UNIT TEST REPORT

PRT582

Lin Yang s364263

Table of Contents

Introduction

The project is to use Test Driven Development (TDD) method to build up a number guessing game in Python called “Guess the Number”. The player of the game will have to accurately guess a randomly generated four-digit number with provided clues about the number. Once the player successfully guesses the number, the program will display the total number of attempts made.

In our case, the automated unit testing tool used is Python's built-in "unittest" module, which allows for the generation and execution of tests to check the functioning of code units. Tests are organised into classes called "test cases," which contain methods with the test_ prefix. Assertions such as assertEqual and assertRaises confirm predicted results. Test cases can be organised into suites for fast execution, and a built-in test runner runs the testing process and provides feedback on outcomes. The "unittest" framework emphasises TDD concepts, assuring code accuracy and maintainability.

Objectives

The objectives of the projects are listed as follows:

1. Develop a functional "Guess the Number" game following the principles of TDD.
2. Allow the player to guess a four-digit randomly generated number.
3. Provide clues to the player about the accuracy of their guesses.
4. Display the number of attempts taken to guess the correct number.
5. Offer the option to play again or to quit the game.
6. Implement the game in a modular and well-tested manner.

Requirements

According to the principles of TDD, the requirement for generating proper Python code can be described below:

1. Generate a random four-digit number:
    - Method: **generate_random_number()**
    - Description: Generate and return a random four-digit number as a string.
2. Provide Hints:
    - Method: **provide_hints(random_number, guessed_number)**
    - Description: Compare the guessed number with the secret number and generate hints using 'circle' and 'x'.
3. Main Game Loop:
    - Method: **play_game()**
    - Description: Implement the main game loop, allowing the player to make guesses until they guess correctly or quit.

4. Main Program:
    - Method: **main()**
    - Description: Initialize the game and start the main game loop; ask the player if they want to play again or quit once they have guessed the correct number or choose to quit.

Process

First, we create the test file test_Guess_the_Number_Game.py and project file Guess_the_Number_Game.py . We import module unittest as well as the project file in the test file.

```
import unittest

import Guess_the_Number_Game
```

And then set the Class GuesstheNumberGame and employ setup() to set up an instance of Guess_the_Number_game.

```
class TestGuesstheNumberGame(unittest.TestCase):
    def setUp(self):
        # This method is called before each test case
        self.game = Guess_the_Number_Game
```

Add .main() to execute the test case in the Python editor:

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

Now let's start generating test cases according to the requirements:

1. Generate random numbers

We have created a test case that ensures the generate_random_number() method generates a random four-digit number of type of string.

```
    def test_generate_random_number(self):
        self.assertIsInstance(self.game.generate_random_number(), str, "Returned value is not a string")
        self.assertEqual(len(self.game.generate_random_number()), 4, "Returned string does not have a length of 4")
```

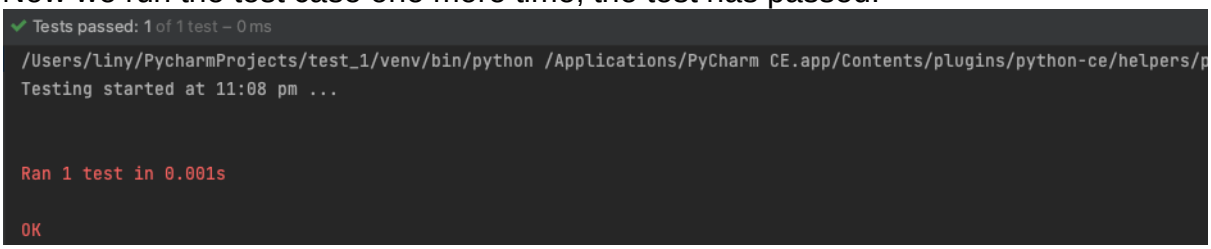
Let's run the first test, the error shows because we haven't defined the method generate_random_number in the project file.

```
Tests failed: 1 of 1 test – 3ms

FAILED (errors=1)
Launching unittests with arguments python -m unittest /Users/liny/PycharmProjects/test_1/test_Guess_the_Number_Game.


Error
Traceback (most recent call last):
  File "/Users/liny/PycharmProjects/test_1/test_Guess_the_Number_Game.py", line 10, in test_generate_random_number
    self.assertIsInstance(self.game.generate_random_number(), str, "Returned value is not a string")
                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: module 'Guess_the_Number_Game' has no attribute 'generate_random_number'
```

Let’s create the method in Guess_the_Number_Game.py to generate a four-digit number and convert it into string.

```
import random

def generate_random_number():
    # Generate a random number between 1000 and 9999
    random_number = random.randint(1000, 9999)
    return str(random_number)
```

Now we run the test case one more time, the test has passed.



2. Provide hints

We create a test that calls provide_hints(random_number, guessed_number) with different secret and guessed numbers and checks if the generated hints are valid.

```
def test_provide_hints(self):
    random_number = "1234"
    guessed_number = "5678"
    hints = self.game.provide_hints(random_number, guessed_number)

# Check if the hints are valid
valid_hint_characters = {'-'}
for hint in hints:
    self.assertIn(hint, valid_hint_characters, f"Invalid hint character: {hint}")
```

Now we write the provide_hints(random_number, guessed_number) function in the project file to compare the guessed number with the random number and generate hints.

```
def provide_hints(random_number, guessed_number):
    hints = []

    for i in range(len(random_number)):
        # Matching digit, add to hints
        if random_number[i] == guessed_number[i]:
            hints.append(random_number[i])
        # Guessed digit is in random number, but not in the same position
        elif guessed_number[i] in random_number:
            hints.append('0')
        # No match
        else:
            hints.append('-')

    # Combine hints into a single string
    return ''.join(hints)
```

Initially we have set up the random_unmber to be 1234, and guessed_number to be 5678, if the provide_hints runs correctly, the result should be“----” which would be in

the list of valid_hint_characters. As the test result shows, the function passes the test.

If we make the guessed number exactly as the random_number, the hints should be which means the function is well defined.

3. Main game loop

Write a test that calls play_game() and checks if the function runs without errors (e.g., no crashes, infinite loops).

Write the play_game() function to implement the main game loop, allowing the player to make guesses until they guess correctly or quit.
Run the test again (it should pass this time).

```
def play_game():
    random_number = generate_random_number()
    attempts = 0

    while True:
        guessed_number = input("Enter your guess (4-digit number) or 'q' to quit: ")

        if guessed_number.lower() == 'q':
            print(f"The secret number was: {random_number}")
            break

        if not guessed_number.isdigit() or len(guessed_number) != 4:
            print("Invalid input. Please enter a valid 4-digit number.")
            continue

    attempts += 1
    hints = provide_hints(random_number, guessed_number)

    if guessed_number == random_number:
        print(f"Congratulations! You've guessed the number {random_number} in {attempts} attempts.")
        break
    else:
        print(f"Hints: {hints}")
```

To test the play_game function, we type in random 4 digit number “1247”, “000”, letters“lojv”, punctuation, “1596” and finally “q” to quit the game.

```
Guess_the_Number_Game
/Users/liny/PycharmProjects/test_1/venv/bin/python /Users/liny/PycharmProjects/test_1/Guess_the_
Enter your guess (4-digit number) or 'q' to quit: 1247
Hints: ----
Enter your guess (4-digit number) or 'q' to quit: 0000
Hints: ----
Enter your guess (4-digit number) or 'q' to quit: loju
Invalid input. Please enter a valid 4-digit number.
Enter your guess (4-digit number) or 'q' to quit: -=!@
Invalid input. Please enter a valid 4-digit number.
Enter your guess (4-digit number) or 'q' to quit: 1596
Hints: -5-0
Enter your guess (4-digit number) or 'q' to quit: q
The secret number was: 6553
```

The test results are expected as the method is designed.

4. Main program

First, we import patch.

```
import unittest
from unittest.mock import patch
import Guess_the_Number_Game
```

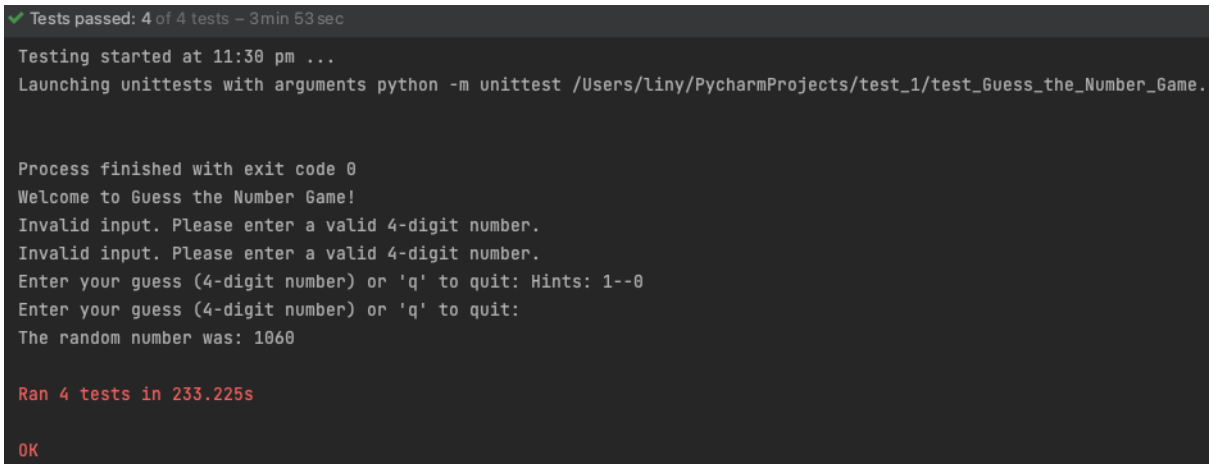
Then we create a test that calls main() and checks if the function runs without errors (e.g., no crashes).

```python
def test_main(self):
    # Mock the input to simulate user responses
    user_input = ['yes', 'no']
    user_input_index = 0

    with patch('builtins.input', side_effect=user_input):
        try:
            self.game.main()
        except StopIteration:
            pass  # StopIteration will be raised when input is exhausted
```

Then we write the main() function to initialize the game and start the main game loop using the previously defined methods.

```python
def main():
    print("Welcome to Guess the Number Game!")

    while True:
        play_game()

        play_again = input("Do you want to play again? (yes/no): ").lower()
        if play_again != 'yes':
            print("Thank you for playing!")
            break

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

Now we run the test again and see the tests run smoothly with no error.



Flake8 and Pylint checking

If we run flake8 and pylint on the two files, we can see the files have to be modified accordingly. While modifying the files, we have to ensure that the test functions of the test file and the game functions of the project file are not affected. Therefore, as long as we have done the modification, we would test both the files interchangeably to guarantee the requested functionalities.

Flake8 suggests that test file test_Guess_the_Number_Game.py has some errors.

```
(venv) liny@LintekiMacBook-Air test_1 % flake8 test_Guess_the_Number_Game.py
test_Guess_the_Number_Game.py:5:1: E302 expected 2 blank lines, found 1
test_Guess_the_Number_Game.py:11:80: E501 line too long (104 > 79 characters)
test_Guess_the_Number_Game.py:12:80: E501 line too long (115 > 79 characters)
test_Guess_the_Number_Game.py:22:80: E501 line too long (89 > 79 characters)
test_Guess_the_Number_Game.py:33:9: F841 local variable 'user_input_index' is assigned to but never used
test_Guess_the_Number_Game.py:41:1: E305 expected 2 blank lines after class or function definition, found 1
test_Guess_the_Number_Game.py:49:1: W391 blank line at end of file
```

After modifying the file, the new test file test_guess_the_number_game.py shows no errors in flake8.

```
(venv) liny@LintekiMacBook-Air test_1 % flake8 test_guess_the_number_game.py
(venv) liny@LintekiMacBook-Air test_1 %
```

Flake8 indicates that the project file Guess_the_Number_Game.py has some errors.

```
(venv) liny@LintekiMacBook-Air test_1 % flake8 Guess_the_Number_Game.py
Guess_the_Number_Game.py:3:1: E302 expected 2 blank lines, found 1
Guess_the_Number_Game.py:10:1: E302 expected 2 blank lines, found 1
Guess_the_Number_Game.py:27:1: E302 expected 2 blank lines, found 1
Guess_the_Number_Game.py:32:80: E501 line too long (84 > 79 characters)
Guess_the_Number_Game.py:46:80: E501 line too long (104 > 79 characters)
Guess_the_Number_Game.py:63:1: E305 expected 2 blank lines after class or function definition, found 1
(venv) liny@LintekiMacBook-Air test_1 %
```

After changing the file, the new project file test_guess_the_number_game.py shows no errors in flake8.

```
(venv) liny@LintekiMacBook-Air test_1 % flake8 guess_the_number_game.py
(venv) liny@LintekiMacBook-Air test_1 %
```

Pylint checking for file test_Guess_the_Number_Game.py shows that many lines must be improved.

```
(venv) liny@LintekiMacBook-Air test_1 % pylint test_Guess_the_Number_Game.py
************* Module test_Guess_the_Number_Game
test_Guess_the_Number_Game.py:13:0: C0301: Line too long (104/100) (line-too-long)
test_Guess_the_Number_Game.py:14:0: C0301: Line too long (115/100) (line-too-long)
test_Guess_the_Number_Game.py:50:0: C0305: Trailing newlines (trailing-newlines)
test_Guess_the_Number_Game.py:1:0: C0114: Missing module docstring (missing-module-docstring)
test_Guess_the_Number_Game.py:1:0: C0103: Module name "test_Guess_the_Number_Game" doesn't conform to snake_case nami
test_Guess_the_Number_Game.py:5:0: C0115: Missing class docstring (missing-class-docstring)
test_Guess_the_Number_Game.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
test_Guess_the_Number_Game.py:16:4: C0116: Missing function or method docstring (missing-function-docstring)
test_Guess_the_Number_Game.py:26:4: C0116: Missing function or method docstring (missing-function-docstring)
```

After modifying the file, the new test file test_guess_the_number_game.py has achieved 10 out of 10 in Pylint.

```
Your code has been rated at 10.00/10 (previous run: 9.68/10, +0.32)

(venv) liny@LintekiMacBook-Air test_1 %
```

Pylint checking for file Guess_the_Number_Game.py shows that the file needs to be modified accordingly.

```
(venv) liny@LintekiMacBook-Air test_1 % pylint Guess_the_Number_Game.py
************* Module Guess_the_Number_Game
Guess_the_Number_Game.py:46:0: C0301: Line too long (104/100) (line-too-long)
Guess_the_Number_Game.py:1:0: C0114: Missing module docstring (missing-module-docstring)
Guess_the_Number_Game.py:1:0: C0103: Module name "Guess_the_Number_Game" doesn't conform to snake_case naming style (
Guess_the_Number_Game.py:10:0: C0116: Missing function or method docstring (missing-function-docstring)
Guess_the_Number_Game.py:13:4: C0200: Consider using enumerate instead of iterating with range and len (consider-usin
Guess_the_Number_Game.py:27:0: C0116: Missing function or method docstring (missing-function-docstring)
Guess_the_Number_Game.py:45:8: R1723: Unnecessary "else" after "break", remove the "else" and de-indent the code insi
Guess_the_Number_Game.py:52:0: C0116: Missing function or method docstring (missing-function-docstring)
```

After adjusting the file based on the feedback of pylint, the new project file guess_the_number_game.py has reached 9.51 out of 10.

```
--------------------------------------------------------------------
Your code has been rated at 9.51/10 (previous run: 9.27/10, +0.24)
```

Conclusion

This project has demonstrated the value of TDD as a systematic way to develop reliable software. Writing tests before developing the code would ensure that each component worked properly. The unittest framework provides a standardised method for creating and managing test cases. The project provides an opportunity to gain more insight into the real-world testing process and motivation to engage more in software engineering studying and practicing.

Following TDD helps detect possible issues earlier, resulting in higher code quality. The modular architecture of code improves readability and maintainability by separating methods for generating numbers, providing hints, and playing the game. The main loop provides users the option to continue playing or exit, increasing their engagement.

To obtain more thorough coverage, precise and detailed test cases should be included. Furthermore, by introducing input validation for replies indicating whether to play again, the game's experience might be improved. This might be used with improved user prompts to guarantee clarity in interactions.

To realize these enhancements, we can increase the breadth of the test suite to include a broader range of scenarios, including faulty inputs and boundary conditions. Meanwhile, error-handling methods shall carefully check user inputs and provide clear instructions.

Github link is provided as follows:

https://github.com/Linyy0123/PRT582.git