**PRT 582: SOFTWARE ENGINEERING: PROCESS AND TOOLS**

**Number Guessing Game**

**By**

Uttam Sapkota

S373292

Contents

[Introduction 3](#_Toc143995178)

[*Objective* 3](#_Toc143995179)

[*Requirements* 3](#_Toc143995180)

[*Testing Tools* 3](#_Toc143995181)

[Process 4](#_Toc143995182)

[Conclusion 9](#_Toc143995183)

# Introduction

## Objective

The objective of the number guessing game is to create an interactive program in which the player gets multiple entries to guess a randomly generated four-digit number. The game gives the player hints after each guess to help them figure out the correct number. A circle is shown when one digit is correct and is in the right spot. A ‘x’ is shown when one digit is correct but in the wrong spot. Lastly, ‘-‘ is shown when the digits provided are in the randomly generated number. The game continues until the player either guesses the correct number or decides to quit. Once the player guesses the correct number, the game congratulates the player and the player is given the option to play again or exit.

## Requirements

The key requirements of the game are as follows:

1. **Randomly Generate Number:** The program should generate a four-digit number with unique digits at the beginning of each game.
2. **User Input:** The program prompts the user to input their guess for the four-digit number.
3. **Input Validation:** The program should validate user inputs to ensure they are four-digit numbers or the 'quit' command.
4. **Hints:** After each guess, the program provides hints using 'circle' and 'x' symbols to indicate correct digits in the correct position and correct digits in the wrong position, respectively. The program also shows “-“ to show that the digit entered in that position is not in the corresponding index of the randomly generated number to make it easier for user to see.
5. **Game Continuation:** The game continues until the player guesses the correct number or chooses to quit by entering 'quit'. Once the user guesses the number correctly, the user is congratulated and is shown the number of attempts required to guess to the correct number.
6. **Play Again Option:** After the game ends, the player is given the option to play again. The game restarts if the player chooses to play again and exits if the player chooses not to. The program also should validate user inputs when asking the user if they want to play again at the end of the game to ensure the input is ‘yes’ or ‘no’.

The code developed implements aforementioned requirements by utilizing functions, loops, input validation, hints, and the play again mechanism.

Overall, the game provides an interactive and engaging experience for the player, challenging them to guess the correct number using the given hints and logic.

## Testing Tools

I used *unittest* testing framework that comes built-in with Python and is part of the standard library. It provides a structured and organized way to write and run tests for any python code.

# Process

I used the concept of test-driven development (TDD) using Unit Testing to write tests before writing the actual code to ensure that the program developed meets the specified requirements and maintains its functionality as I made changes.

There were 6 requirements for this program as listed in requirements section. The following section describes how I used TDD and Unit testing method to satisfy each of the aforementioned requirements.

1. Randomly Generate Number: The program should generate a four-digit number with unique digits at the beginning of each game.

For this requirement, I imported the random module of python and generated 4 digits random number using a sample of numbers from 0 to 9 with a size of 4. I used the random function to create a string of the 4 digits each separated by “ “ null value in order to make searching through each digit and comparing it to the corresponding digit in user’s guess easier.

This is handled by the code – screeshot below:



This satisfies the requirement #1.

1. User Input: The program prompts the user to input their guess for the four-digit number.
2. Input Validation: The program should validate user inputs to ensure they are four-digit numbers or the 'quit' command.

Once the random number is generated, I then ask the user to enter their 4-digit guess. This input is then validated to ensure that it is indeed a 4 digit integer. The input is also validated to ensure that the program exits if the user enters ‘quit’ instead of the 4 digits.

This portion of the requirement is handled by the code – screenshot below:

A screen shot of a computer program

Description automatically generatedI first generated the test case in the unit testing python file in Visual Studio. I checked for 4 possible scenarios:

1. Input is a 4 digit integer
2. Input is more than 4 digits
3. Input is less than 4 digits
4. Input is not an integer

The program will only run if the input is 4 digits and is an integer.

Once the possible cases were developed, I used the code – screenshot below – to ask the user to input a number or ‘quit’ to exit from the program.

A computer screen shot of text

Description automatically generated

***Case 1: User inputs Quit***

* Once the user entered a number, the program firstly checks to see if the user wanted to quit the program by checking if the input is ‘quit’.
* I ensured that all possible versions or capitalization of quit was accepted by using .lower() python module to check if the lowered input included ‘quit’.
* If the user entered ‘quit’, the program thanks the user for playing the game and the game exits.

***Case 2: User provides an input other than ‘quit’***

* The program checks if the length of the input is 4. I use a while loop to ensure that the input prompt is shown until the user enters a 4 digit integer.
* The program ensures that the user this by checking two conditions in the while loop

1. length of the input does not equal 4
2. Boolean value of guess.isdigit() is false (invoked by using not in python)

I generated a truth table to outline all possible input combinations (as listed in the unit testing code above) and their corresponding outputs. In this case, the conditions are related to the input and checking whether it's valid or not. The truth table is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | guess.lower() == 'quit' | len(guess) != 4 | not guess.isdigit() | Output |
| "1234" | False | False | False | Continue (valid input) |
| "quit" | True | - | - | Return (ends the game) |
| "12345" | False | True | False | Print "Invalid input..." (not valid) |
| "123a" | False | True | True | Print "Invalid input..." (not valid) |
| "123" | False | True | False | Print "Invalid input..." (not valid) |
| "" | False | True | True | Print "Invalid input..." (not valid) |

Note:

* For "quit", the game ends. Other conditions are not checked.
* For valid inputs, the loop continues.
* For invalid inputs (not 4 digits or not digits at all), the loop prints the error message and continues.
* The "Output" column describes what the program does in response to each combination of inputs and conditions.

These input validation checks ensure that only the correct input is accepted and the program runs smoothly. And thus satisfy requirements 2 and 3.

1. Hints: After each guess, the program provides hints using 'circle' and 'x' symbols to indicate correct digits in the correct position and correct digits in the wrong position, respectively. The program also shows “-“ to show that the digit entered in that position is not in the corresponding index of the randomly generated number to make it easier for user to see.

Once a valid input is entered by the user, the program creates a list with both randomly generated number (actual\_number) and user’s guess (guess) as well as an empty hints list.

The program then uses ***for*** loop to check if each digit in the randomly generated number equals to its corresponding digit in the number user entered.

* If the digit user entered is CORRECT for the index iteration and also in the right spot, ‘circle’ is added to the hints list.
* If the digit user entered IS IN the randomly generated number but in a wrong place (index), ‘x’ is added to the hints list.
* If the digit user entered IS NOT in the randomly generated number in any index, “-“ is added to the hints list to make it easier for the user to read the hints in 4 digit format.

Once all 4 digits in guessed numbers are compared with digits in user’s guess, the ***for*** loop returns the list of hints.

Requirement 4 is performed by the code – screenshot below – in unit test (TDD) first:

A screen shot of a computer code

Description automatically generatedI first generated the test case in the unit testing python file in Visual Studio. I checked for 4 possible scenarios:

1. The guess is correct
2. All digits in wrong position
3. Some digits in right and some in wrong position
4. No digits match

Once the possible cases were developed, I used the code – screenshot below – to build the hints list after comparing each digit in guessed number to randomly generated 4 digits.

A computer screen shot of a code

Description automatically generated

Below I have created a table to show how the program actually creates the hint list after comparing every digit assuming the randomly generated number is 1234 and the users guess is 1253

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| actual\_digit | guessed\_digit | | Condition Met | Output |
| '1' | | '1' | actual\_digit == guessed\_digit | 'circle' |
| '2' | | '2' | actual\_digit == guessed\_digit | 'circle' |
| '3' | | '5' | guessed\_digit in actual\_digit == FALSE | '-' |
| '4' | | '3' | guessed\_digit in actual\_digit == TRUE | 'x' |

In this truth table, each row represents a comparison between the actual\_digit and the corresponding guessed\_digit. The "Condition Met" column describes whether the condition for a 'circle' or 'x' hint is met, and the "Output" column indicates the resulting hint.

The hints list is returned after all the digits are compared and hints are appended to the list. With final hints list looking like the ones as shown in the table below:

|  |  |  |
| --- | --- | --- |
| actual\_number | guess | hints |
| '1234' | **'1234'** | **['circle', 'circle', 'circle', 'circle']** |
| '1234' | **'4321'** | **['x', 'x', 'x', 'x']** |
| '1234' | **'1243'** | **['circle', 'circle', 'x', 'x']** |
| '1234' | **'5678'** | **['-', '-', '-', '-']** |
| ‘1234’ | **‘1253’** | **[‘circle’, ‘circle’, ‘-‘, ‘x’]** |

These digits comparison and hints list therefore satisfy requirement 4.

1. Game Continuation: The game continues until the player guesses the correct number or chooses to quit by entering 'quit'. Once the user guesses the number correctly, the user is congratulated and is shown the number of attempts required to guess to the correct number.

Once the user inputs the first valid input, I initiate the counter using ‘attempts’ variable. Attempts with invalid input are not counted.

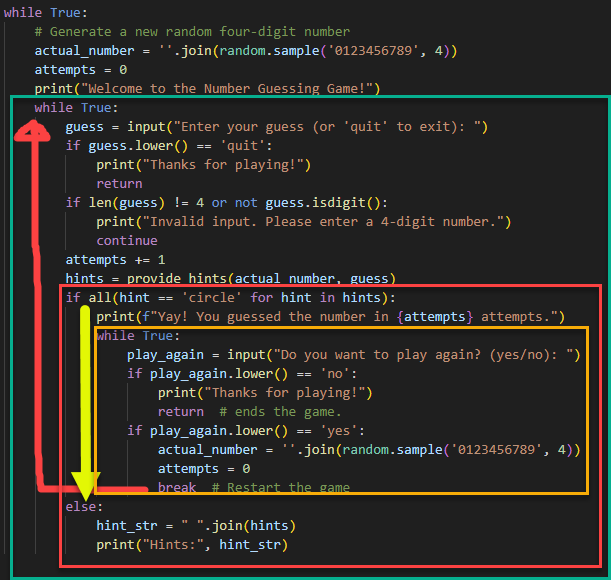
1. Play Again Option: After the game ends, the player is given the option to play again. The game restarts if the player chooses to play again and exits if the player chooses not to. The program also should validate user inputs when asking the user if they want to play again at the end of the game to ensure the input is ‘yes’ or ‘no’.

The program uses hints to determine if the user’s guess is correct. If the hints for all four digits corresponding to the user’s input are circle, then the user have guessed the random number correctly. The program uses this method to determine and congratulate when the user guessed the number correctly.

Once the user guesses the number correctly, the program also shows the number of attempts it took the user to guess the number correctly.

The program also asks the user if the user wants to play the game again. The only acceptable inputs are ‘yes’ and ‘no. ‘yes’ restarts the game and ‘no’ exits the game.

This is achieved by nested while loops as shown in the code snippet below:

*lists of loops used for game continuity:*

Blue rectangle: while loop

Red rectangle: if else statement

Yellow rectangle: nested while loop

Hints returned from hints function in page 7 is used to check if all the hint in hints are are circle. If all the digits are circles, this means the user guessed the number correctly. The program will congratulate the user and show the number of attempts taken by the user to guess the number correctly *(depicted by code enclosed by red rectangle)*

New while nested loop (depicted by code enclosed by yellow rectangle) asks the user if they want to play the game again or not. If the answer is ‘yes’, the game will restart (flow depicted by red arrow in the code snippet above. If the answer is ‘no’, the while statement is broken and the game ends. Validation is incorporated into this code to ensure that the user must say ‘yes’ to play and ‘no’ to quit. Any other input will prompt the user to enter the valid response.

This code satisfies requirements 5 and 6.

# Conclusion

As part of this project, I learned how to create programs using TDD methods and by using testing tool – unittest – in python. I learned a lot of valuable lessons in programming through this exercise such as:

* Random Number Generation: The random module was used to generate random numbers, demonstrating how to create and manage randomness in programs.
* Input Validation: I incorporated input validation in the code to ensure that user inputs are within expected ranges or formats. This helped prevent errors and handle unexpected inputs.
* User-Friendly Interface and interaction: The code provides clear prompts and instructions for the user, making the game more user-friendly and interactive. The code interacts with the user by displaying messages, receiving input, and providing hints. This showcases the importance of user engagement in programs.
* Nested Loops: The code effectively uses nested loops, combining a loop for the game rounds (while True:) with a loop for handling user input and restarting the game.
* String Formatting: The code uses string formatting to display messages with dynamic content, improving the readability of the output.
* Logical Conditions: logical conditions are used to control the flow of the game and make decisions based on user inputs and game state.
* Modularization: I divided the code into separate functions (play\_game and provide\_hints), which makes the code more organized, modular, and easier to understand.

These lessons demonstrate good practices for structuring, organizing, and writing code in a way that is both functional and maintainable. This exercise has helped me learn various fundamentals of software engineering.

**Areas of Improvement:** I had a very difficult time coming up with automated tests within nested while loops even though the tests were just for input validations. Also, testing the state or the exiting behaviour of the program when the user input ‘quit’ or ‘no’ was difficult since I do not know how to do that yet. As I progress in the course, I hope to learn these things and better implement them in my codes.

GitHub Link: