PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS

SOFTWARE UNIT TESTING REPORT

Guess a Number game using TDD in Python

Github: https://github.com/7uv4n/prt582_unit_testing_report_s364626

YUVANSHANKAR AZHAGUMURUGAN

STUDENT ID: S364626

Table of Contents

Introduction

The primary aim of this project is to develop a game, "Guess the Number" with Python and implement an automated unit testing for ensuring its accuracy and functionality. This game revolves around challenge of predicting a secret four digit number. This report showcases concept of automated testing, a critical aspect in software development process

At start of the game, players are asked to input their guess for secret number. The game provides clues based on the input: 'circle' for digit in correct place, 'x' for correct digit in wrong place. The player continues to guess the number until the correct number is found or the player choose to quit. This game then displays the number of attempts made to guess the number.

In order to automatically verify the game's key functionalities, we are incorporating automated testing into this project, using Python's Unit-testing module '*unittest'*. This package can be employed to create distinct test cases for assessing specific components of game's functionalities.

Unit testing can be defined as a software development process, where the smallest testable parts of a software, project or application (known as units) are tested and scrutinised individually.

This automated testing method can be achieved through predefined test cases to systematically validate functions rather than manual testing. Testing is done for checking the validity of random number generated, format of clue generated and input validation functions. Each test case has predefined conditions that are expected to pass or fail and produce results to indicate reliablity of game's features.

In this report, two codes are created "number_guessing_game.py" (contains main game code) and "testing.py" (contains automated testing code).

Popular Culture: A version of this game is found as one of the in-game puzzles in a popular video game "Sleeping Dogs", where the protoganist solves puzzle similar to this game to complete some missions.

Process

"Guess the Number" is developed using Test Driven Development (TDD) approach in Python using the functionalities and conditions listed in this report. The requirements of the game are listed as follows:

1. The game must generate a random four-digit number and save it as a secret number
2. The game should continuously prompt the user to guess the number until they guess it correctly or choose to quite
3. At each guess, the game provides clues to indicate the accuracy of the guess
    a. 'o' – denotes that the digit is correct and in correct position
    b. 'x' – denotes that the digit is correct but in incorrect position
    c. '#' – denotes that the digit is incorrect.
4. After completion of game:
    a. Total attempts made is displayed
    b. User can choose the option to quit game or play again.
5. User has the freedom anytime they want

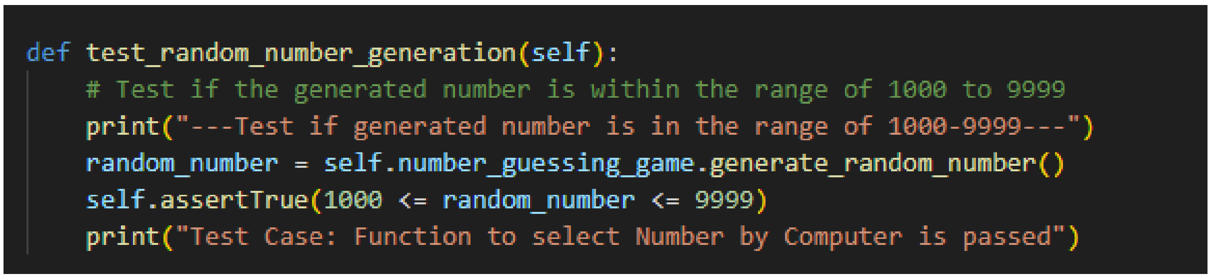
The development process involves breaking down the requirements into functionalities and each to fulfill a specific role. These functionalities are implemented in 'number_guessing_game.py' file. A suite of unit tests are created in a separate file, 'testing.py', and automated.

Functionality 1 (Generating Random number for Secret number):

The game will generate a random four digit number first. A class *'MainGame'* is utilised to create the game and employs object oriented Paradigm to encapsulate game's logic and state.

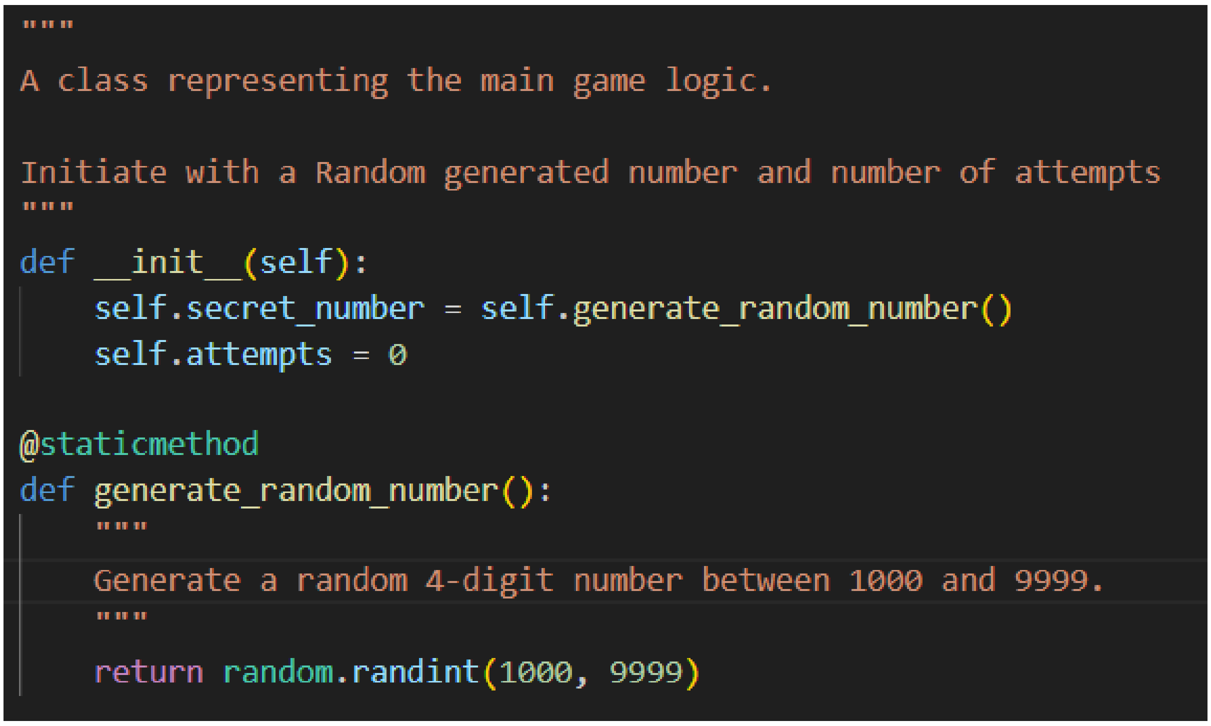
Test Case Function unit: To validate functionality of generating a random number, a unit test function is provided. This function sees whether the random number lies in the range of 1000 to 9999. This can be achieved using 'assertTrue' asserting method from testing framework. (Figure 1)

Relevant Game Function Unit: Upon instantiation, the class initialises two attributes: 'secret_number' and 'attempts'. The 'secret_number' attribute is set badsed on result of 'generate_random_number()' static method, which generates a random integer between 1000 and 9999. This method also utilises Python's buit-in 'random' library to ensure unpredictable selection of secret number. The 'attempts' attribute is initialised to 0, to indicate the number of attempts player has created. (Figure 2)



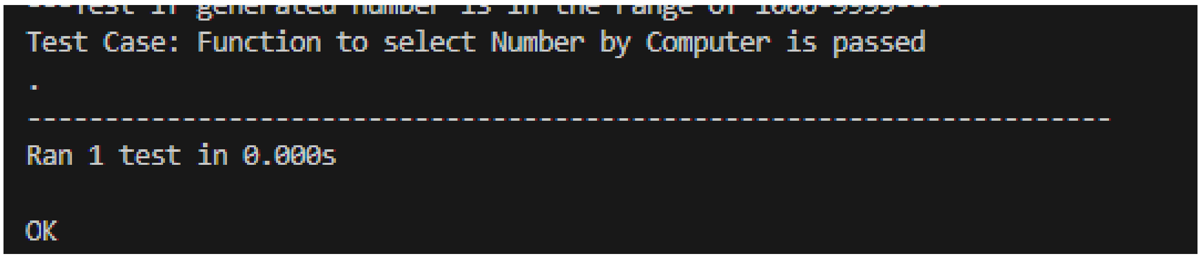
```python
def test_random_number_generation(self):
    # Test if the generated number is within the range of 1000 to 9999
    print("---Test if generated number is in the range of 1000-9999---")
    random_number = self.number_guessing_game.generate_random_number()
    self.assertTrue(1000 <= random_number <= 9999)
    print("Test Case: Function to select Number by Computer is passed")
```

Figure 1: test_random_number_generation test case from testing.py



```python
"""
A class representing the main game logic.

Initiate with a Random generated number and number of attempts
"""
def __init__(self):
    self.secret_number = self.generate_random_number()
    self.attempts = 0

@staticmethod
def generate_random_number():
    """
    Generate a random 4-digit number between 1000 and 9999.
    """
    return random.randint(1000, 9999)
```

Figure 2: Initiation and generate_random_number method in game coding



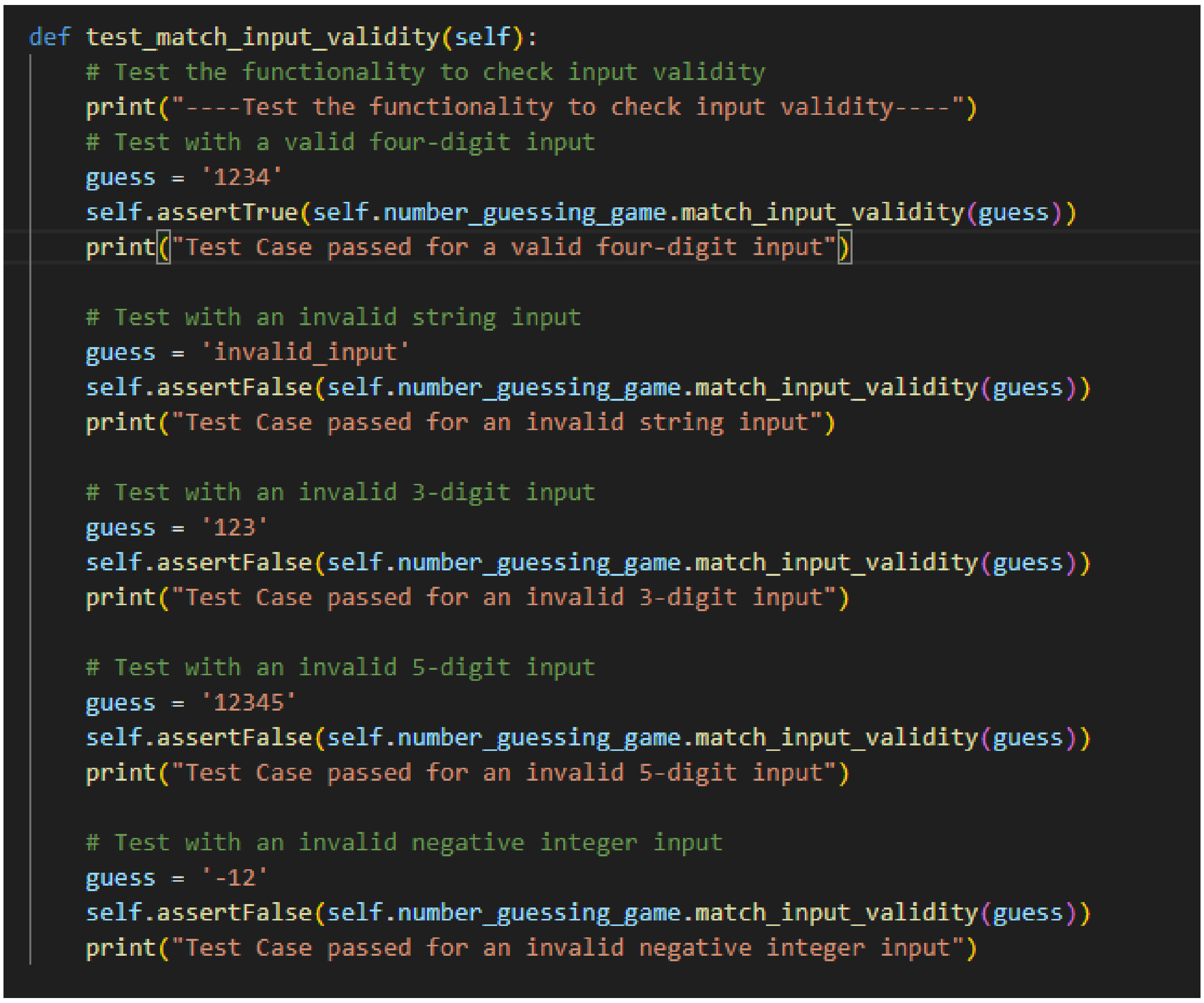
```
---Test if generated number is in the range of 1000-9999---
Test Case: Function to select Number by Computer is passed
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Figure 3: Output of Test Case

Functionality 2 (Check Validity of Input):

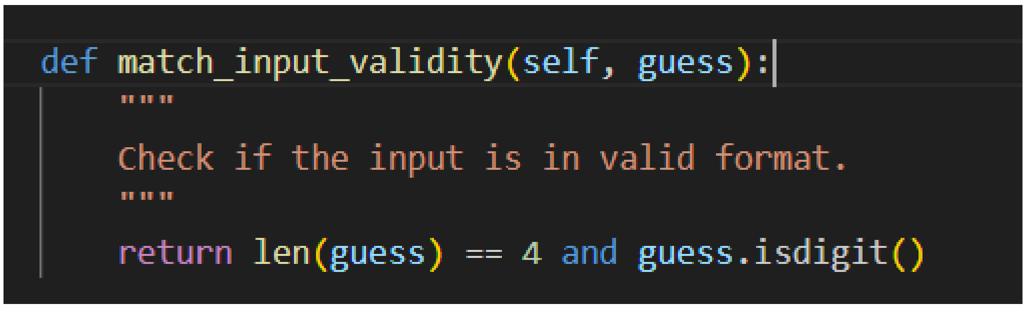
__Test Case Function unit:__ In order to test the correctness of input, a test case function 'test_match_input_validity()' is created. This function ensures whether the user gives a four-digit positive integer as input. It evaluates scenarios to confirm the functions accuracy in distinguishing valid and invalid inputs. The tests include verifying valid inputs, detecting invalid string inputs and to identify whether the input is too short, too long or a negative integer. (Figure 4)

Relevant Game Function Unit: This test case is tested on a function 'match_input_validity' which revolves around a simple check. It examines whether the provided input is exactly four characters in length. If the conditions are met, the input is deemed valid and function returns 'True', else 'False'. (Figure 5)



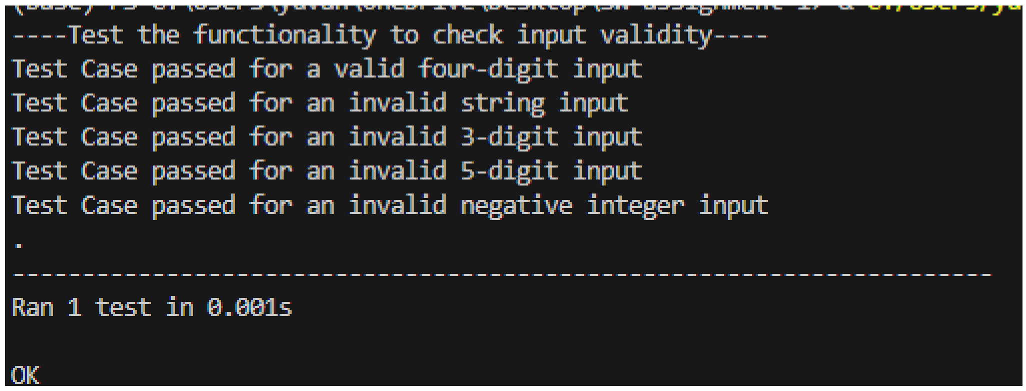
```python
def test_match_input_validity(self):
    # Test the functionality to check input validity
    print("----Test the functionality to check input validity----")
    # Test with a valid four-digit input
    guess = '1234'
    self.assertTrue(self.number_guessing_game.match_input_validity(guess))
    print("Test Case passed for a valid four-digit input")

    # Test with an invalid string input
    guess = 'invalid_input'
    self.assertFalse(self.number_guessing_game.match_input_validity(guess))
    print("Test Case passed for an invalid string input")

    # Test with an invalid 3-digit input
    guess = '123'
    self.assertFalse(self.number_guessing_game.match_input_validity(guess))
    print("Test Case passed for an invalid 3-digit input")

    # Test with an invalid 5-digit input
    guess = '12345'
    self.assertFalse(self.number_guessing_game.match_input_validity(guess))
    print("Test Case passed for an invalid 5-digit input")

    # Test with an invalid negative integer input
    guess = '-12'
    self.assertFalse(self.number_guessing_game.match_input_validity(guess))
    print("Test Case passed for an invalid negative integer input")
```

Figure 4: test_match_input_validity test case from testing.py



```python
def match_input_validity(self, guess):
    """
    Check if the input is in valid format.
    """
    return len(guess) == 4 and guess.isdigit()
```

Figure 5: Input validity check function in-game coding



```
----Test the functionality to check input validity----
Test Case passed for a valid four-digit input
Test Case passed for an invalid string input
Test Case passed for an invalid 3-digit input
Test Case passed for an invalid 5-digit input
Test Case passed for an invalid negative integer input
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

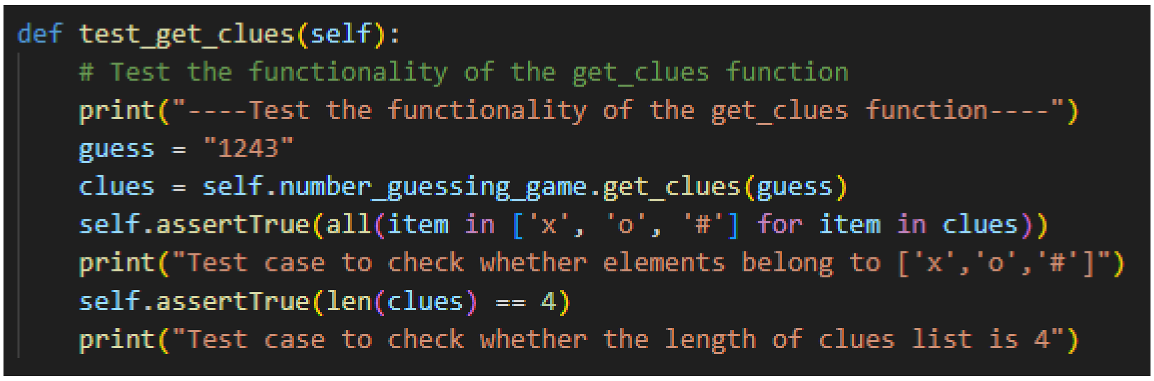
Figure 6: Output of Test Case

Functionality 3 (Generate clues):

This unit test aims to validate function 'get_clues', which generates clues from the guessed number and a secret number. The game involves users giving a 4-digit input.

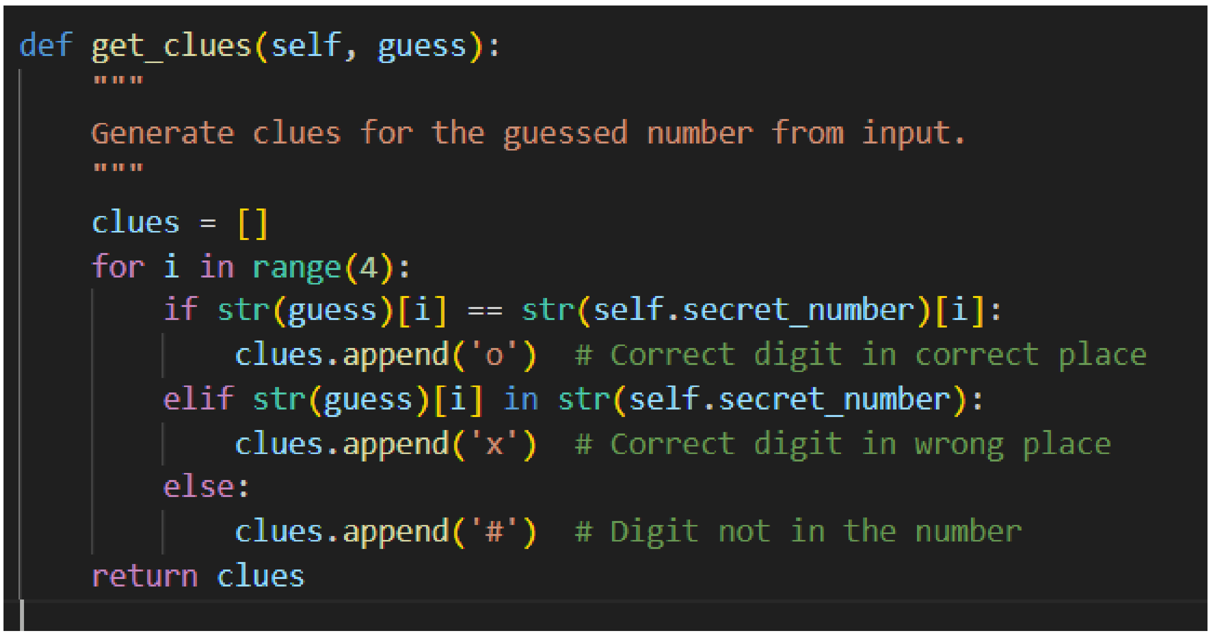
Test Case Function unit: The test method 'test_get_clues' begins by initialising a test guess as "1234". It calls the 'get_clues' function and captures the returned clues in a list datatype. The test ensures that all the list elements are either 'x', 'o' or '#', which is indicated by the assertTrue function. It also checks whether the length of given list is equal to 4, which ensures the appropriate number of clues generated for the given guess. Through these assertions and print statements, the test method ensures whether 'get_clues' function generates accurate and expected clues, and validating a correct implementation.

Relevant Game Function Unit: The 'get_clues' function is an integral part of "Guess the Number" game. When user gives a number as input, the function provides clues to provide feedback to player on the accuracy of their guess. The function iterates through each digit and compares it with respective digit in secret number (generated and validated in functionalities discussed previously). If digits match both value and position, function appends 'o' to the 'clues' list. If digit is present in secret number but at different position, function appends 'x' to the ‘clues’ list. If the digit is not present in the list, function appends ‘#’ to the ‘clues’ list. The resulting ‘clues’ list is printed to user, providing information on their guess.



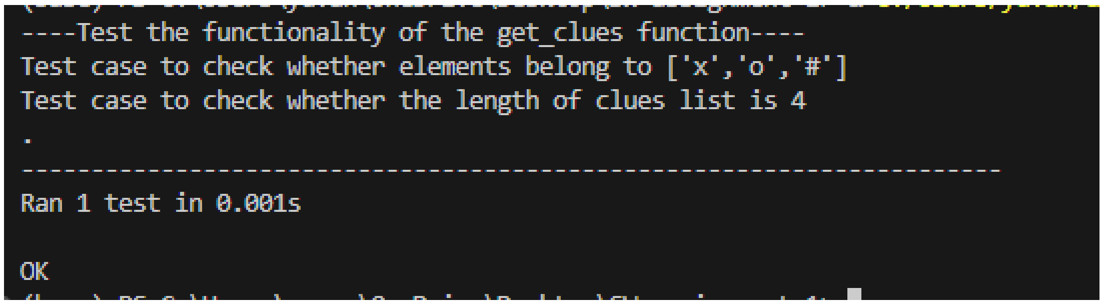
```python
def test_get_clues(self):
    # Test the functionality of the get_clues function
    print("----Test the functionality of the get_clues function----")
    guess = "1243"
    clues = self.number_guessing_game.get_clues(guess)
    self.assertTrue(all(item in ['x', 'o', '#'] for item in clues))
    print("Test case to check whether elements belong to ['x','o','#']")
    self.assertTrue(len(clues) == 4)
    print("Test case to check whether the length of clues list is 4")
```

Figure 7: test_get_clues() function in testing.py



```python
def get_clues(self, guess):
    """
    Generate clues for the guessed number from input.
    """
    clues = []
    for i in range(4):
        if str(guess)[i] == str(self.secret_number)[i]:
            clues.append('o')  # Correct digit in correct place
        elif str(guess)[i] in str(self.secret_number):
            clues.append('x')  # Correct digit in wrong place
        else:
            clues.append('#')  # Digit not in the number
    return clues
```

Figure 8: Function to create clues from user's input in game coding



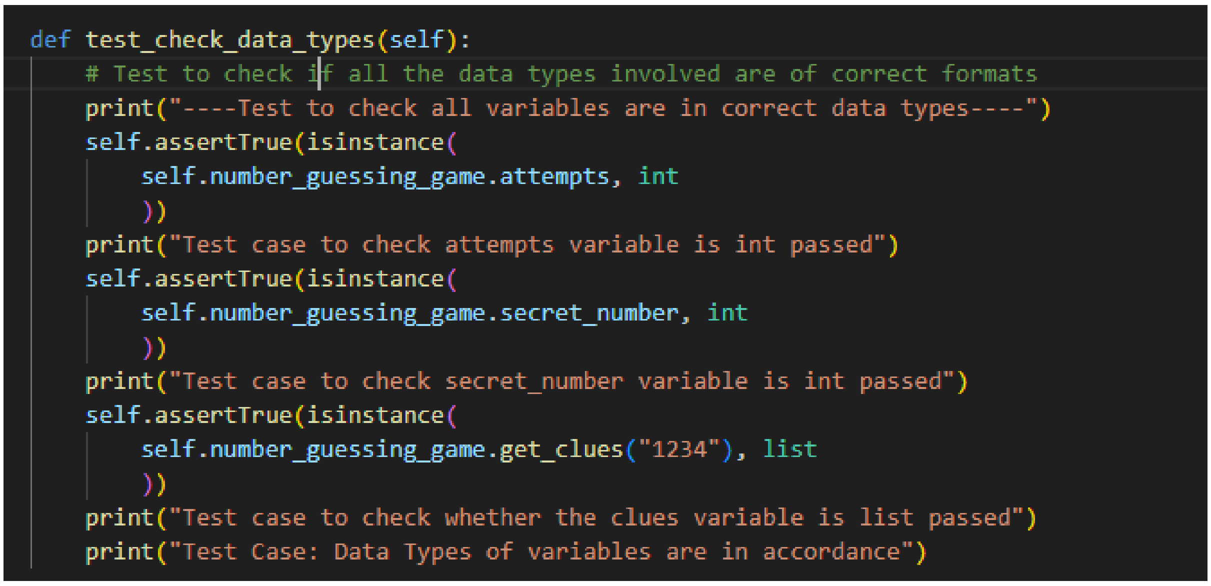
```
----Test the functionality of the get_clues function----
Test case to check whether elements belong to ['x','o','#']
Test case to check whether the length of clues list is 4
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Figure 9: Output of the test code

Functionality 4 (Checking the data type and play() function):

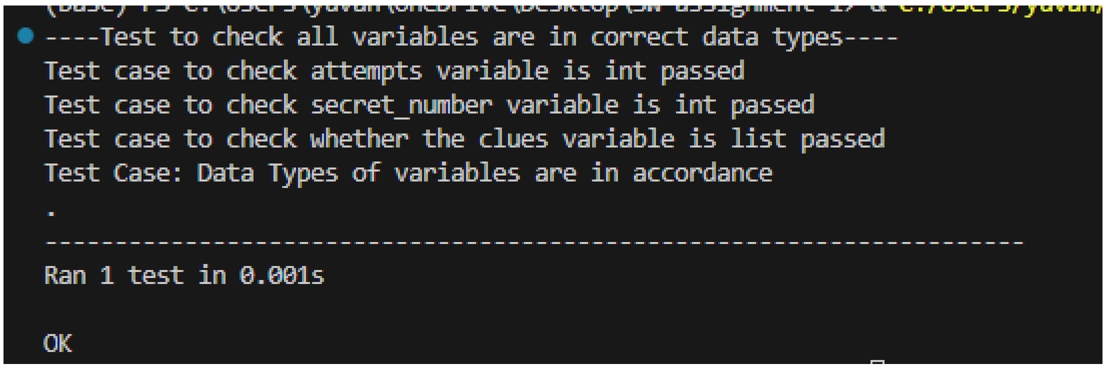
Before passing the variables to the main function ‘play()’ (This main function is responsible for user’s input and relevant output), all the variables need to go through a data type check.

<u>Test Case Function unit:</u> The unit code ‘test_check_data_types’ focuses on ensuring the correctness of data type inside the “Guess the Number” game. It systematically hecks whether the data attributes of the game object, such as ‘attempts’, ‘secret number’ and result of ‘get_clues’ method. By using assertion functions, the code confirms whether the attributes possess intended data types, mainly integres for ‘attempts’ and ‘secret_number’ and lists for ‘get_clues’. Using these validations unit test guarantees data types associated and utilised in ‘play()’ method, to adhere expected formats.



```python
def test_check_data_types(self):
    # Test to check if all the data types involved are of correct formats
    print("----Test to check all variables are in correct data types----")
    self.assertTrue(isinstance(
        self.number_guessing_game.attempts, int
        ))
    print("Test case to check attempts variable is int passed")
    self.assertTrue(isinstance(
        self.number_guessing_game.secret_number, int
        ))
    print("Test case to check secret_number variable is int passed")
    self.assertTrue(isinstance(
        self.number_guessing_game.get_clues("1234"), list
        ))
    print("Test case to check whether the clues variable is list passed")
    print("Test Case: Data Types of variables are in accordance")
```

Figure 10: test_check_data_types function in testing.py

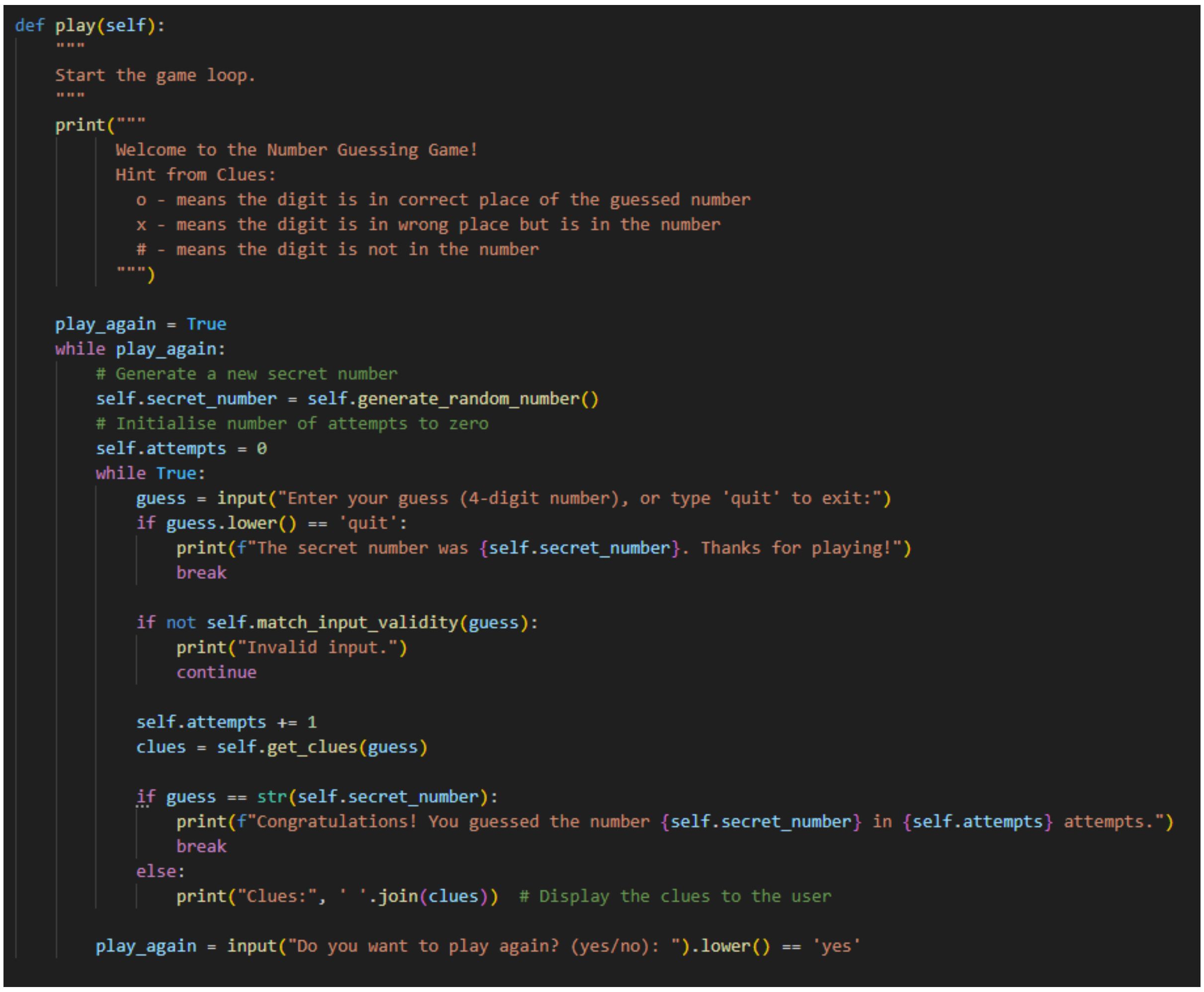


```
----Test to check all variables are in correct data types----
Test case to check attempts variable is int passed
Test case to check secret_number variable is int passed
Test case to check whether the clues variable is list passed
Test Case: Data Types of variables are in accordance
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Figure 11: Output of the test case

<u>Game Function Unit:</u> The ‘play()’ function is responsible for creating a game loop in “Guess the Number” game. On starting, player is introduced to games

mechanics and meaning of clues ((i.e.) ‘o’, ‘x’ and ‘#’). The loop allows the player for repeated gameplay, where whenever restarted, a new random secret number is generated. The code validates the input for correctness and also maintains the count number of attempts. The game provides clues on the correctness of guessed input and their positions. If player guesses the correct number, the game displays the number of attempts to do the guess and asks whether the user wants to replay the game, leading to a fluid game experience.



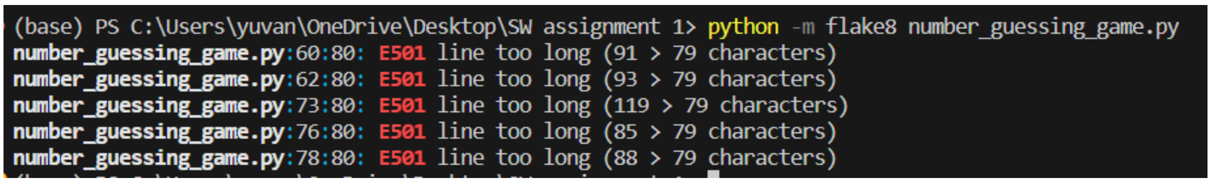
```python
def play(self):
    """
    Start the game loop.
    """
    print("""
          Welcome to the Number Guessing Game!
          Hint from Clues:
            o - means the digit is in correct place of the guessed number
            x - means the digit is in wrong place but is in the number
            # - means the digit is not in the number
          """)

    play_again = True
    while play_again:
        # Generate a new secret number
        self.secret_number = self.generate_random_number()
        # Initialise number of attempts to zero
        self.attempts = 0
        while True:
            guess = input("Enter your guess (4-digit number), or type 'quit' to exit:")
            if guess.lower() == 'quit':
                print(f"The secret number was {self.secret_number}. Thanks for playing!")
                break

            if not self.match_input_validity(guess):
                print("Invalid input.")
                continue

            self.attempts += 1
            clues = self.get_clues(guess)

            if guess == str(self.secret_number):
                print(f"Congratulations! You guessed the number {self.secret_number} in {self.attempts} attempts.")
                break
            else:
                print("Clues:", ' '.join(clues))  # Display the clues to the user

        play_again = input("Do you want to play again? (yes/no): ").lower() == 'yes'
```

Figure 12: Game Function play()

Quality of code:

The quality of code is tested out using Flake8 and Pylint. The output of these modules are shown below and the description are written as required.

Flake8 Quality Check on “number_guessing_game.py”:

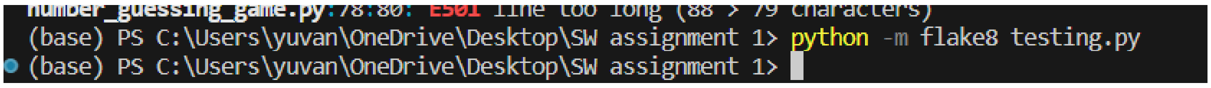


```
(base) PS C:\Users\yuvan\OneDrive\Desktop\SW assignment 1> python -m flake8 number_guessing_game.py
number_guessing_game.py:60:80: E501 line too long (91 > 79 characters)
number_guessing_game.py:62:80: E501 line too long (93 > 79 characters)
number_guessing_game.py:73:80: E501 line too long (119 > 79 characters)
number_guessing_game.py:76:80: E501 line too long (85 > 79 characters)
number_guessing_game.py:78:80: E501 line too long (88 > 79 characters)
```

Figure 13: Flake8 Quality Check on “number_guessing_game.py"

The lines described here long are print statements. As these print statements are essential, these discrepancies are not solved in code.

Flake8 Quality Check on “testing.py”:

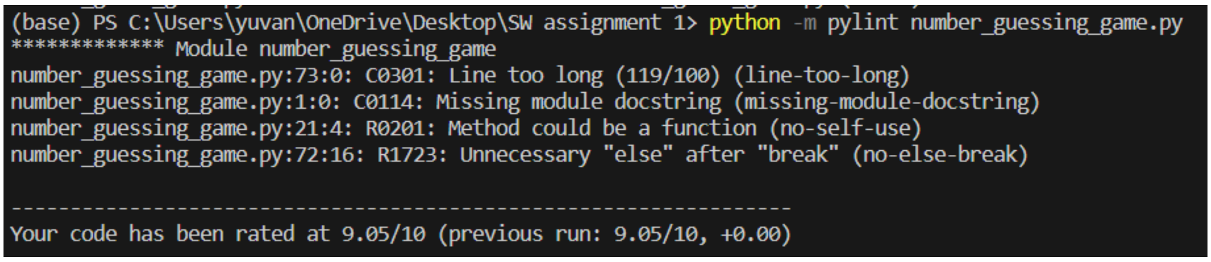


```
 number_guessing_game.py:78:80: E501 line too long (88 > 79 characters)
 (base) PS C:\Users\yuvan\OneDrive\Desktop\SW assignment 1> python -m flake8 testing.py
 (base) PS C:\Users\yuvan\OneDrive\Desktop\SW assignment 1>
```

Figure 14: Flake8 Quality Check on “testing.py”

‘testing.py’ had no discrepancies arised due to perfect formatting of code.

Pylint Quality Check on “number_guessing_game.py”:

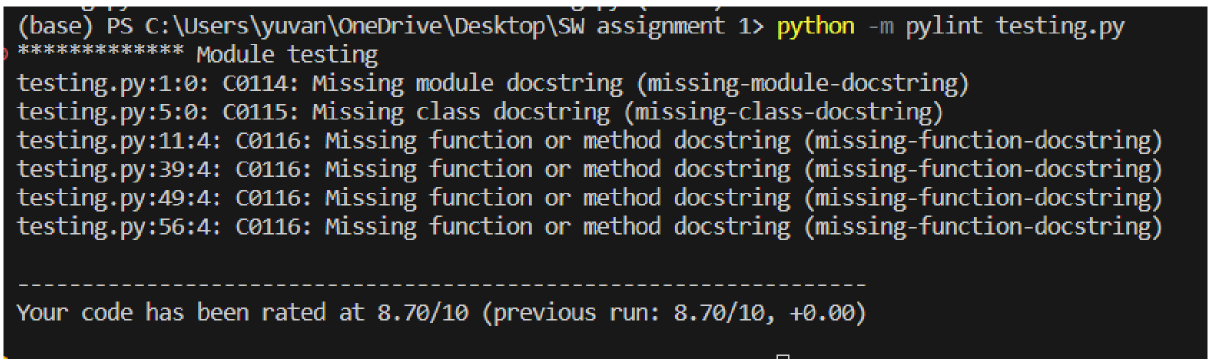


```
(base) PS C:\Users\yuvan\OneDrive\Desktop\SW assignment 1> python -m pylint number_guessing_game.py
************* Module number_guessing_game
number_guessing_game.py:73:0: C0301: Line too long (119/100) (line-too-long)
number_guessing_game.py:1:0: C0114: Missing module docstring (missing-module-docstring)
number_guessing_game.py:21:4: R0201: Method could be a function (no-self-use)
number_guessing_game.py:72:16: R1723: Unnecessary "else" after "break" (no-else-break)

------------------------------------------------------------------
Your code has been rated at 9.05/10 (previous run: 9.05/10, +0.00)
```

Figure 15: Pylint Quality check on "number_guessing_game.py"

“number_guessing_game.py” has been rated 9.05/10 which is perfect and accurate according to pylint. The comments arose due to longer lines for print statements and unnecessary else statements after break command. The comments also suggest to turn a function into static method to reduce memory usage.

Pylint Quality Check on “testing.py”:



```
(base) PS C:\Users\yuvan\OneDrive\Desktop\SW assignment 1> python -m pylint testing.py
************* Module testing
testing.py:1:0: C0114: Missing module docstring (missing-module-docstring)
testing.py:5:0: C0115: Missing class docstring (missing-class-docstring)
testing.py:11:4: C0116: Missing function or method docstring (missing-function-docstring)
testing.py:39:4: C0116: Missing function or method docstring (missing-function-docstring)
testing.py:49:4: C0116: Missing function or method docstring (missing-function-docstring)
testing.py:56:4: C0116: Missing function or method docstring (missing-function-docstring)

------------------------------------------------------------------
Your code has been rated at 8.70/10 (previous run: 8.70/10, +0.00)
```

Figure 16: Pylint Quality Check on “testing.py”

“testing.py” has been rated 8.70/10 which is accurate. The comments are about lacking in module-level docstring, which could provide a brief overview on module’s purpose, which is beyond the scope of the intended task and project.

Conclusion:

In summary, the task aims to create a game “Guess the Number” using TDD approach for a functional game and automated testing (Unit testing). Key functionalities are implemented, such as random number generation, user input validation and clue generation methods. Automated testing is implemented using ‘unittest’ in Python to ensure the accuracy of game. Furthermore, coding quality is checked using Flake8 and Pylint.

Overall, this project emphasises value of automated testing in a real-world environment to create out softwares in a much efficient methods using automated testing.

Github Link of my Project:

https://github.com/7uv4n/prt582_unit_testing_report_s364626

Table of Figures