SOFTWARE UNIT TESTING REPORT

Guess the Number game by using Test Driven Development (TDD) in Python Programming Language

AUGUST 25, 2023

STUDENT NAME: AMAIR AHMED
STUDENT ID: S366847
CAMPUS: CASUARINA

Table of Contents:

1. INTRODUCTION:

The main objective of the task is to develop GUESS THE NUMBER game in python programming language by using automate unit test environment. participants try to figure out a secret 4-digit number using the supplied code. The game allows users to guess, and it offers tips to assist them restrict their options after each guess. Until the user correctly guesses the secret number or decides to give up, the game cycle continues. Users have the choice to continue playing or leave each game once it has ended. Furthermore, participants enter their own 4-digit predictions once the game starts with a randomly generated 4-digit secret number. The suggestions are then given based on how the predicted digits are positioned in relation to the digits of the secret number. An 'O' is shown if a guesses digit is in the right spot. An "x" is displayed if a predicted digit is accurate but in the wrong location. A '_' is shown if the digit is absent from the secret number.

By entering "quit," users can exit the game at any moment and see their secret number and the amount of attempts they made. When the user correctly guesses the secret number, the script verifies that the input is correct, keeps track of how many times they have tried, and displays congratulatory messages. Until the user chooses not to play again, the game is played endlessly.

Demonstrating the phrase automated testing, which covers the automatic testing of the application, in which the developer will construct a unique and different script for the testing file using various automated testing tools. Following the creation of the testing file, the programmed file will automatically test the application and display the results. This is the basic principle of an automated testing tool.

In this project, I am using unit testing, which will automatically test the functioning of the game's key functionalities by inserting pass/fail situations into the test cases. Consider the game to be the unit of code. The core capability of unit testing is to test that unit of code using various test cases, analysing their behaviour and results.

We need to import the Unit test package module in testing file, which is already in the standard library, to do unit testing in Python. We don't need to install any other modules. Each of the test cases has been expanded further with appropriate screenshots.

2. Process:

Guess number game has been developed using the Test-Driven Development (TDD) in python programming language by using the following steps and conditions.

I have written the different functions for the major functionality of the game in the guessing_game.py file which is the major game coding file and I have written the test cases of these major functionalities into the other unit testing file called as test_guessing_game.py. test_guessing_game.py file will automatically test the behaviour and functionality of the game as I have written different test cases with different inputs and also provided different scenario.

2.1 Functionality 1: (Guess number & How many attempts for guessing secret number)

A user will attempt to guess a four-digit secret number. The computer provides you with tips to assist you in determining the number each time you guess. How many attempts you make are recorded by the game. You can type "quit" to end the game, and the computer will then reveal the secret number. The game notifies you of your victory and the number of attempts it required if you correctly predict the number. Overall, this game works by producing a secret number, asking your guesses, providing suggestions, and telling you of your results or desire to give up. simultaneously, the testing file will also verify the functionality of the match input function by providing the different input options through the testing file and check the response of the function automatically through coding. Here are the screen shots of both game and testing file.

```python
# Function to play the guessing game
def play_game():
    # Generate a random secret number for the current game
    secret_number = generate_random_number()

    # Initialize the number of attempts made by the player
    attempts = 0

    # Start a loop for each game session
    while True:
        # Get the player's guess as input
        guess = input("Guess the four-digit number: ")

        # Check if the player wants to quit the game
        if guess == 'quit':
            # Display the secret number and the number of attempts made
            print(f"The secret number was {secret_number}.")
            if attempts == 0:
                print("You took 0 attempts.")
            else:
                print(f"You took {attempts} attempts.")
            break
```

Figure 1: Guess number functionality from coding game

```python
# Define a test class that inherits from unittest.TestCase
class TestGuessingGame(unittest.TestCase):

    # Test case for invalid inputs (non-digit and incorrect length)
    @patch('builtins.input', side_effect=['123a', 'abcd', '12345', '1234', 'quit'])
    def test_invalid_input(self, mock_input):
        self.assertEqual(guessing_game.play_game(), None)
```

Figure 1.1: Guess number functionality from test_guessing_game.py file

```python

    # Test case for checking the displayed number of attempts
    @patch('builtins.input', side_effect=['1234', '5678', 'quit'])
    def test_attempts(self, mock_input):
        with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
            guessing_game.play_game()
            self.assertIn("You took", mock_stdout.getvalue())

```

Figure 1.2: Attempts user took functionality from test_guessing_game.py file

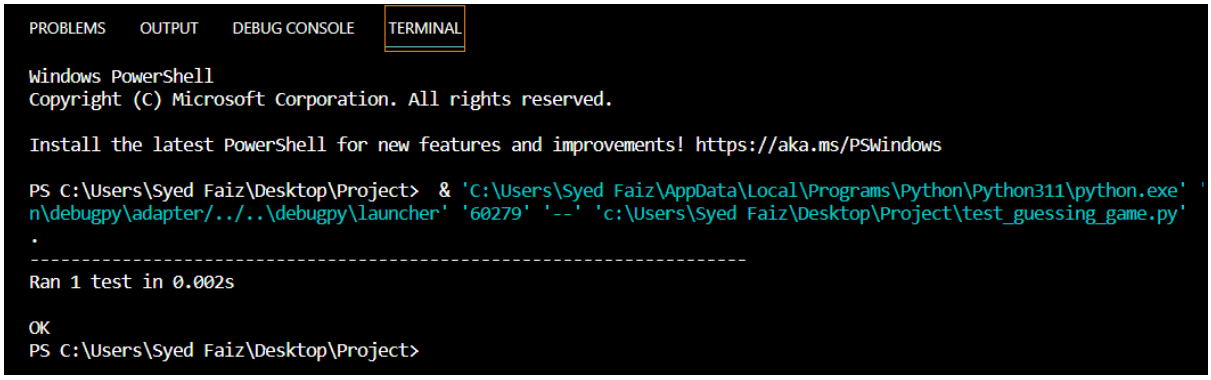


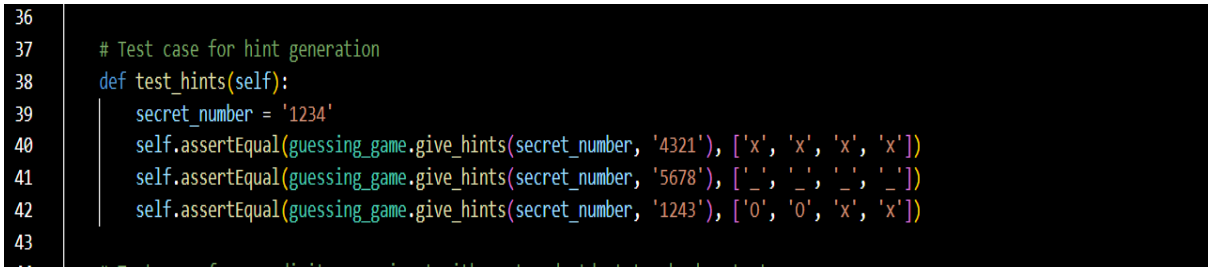
Figure 1.4: Test case passed for the functionality 1.

2.2 Functionality 2: (Provide hints to user)

After user written the guess number afterwards the game will provide the hints for the user to identify which position of the digit is correct. In which if the position is correct the game will give an indicator like ‘O’ that means the position of the digit is correct. On the other hand, side if the user enters the incorrect digit, then game will provide a indicator named as ‘X’ which means that the number is incorrect digit. Additionally, if the digit is absence, then it will give the user as ‘_’ that means empty value. I have written the test case as well for this functionality that will clearly provide the hint to the user that user can change the value accordingly until the whole digit come correct. Here are the screen shots from both the files.

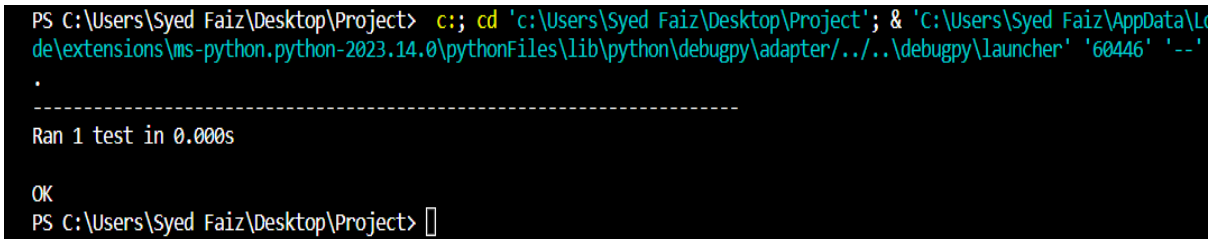
```
7
8   # Function to provide hints based on the secret number and the player's guess
9   def give_hints(secret_number, guess):
10      # Initialize an empty list to store hints
11      hints = []
12
13      # Iterate through each digit's position
14      for i in range(len(secret_number)):
15          # Compare the digit at the current position in secret_number and guess
16          if secret_number[i] == guess[i]:
17              hints.append('O')  # Add 'O' if the digit and position match
18          elif guess[i] in secret_number:
19              hints.append('x')  # Add 'x' if the digit is correct but in the wrong position
20          else:
21              hints.append('_')  # Add '_' if the digit is not present in the secret number
22      return hints
23
```

Figure 2: Game function of random input from the user



```
36
37      # Test case for hint generation
38      def test_hints(self):
39          secret_number = '1234'
40          self.assertEqual(guessing_game.give_hints(secret_number, '4321'), ['x', 'x', 'x', 'x'])
41          self.assertEqual(guessing_game.give_hints(secret_number, '5678'), ['_', '_', '_', '_'])
42          self.assertEqual(guessing_game.give_hints(secret_number, '1243'), ['O', 'O', 'x', 'x'])
43
```

Figure 2.1: test case to verify the random input from the user



```
PS C:\Users\Syed Faiz\Desktop\Project>  c:; cd 'c:\Users\Syed Faiz\Desktop\Project'; & 'C:\Users\Syed Faiz\AppData\Lo
de\extensions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '60446' '--'
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Syed Faiz\Desktop\Project>
```

Figure 2.2: Output of the testing case

2.2 Functionality 3: (Display the success message and ask the user to play again or not)

After successful attempts of guessing the secret number, the game will provide a congratulatory message and game will ask the user to whether user wants to "Do you want to play again? (yes/no): “. If user write down ‘yes’ then the game will restart again with the same procedure asking for guessing the secret number and provide the hints accordingly to the user entered secret number after that it will shows up the congratulatory message and asked them to "Do you want to play again? (yes/no): " this will continuously repeated when the user says ‘no’ to play again. Moreover, after selecting ‘no’ the user will get the message such as ‘Thanks for playing the game’. Here are the screenshots from both the files.

```
57
58          # Check if the player's guess matches the secret number
59          if guess == secret_number:
60              # Display a congratulatory message with the secret number and attempts made
61              print(f"Congratulations! You guessed the number {secret_number} in {attempts} attempts.")
62              break
63          else:
64              # Display the hints provided for the player's guess
65              print("Hints:", ' '.join(hints))
66
67  # Function to manage the entire game
68  def main():
69      # Start an infinite loop for playing multiple game sessions
70      while True:
71          # Call the play_game function to play a single game
72          play_game()
73
74          # Ask the player if they want to play again
75          play_again = input("Do you want to play again? (yes/no): ")
76
77          # Check if the player wants to end the game
78          if play_again.lower() != 'yes':
79              print("Thanks for playing!")
80              break
```

Figure 3: functionality for game play again or not

```
14
15      # Test case for invalid inputs with captured stdout to check output
16      @patch('builtins.input', side_effect=['4321', 'quit'])
17      def test_invalid_input_output(self, mock_input):
18          with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
19              guessing_game.play_game()
20              print(mock_stdout.getvalue())
21              self.assertNotIn("Congratulations! You guessed the number", mock_stdout.getvalue())
22
```

Figure 3.1 Test for the functionality from test_guessing_game.py file

```
Guess the four-digit number: 6542
Hints: _ x x _
Guess the four-digit number: 0452
Hints: _ x x _
Guess the four-digit number: 3987
Hints: x _ O _
Guess the four-digit number: 2189
Hints: _ _ O _
Guess the four-digit number: 5689
Hints: O _ O _
Guess the four-digit number: 5080
Hints: O _ O _
Guess the four-digit number: 5683
Hints: O _ O x
Guess the four-digit number: 5484
Hints: O x O O
Guess the four-digit number: 5084
Hints: O _ O O
Guess the four-digit number: 5184
Hints: O _ O O
Guess the four-digit number: 5284
Hints: O _ O O
Guess the four-digit number: 5384
Congratulations! You guessed the number 5384 in 22 attempts.
Do you want to play again? (yes/no): no
Thanks for playing!
PS C:\Users\Syed Faiz\Desktop\Project>
```

Figure 3.2 Output of the success message &play again functionality

2.4 Functionality4: (Validate the input for a 4-digit number)

While guessing the secret the user might enters the more digits instead of four digits then game will provide an error message to the user says that Invalid input. Please enter a four-digit number." Afterwards, the game will continue will there itself unless and until user doesn't provide the 4 digits number. By having the this restricted the user will get know that they are doing mistake while playing the game. This will be beneficial for the user and improve the intangibility for the game simultaneously. Here the below screenshots from both files.

```
        # Validate the input for a 4-digit number
        if len(guess) != 4 or not guess.isdigit():
            print("Invalid input. Please enter a four-digit number.")
            continue
```

Figure 4: Invalid input from the user

```
@patch('builtins.input', side_effect=['abcd', '1a34', 'quit'])
def test_non_digit_guess(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("Invalid input. Please enter a four-digit number.", mock_stdout.getvalue())
```

Figure 4.1: Test case for the Invalid Input

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Syed Faiz\Desktop\Project> & 'C:\Users\Syed Faiz\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Syed
n\debugpy\adapter/../..\debugpy\launcher' '60798' '--' 'c:\Users\Syed Faiz\Desktop\Project\guessing_game.py'
Guess the four-digit number: 1234
Hints: _ _ _ _
Guess the four-digit number: 13456
Invalid input. Please enter a four-digit number.
Guess the four-digit number:
```

Figure 4.2: Output for the more than 4 digits entered by the user

2.5 Functionality 5: (Quit the Game)

This functionality is very useful for the user if the user doesn't want to play the game and they no longer to interested to play they can quit the game anytime during the whole game. The user need s to just type 'quit' the game will directly quits and display the message says that 'The secret number was 'whatever the number is' and it also says that how many attempts it will take 'You took 1 attempt. Moreover, it will ask the user played again or not. If user say 'yes' then the game will continue as usually and if 'no' it will game ends.

```
36
37          # Check if the player wants to quit the game
38          if guess == 'quit':
39              # Display the secret number and the number of attempts made
40              print(f"The secret number was {secret_number}.")
41              if attempts == 0:
42                  print("You took 0 attempts.")
43              else:
44                  print(f"You took {attempts} attempts.")
45              break
46
```

Figure 5: Functionality for quitting the game

```
# Test case for quitting without guessing the number
@patch('builtins.input', side_effect=['1234', '5678', 'quit'])
def test_quit_guess(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("The secret number was", mock_stdout.getvalue())

# Test case for checking the displayed number of attempts
@patch('builtins.input', side_effect=['1234', '5678', 'quit'])
def test_attempts(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("You took", mock_stdout.getvalue())
```

Figure 5.1: Test case for the quitting the game.

```
Guess the four-digit number: quit
The secret number was 6888.
You took 1 attempts.
Do you want to play again? (yes/no): nno
Thanks for playing!
PS C:\Users\Syed Faiz\Desktop\Project>
```

Figure 5.2: Output for quitting the game

2.6 Functionality 6: (Test case for Invalid Length guess, Invalid short number, non-digit guess)

This functionality is very efficient for the users who are making mistakes every time here and then. It will provide the number the error according the input such as if the user enters the invalid length guess input which means entering more digits to the standard 4 digits, then it will provide a message says that "Invalid input. Please enter a four-digit number." Similarly, if the user hits alphabetical letter as an input it will shows the same error and same for the short number guess.

```python
# Test case for non-digit guess input with captured stdout to check output
@patch('builtins.input', side_effect=['abcd', '1a34', 'quit'])
def test_non_digit_guess(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("Invalid input. Please enter a four-digit number.", mock_stdout.getvalue())

# Test case for invalid length guess input with captured stdout to check output
@patch('builtins.input', side_effect=['12', '12345', 'quit'])
def test_invalid_length_guess(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("Invalid input. Please enter a four-digit number.", mock_stdout.getvalue())

# Test case for a short number guess input with captured stdout to check output
@patch('builtins.input', side_effect=['123', 'quit'])
def test_short_number(self, mock_input):
    with patch('sys.stdout', new_callable=io.StringIO) as mock_stdout:
        guessing_game.play_game()
        self.assertIn("Invalid input. Please enter a four-digit number.", mock_stdout.getvalue())
```

Figure 6: Test case for the above functionalities

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Syed Faiz\Desktop\Project>  & 'C:\Users\Syed Faiz\AppData\Local\Programs\Python\Python311\python.exe' 'c:\User
n\debugpy\adapter/../..\debugpy\launcher' '61382' '--' 'c:\Users\Syed Faiz\Desktop\Project\guessing_game.py'
Welcome to the Guess Number Game!
Guess the four-digit number: or write quit for quitting the game: abcd
Invalid input. Please enter a four-digit number.
Guess the four-digit number: or write quit for quitting the game: 12345
Invalid input. Please enter a four-digit number.
Guess the four-digit number: or write quit for quitting the game: 123
Invalid input. Please enter a four-digit number.
```

Figure 6.1: Output of a restrictions for invalid inputs from the user

3. Conclusion:

Through this project, I successfully made a number guessing game that follows the rules we set. I used something called Test-Driven Development (TDD) and a tool called "unittest" to test the game automatically. This helped me make sure that the game works well and does what it should. TDD also helped me find and fix problems early. This project taught me the importance of testing and how it makes software strong and reliable.

This game also taught me about making things step by step using TDD. First, I tested my ideas, then I built the game. This way, I caught problems early and made a strong foundation for the game. As I worked on the game, I realized that TDD is difficult then I started doing one piece at a time then it made me feel more confident about each step I took. Even when there were challenges, like figuring out how to handle player guesses and give hints, TDD guided me in the right direction. It showed me that improving the game bit by bit and testing along the way is a smart way to build things. This project gave me a idea of how to write a code in efficient manner.

Please find the below GitHub link for my project:
https://github.com/ahmedau024/Software-Unit-Testing