Snake Water Gun Game

Project Report

Submitted by:
HERAMB SUMUKH
Reg. No. 25BME10008

Technology Stack:
Python 3

Date: November 22, 2025

1. INTRODUCTION

This report documents the development of ”Snake Water Gun,” a command-line game developed using the Python programming language. It is a digital variation of the classic ”Rock Paper Scissors” game. The application allows a single player to compete against the computer, which makes random moves, ensuring that each round is unpredictable and engaging.

2. PROBLEM STATEMENT

The objective of this project is to create a digital simulation of the traditional hand game. The system must:

- Accept user input corresponding to the three game choices.
- Generate a random choice for the opponent (computer).
- Compare the two choices based on the game’s predefined rules.
- Declare the correct winner (User, Computer, or Draw).

3. FUNCTIONAL REQUIREMENTS

The system fulfills the following functional requirements:

- **FR1 Input Handling:** The system accepts inputs ’s’ (Snake), ’w’ (Water), or ’g’ (Gun).
- **FR2 Randomization:** The system uses the `random` library to ensure the computer’s move is not pre-determined.
- **FR3 Game Logic:** The system implements the specific win/loss conditions:
  - Snake drinks Water.
  - Water douses Gun.
  - Gun shoots Snake.
- **FR4 Output:** The system displays the choices made by both parties and the final result text.

4. NON-FUNCTIONAL REQUIREMENTS

- **Usability:** The interface is a simple Command Line Interface (CLI) suitable for all users.
- **Performance:** The game logic executes instantly without latency.
- **Reliability:** The game produces a valid result for every possible combination of inputs.

5. SYSTEM ARCHITECTURE

The system follows a linear procedural architecture:

1. **Input Module:** Captures user strings via standard input.
2. **Processing Unit:**

  - Maps string inputs to integers for efficient comparison.
  - Generates a random integer for the computer.
  - Evaluates conditions using Control Flow (if/elif/else).

3. **Output Module:** Prints formatted strings to the console.

6. DESIGN DIAGRAMS

6.1 Use Case Diagram

- **Actor:** Player
- **System:** Snake Water Gun Game
- **Use Cases:** Start Game, Input Choice, View Result.

6.2 Workflow Diagram

`Start` → `Computer Selects (Hidden)` → `User Inputs Choice` → `Map to Integers` → `Compare Logic` → `Print Result` → `End`.

6.3 Sequence Diagram

1. User enters ’s’, ’w’, or ’g’.
2. System validates and converts input.
3. System calls `random.choice()`.
4. System compares values.
5. System prints ”You Win” or ”You Lose”.

7. DESIGN DECISIONS & RATIONALE

- **Language Selection:** Python was chosen for its readability and concise syntax, which is ideal for logic-based scripts.
- **Data Mapping:** A dictionary approach was used (`youDict`) to map user keystrokes (’s’, ’w’, ’g’) to integers (1, -1, 0). This allows the use of integer comparison rather than string comparison, reducing the likelihood of typo-related bugs in the logic.
- **Random Module:** The built-in `random` module is efficient and sufficient for generating non-deterministic behavior required for the game.

8. IMPLEMENTATION DETAILS

The core logic utilizes a nested conditional structure. Below is a snippet of the implementation:

```
import random

# 1: Snake, -1: Water, 0: Gun
computer = random.choice([-1, 0, 1])
```

```python
youstr = input("Enter your choice: ")
youDict = {"s": 1, "w": -1, "g": 0}
you = youDict[youstr]

if(computer == you):
    print("Its a draw")
else:
    if(computer == -1 and you == 1):
        print("You Win!") # Snake drinks Water
    elif(computer == -1 and you == 0):
        print("You Lose!") # Water douses Gun
    # ... remaining logic ...
```

9. SCREENSHOTS / RESULTS

Typical output scenarios:

Scenario 1: Victory

```
Enter your choice: s
You chose Snake
Computer chose Water
You Win!
```

Scenario 2: Defeat

```
Enter your choice: g
You chose Gun
Computer chose Water
You Lose!
```

10. TESTING APPROACH

Manual testing was performed to verify all possible permutations of the game state (9 total states):

User	Computer	Result
Snake	Snake	Draw
Snake	Water	Win
Snake	Gun	Lose
Water	Snake	Lose
Water	Water	Draw
Water	Gun	Win
Gun	Snake	Win
Gun	Water	Lose
Gun	Gun	Draw

11. CHALLENGES FACED

- **Input Validation:** The initial prototype would crash if the user entered a key other than ’s’, ’w’, or ’g’.

- **Logic Complexity:** Ensuring that all 9 conditions were covered without logical redundancy required careful structuring of the `if-elif` ladder.

12. LEARNINGS & KEY TAKEAWAYS

- Gained proficiency in Python control flow and conditional logic.
- Understood the importance of mapping internal data representations (integers) to external user interfaces (strings).
- Learned to use the Python standard library (`random`) effectively.

13. FUTURE ENHANCEMENTS

- **Looping Mechanism:** Allow the user to play multiple rounds without restarting the program.
- **Score Tracking:** Maintain a running score (e.g., Best of 5).
- **GUI:** Implement a graphical interface using Tkinter or PyGame for a better user experience.

14. REFERENCES

- Python 3 Documentation - `random` module.
- Standard Game Rules for Rock-Paper-Scissors variants.