**ASSIGNMENT 2**

**SOFTWARE UNIT TESTING REPORT**

**SOFTWARE ENGINEERING: PROCESS AND TOOLS**

**S223 PRT582**

**GUESS THE NUMBER GAME USING TEST-DRIVEN DEVELOPMENT IN PYTHON**

**STUDENT NAME – NITHYA ROMESHIKA YAMASINGHE**

**STUDENT ID – S370170**

**TABLE OF CONTENT**

LIST OF FIGURES……………………………………………………………………………………………………………………………………..iii

[1. INTRODUCTION 4](#_Toc143881712)

[1.1 OBJECTIVES OF THE GAME 4](#_Toc143881713)

[1.1.1 OBJECTIVES 5](#_Toc143881714)

[1.2.1 REQUIREMENTS OF THE GAME 7](#_Toc143881715)

[1.2.1 FUNCTIONAL REQUIREMENTS 7](#_Toc143881716)

[1.2.2 EXTERNAL INTERFACE REQUIREMENTS: 8](#_Toc143881717)

[1.2.2.1 USER INTERFACES 8](#_Toc143881718)

[1.2.2.2 HARDWARE INTERFACES 8](#_Toc143881719)

[1.2.2.3 SOFTWARE INTERFACES 8](#_Toc143881720)

[1.2.2.4 COMMUNICATION INTERFACES 8](#_Toc143881721)

[1.2.3 NON FUNCTIONAL REQUIREMENTS 8](#_Toc143881722)

[1.2.3.1 PERFORMANCE REQUIREMENTS 8](#_Toc143881723)

[1.2.3.2 SAFETY REQUIREMENTS 8](#_Toc143881724)

[1.2.3.3 SECURITY REQUIREMENTS 9](#_Toc143881725)

[1.2.3.4 SOFTWARE QUALITY REQUIREMENTS 9](#_Toc143881726)

[1.3 AUTOMATED SOFTWARE UNIT TESTING TOOL USED 9](#_Toc143881727)

[2. PROCESS OF THE TEST-DRIVEN DEVELOPMENT OF THE GAME 10](#_Toc143881728)

[2.1 TEST DRIVEN DEVELOPMENT LIFE CYCLE 11](#_Toc143881729)

[2.2 HOW THE TEST-DRIVEN DEVELOPMENT USED IN THE GAME 11](#_Toc143881730)

[2.3 HOW THE AUTOMATED SOFTWARE UNIT TESTING TOOL USED IN THE GAME 15](#_Toc143881731)

[2.4 HOW IT USED STEP-BY-STEP WITH SCREENSHOTS 17](#_Toc143881732)

[3. THE FINAL PRODUCT 28](#_Toc143881733)

[3.1 HOW THE GAME LOOK LIKE WITH SCREENSHOTS 28](#_Toc143881734)

[4. CONCLUSION 31](#_Toc143881735)

[4.1 LESSONS LEARNED 31](#_Toc143881736)

[4.1.1 WHAT WENT WELL 33](#_Toc143881737)

[4.1.2 WHAT WENT WRONG 34](#_Toc143881738)

[4.2 FUTURE IMPROVEMENTS TO THE GAME 34](#_Toc143881739)

[4.3 FLAKE8 AND PYLINT SCORE DETAILS 36](#_Toc143881740)

[4.4 GITHUB REPOSITORY LINK 38](#_Toc143881741)

# LIST OF FIGURES

[Figure 2. 1: Test-Driven Development Cycle 8](#_Toc143883905)

[Figure 2. 2: Sample Test Case – test\_calculate\_hints\_all\_correct 9](#_Toc143883906)

[Figure 2. 3: Calculate Hints Method 9](#_Toc143883907)

[Figure 2. 4: Sample Test Case – test\_calculate\_hints\_mixed\_correct\_positions 10](#_Toc143883908)

[Figure 2. 5: Guess the Number App Class 10](#_Toc143883909)

[Figure 2. 6: Sample Test Case – handle\_wrong\_guess 11](#_Toc143883910)

[Figure 2. 7: Initial test cases without any implementation 14](#_Toc143883911)

[Figure 2. 8: Without adding any logic to the Guess the Number Logic Class in main.py 14](#_Toc143883912)

[Figure 2. 9: Running test cases without adding logic to main.py 15](#_Toc143883913)

[Figure 2. 10: After adding some logic to Guess the Number Game Logic Class in main.py 16](#_Toc143883914)

[Figure 2. 11: After adding logic and testing, all test cases have passed 16](#_Toc143883915)

[Figure 2. 12: Without adding any logic, Guess the Number Game App Class 17](#_Toc143883916)

[Figure 2. 13: After adding some logic, Guess the Number Game App Class in the main.py 17](#_Toc143883917)

[Figure 2. 14: After adding some logic, Guess the Number Game App Class in the main.py 18](#_Toc143883918)

[Figure 2. 15: After adding some logic, Guess the Number Game App Class in the main.py 18](#_Toc143883919)

[Figure 2. 16: After adding some logic, Guess the Number Game App Class in the main.py 19](#_Toc143883920)

[Figure 2. 17: The complete logic added to the Guess the Number Game App Class in main.py 19](#_Toc143883921)

[Figure 2. 18: After adding logic and testing, all test cases have passed 20](#_Toc143883922)

[Figure 2. 19: After adding minimal implementation of Guess, the Number Game Logic Class 21](#_Toc143883923)

[Figure 2. 20: After adding minimal implementation of Guess the Number Game App Class 21](#_Toc143883924)

[Figure 2. 21: Initially failing test sample 22](#_Toc143883925)

[Figure 2. 22: After adding more implementation to the Guess, the Number Game Logic Class 22](#_Toc143883926)

[Figure 2. 23: After adding more implementation to the Guess the Number Game App Class 23](#_Toc143883927)

[Figure 2. 24: All tests successful of Guess the Number Game Logic Class 23](#_Toc143883928)

[Figure 2. 25: All tests successful of Guess the Number Game App Class 24](#_Toc143883929)

[Figure 2. 26: Sample refactoring 24](#_Toc143883930)

[Figure 3. 1: Interface 01 25](#_Toc143883931)

[Figure 3. 2: Interface 02 26](#_Toc143883932)

[Figure 3. 3: Interface 03 26](#_Toc143883933)

[Figure 3. 4: Interface 04 27](#_Toc143883934)

[Figure 3. 5: Interface 05 27](#_Toc143883935)

[Figure 3. 6: Interface 06 28](#_Toc143883936)

[Figure 4. 1: Flake8 main.py 33](#_Toc143884163)

[Figure 4. 2: Pylint main.py 33](#_Toc143884164)

[Figure 4. 3: test\_main.py test passing 34](#_Toc143884165)

[Figure 4. 4: Flake8 test\_main.py 34](#_Toc143884166)

[Figure 4. 5: Pylint test\_main.py 35](#_Toc143884167)

# INTRODUCTION

The "Guess the Number" game is developed using Python. Players are invited to decipher a secret four-digit number, diligently generating guesses while decoding a trail of hints. The game's mechanics are straightforward: with every Guess, players receive cues like 'circles' and 'x's, guiding them toward the solution. A 'circle' signifies a correct digit in the correct position, while an 'x' suggests a correct digit in an alternate slot. Through this engaging interplay of deduction and intuition, players experience the exhilaration of solving a digital riddle. As we dive into the development of this game, we not only unravel an absorbing entertainment experience but also grasp the tenets of Test-Driven Development (TDD), ensuring that every aspect of the game is enjoyable and meticulously crafted.

The development of the captivating "Guess the Number" game uses the integration of key technologies that collectively create an engaging and seamless experience. Python, renowned for its versatility and readability, forms the backbone of the game's logic, providing a robust foundation for implementing its mechanics. Leveraging Python's dynamic capabilities, the game's codebase comes to life, enabling the generation of random numbers, user input handling, and feedback processing. Python Tkinter, the standard graphical user interface library, further enriches the game's interactive facet. Tkinter empowers the creation of an intuitive and user-friendly interface, facilitating player interactions through visual prompts and options. Furthermore, the game's reliability is fortified by adopting Python Unittest, a testing framework that embodies the principles of Test-Driven Development (TDD). Through systematically crafted test cases, Python Unittest ensures the accuracy and resilience of the game's functionalities, offering players an enjoyable and flawlessly executed gaming experience.

## OBJECTIVES OF THE GAME

Several compelling objectives drive player engagement and software development finesse in the "Guess the Number" game. As players embark on the journey to crack a concealed four-digit code, they are presented with a unique challenge that encourages intuitive deduction and strategic thinking. The game's core objective revolves around interpreting the subtle hints it provides, conveyed through 'circles' and 'x's, which guide players toward the correct number. Players refine their guesses with each attempt, strategically leveraging the feedback to inch closer to the solution. This gameplay dynamic fosters an atmosphere of entertainment and education as players decipher numbers and learn the art of interpreting feedback cues. Moreover, the game's architecture embodies the principles of Test-Driven Development (TDD), providing an opportunity to practice this vital software engineering approach. The "Guess the Number" game encapsulates a synergy between engaging gameplay and skillful software design by achieving these objectives.

## OBJECTIVES

1. Number Guessing Challenge -

The game's primary objective is to guess a randomly generated four-digit number.

2. Intuitive Deduction -

Engage players in deducing the correct number through logical reasoning and educated guesses.

3. Feedback Interpretation -

Teach players to interpret the feedback provided by the game in the form of 'circles' and 'x's, aiding them in refining their guesses.

4. Strategic Guessing -

Encourage players to refine their guesses based on feedback after each attempt, fostering a strategic approach to number guessing.

5. Game Completion -

Challenge players to continue guessing until they successfully decipher the entire four-digit number.

6. Attempt Count Tracking -

Keep track of the player's attempts to solve the puzzle.

7. User Interaction -

Facilitate player and program interaction by providing input prompts and game-related information.

8. Game Loop -

Implement a loop structure that allows players to make multiple guesses until they solve the puzzle or decide to quit.

9. Play Again Option -

Provide players with the choice to play the game again after successfully solving the puzzle or quitting.

10. Test-Driven Development (TDD) Practice -

Implement the game using the TDD approach, adhering to the principles of writing tests before writing the code.

11. Robust Software Design -

Develop a well-tested and reliable game that entertains and showcases software engineering principles.

12. Learning TDD Concepts -

Use the game development process to gain a practical understanding of Test-Driven Development, applying it to create a functional and enjoyable game.

By achieving these objectives, the "Guess the Number" game provides an engaging entertainment experience and imparts lessons in logical deduction, feedback interpretation, and applying TDD principles in software development.

## REQUIREMENTS OF THE GAME

## 1.2.1 FUNCTIONAL REQUIREMENTS

* Random Number Generation

The system must be capable of generating a random four-digit number to serve as the target for players to guess.

* User Interaction

The system should repeatedly prompt the player to input their Guess. The game loop continues until the player correctly guesses the four-digit number or decides to quit.

* Guess Evaluation and Feedback

After each Guess, the system must provide feedback to the player. This feedback will consist of 'circles' and 'x's, where 'circles' denote a correct digit in the correct position and 'x's denote a correct digit in the wrong position.

* Game Completion

The game loop must continue until the player successfully guesses the number or chooses to quit. Upon successfully guessing the number, the system should display the number of attempts the player took.

* Play Again Option

After the game is completed, the system should prompt the player to decide whether to play the game again or exit.

* Quit Option

The system should allow the player to quit the game at any point during the gameplay.

## 1.2.2 EXTERNAL INTERFACE REQUIREMENTS:

## 1.2.2.1 USER INTERFACES

The system's user interface should provide clear and concise prompts to guide the player's interactions with the game. It should also display the feedback understandably.

## 1.2.2.2 HARDWARE INTERFACES

No specific hardware interfaces are required for the game, as it is a text-based application.

## 1.2.2.3 SOFTWARE INTERFACES

The system should be implemented using Python programming language with an inbuilt tkinter module for the GUI, utilizing standard libraries for random number generation and user input/output.

## 1.2.2.4 COMMUNICATION INTERFACES

No external communication interfaces are required for this standalone game.

## 1.2.3 NON FUNCTIONAL REQUIREMENTS

## 1.2.3.1 PERFORMANCE REQUIREMENTS

The system should be responsive to user inputs and provide timely feedback. It should handle typical gameplay scenarios efficiently.

## 1.2.3.2 SAFETY REQUIREMENTS

The game does not involve any safety-critical aspects and does not pose any risks to users.

## 1.2.3.3 SECURITY REQUIREMENTS

No specific security requirements are necessary for this offline game, as it does not involve external data inputs or sensitive information.

## 1.2.3.4 SOFTWARE QUALITY REQUIREMENTS

The system's code should be well-structured, modular, and adequately commented to ensure readability and maintainability. Test-driven development (TDD) practices should ensure the software is thoroughly tested for correctness and reliability.

Organizing the requirements in this structure makes it easier to understand the different aspects of the game and its development expectations.

## AUTOMATED SOFTWARE UNIT TESTING TOOL USED

In the development journey of the "Guess the Number" game, the implementation is fortified by integrating an automated unit testing framework, Python's `unit test`. This robust testing tool forms the cornerstone of Test-Driven Development (TDD), enabling developers to validate their code's accuracy, reliability, and functionality. Through a meticulously designed suite of test cases, `unit test` systematically exercises various aspects of the game's logic and user interactions, ensuring that each component functions as intended. By executing these tests automatically, discrepancies and bugs are promptly identified, fostering a continuous cycle of refinement during the development process. `unit test` provides a structured and repeatable approach to testing, enhancing the reliability of the software and aiding in the rapid identification of regressions as the codebase evolves. This systematic validation mechanism, encapsulated within the `unittest` framework, aligns perfectly with the principles of TDD, contributing to creating a polished and dependable "Guess the Number" game.

# PROCESS OF THE TEST-DRIVEN DEVELOPMENT OF THE GAME

The development of the "Guess the Number" game followed a rigorous Test-Driven Development (TDD) methodology, an iterative process driven by automated testing. The initial phase involved formulating an extensive suite of test cases that comprehensively assessed distinct aspects of the game's logic and user interactions. These tests were designed to validate key functionalities like input validation, feedback generation, and core game logic.

Subsequently, the actual implementation of the game's code began in alignment with the outlined tests. Each function was developed incrementally, emphasizing adhering to the expected test outcomes. This entailed writing code to meet the specific conditions and constraints of the established tests, ensuring that each function correctly fulfilled its designated purpose. Throughout this process, regular execution of the tests provided continuous feedback on the accuracy and robustness of the codebase.

As the codebase evolved, the tests themselves evolved as well. Modifications to the code triggered corresponding updates in the tests, reflecting the dynamic nature of the development process. This synchronization between the tests and the codebase fostered a reliable and adaptable foundation for the game. Notably, the adherence to TDD principles enabled swift detection and rectification of potential issues. Any deviation from the expected test outcomes indicated areas that required further refinement. This approach also facilitated the introduction of new features and improvements while maintaining a stable baseline, reducing the likelihood of introducing regressions.

In conclusion, successfully implementing the "Guess the Number" game through TDD underscored the symbiotic relationship between meticulous testing and code development. The resulting application is a testament to TDD's efficacy in fostering precision, adaptability, and reliability in software engineering endeavors.

## 2.1 TEST DRIVEN DEVELOPMENT LIFE CYCLE

A diagram of a test

Description automatically generated

Figure 2. : Test-Driven Development Cycle

Test-Driven Development (TDD) is a coding approach where we write tests before writing the actual code. The process starts with creating small tests that describe how a specific part of the code should behave. These tests initially fail, as no code fulfills their expectations. Then, the actual code is written to make the tests pass. This iterative cycle of writing a test, observing it fail, and then writing the code to pass the test forms the basis of TDD. By following this approach, we ensure that each piece of code is thoroughly tested and that the code evolves in small, controlled steps, resulting in more robust and reliable software.

## 2.2 HOW THE TEST-DRIVEN DEVELOPMENT USED IN THE GAME

1. Setting up Test Cases

In TDD, we start by writing tests before implementing the actual functionality. In the game's case, we created test cases for the `GuessNumberGameLogic` class to validate the logic for calculating hints based on the user's Guess. For instance, the `test\_calculate\_hints\_all\_correct` method verifies the correct calculation of hints when all digits and positions are correct:

A screenshot of a computer program

Description automatically generated

Figure 2. : Sample Test Case – test\_calculate\_hints\_all\_correct

1. Incremental Implementation

Once tests are in place, we incrementally implement the corresponding methods. For instance, in the `calculate\_hints` method, we iterated through the digits of the Guess and compared them to the actual random number. The code incrementally fulfilled the test cases:

A screenshot of a computer

Description automatically generated

Figure 2. : Calculate Hints Method

3. Testing Different Scenarios

TDD ensures that various scenarios are considered. We tested scenarios like incorrect guesses with mixed correct digits and positions and duplicates. For example, the `test\_calculate\_hints\_mixed\_correct\_positions` method checks the calculation of hints for mixed correct digits and positions:

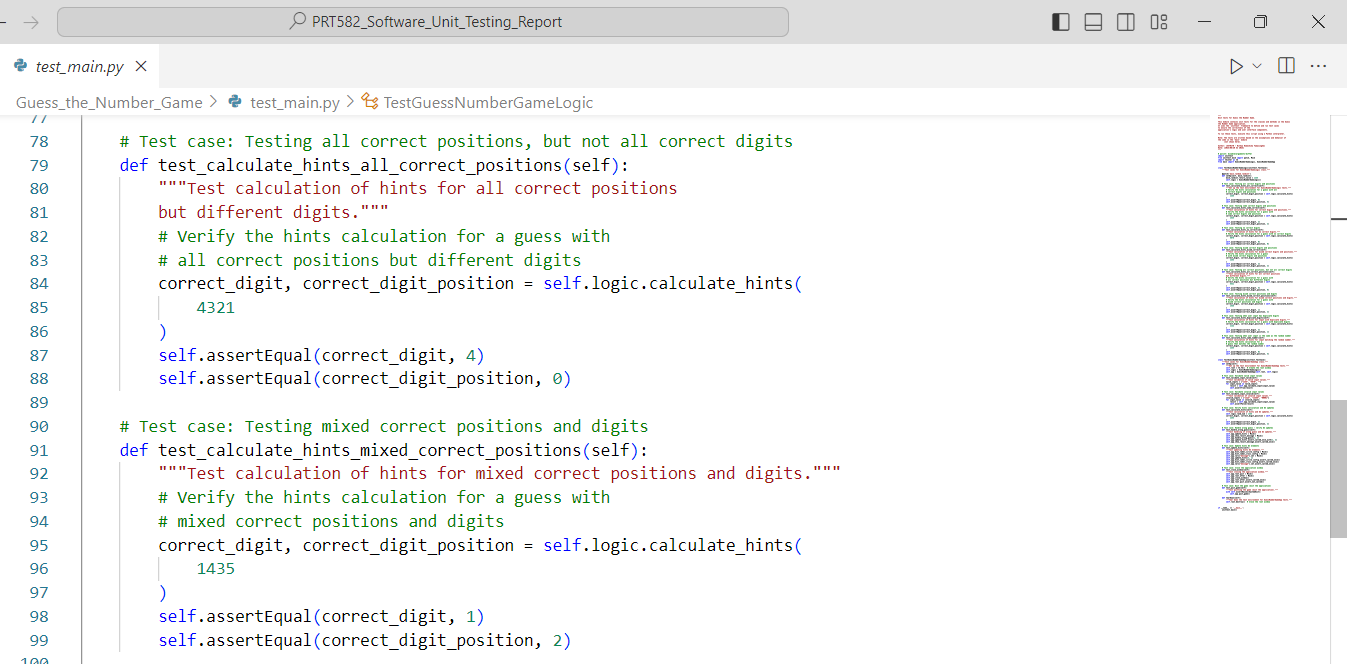


Figure 2. : Sample Test Case – test\_calculate\_hints\_mixed\_correct\_positions

4. User Interface Testing

In the `GuessNumberGameApp` class, TDD was also applied to the UI. We tested user input validation, such as valid and invalid inputs.

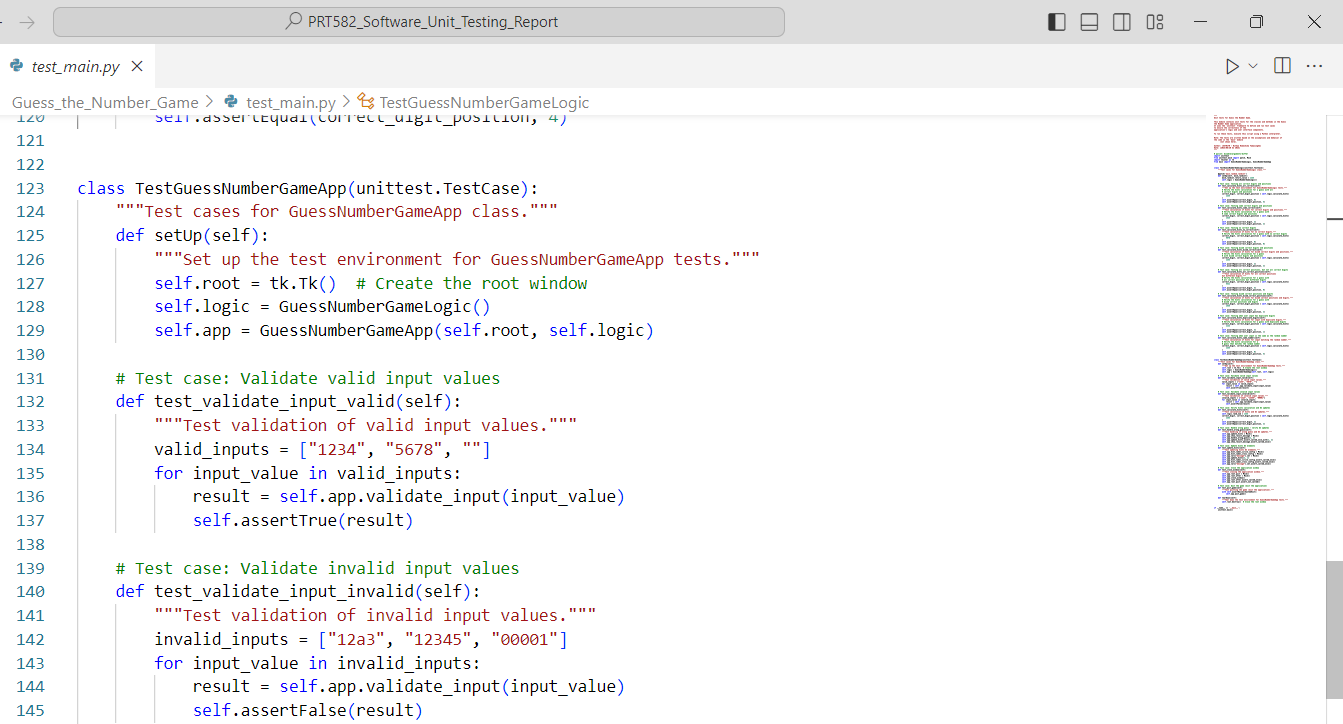


Figure 2. : Guess the Number App Class

5. Interaction Testing

Using TDD, user interaction methods such as `guess\_number` and `handle\_wrong\_guess` were tested. For example, `test\_handle\_wrong\_guess` verifies the UI updates after a wrong guess:

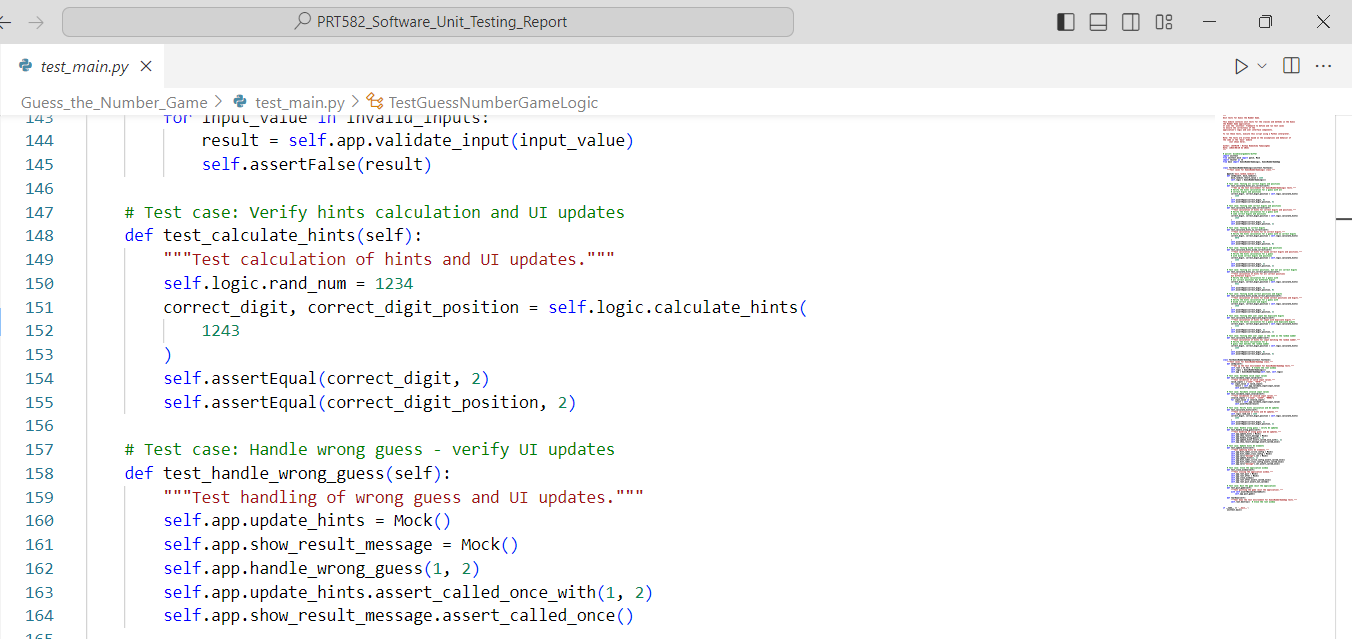


Figure 2. : Sample Test Case – handle\_wrong\_guess

6. Integration and Refinement

Tests were executed frequently throughout the development, ensuring each modification adhered to the expected behavior. Any deviations resulted in immediate rectification. The code and tests evolved, leading to a refined and accurate game implementation.

The "Guess the Number" game was developed using the TDD approach, yielding reliable and effective results by systematically implementing, testing, and refining different methods and functionalities through a series of tests.

## 2.3 HOW THE AUTOMATED SOFTWARE UNIT TESTING TOOL USED IN THE GAME

* Testing Tool

In the "Guess the Number" game, the automated software unit testing tool used is Python's built-in `unittest` framework. This framework provides the tools and methods to write and execute unit tests for the game's components.

* Testing Process

The testing process involved creating separate test cases for each game's logic and user interface component. Each test case was developed as a class inheriting from `unittest.TestCase`. Within each test case, specific methods were defined to test different aspects of the component.

* Testing Modules Used

Several modules from the `unittest` framework were utilized in the testing process:

1. `unittest.TestCase`: The base class for all test cases. It provides methods for setting up and tearing down the testing environment.
2. `unittest.mock.patch`: This module allowed the mocking of external functions or classes for controlled testing. It stimulates the `random.randint` behavior in the game logic tests.
3. `unittest.TestCase.assertEqual`: A method for asserting that two values are equal.
4. `unittest.TestCase.assertTrue` and `unittest.TestCase.assertFalse`: Methods to assert that a condition is true or false, respectively.
5. `unittest.TestCase.assertRaises`: A method to verify that a specific exception is raised.

* Testing Structure

1. The testing structure was organized around separate test cases for the `GuessNumberGameLogic` class and the `GuessNumberGameApp` class. Each test case was a subclass of `unittest.TestCase`, and within each test case, multiple test methods were defined to test different scenarios and functionalities.
2. For instance, the `TestGuessNumberGameLogic` test case contained methods like `test\_calculate\_hints\_all\_correct`, `test\_calculate\_hints\_some\_correct`, and so on. Each method tested a specific scenario, such as calculating hints for different user inputs and comparing the expected outcomes.
3. Similarly, the `TestGuessNumberGameApp` test case had methods like `test\_validate\_input\_valid`, `test\_calculate\_hints`, and others. These methods validated the user interface and interaction aspects of the game.

* Execution and Reporting

The tests were executed by running the test script, and the `unittest` framework handled the execution of each test method within the test cases. The framework provided informative output, indicating which tests passed and which failed. Any failures indicated discrepancies between expected and actual outcomes.

In conclusion, the "Guess the Number" game's testing process was effectively managed using Python's `unittest` framework. The testing structure was organized, with each test case focusing on specific functionalities. The utilization of `unittest.mock.patch` allowed controlled testing, and the assertion methods verified expected behaviors. The overall process ensured that the game's components functioned as intended and provided a reliable user experience.

## 2.4 HOW IT USED STEP-BY-STEP WITH SCREENSHOTS

The TDD approach followed to implement the game using the `unittest` framework and includes test cases for both the `GuessNumberGameLogic` class and the `GuessNumberGameApp` class. Here's a step-by-step breakdown of the Test-Driven Development (TDD) approached

1. Test the `GuessNumberGameLogic` Class

This class is responsible for the game logic.

A screenshot of a computer

Description automatically generated

Figure 2. : Initial test cases without any implementation

A screenshot of a computer

Description automatically generated

Figure 2. : Without adding any logic to the Guess the Number Logic Class in main.py

a. `test\_calculate\_hints\_all\_correct`:

Test the logic when all digits and positions are correct.

b. `test\_calculate\_hints\_some\_correct`:

Test the logic when some digits and positions are correct.

c. `test\_calculate\_hints\_no\_correct`:

Test the logic when no digits are correct.

d. `test\_calculate\_hints\_mixed\_correct`:

Test the logic when some digits are correct but in different positions.

e. `test\_calculate\_hints\_all\_correct\_positions`:

Test the logic when all positions are correct but not all digits.

f. `test\_calculate\_hints\_mixed\_correct\_positions`:

Test the logic when some positions and digits are correct.

g. `test\_calculate\_hints\_duplicate\_digits`:

Test the logic when there are duplicate digits in the input.

h. `test\_calculate\_hints\_same\_number`:

Test the logic when the input matches the random number.

A screenshot of a computer

Description automatically generated

Figure 2. : Running test cases without adding logic to main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding some logic to Guess the Number Game Logic Class in main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding logic and testing, all test cases have passed

2. Test the `GuessNumberGameApp` Class

This class deals with the user interface and interaction with the game logic.

A screenshot of a computer

Description automatically generated

Figure 2. : Without adding any logic, Guess the Number Game App Class

A screenshot of a computer

Description automatically generated

Figure 2. : After adding some logic, Guess the Number Game App Class in the main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding some logic, Guess the Number Game App Class in the main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding some logic, Guess the Number Game App Class in the main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding some logic, Guess the Number Game App Class in the main.py

A screenshot of a computer

Description automatically generated

Figure 2. : The complete logic added to the Guess the Number Game App Class in main.py

A screenshot of a computer

Description automatically generated

Figure 2. : After adding logic and testing, all test cases have passed

a. `test\_validate\_input\_valid`:

Test input validation for valid input values.

b. `test\_validate\_input\_invalid`:

Test input validation for invalid input values.

c. `test\_calculate\_hints`:

Test the UI's reaction when hints are calculated and updated.

d. `test\_handle\_wrong\_guess`:

Test how the UI updates when the Guess is incorrect.

e. `test\_update\_hints`:

Test updating UI elements for hints.

f. `test\_close\_window`:

Test closing the application window.

g. `test\_quit\_game`:

Test quitting the game (exit the application).

3. Write Minimal Implementation

Before writing the actual class implementations, I created minimal implementations of the classes and methods to make the tests run without errors.

A screenshot of a computer

Description automatically generated

Figure 2. : After adding minimal implementation of Guess, the Number Game Logic Class

A screenshot of a computer

Description automatically generated

Figure 2. : After adding minimal implementation of Guess the Number Game App Class

4. Run Tests

Run the tests I've written. Since the implementations are minimal, most tests will fail.

A screenshot of a computer

Description automatically generated

Figure 2. : Initially failing test sample

5. Write Implementations

Started writing the actual code for the classes and methods, adhering to the expected behavior defined by the tests. Refactored as needed to ensure the code meets the expectations of the tests.

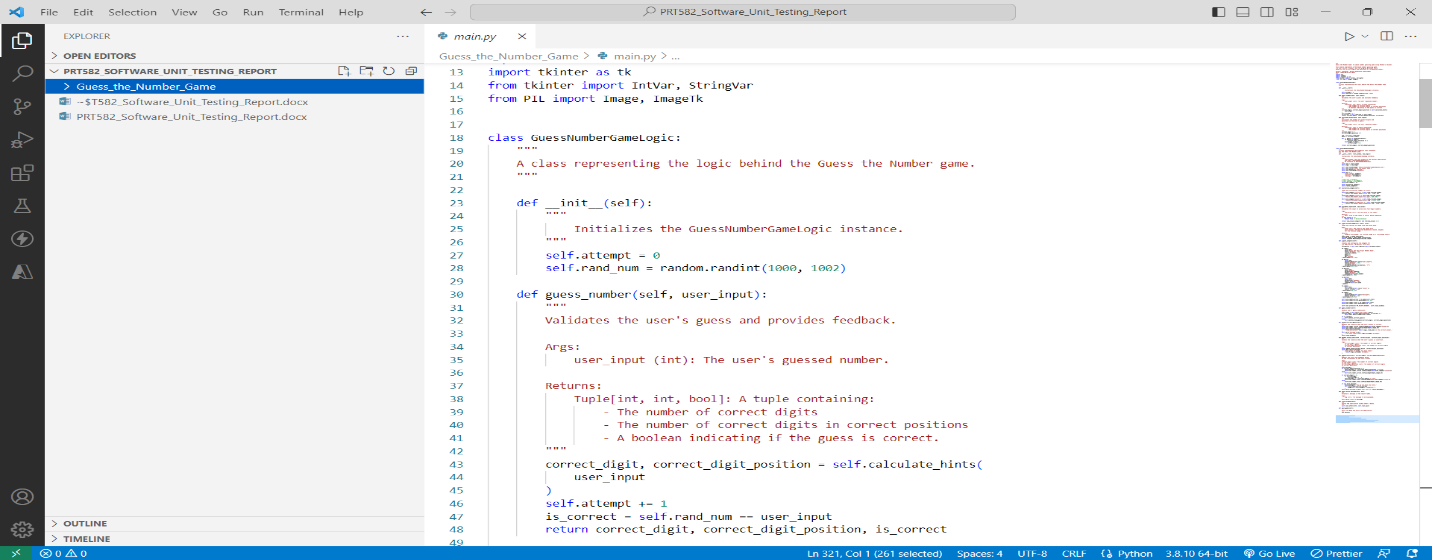


Figure 2. : After adding more implementation to the Guess, the Number Game Logic Class

A screenshot of a computer

Description automatically generated

Figure 2. : After adding more implementation to the Guess the Number Game App Class

6. Run Tests Again

After writing the implementations, run your tests again. This time, my goal was to make all tests pass.

A screenshot of a computer

Description automatically generated

Figure 2. : All tests successful of Guess the Number Game Logic Class

A screenshot of a computer

Description automatically generated

Figure 2. : All tests successful of Guess the Number Game App Class

7. Refactor and Improve

Once the tests are passed, refactor it if necessary to make it cleaner, more efficient, and better organized. Ensured that the code remained maintainable even as new features were added or requirements changed.

A screenshot of a computer

Description automatically generated

Figure 2. : Sample refactoring

# THE FINAL PRODUCT

The culmination of development efforts yields a polished final product as a standalone GUI application created using Python and the Tkinter library. This engaging graphical interface seamlessly encapsulates the essence of the "Guess the Number" game, providing users with an immersive experience. The Tkinter-powered user interface showcases an intuitive design, allowing players to input their guesses, receive real-time feedback, and interact with the game effortlessly. The self-contained nature of the application ensures its independence, enabling users to enjoy the game without any external dependencies. The amalgamation of Python's dynamic capabilities and Tkinter's graphical prowess results in a user-friendly, standalone game that blends entertainment with interactivity.

## HOW THE GAME LOOK LIKE WITH SCREENSHOTS

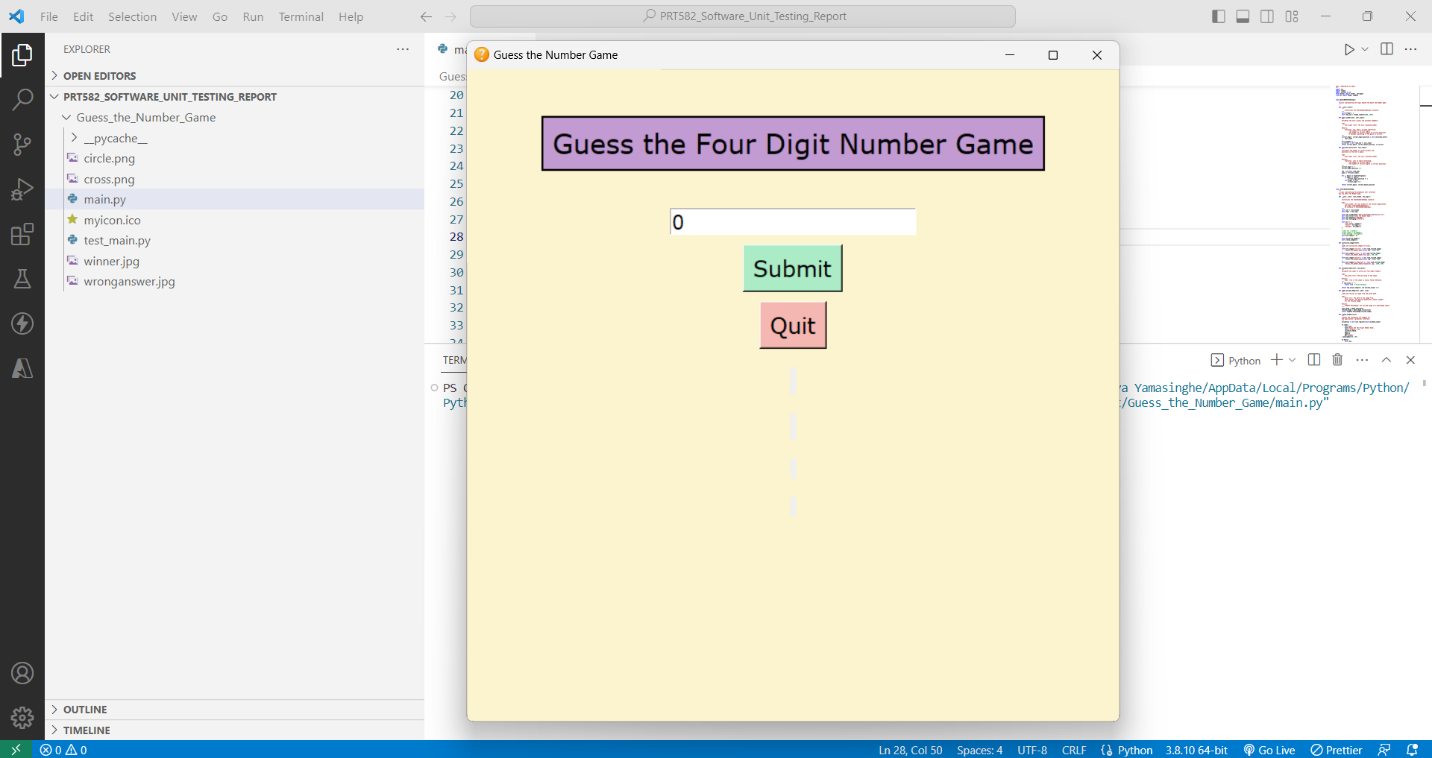


Figure 3. : Interface 01

A screenshot of a computer

Description automatically generated

Figure 3. : Interface 02

A screenshot of a computer

Description automatically generated

Figure 3. : Interface 03

A screenshot of a computer

Description automatically generated

Figure 3. : Interface 04

A screenshot of a computer

Description automatically generated

Figure 3. : Interface 05

A screenshot of a computer

Description automatically generated

Figure 3. : Interface 06

# CONCLUSION

The "Guess the Number" game emerges as a meticulously engineered application that harmonizes user interaction, game logic, and graphical representation. Employing Test-Driven Development (TDD) techniques, the game's core functionalities were rigorously validated through automated tests, ensuring accurate behavior and resilience against regressions. The integration of Python's Tkinter library facilitated the creation of an intuitive graphical interface, enabling users to engage with the game effortlessly. From user input validation to real-time feedback generation, each component was meticulously crafted to ensure precision and reliability. As a standalone GUI application, the game seamlessly encapsulates the intricacies of number guessing within an elegant and user-friendly interface, culminating in a technically sound and enjoyable software product.

## 4.1 LESSONS LEARNED

While developing the "Guess the Number" game using Test-Driven Development (TDD), several valuable lessons were learned that underscore the significance of this approach in software development.

1. Early Validation - TDD emphasizes creating test cases before writing the code. This practice highlighted the importance of clarifying expectations and requirements upfront, which led to a clearer understanding of the desired functionalities.

2. Incremental Development - TDD encourages incremental development by fulfilling one test case at a time. This incremental approach helped maintain a clear focus on a specific functionality while avoiding the temptation to overcomplicate the implementation.

3. Maintainable Code - TDD necessitates modular code to pass specific tests. This focus on modularity led to easier code to understand, maintain, and refactor, promoting cleaner architecture.

5. Improved Debugging - When tests fail, they indicate what went wrong. This facilitated pinpointing issues in the codebase, accelerating the debugging process.

7. Confidence in Refactoring - The extensive test suite fostered confidence during refactoring, allowing for code enhancements without fearing unintended side effects.

8. Quality Assurance - TDD places a strong emphasis on comprehensive testing. This commitment to quality assurance ensured that the final product was reliable and robust, minimizing potential user-facing issues.

9. Design Flexibility - TDD allows for design improvements without disrupting existing functionalities. The iterative nature of TDD encourages adapting the design to evolving requirements.

10. Structured Development - TDD provided a structured approach, which helped maintain focus, organize tasks, and avoid scope creep.

In conclusion, the application of Test-Driven Development throughout the development of the "Guess the Number" game brought to light the effectiveness of this approach in fostering precise code, dependable functionalities, and overall software quality.

## 4.1.1 WHAT WENT WELL

During the implementation of the "Guess the Number" game using Test-Driven Development (TDD), several aspects of the process went remarkably well. One standout factor was the clarity that emerged from defining test cases upfront. This step forced me to understand the game's requirements thoroughly, ensuring every functionality was well-defined and aligned with my intentions. TDD's incremental approach was another win. It kept me focused on small pieces of functionality at a time, ensuring that each component was comprehensively tested and accurately implemented. What I found particularly beneficial was the quick feedback loop TDD provided. Writing tests, observing and fixing failures happened in minutes, offering rapid and actionable insights into any issues. This constant feedback accelerated my development pace and helped me spot problems early.

Having a suite of tests gave me confidence in my code. I could make changes and refactor without worrying about breaking things, as the tests acted as my safety net against regressions. Collaborating with team members became smoother due to the tests serving as documentation. They communicated the expected behavior of functions and features, ensuring a shared understanding. Moreover, TDD allowed me to be flexible with design adjustments as the project evolved. This adaptability was crucial when requirements shifted slightly, and the modular nature of testing made it easier to pivot and refactor accordingly. The systematic approach provided by TDD brought order to the development process, and the organized way of writing tests before code minimized chaos.

Perhaps one of the most valuable aspects was enhanced debugging. When a test failed, it pinpointed the exact issue, making troubleshooting more efficient and targeted. As a result, I could identify and rectify problems swiftly. Overall, TDD's rigorous testing regimen boosted the quality assurance of the final product. The game's functionalities turned out to be reliable and solid, reducing the chances of users encountering unexpected issues. All in all, the experience of implementing the game using TDD highlighted its effectiveness in fostering a disciplined and organized development process, resulting in a robust and thoroughly tested outcome.

## 4.1.2 WHAT WENT WRONG

Reflecting on the implementation journey of the "Guess the Number" game with Test-Driven Development (TDD), a few areas presented challenges. One notable aspect was the initial learning curve associated with TDD. As a newcomer to this approach, it took me some time to fully grasp the methodology and the sequence of writing tests before the actual code. This occasionally led to a slower start as I navigated this new way of thinking.

Another aspect that posed some difficulties was the intricate nature of crafting test cases. While TDD encourages comprehensive test coverage, there were instances where I struggled to capture every possible scenario in the tests. Ensuring that the tests encompassed all edge cases and corner scenarios required extra consideration and effort.

Additionally, the iterative nature of TDD sometimes causes a bit of back-and-forth. While this is inherent to TDD's approach of writing tests incrementally, it occasionally led to adjustments in both the tests and the code to accommodate new insights or requirements that emerged later in the development process.

Despite these challenges, I found that each hurdle provided valuable learning experiences. The learning curve eventually transformed into a deeper understanding of TDD's advantages. The complexity of crafting test cases led me to think critically about the various scenarios, ultimately enhancing my testing skills. And the iterative process, while sometimes requiring extra steps, ultimately contributed to a more refined final product. Ultimately, what went wrong turned into opportunities for growth and improvement, aligning with the overarching spirit of iterative development encouraged by TDD.

## 4.2 FUTURE IMPROVEMENTS TO THE GAME

As I reflect on the development journey of the "Guess the Number" game using Test-Driven Development (TDD), I can't help but envision future possibilities for enhancing the game even further. One area that stands out is the user interface. While the current interface is functional and intuitive, there's room to elevate the visual appeal and user experience. Incorporating more engaging graphics and animations could add an extra layer of excitement to the game.

Moreover, expanding the game's complexity could make it even more captivating. Introducing different game modes or difficulty levels, each with its own rules and challenges, could cater to a wider range of players. This expansion could require modifications to the existing game logic, prompting a revisit of the TDD process to ensure the new functionalities are well-tested.

Another avenue for improvement lies in incorporating social features. Adding the ability to compete with friends, share scores, or even integrate the game with online leaderboards could foster a sense of community and healthy competition among players.

Furthermore, enhancing accessibility features could make the game more inclusive. Implementing features like adjustable font sizes, alternative input methods, or compatibility with screen readers could make the game enjoyable for a broader audience.

Lastly, considering the technological landscape, creating a mobile version of the game could extend its reach to users on various devices. This endeavor would require adapting the user interface to fit smaller screens and touch interactions while maintaining the core gameplay mechanics.

In conclusion, the journey of developing the "Guess the Number" game using TDD has left me excited about the potential future improvements. With avenues to refine the user interface, introduce new complexities, embrace social aspects, enhance accessibility, and explore new platforms, the game has the potential to evolve into an even more captivating and inclusive experience.

## 4.3 FLAKE8 AND PYLINT SCORE DETAILS

* Flake8 main.py

A screenshot of a computer

Description automatically generated

Figure 4. : Flake8 main.py

* Pylint main.py

A screenshot of a computer

Description automatically generated

Figure 4. : Pylint main.py

* Test Passing test\_main.py

A screenshot of a computer

Description automatically generated

Figure 4. : test\_main.py test passing

* Flake8 test\_main.py

A screenshot of a computer

Description automatically generated

Figure 4. : Flake8 test\_main.py

* Pylint test\_main.py

A screenshot of a computer

Description automatically generated

Figure 4. : Pylint test\_main.py

## 4.4 GITHUB REPOSITORY LINK

GitHub Repository Link -

<https://github.com/NithyaYamsinghe/PRT582_Software_Unit_Testing_Report>