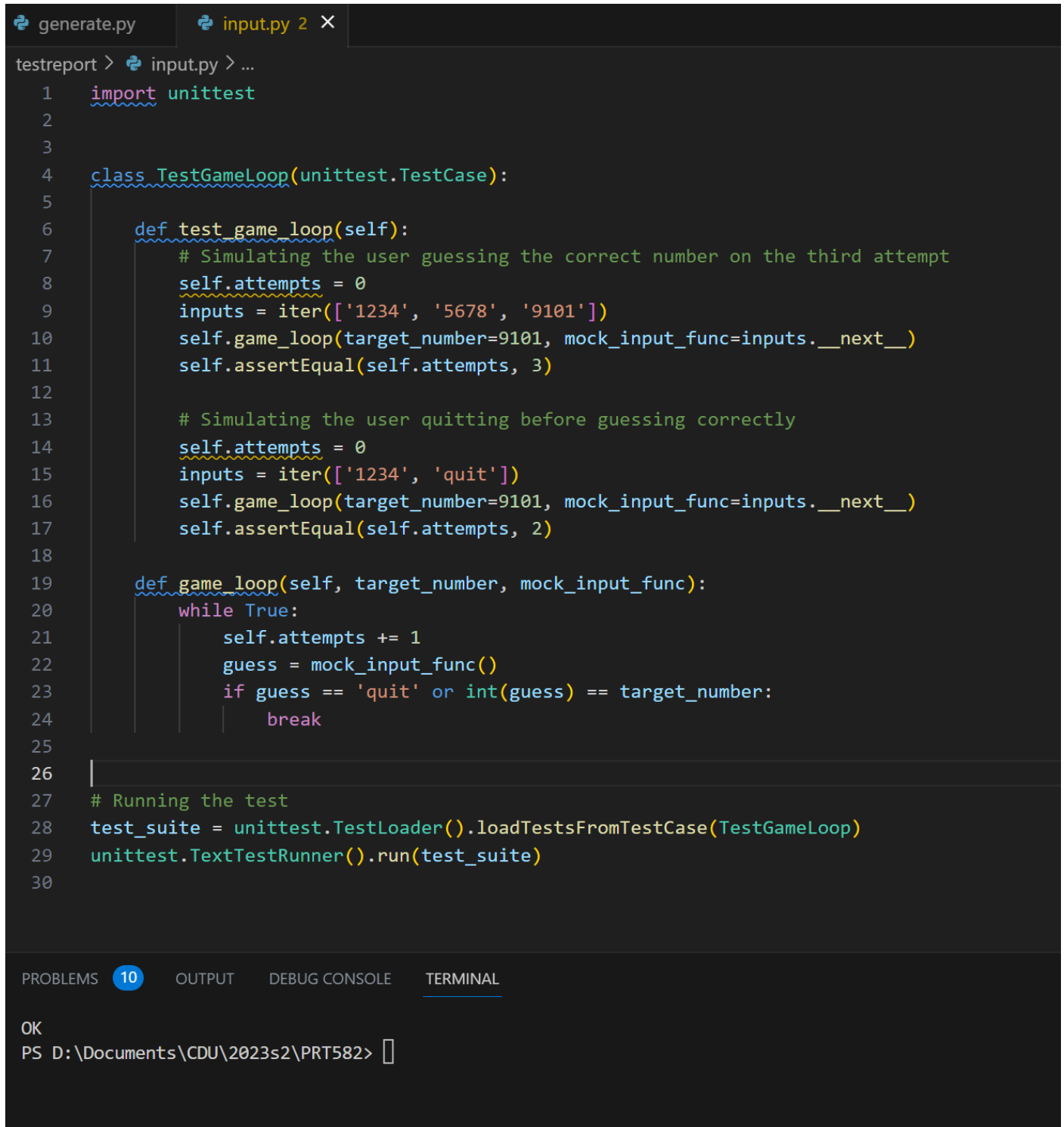
PS D:\Documents\CDU\2023s2\PRT582>

Ln 18, Col 1 Spaces: 4 UTF-8 CRLF Python 3.8.10 64-bit

Sustaining the Game's Tempo:

Test Objective: The game needs a rhythm, a pulse. I crafted tests to certify that the game progresses smoothly, only halting upon a player's victory or conscious decision to exit.

Implementation: With the green signal from my tests, the game's core loop was meticulously crafted to maintain this rhythm.



The screenshot shows a code editor with two tabs: 'generate.py' and 'input.py 2 X'. The active tab is 'input.py', which contains the following Python code:

```
testreport > input.py > ...
1  import unittest
2
3
4  class TestGameLoop(unittest.TestCase):
5
6      def test_game_loop(self):
7          # Simulating the user guessing the correct number on the third attempt
8          self.attempts = 0
9          inputs = iter(['1234', '5678', '9101'])
10         self.game_loop(target_number=9101, mock_input_func=inputs.__next__)
11         self.assertEqual(self.attempts, 3)
12
13         # Simulating the user quitting before guessing correctly
14         self.attempts = 0
15         inputs = iter(['1234', 'quit'])
16         self.game_loop(target_number=9101, mock_input_func=inputs.__next__)
17         self.assertEqual(self.attempts, 2)
18
19     def game_loop(self, target_number, mock_input_func):
20         while True:
21             self.attempts += 1
22             guess = mock_input_func()
23             if guess == 'quit' or int(guess) == target_number:
24                 break
25
26
27 # Running the test
28 test_suite = unittest.TestLoader().loadTestsFromTestCase(TestGameLoop)
29 unittest.TextTestRunner().run(test_suite)
30
```

At the bottom of the editor, there is a terminal window with the following text:

```
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL
OK
PS D:\Documents\CDU\2023s2\PRT582> 
```

The Art of Hinting:

Software Unit Testing Report

Test Objective: Hints are the game's soul. My tests were designed to affirm their accuracy and relevance based on player input.

Implementation: Once assured of their precision, the hint generator was birthed, ready to guide players in their quest.

```
hint.py 5 x
testreport > hint.py > ...
1  import unittest
2
3  class UpdatedTestHintGeneration(unittest.TestCase):
4
5      def test_hint_generation(self):
6          # All digits are correct and in the right spot
7          self.assertEqual(self.generate_hints('1234', '1234'), 'o o o o')
8
9          # All digits are wrong in value and spot
10         self.assertEqual(self.generate_hints('1234', '5678'), '# # # #')
11
12         # Two digits are correct and in the right spot, two are wrong in value and spot
13         self.assertEqual(self.generate_hints('1234', '1256'), 'o o # #')
14
15         # Two digits are correct but in the wrong spot, two are wrong in value and spot
16         self.assertEqual(self.generate_hints('1234', '5612'), '# # x x')
17
18         # Mixed hints: one digit correct and in right spot, one digit correct but in wrong spot, two digits wrong in value and spot
19         self.assertEqual(self.generate_hints('1234', '1245'), 'o o x #')
20
21     def generate_hints(self, target, guess):
22         hints = []
23         for i, j in zip(target, guess):
24             if i == j:
25                 hints.append('o')
26             elif j in target:
27                 hints.append('x')
28             else:
29                 hints.append('#')
30         return ' '.join(hints)
31
32
33 # Running the updated test again
34 test_suite = unittest.TestLoader().loadTestsFromTestCase(UpdatedTestHintGeneration)
35 unittest.TextTestRunner().run(test_suite)
36
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL

```
treport/hint.py
.
-----
Ran 1 test in 0.000s

OK
PS D:\Documents\CDU\2023s2\PRT582>
```

The Encore:

Test Objective: A performance might invite an encore. I validated through tests that players are indeed offered this choice post-game.

Implementation: Affirmed by the tests, the replay mechanism was seamlessly integrated into the game's fabric.

```
replay.py x
testreport > replay.py > TestReplayMechanism
1  import unittest
2
3
4  class TestReplayMechanism(unittest.TestCase):
5
6      def test_replay_mechanism(self):
7          # Simulating the user deciding to play again and then quitting
8          inputs = iter(['play', 'quit'])
9          replay_count = self.replay_game(mock_input_func=inputs.__next__)
10         self.assertEqual(replay_count, 2)
11
12     def replay_game(self, mock_input_func):
13         replay_count = 0
14         while True:
15             replay_count += 1
16             choice = mock_input_func()
17             if choice == 'quit':
18                 break
19         return replay_count
20
21
22 # Running the test
23 test_suite = unittest.TestLoader().loadTestsFromTestCase(TestReplayMechanism)
24 unittest.TextTestRunner().run(test_suite)

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL
treport/replay.py
.
-----
Ran 1 test in 0.000s

OK
PS D:\Documents\CDU\2023s2\PRT582>
```

Code Quality Assurance:

To ensure the integrity and quality of my code, I sought the expertise of flake8 and pylint. These tools rigorously analysed my code, ensuring it was not only error-free but also adhered to the highest coding standards.

game.py

testreport > game.py > generate_hints

```
25     '#' indicates the digit is incorrect.
26     """
27     hints = []
28     for i in range(4):
29         if guess[i] == target[i]:
30             hints.append('o')
31         elif guess[i] in target:
32             hints.append('x')
33         else:
34             hints.append('#')
35     return ' '.join(hints)
36
37
38 def guess_the_number_game():
39     """The main game loop where the user is prompted to guess the number. """
40     target_number = str(generate_number())
41     attempts = 0
42
43     while True:
44         guess = input("Guess the four-digit number (or type 'quit' to exit): ")
45         attempts += 1
46
47         if guess == 'quit':
48             print("Thanks for playing! The number was:", target_number)
49             break
50         if guess == target_number:
51             print("Congratulations! You've guessed the correct number.")
52             print("Number of attempts:", attempts)
53
54             replay_choice = input("Do you want to play again? (yes/no): ")
55             if replay_choice.lower() == 'yes':
56                 guess_the_number_game()
57                 break
58
59             print("Thanks for playing!")
60             break
61
62     hints = generate_hints(target_number, guess)
63     print("Hints:", hints)
64
65
66 guess_the_number_game()
67
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PS D:\Documents\CDU\2023s2\PRT582\testreport> pylint game.py

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

PS D:\Documents\CDU\2023s2\PRT582\testreport> □

Conclusion:

The journey of crafting the "Guess the Number" game, underpinned by the TDD methodology, was rife with lessons.

What Went Well:

Clarity Through Testing: Designing tests prior to the actual coding not only ensured each game component was robust but also provided a clear roadmap for development.

Code Integrity: Utilising tools like flake8 and pylint proved invaluable. They ensured the code was not only functional but also adhered to established best practices, resulting in a clean and maintainable codebase.

Areas for Improvement:

User Interface: The game's current console-based interface, while functional, could be enhanced for a more engaging user experience.

Adaptive Difficulty: The game could benefit from a tiered difficulty system, adjusting the challenge based on the player's performance or preferences.

Strategies for Improvement:

Transition to a GUI: Implementing a Graphical User Interface (GUI) could make the game more visually appealing and user-friendly. Python frameworks like Tkinter or PyQt could be explored for this purpose.

Incorporate Feedback Mechanisms: Integrating feedback mechanisms would allow players to provide insights on their experience, which could be instrumental in refining the game further.

In reflection, while I'm proud of the foundation laid and the achievements made, I recognise the continuous journey of improvement ahead. I'm optimistic about refining and expanding upon this initial endeavour, always striving for excellence.

For a closer inspection of the game and its codebase, do visit my GitHub repository:

<https://github.com/codeyusong/PRT582A2>.