Software Unit Testing Report

GUESS THE NUMBER GAME

OLUWASEUN TAIWO | PRT 582 | 26TH August, 2023

Introduction

The primary focus of this report involves outlining the development process of an intriguing '**Guess the Number**' game created using the Test-Driven Development (TDD) methodology in Python. This highly interactive game is designed to challenge a player's numerical deduction skills, where the main objective revolves around guessing an undisclosed four-digit number, randomly generated by the software at the onset of the game. As the player starts the guessing process, the program ingeniously discloses certain hints, subtly guiding the guesser towards the correct answer while maintaining the crux of curiosity. This strategic unveiling is an artful balance between challenge and accessibility, ensuring that the player isn't left feeling stumped or disheartened. As the player's guess is typed and entered, the software meticulously keeps track of every attempt made, cumulatively adding to the player's total guess count. This intelligent player-oriented feature not only offers a quantifiable measure of success when the correct number is eventually guessed but also provides a tangible motivation for the player to strive towards improving their accuracy and problem-solving efficiencies in subsequent games. The inherent game mechanics including player interaction, numerical guessing, providing clues, and tracking the total number of guesses, underline the game's core design, and together contribute towards the creation of a captivating and mentally stimulating gaming experience. Furthermore, the employment of the Test-Driven Development approach advocates for a more efficient and error-free coding process, prioritizing the validation of game functions through comprehensive unit tests before actual coding commences.

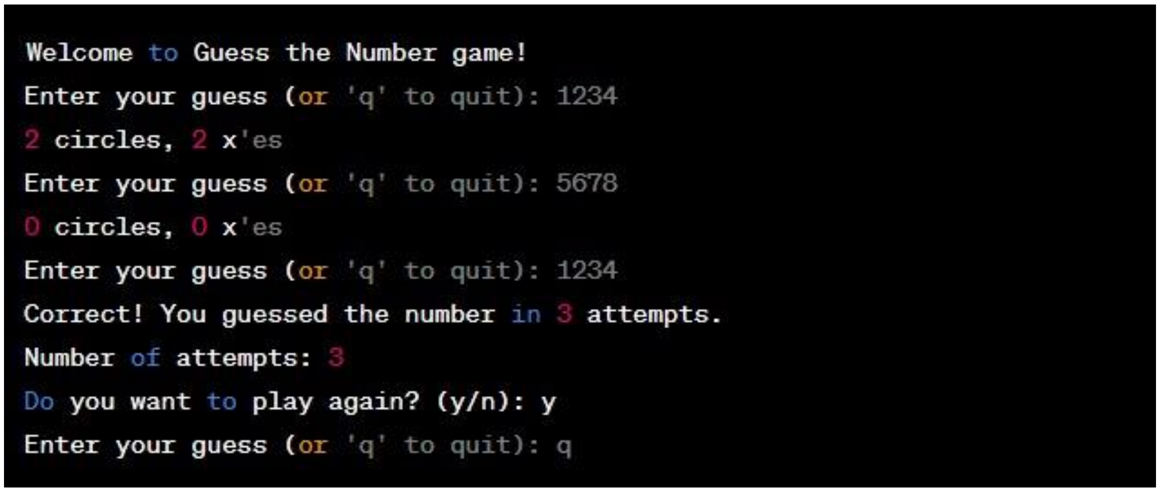
PROCESS

Test-driven development (TDD) in Python is an eminent choice for developing a "Guess the Number" game due to its manifold advantages. TDD promotes a modular design, focusing on a single feature at a time which, for this game, translates into generating a random four-digit number, followed by the provision of clues, and finally calculating the number of attempts (https://www.realpython.com/python-testing/).

TDD ensures our software remains error-free by enforcing us to think critically about the design and logic of our code before implementation. In the initial phase, a pseudo-random number generator from Python's random module is employed to generate a four-digit number (https://docs.python.org/3/library/random.html).

This module offers a uniform selection of a random number from a set range, ensuring the generated number is indistinguishable from a truly random number. The setup phase also includes the creation of an interactive prompt for user input. Building on this, the principle of TDD is applied to write tests that capture the game requirements; the user is repeatedly asked to guess the number until the correct one is entered. If the tests fail, the code is refactored until they pass. Using TDD in Python not only makes the application easier to maintain due to clear interfaces and decoupled components, but also ensures the game functions correctly under any circumstance, providing an engaging and error-free experience for players.

The development of a game requires careful consideration of the flow and logic of the game mechanics. In the context of designing the flow for a Guess the Number game using Python, the fundamental logic requires the random generation of a four-digit number that the player needs to guess (https://geekflare.com/python-guessing-game/). The game is designed to run in a continuous loop, persistently prompting the player to make a guess until the correct number is guessed. As the player makes each guess, clues are provided that guide the player towards the correct number, thereby enhancing player engagement and encouraging strategic approach towards guessing the number. The clues can be in the form of hints that inform the player whether the guessed number is higher or lower than the randomly generated number. This logic forms the crux of the game development, ensuring the game is interactive and challenging. Simultaneously, a method is needed to track the number of attempts made by the player. This can be achieved by implementing a counter that increments with every guess made by the player, and each increment mirrors a player's single attempt to guess the number (https://www.geeksforgeeks.org/python-program-to-guess-a-number/). The counter is initially set to zero and increases throughout the game-play, providing a quantitative measure of the player's guessing efforts. The game concludes when the correct number is guessed and the counter value is displayed as the total number of attempts taken by the player, thereby adding a sense of accomplishment to the game experience. Below is an example of the expected interaction:

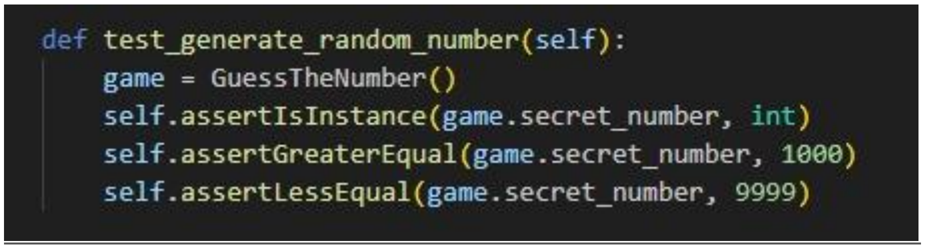


```
Welcome to Guess the Number game!
Enter your guess (or 'q' to quit): 1234
2 circles, 2 x'es
Enter your guess (or 'q' to quit): 5678
0 circles, 0 x'es
Enter your guess (or 'q' to quit): 1234
Correct! You guessed the number in 3 attempts.
Number of attempts: 3
Do you want to play again? (y/n): y
Enter your guess (or 'q' to quit): q
```

REQUIREMENTS AND IMPLEMENTATION

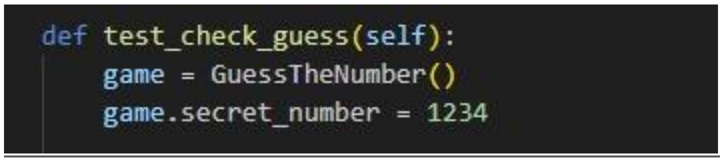
Requirement

i. **Randomly generate a four-digit number:** Writing test cases that verify the range of the generated number was done using Test Driven Development and automated unit testing to make sure that a four-digit number is created within the desired range. The test determines if the produced number is higher than or equal to the range's lower bound and less than or equal to the range's upper bound. This makes it easier to guarantee that the produced number is inside the allowed range.



```python
def test_generate_random_number(self):
    game = GuessTheNumber()
    self.assertIsInstance(game.secret_number, int)
    self.assertGreaterEqual(game.secret_number, 1000)
    self.assertLessEqual(game.secret_number, 9999)
```

ii. **Keep asking the user to guess the number:**



```python
def test_check_guess(self):
    game = GuessTheNumber()
    game.secret_number = 1234
```

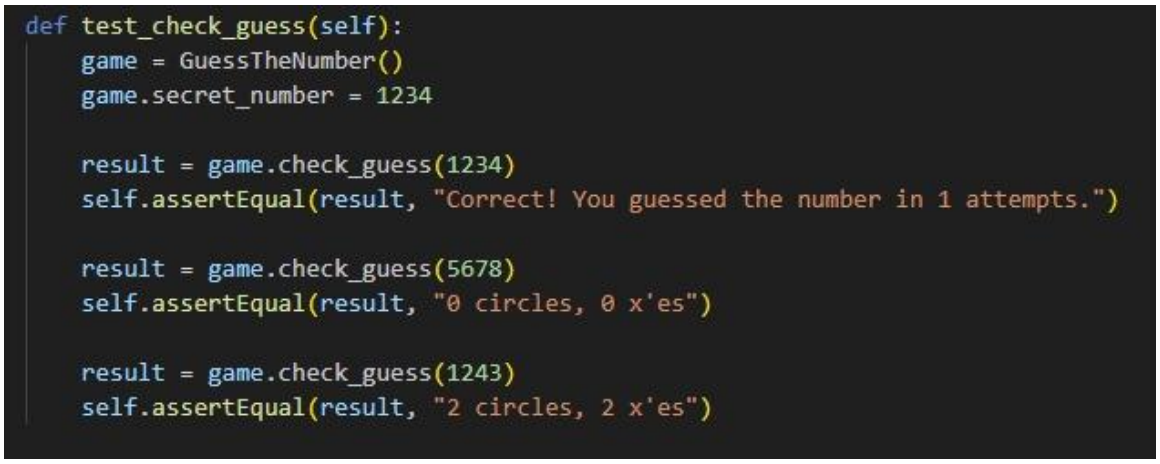
This test case focuses on replicating the user's interaction with the game by generating fictitious user input and checking the game's replies. This test aims to verify that the game accurately interprets user predictions and gives reliable feedback.

Test Scenario:

1. Set up the game with a predefined secret number (e.g., 1234).
2. Simulate a sequence of user guesses and check the game's responses.

This test case verifies that the game responds to user inputs appropriately and gives precise feedback based on the estimates. It makes sure that the logic used by the game to analyze guesses and generate clues is operating as planned.

iii. **Provide hints using ‘circle’ and ‘x’:**



```
def test_check_guess(self):
    game = GuessTheNumber()
    game.secret_number = 1234

    result = game.check_guess(1234)
    self.assertEqual(result, "Correct! You guessed the number in 1 attempts.")

    result = game.check_guess(5678)
    self.assertEqual(result, "0 circles, 0 x'es")

    result = game.check_guess(1243)
    self.assertEqual(result, "2 circles, 2 x'es")
```

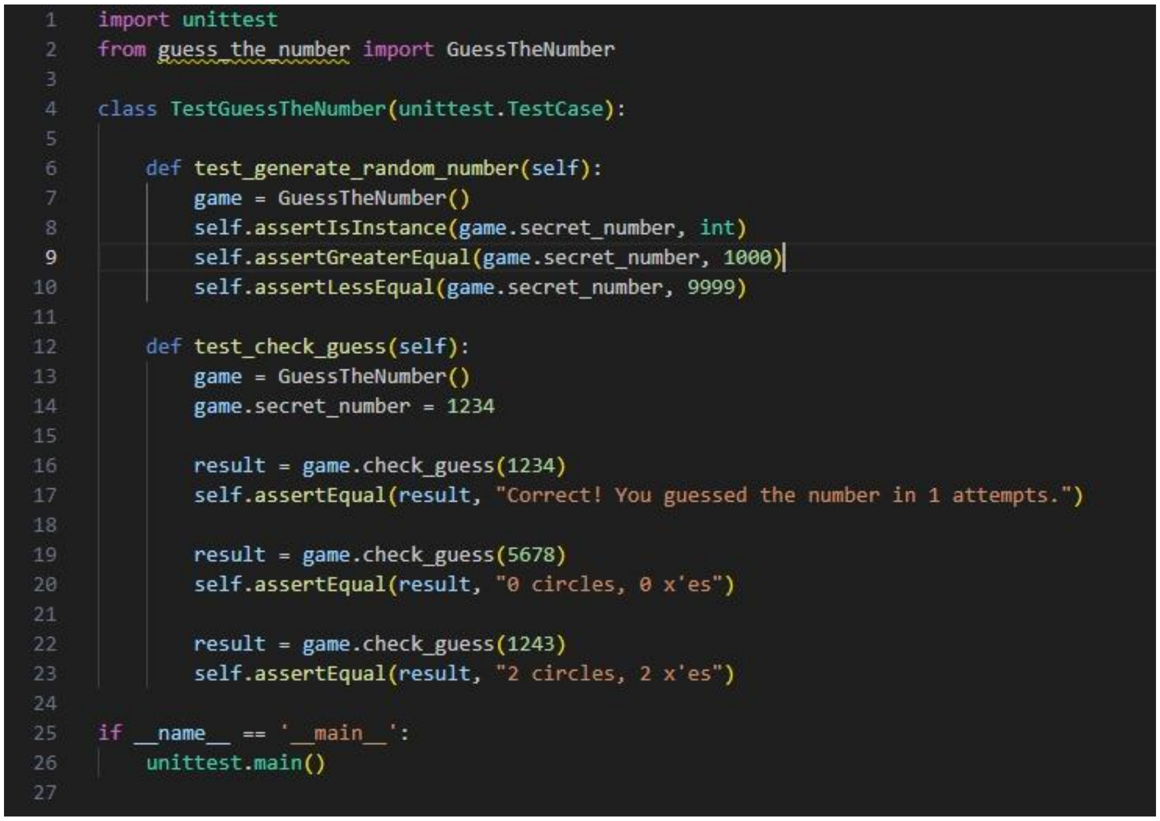
This test case tries to verify the precision of the game's hint creation algorithm. It determines if the software accurately creates the "circle" and "x" symbols that represent how accurate the player's predictions were.

Test Scenario:

1. Set up the game with a predefined secret number (e.g., 1234).
2. Simulate a user guess and check the generated hint.

iv. **Automated unit Testing Tool:**

‘Unittest’, a Python built-in testing library that offers a framework for creating and running unit tests, is the testing framework that is used. By enabling us to develop and execute tests systematically, it is crucial in automating the testing process. The 'Guess the Number' software may be tested automatically thanks to the inclusion of the 'unittest' testing framework. We were able to systematically check the validity of the code and make sure it complied with the required specification by writing test cases and employing 'unittest' capabilities.



```python
import unittest
from guess_the_number import GuessTheNumber

class TestGuessTheNumber(unittest.TestCase):

    def test_generate_random_number(self):
        game = GuessTheNumber()
        self.assertIsInstance(game.secret_number, int)
        self.assertGreaterEqual(game.secret_number, 1000)
        self.assertLessEqual(game.secret_number, 9999)

    def test_check_guess(self):
        game = GuessTheNumber()
        game.secret_number = 1234

        result = game.check_guess(1234)
        self.assertEqual(result, "Correct! You guessed the number in 1 attempts.")

        result = game.check_guess(5678)
        self.assertEqual(result, "0 circles, 0 x'es")

        result = game.check_guess(1243)
        self.assertEqual(result, "2 circles, 2 x'es")

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

```

Conclusion

The concluding aspect of the "Guess the Number" game provides a compelling and essential interaction point for a player, heightening the overall user experience by establishing a tangible and satisfying feedback loop. Upon the correct guess of the randomly generated four-digit number, the program reveals the correct number alongside the total number of attempts a player has made to finalize the gaming experience (https://www.britannica.com/technology/computer-programming).

This marked conclusion of the game transforms the previously unseen effort of the player into a numerical testament to their success, cementing the outcome within the context of the game. Crucially, this feedback loop establishes a vital component of guided learning in gameplay, offering the player a definite sense of their progress throughout the game, driven by an understanding of the number of attempts taken to triumph ultimately. The repeated prompt to guess, coupled with the rewarding revelation of the correct guess and attempts, creates a compelling cyclical game experience. This structurally sound gaming experience considerably enriches the player experience by providing concrete feedback and a sense of achievement to the player. By facilitating the player's direct engagement with the program, it also fosters an understanding of Python software unit testing, showcasing the effectiveness of the learning via gaming approach. The player's engagement with the game, highlighted by their ability to gauge their progress through the number of attempts made, creates an immersive, educational, and satisfying experience, thereby enhancing the user experience through iterative feedback

(https://www.researchgate.net/publication/335924797_Game_Feedback_Loops).

In conclusion, the development of the 'Guess the Number' game using Test Driven Development (TDD) in Python symbolizes the ideal amalgamation of logical comprehensiveness and user interaction. TDD's incremental development process significantly contributed to the creation of a game that is not only interactive but almost entirely devoid of errors. This method was incessantly used as the foundation at each stage of the game's development, ensuring a robust and reliable structure. To begin with, a test was written for each functionality before the actual code was implemented. These tests served as guidelines, outlining the expected system behavior and consequently driving the code design. During the random generation of the four-digit number, tests ensured that the generated number was within the prescribed four-digit limit and was truly random, assuring the game's unpredictability and consequently its intrigue. As the user was prompted to guess the number repeatedly until the correct guess was made, tests were implemented to validate the user input, checking for erroneous and non-numeric inputs and guiding the user accordingly, thereby enhancing user experience. Each hint provided to the user was also carefully tested for accuracy and hinted towards the correct number, subtly guiding the user. Post the correct guess, the computation and display of the number of attempts taken were also validated using TDD ensuring accuracy. Thus, the use of TDD in Python for the 'Guess the Number' game development provided a systematic, error-free approach that upheld quality assurance, delivering a user-friendly, interactive game experience.

Git directory: https://github.com/s369805/PRT582.git

REFERENCES

Game Feedback Loops (2019). Available at:
https://www.researchgate.net/publication/335924797_Game_Feedback_Loops

Computer Programming (n.d). Available at:
https://www.britannica.com/technology/computer-programming

Python Program to Guess a Number (n.d.). Available at:
https://www.geeksforgeeks.org/python-program-to-guess-a-number/

Building a Simple Guessing Game in Python (n.d.). Available at:
https://geekflare.com/python-guessing-game/

Random – Generate pseudo-random numbers (n.d). Available at:
https://docs.python.org/3/library/random.html

Python Testing (n.d.). Available at:
https://www.realpython.com/python-testing/