Project Title: Number Guessing Game

Course: Introduction to Problem Solving & Programming

Student Name: Anupma Mishra

Reg. No:- 25BAI11311

Institution: VIT Bhopal

Submission Date: 25 November 2025

1. Introduction

Games are one of the simplest ways to understand programming logic.

The Number Guessing Game is a beginner-friendly Python project where the computer selects a random number, and the user tries to guess it.

This small application demonstrates basic programming concepts such as loops, condition checking, user input, and random number generation.

The project follows a structured development approach and aligns with the requirements of the VITyarthi flipped course.

3. Problem Statement

Users enjoy simple interactive games that test logic and guessing skills.

The problem is to design a small game where the computer thinks of a number and the user must guess it.

The program should respond to the user's input, check whether the guess is correct, and continue the interaction until the correct guess is made.

An additional feature should allow the user to reveal the answer anytime using a special input**.**

4. Functional Requirements

1. The system must generate a random number between 1 and 100.
2. The user must be able to input guesses.
3. If the user types “answer”, the system must reveal the secret number.
4. The program must compare the user’s guess with the secret number.
5. The system must display a success message when the user guesses correctly.
6. Incorrect guesses should prompt the user to try again.
7. The program must terminate after a correct guess or after revealing the answer.

5. Non-Functional Requirements

1. Usability: The interface must be simple, readable, and beginner-friendly.
2. Performance: The program should execute instantly without noticeable delay.
3. Reliability: The program should run correctly for all valid inputs.
4. Maintainability: The code should be clean with proper comments for easy understanding.
5. Error Handling: The program should not crash for the special input “answer”.
6. Portability: The project should run on any system with Python installed.

6. System Architecture

This system follows a simple linear architecture:

User Input -> Input Processing -> Number Comparison -> Output Response

The user gives input through the console.

- The program processes the input to determine whether it's a guess or "answer".
- The system then compares the guess with the generated number.
- A message is displayed based on the result.

7. Design Diagrams

7.1 Use Case Diagram

Actors: User

Use Cases:

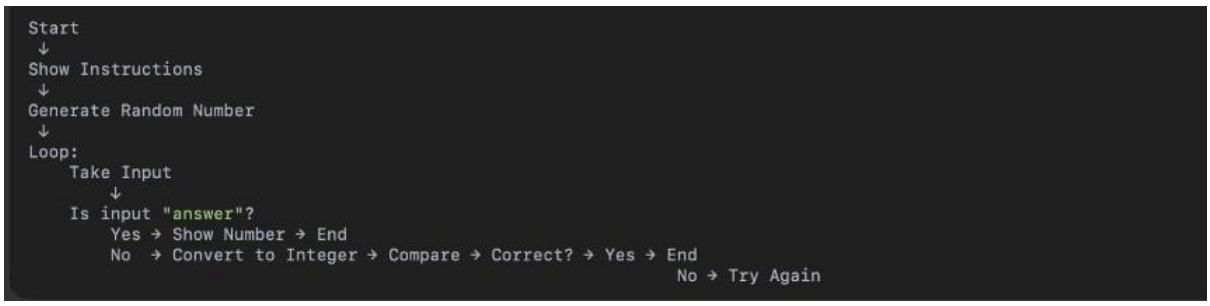
- Enter guess
- Reveal answer
- Get feedback
- Win the game

Use Case Flow:

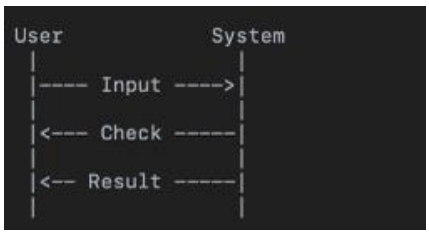
User -> Enter Guess -> System Checks -> Shows Results

User -> Type "answer" -> System Reveals Number

7.2 Workflow Diagram



7.3 Sequence Diagram



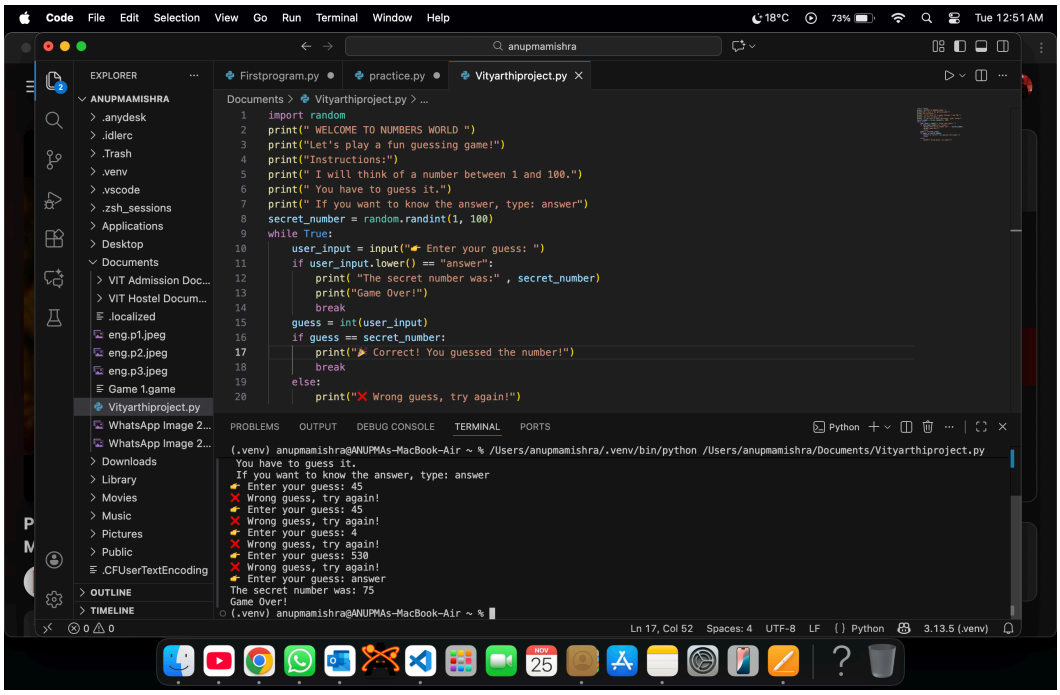
8. Design Decisions & Rationale

- Python was chosen as it is beginner-friendly.
- The random module was used to generate unpredictable numbers.
- A while-loop was selected to allow repeated guessing until correct.
- The “answer” feature was added to improve interactivity and user control.
- Simple prints were used for clear communication with the user.

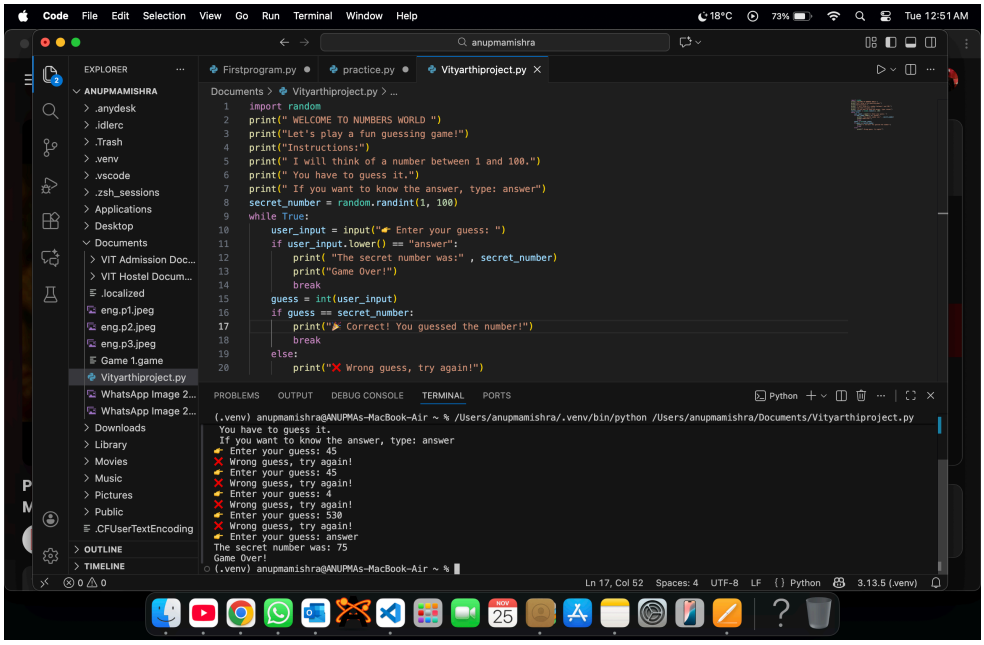
9. Implementation Details

```
import random
print(" WELCOME TO NUMBERS WORLD ")
print("Let's play a fun guessing game!")
print("Instructions:")
print(" I will think of a number between 1 and 100.")
print(" You have to guess it.")
print(" If you want to know the answer, type: answer")
secret_number = random.randint(1, 100)
while True:
    user_input = input("👉 Enter your guess: ")
    if user_input.lower() == "answer":
        print( "The secret number was:" , secret_number)
        print("Game Over!")
        break
    guess = int(user_input)
    if guess == secret_number:
        print("🎉 Correct! You guessed the number!")
        break
    else:
        print("❌ Wrong guess, try again!")
```

10. Screenshots / Results



11.



Testing

Approach

Test Case 1: Correct Guess

- Input: 45
- Secret Number: 45
- Output: Correct!

Test Case 2: Wrong Guess

- Input: 15
- Output: Wrong guess

Test Case 3: Reveal Answer

- Input: answer
- Output: Secret number revealed

Test Case 4: Multiple Guesses

- 1st: 20
- 2nd: 55
- 3rd: 77 (correct)

12. Challenges Faced

- Handling the "answer" keyword without crashing input conversion.
- Making sure the loop stops properly after correct guess.
- Ensuring the code remains simple and readable for beginners.

13. Learnings & Key Takeaways

- Learned how loops, conditionals, and user input work together.
- Understood how the random module generates unpredictable values.
- Improved debugging and logical thinking skills.
- Gained experience writing structured documentation and GitHub repository.
- Understood the importance of user-friendly communication in console applications.

14. Future Enhancements

1. Add hints: "Too high" / "Too low".
2. Add limited attempts (e.g., 7 tries only).
3. Create a GUI version using Tkinter.
4. Add difficulty levels (Easy/Medium/Hard).
5. Add score tracking for multiple rounds.

15. References

1. Python Official Documentation – random module
2. Class notes and lectures
3. VITyarthi course materials