

Software Unit testing report

~Prasanna Adhikari (s371855)



Contents

[Introduction 2](#_Toc143881139)

[Process 2](#_Toc143881140)

[Conclusion 4](#_Toc143881141)

[Screenshots 6](#_Toc143881142)

**Table of Figure**

[Figure 1: Initial Test Case 2](#_Toc143881171)

[Figure 2: Final Test Case 3](#_Toc143881172)

[Figure 3: patch, builtins.input and side\_effect 3](#_Toc143881173)

[Figure 4: Number Guessing Game 4](#_Toc143881174)

[Figure 5: Working demo 6](#_Toc143881175)

[Figure 6: Demo of all the tests has been passed 7](#_Toc143881176)

[Figure 7: Flake8 and Pylint check screenshot 7](#_Toc143881177)

[Figure 8: Pylint and Flake8 check screenshot 8](#_Toc143881178)

# Introduction

The objective of the "Number Guessing Game" is to provide players with a fun experience by asking them to guess a four-digit secret number that the computer has generated. The game's primary objective is to engage logical and critical thinking, prompting players to strategically guess the correct number within a limited number of attempts. The game has various requirements. It should generate a random four-digit number with unique digits, accept user input for guesses while validating the input's format, provide informative hints after each guess to guide players and determine when a player successfully guesses the secret number. The game loop should offer the option to play again, maintaining a user-friendly interface with clear messages and prompts. To guarantee the game's dependability and accuracy, an automatic unit testing tool is used. Valid and invalid user inputs, accurate and inaccurate guesses, and exit situations are all covered by the testing process. With the help of these goals and specifications, the "Number Guessing Game" intends to give users a joyful and intellectually stimulating experience, along with a reliable testing system to ensure its performance.

# Process

In the "Number Guessing Game", Test-Driven Development (TDD) techniques and an automated unit testing tool have been used to improve the game's development process and guarantee its dependability. In developing the "Number Guessing Game," a strategy known as Test-Driven Development (TDD) was used. This approach involves writing tests before creating the actual game features. For example, before coding the part that generates random numbers, a test case named "test\_generate\_random\_number" was written. This test case described what the random number should look like.

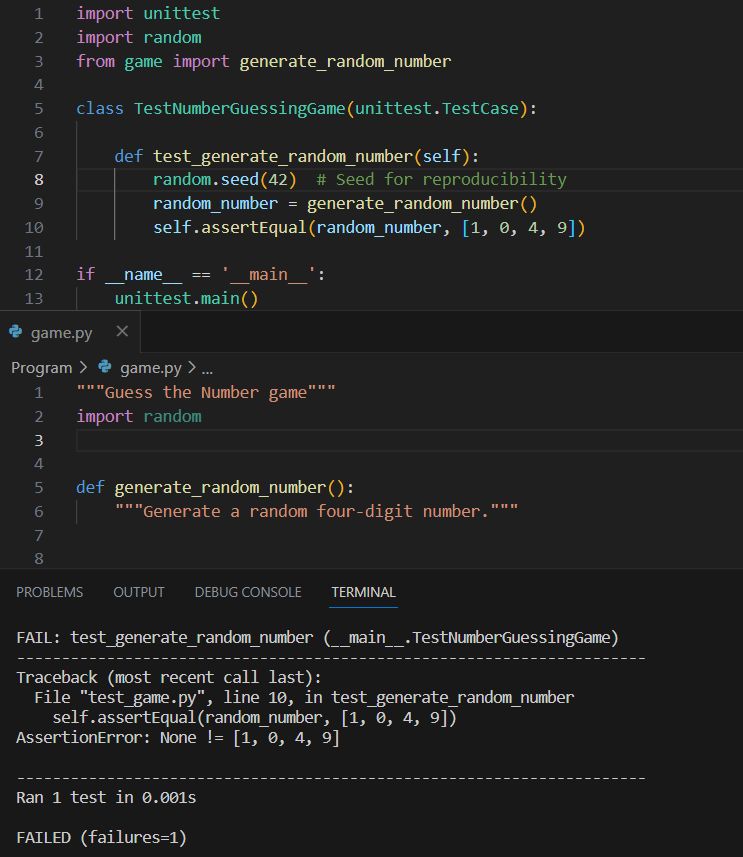


Figure 1: Initial Test Case

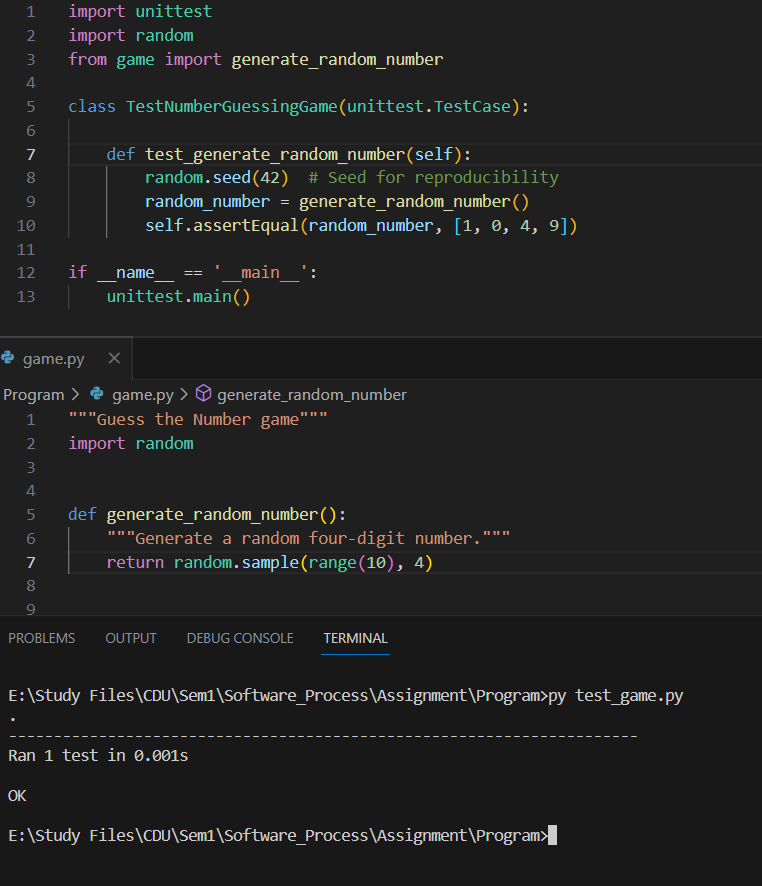


Figure 2: Final Test Case

This method helped in focusing on writing the code that could create random numbers exactly as needed. Similar to this, other aspects like receiving user guesses and providing hints were also put through tests to ensure they worked correctly. These tests ensured that each part of the game functioned as intended.

To make testing easier, special tools from the **unittest** framework were used, including **patch**, **builtins.input**, and **side\_effect**. The process began by creating tests that outlined how each part of the game should work. This provided a clear guide for writing the code. The **patch** tool simulated interactions with the game, while **builtins.input** and **side\_effect** helped type things as if we were players.

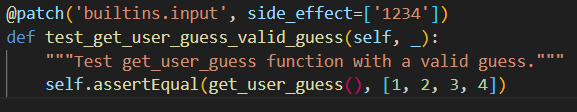


Figure 3: patch, builtins.input and side\_effect

Here's how it was done: First, the game's expected behavior was written down in tests. Then, a tool named **patch** pretended to talk to the game, and **builtins.input** with **side\_effect** acted like players typing things. This made it possible to test the game step by step without needing to play through it manually. Finally, the game had been thoroughly tested and was ready for use.

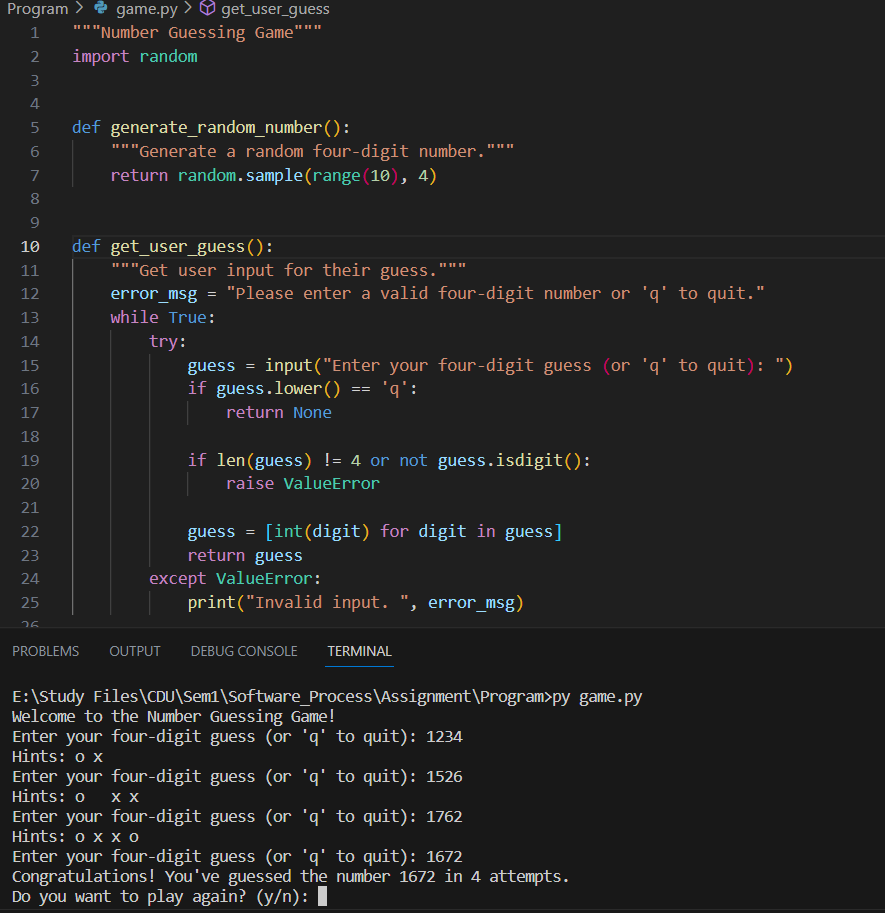


Figure 4: Number Guessing Game

# Conclusion

Developing the "Number Guessing Game" with Test-Driven Development (TDD) and testing tools had some strong points and spots to improve. TDD was helpful in planning tests before writing the code, which made the path clearer. Tools like `patch`, `builtins.input`, and `side\_effect` aided in controlled testing. Additionally, the teamwork of TDD and tools ensured that the game was dependable and robust, making it more trustworthy.

However, there are areas with room for improvement. To expand testing, incorporating a wider array of test cases for different game scenarios is advisable. Exploring advanced capabilities within the `unittest` framework could offer more testing options. For greater efficiency, setting up automated test suites could streamline the process and lessen manual involvement. Lastly, involving team members in collaborative code reviews can offer new insights and identify areas for enhancement.

In conclusion, the process showcased the value of TDD and testing tools, highlighting strengths in systematic development and validation. By addressing areas of improvement such as diversified testing, exploring advanced `unittest` features, and optimizing the testing process, the development process could be further refined, ultimately leading to more efficient and robust outcomes.

**Github link**: <https://github.com/prasanna-adhikari/Software-Unit-Testing-Report>

# Screenshots

1. Working game screenshots

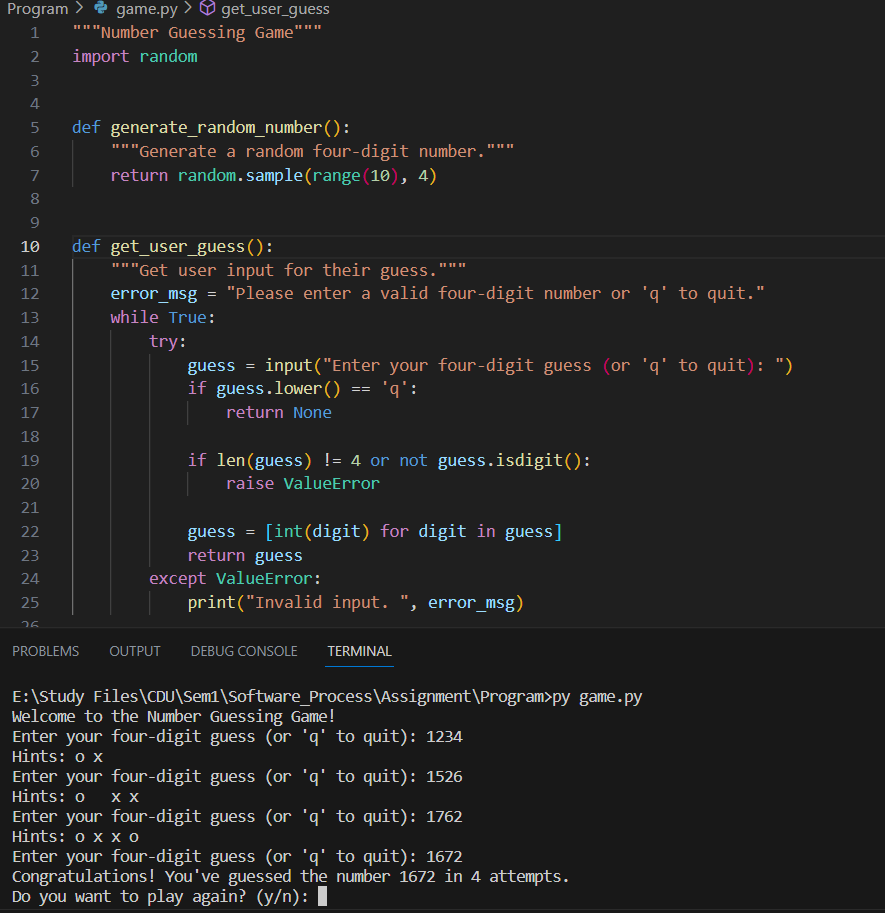


Figure 5: Working demo

1. Demo of Test

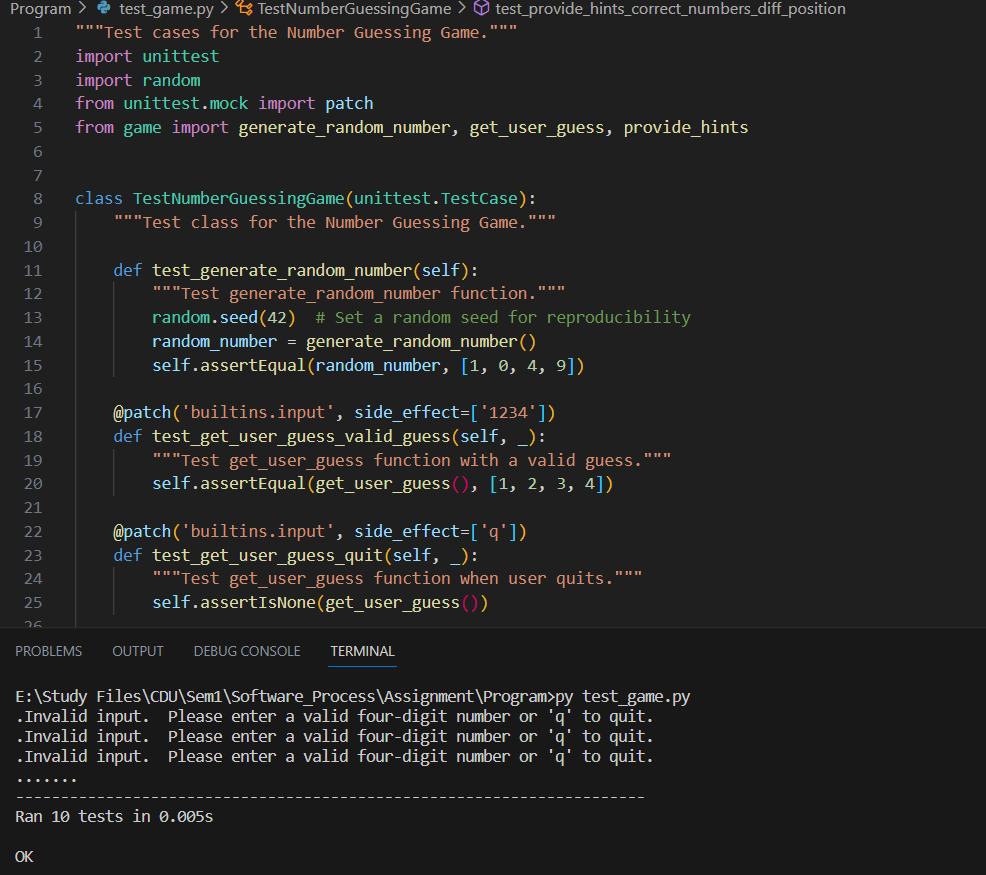


Figure 6: Demo of all the tests has been passed

1. Pylint and Flake8 screenshots of the main program

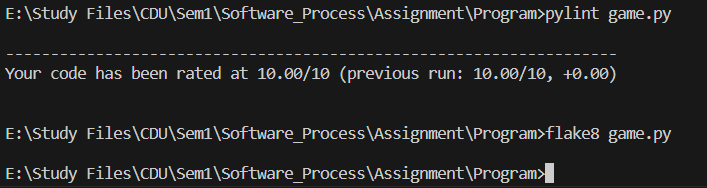


Figure 7: Flake8 and Pylint check screenshot

1. Pylint and Flake8 screenshots of the Test program

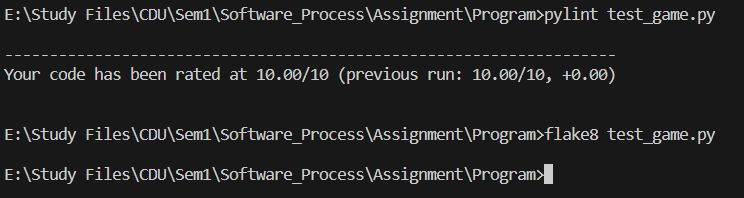


Figure 8: Pylint and Flake8 check screenshot