UNIT CODE PRT582

SOFTWARE ENGINEERING: PROCESS AND TOOLS

Guess the Number game using Test Driven Development in Python

Submitted By:

Name: Niket Kiritkumar Patel

Student ID:

S364848

Table of Content

Table of Figure

1. Introduction

1.1 Objectives of the Project

The main objective of this project is to develop a "Guess the Number" game using the principles of Test-Driven Development (TDD) in Python. The game challenges players to guess a randomly generated four-digit number while providing clues to aid their guesses. By implementing TDD, we aim to ensure the reliability and accuracy of the game's functionality through a systematic testing approach. This project not only hones our coding skills but also deepens our understanding of TDD methodologies. Through successful completion, we strive to demonstrate proficiency in TDD, Python programming, and game development, while delivering an engaging and functional game.

1.2 Requirements of the Game

The game's requirements are outlined to create an engaging and interactive player experience:

- Random Number Generation:
  - A four-digit number is randomly generated as the target for players to guess.
- Guessing Loop:
  - Players continue to make guesses until the correct number is guessed or they quit.
- Hint System:
  - The program responds to guesses with specific hints:
    - 'Circle': Correct digit in the correct position.
    - 'X': Correct digit in the wrong position.
    - '-': Incorrect digit.
- Game Completion:
  - Upon successful guess:
    - The number of attempts made is displayed.
    - Players are given the option to play again or quit.
- User-Friendly Exit:
  - Players can exit the game at any point, providing flexibility and convenience.
- Educational Value:
  - The game exercises players' deductive reasoning skills and adds an educational element.
- Clear Gameplay Flow:
  - The game follows a logical and clear sequence, enhancing user understanding.

1.3 Overview of Automated Unit Testing Tool

The unittest module in Python facilitates efficient automated unit testing, enhancing the quality and reliability of code through:

- **Structured Test Cases:**
  - unittest provides a framework for creating organized test cases, promoting modular and maintainable testing code.
- **Assertions and Test Discovery:**
  - Developers can utilize various assertions to validate expected outcomes, while automated test discovery streamlines the testing process.
- **Batch Testing and Reporting:**
  - unittest allows for batch execution of test cases and offers comprehensive reporting, enabling rapid identification of issues and errors.

By leveraging the unittest module, developers can systematically validate code functionality, improve code stability, and expedite the development process.

2. Process

2.1 Test-Driven Development (TDD) and Automated Unit Testing

Test-Driven Development (TDD) is a software development approach that emphasizes creating tests before writing the actual code. This method guides the development process and ensures code functionality aligns with requirements. Automated Unit Testing, a vital component of TDD, streamlines this process by enabling the creation and execution of tests automatically.

In TDD, the cycle starts with writing a failing test case based on a specific requirement. This prompts the developer to design code that fulfils the test's criteria. As development progresses, automated unit tests continuously validate the code's correctness, catching regressions early and promoting code stability.

The synergy of TDD and automated unit testing offers several advantages. It enforces clear code structure, identifies issues promptly, and supports code refactoring without sacrificing functionality. By incorporating TDD and automated unit testing in this project, we ensure our "Guess the Number" game adheres to the requirements while maintaining code integrity throughout its development lifecycle.

2.2 Implementation Steps

2.2.1 Requirement 1: Game Intro

The game's introduction serves as the players' first interaction with the "Guess the Number" game, providing crucial information and setting the stage for their gameplay experience. This requirement is met by the following test case function, relevant game function, and their explanation:

- **Test Case Function - test_print_game_intro**

This test case function checks if the game's introduction is displayed correctly when the **print_game_intro** function is called.

```
@patch('sys.stdout', new_callable=StringIO)
def test_print_game_intro(self, mock_output):
    """Test if the game intro is printed correctly."""
    expected_output = (
        "============================================================\n"
        "                    GUESS THE NUMBER GAME                   \n"
        "============================================================\n"
        "Welcome, brave challenger!\n"
        "Can you crack the code and reveal\n"
        "the secret 4-digit number?\n"
        "You'll receive hints after each guess.\n"
        "Legend:\n"
        "     O - Correct digit in correct position\n"
        "     X - Correct digit in wrong position\n"
        "     - - Incorrect digit\n"
        "============================================================"
    )

    self.game.print_game_intro()
    output = mock_output.getvalue().strip()

    self.assertIn(expected_output, output)

print("Print game intro test passed: "
      "The game introduction is printed correctly.")
```

*Figure 1 : **test_print_game_intro***

- **Relevant Game Function - print_game_intro**

The **print_game_intro** function is a static method responsible for printing the game's introduction.

```
@staticmethod
def print_game_intro():
    """Prints game introduction with rules and hints for the Guess The Number game."""

    print("============================================================")
    print("                    GUESS THE NUMBER GAME                   ")
    print("============================================================")
    print("Welcome, brave challenger!")
    print("Can you crack the code and reveal")
    print("the secret 4-digit number?")
    print("You'll receive hints after each guess.")
    print("Legend:")
    print("     O - Correct digit in correct position")
    print("     X - Correct digit in wrong position")
    print("     - - Incorrect digit")
    print("============================================================")
```

Figure 2 : print_game_intro

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Use
\NIKET\AppData\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugp
adapter/../..\debugpy\launcher' '56613' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
.
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
```

Activate Windows
Go to Settings to activate W

Figure 3 : output of test into

2.2.2 Requirement 2: Random Number Generation

This requirement pertains to the generation of a random 4-digit number within a specified range. The following test case function, relevant game function, and requirement details ensure the proper functioning of this aspect:

- **Test Case Function - test_generate_random_number**

This test case verifies that the generated number falls within the valid range.

```python
def test_generate_random_number(self):
    """Test random number generation within the valid range."""

    self.assertTrue(1000 <= int(self.game.number) <= 9999)
    print("Random number generation test passed: "
          "The generated number is within the valid range.")
```

*Figure 4 : **test_generate_random_number***

- **Relevant Game Function - __init__**

The __init__ function initializes the game by generating a random 4-digit number.

```python
def __init__(self):
    self.number = str(random.randint(1000, 9999))
    # print(self.number)
    self.attempts = 0
```

Figure 5 : random_number

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Users
\NIKET\AppData\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\
adapter/../..\debugpy\launcher' '56913' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
Random number generation test passed: The generated number is within the valid range.
..
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

Activate Windows
Go to Settings to activate Windows.

Figure 6 : output of random_number

2.2.3 Requirement 3: Correct Guess and Output

This requirement validates the game's ability to handle a correct guess, display the appropriate output messages, and accurately track the number of attempts. The relevant test case function, the involved game function, and the requirement specifics are outlined below:

- **Test Case Function - test_correct_guess**

This test case simulates a scenario where the player makes a correct guess. It checks if the game outputs the correct messages and tracks the number of attempts.

```
@patch('builtins.input', side_effect=['1234', 'n'])
def test_correct_guess(self, _):
    """Test correct guessing, output messages, and attempts tracking."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.number = '1234'
        self.game.play_game()
        output = mock_output.getvalue().strip()
        self.assertIn("Congratulations! You guessed the number correctly.", output)
        self.assertIn("Number of attempts: 1", output)
        self.assertIn("Thank you for playing!", output)
    print("Correct guess test passed: "
          "Output messages and attempts tracking are working as expected.")
```

Figure 7 : test_correct_guess

- **Relevant Game Function – play_game()**

The **play_game** function orchestrates the gameplay, including checking for correct guesses and displaying relevant messages.

```
def play_game(self):
    """Play the number guessing game, providing hints and tracking attempts."""

    self.print_game_intro()

    while True:
        guess = input("Enter your guess (4 digits) or 'q' to quit: ")
        if guess.lower() == 'q':
            print("\nOh no, you've decided to leave the challenge...")
            print(f"But wait, the secret number you were after is: {self.number}!")
            self.attempts = 0
            break
        if not guess.isdigit() or len(guess) != 4:
            print("Invalid input. Please enter a valid 4-digit number.")
            continue
        self.attempts += 1
        self.give_hints(guess)
        if guess == self.number:
            print("Congratulations! You guessed the number correctly.")
            print("Number of attempts:", self.attempts)
            self.ask_to_play_again()
            break
```

Figure 8 : correct_guess

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Users\NIKET\AppData\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '57153' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
Correct guess test passed: Output messages and attempts tracking are working as expected.
.Random number generation test passed: The generated number is within the valid range.
..
----------------------------------------------------------------------
Ran 3 tests in 0.005s

OK
```

Figure 9 : output of random_number

2.2.4 Requirement 4: Handling Invalid Inputs

This requirement focuses on the game's capability to handle various scenarios involving invalid inputs during gameplay. The following test case function, pertinent game function, and requirement details elaborate on this aspect:

- **Test Case Function - test_invalid_inputs**

This test case simulates different scenarios where the player enters invalid inputs and assesses whether the game handles them correctly.

```
@patch('builtins.input', side_effect=['123a', '12345', 'abcd',
                                      '6789', '4567', '9876', 'q'])
def test_invalid_inputs(self, _):
    """Test handling of various invalid inputs during gameplay."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.play_game()
        output = mock_output.getvalue().strip()
        self.assertIn("Invalid input. Please enter a valid 4-digit number.", output)
        self.assertIn("Invalid input. Please enter a valid 4-digit number.", output)
        self.assertIn("Invalid input. Please enter a valid 4-digit number.", output)
    print("Invalid inputs test passed: "
          "Handling of invalid inputs during gameplay is correct.")
```

Figure 10 : test_invalid _input

- **Relevant Game Function - play_game**

  The **play_game** function is responsible for capturing and handling user inputs during the gameplay loop.

  Requirement Details

  - When a player enters an input that is not a valid 4-digit number, the game should recognize it as an invalid input.
  - The game should output the message "Invalid input. Please enter a valid 4-digit number."
  - This message is presented using the **print** function, addressing each instance of invalid input within the gameplay loop.

```python
def play_game(self):
    """Play the number guessing game, providing hints and tracking attempts."""

    self.print_game_intro()

    while True:
        guess = input("Enter your guess (4 digits) or 'q' to quit: ")
        if guess.lower() == 'q':
            print("\nOh no, you've decided to leave the challenge...")
            print(f"But wait, the secret number you were after is: {self.number}!")
            self.attempts = 0
            break
        if not guess.isdigit() or len(guess) != 4:
            print("Invalid input. Please enter a valid 4-digit number.")
            continue
        self.attempts += 1
        self.give_hints(guess)
        if guess == self.number:
            print("Congratulations! You guessed the number correctly.")
            print("Number of attempts:", self.attempts)
            self.ask_to_play_again()
            break
```

Figure 11 : invalid_input

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Users\NIKET\AppData\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '57386' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
Correct guess test passed: Output messages and attempts tracking are working as expected.
.Random number generation test passed: The generated number is within the valid range.
.Invalid inputs test passed: Handling of invalid inputs during gameplay is correct.
..
----------------------------------------------------------------------
Ran 4 tests in 0.006s

OK
```

Figure 12 : output of invalid input

2.2.5 Requirement 5: Hints Generation for Partially Correct Guesses

This requirement involves generating appropriate hints when the player's guess contains partially correct digits. It also addresses scenarios where no correct digits are present in the hints. The following test case functions, relevant game function, and requirement details consolidate these aspects:

- **Test Case Function - test_partial_correct_guess**
  This test case examines the generation of hints for a partially correct guess and assesses whether the hints are produced accurately.
- **Test Case Function - test_no_correct_digits**
  This test case simulates a scenario where no correct digits are present in the hints generated for the player's guess.

```python
@patch('builtins.input', side_effect=["1253"])
def test_partial_correct_guess(self, _):
    """Test hints for partially correct guesses and hints generation."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.number = "1234"
        self.game.give_hints(input())
        output = mock_output.getvalue().strip()
        expected_hints = "O O - X"
        expected_output = f"Hints: {expected_hints}"
        self.assertEqual(output, expected_output)
    print("Partial correct guess test passed: "
          "Hints generation for partially correct guesses is correct.")

@patch('builtins.input', side_effect=["5678"])
def test_no_correct_digits(self, _):
    """Test scenario with no correct digits in the hints."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.number = "1234"
        self.game.give_hints(input())
        output = mock_output.getvalue().strip()
        expected_hints = "- - - -"
        expected_output = f"Hints: {expected_hints}"
        self.assertEqual(output, expected_output)
    print("No correct digits hints test passed: "
          "Handling of no correct digits scenario is correct.")
```

Figure 13 : test_partial_correct_hint

- **Relevant Game Function - give_hints**
  The **give_hints** function generates and displays hints based on the player's guess and the target number.

Requirement Details

- The game should generate hints to guide the player towards the correct answer, particularly for partially correct guesses.
- In cases where digits are correctly guessed and correctly positioned, the hint should be represented as 'O'.
- For digits that are correctly guessed but positioned incorrectly, the hint should be represented as 'X'.
- Digits that are incorrect both in value and position should be indicated with '-'.
- If no digits in the guess are correct, the game should print "No digits are correct."
- In scenarios where no correct digits are present in the hints, the game should output the message "Hints: - - - -" to indicate that none of the guessed digits are correct.
- invalid input within the gameplay loop.

```python
def give_hints(self, guess):
    """Generate and display hints for the guessed number."""
    hints = []
    for i, digit in enumerate(guess):
        if digit == self.number[i]:
            hints.append('O')
        elif digit in self.number:
            hints.append('X')
        else:
            hints.append('-')
    if len(hints) == 0:
        print("No digits are correct.")
    else:
        if " ".join(hints) != "O O O O":
            print("Hints:", " ".join(hints))
```

Figure 14 : give hint

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Users\NIKET\AppD
ata\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\
launcher' '57597' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
Correct guess test passed: Output messages and attempts tracking are working as expected.
.Random number generation test passed: The generated number is within the valid range.
.Invalid inputs test passed: Handling of invalid inputs during gameplay is correct.
.No correct digits hints test passed: Handling of no correct digits scenario is correct.
.Partial correct guess test passed: Hints generation for partially correct guesses is correct.
..
----------------------------------------------------------------------
Ran 6 tests in 0.008s

OK
```

Activate Windows

Figure 15 : output of give hint

2.2.6 Requirement 6: Play Again and Handling User Choices

This requirement encompasses the game's functionality of managing user choices to play again or quit after a game session. It also addresses scenarios involving invalid input. The following test case functions, relevant game function, and requirement details consolidate these aspects:

- **Test Case Function - test_play_again_yes**
  This test case validates the game's ability to handle a scenario where the player chooses to play again.
- **Test Case Function - test_play_again_no**

  This test case examines the scenario where the player chooses not to play again.

- **Test Case Function - test_play_again_invalid_then_yes_no**
  This test case assesses scenarios involving invalid input, the choice to play again, and the choice not to play again.

```python
@patch('builtins.input', side_effect=['y', 'q'])
def test_play_again_yes(self, _):
    """Test replay scenario where player chooses to play again."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.ask_to_play_again()
        output = mock_output.getvalue().strip()
        self.assertIn("Let's play another exciting round! 🎮", output)
        self.assertEqual(self.game.attempts, 0)
    print("Play again (yes) test passed: Player's choice to play again is handled correctly.")

@patch('builtins.input', side_effect=['n'])
def test_play_again_no(self, _):
    """Test replay scenario where player chooses not to play again."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.ask_to_play_again()
        output = mock_output.getvalue().strip()
        self.assertEqual(self.game.attempts, 0)
        self.assertEqual("Thank you for playing!", output)
    print("Play again (no) test passed: Player's choice "
          "not to play again is handled correctly.")

@patch('builtins.input', side_effect=['invalid', 'n'])
def test_play_again_invalid_then_yes_no(self, _):
    """Test replay scenarios: invalid input, play again, and not play again."""

    with patch('sys.stdout', new_callable=StringIO) as mock_output:
        self.game.ask_to_play_again()
        output = mock_output.getvalue().strip()
        self.assertIn("Invalid input. Please enter 'y' or 'n'.", output)
        self.assertIn("Thank you for playing!", output)
        self.assertEqual(self.game.attempts, 0)
    print("Play again scenarios test passed: "
          "Handling of invalid input and choices is correct.")
```

Figure 16 : test_play_again_user_choice

- **Relevant Game Function - ask_to_play_again**
  The **ask_to_play_again** function presents the player with the option to play again or quit based on their input.

Requirement Details

- After completing a game session, the player should be given the choice to play again or quit by entering 'y' or 'n' respectively.
- If the player chooses to play again ('y'), the game should reset and generate a new random number for the next round.
- If the player chooses not to play again ('n'), the game should output the message "Thank you for playing!" and conclude.
- When the player provides invalid input (neither 'y' nor 'n'), the game should respond with "Invalid input. Please enter 'y' or 'n'."
- The ask_to_play_again function handles these scenarios based on the player's choice and input validation.

```python
def ask_to_play_again(self):
    """Reset game or exit based on player's choice."""

    self.attempts = 0
    while True:
        play_again = input("Do you want to play again? (y/n): ")
        if play_again.lower() == 'y':
            self.number = str(random.randint(1000, 9999))
            self.attempts = 0
            # print(self.number)
            print("Let's play another exciting round! 🎮")
            self.play_game()
            break
        if play_again.lower() == 'n':
            print("Thank you for playing!")
            break
        print("Invalid input. Please enter 'y' or 'n'.")
```

Figure 17 : : play_again

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>  d:; cd 'd:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number'; & 'C:\Users\NIKET\AppD
ata\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\NIKET\.vscode\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\
launcher' '57983' '--' 'D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number\test_guess_the_number.py'
Print game intro test passed: The game introduction is printed correctly.
Correct guess test passed: Output messages and attempts tracking are working as expected.
.Random number generation test passed: The generated number is within the valid range.
.Invalid inputs test passed: Handling of invalid inputs during gameplay is correct.
.No correct digits hints test passed: Handling of no correct digits scenario is correct.
.Partial correct guess test passed: Hints generation for partially correct guesses is correct.
.Play again scenarios test passed: Handling of invalid input and choices is correct.
.Play again (no) test passed: Player's choice not to play again is handled correctly.
.Play again (yes) test passed: Player's choice to play again is handled correctly.
..
----------------------------------------------------------------------
Ran 9 tests in 0.013s

OK
```

Activate Windows

Figure 18 : output of play again.

In the implementation phase, the following key requirements were fulfilled:

1. **Game Setup:** Developed a number guessing game generating a 4-digit random target number.
2. **Input Validation:** Implemented input validation for guesses, ensuring only valid 4-digit entries are accepted.
3. **Hints Generation:** Created hints indicating the correctness of digits in guesses.
4. **Play Again Handling:** Allowed players to replay or quit, handling user choices effectively.
5. **Invalid Input Handling:** Managed various scenarios of invalid inputs during gameplay.

6. **Partially Correct Hints:** Generated accurate hints for partially correct guesses.
7. **Automated Testing:** Utilized unit tests to validate functions' correctness against defined criteria. By addressing these requirements, the Guess the Number game ensures engaging, accurate, and user-friendly gameplay.

3. Code Quality:

In terms of code quality, thorough assessment was performed using pylint, achieving a perfect score of 10/10. While employing flake8, minor feedback was received regarding line length, although some lines were intentionally extended for demonstration. Overall, the implementation adheres to high coding standards, ensuring both functionality and readability.

- **Pylint**

Guess_the_number.py

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number> python -m pylint .\guess_the_number.py

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

Figure 19 : pylint_game

Test_Guess_the_number.py

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number> python -m pylint .\test_guess_the_number.py

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

Figure 20 : pylint_test

- **Flake8**

  Guess_the_number.py

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number> flake8 .\guess_the_number.py
.\guess_the_number.py:2:80: E501 line too long (93 > 79 characters)
.\guess_the_number.py:38:80: E501 line too long (90 > 79 characters)
.\guess_the_number.py:54:80: E501 line too long (83 > 79 characters)
.\guess_the_number.py:62:80: E501 line too long (87 > 79 characters)
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number>
```

Figure 21 : flake8 game

Test_Guess_the_number.py

```
PS D:\CDU Study\SEM 2\Software Tools\Assignment 2\Guess_the_Number> flake8 .\test_guess_the_number.py
.\test_guess_the_number.py:68:5: E303 too many blank lines (2)
.\test_guess_the_number.py:76:80: E501 line too long (87 > 79 characters)
.\test_guess_the_number.py:90:80: E501 line too long (88 > 79 characters)
.\test_guess_the_number.py:91:80: E501 line too long (88 > 79 characters)
.\test_guess_the_number.py:92:80: E501 line too long (88 > 79 characters)
.\test_guess_the_number.py:133:80: E501 line too long (98 > 79 characters)
.\test_guess_the_number.py:149:80: E501 line too long (83 > 79 characters)
.\test_guess_the_number.py:176:80: E501 line too long (81 > 79 characters)
.\test_guess_the_number.py:177:80: E501 line too long (81 > 79 characters)
.\test_guess_the_number.py:178:80: E501 line too long (81 > 79 characters)
.\test_guess_the_number.py:192:80: E501 line too long (87 > 79 characters)
.\test_guess_the_number.py:196:80: E501 line too long (88 > 79 characters)
.\test_guess_the_number.py:197:80: E501 line too long (88 > 79 characters)
```

Figure 22 : flake8 test

4. Conclusion

4.1 Lessons Learned

Throughout the development process of the Guess The Number game, several valuable lessons were gained. The successes achieved include implementing core game mechanics, effective handling of user inputs, and generating accurate hints. This project also highlighted areas for improvement, such as enhancing code modularity and providing more comprehensive error handling.

By reflecting on these lessons, the project offers insights into both accomplishments and avenues for further enhancement, contributing to continuous learning and growth.

- Successes and Achievements

  The development of the Guess The Number game yielded several notable successes and achievements. Key accomplishments include the successful implementation of the game's fundamental mechanics, such as generating a random target number and providing hints based on user guesses. The input validation system effectively handles various scenarios, enhancing the user experience. The integration of automated unit testing using pylint demonstrated a commitment to code quality, resulting in a flawless score of 10/10. These achievements showcase a solid foundation for a functional and engaging game.
- Areas for Improvement

  While the project achieved its core objectives, there are areas identified for potential improvement. One notable aspect is the optimization of code modularity to enhance

maintainability and extensibility. Additionally, the feedback from flake8 regarding line length highlights the importance of refining code readability without sacrificing clarity. Strengthening error handling mechanisms and incorporating more comprehensive error messages can further enhance user guidance. By addressing these areas, the project can evolve into an even more refined and polished application.

4.2 GitHub Link

https://github.com/Niket16/Guess_the_Number