Final Project Report — Programming Fundamentals

University Name: FAST NUCES Karachi

Department: Department of Computer Science

Course: Programming Fundamentals

Project Title: Word Guessing Game

Submitted By: Mahnoor Farrukh (25k-0742) & Bhoomika Rohra (25k-0784)

Submitted To: Ms. Izzah Salam

Semester: Fall 2025

Date: 23rd November, 2025

Abstract:

"Guess It or L + Ratio" is an interactive word-guessing game developed as the semester-end project for Programming Fundamentals. The game combines logic, quick thinking, and an engaging console interface using colored text, hints, sound effects, and a dynamic battery-style attempt bar. Words and meanings are loaded from external text files, and the game prevents repeated guesses, supports full-word guessing, and provides clear feedback for every action. The project demonstrates practical use of file handling, arrays, loops, conditional logic, functions, and console manipulation.

1. Introduction

This project was created to turn a simple word-guessing game into something more fun, responsive, and visually appealing. Instead of plain text output, the game uses cursor positioning, colors, animated attempt bars, and beeps to create a smooth gameplay experience in the terminal. The player receives a hint, guesses letters, and watches the display update in real-time. The game also handles wrong guesses with custom messages and prevents repeated inputs.
The purpose of this project is to apply core Programming Fundamentals concepts in a practical and creative way: using files to store words and meanings, file handling to load data, arrays for tracking guesses, and functions to organize the entire logic.

2. Objectives

- To familiarize students with the tech vocabulary through gamification.

· To design a user-friendly word-guessing game using C.
· To apply Programming Fundamentals concepts (loops, conditionals, functions, arrays, structures, and file handling) into a single cohesive program.
· To load words and hints from external text files for easy expansion and clean data management.
· To provide a responsive console interface with colors, cursor movement, and sound effects.

3. System Design

System Overview

Flow of the Program:

Start
→ Display Game Menu
→ User Selects Difficulty Level
→ Load Words From Text File Based on Difficulty
→ Randomly Select One Word & Show Hint
→ Display Blank Underscore Pattern
→ Keep Accepting User Input (letters or full-word guesses)
→ Check for repeated guesses / correct / incorrect attempts
→ Update battery-style attempt bar and messages
→ End the game when the word is guessed or attempts run out

→ Show final message and reveal word (if needed)
→ Exit

Flow of the Program:

1) Start the program
2)Display the game title and difficulty menu.
3)Take difficulty choice from the user.
4)Load the corresponding text file (easy.txt, medium.txt, or hard.txt).
5)Randomly select a word and display its hint.
6)Initialize attempts, blank display, and guessed-letter tracking.
7)Repeat until the word is guessed or attempts become zero:
8)Ask the user for a letter or full-word guess.
9)Check if input is valid.
10)Check if the letter was already guessed.
11)If correct → reveal matching letters and play success sound.
12)If incorrect → reduce attempts, show message, update battery bar.
13)If the word is fully guessed → show success message.
14)Otherwise → display "out of attempts" and reveal the word.
15)End the program.

Input & Output

Input:
Difficulty level (Easy / Medium / Hard)
User guesses:
Single letters OR full-word guesses

Output:
Random word (hidden with underscores)
Meaning of the selected word (as a hint)
Updated letter display on every correct guess
Warning messages for repeated or invalid inputs
Wrong/Right messages with colors
Battery-style attempt bar
Final result (win/lose)
The correct word (if user fails)

4. Implementation

Language: C

Compiler/IDE: Code::Blocks / Dev C++ / GCC/ VS Code

Key Features:

1) Difficulty-based word loading
(Easy / Medium / Hard text files with word:meaning format)

2)Random word selection with meaning displayed as a hint

3)Dynamic battery-style chance bar that changes color based on remaining attempts

5)Real-time console UI(cursor positioning, overwriting text, centered blanks)

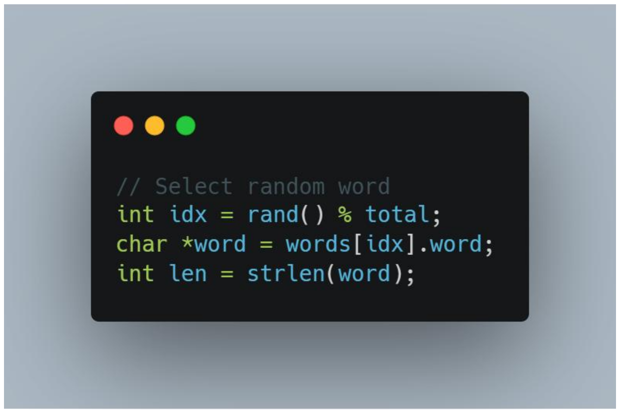
6)Detection of repeated guesses
(prevents user from entering the same letter twice)

7) Color-coded messages:
a)Correct guess (green)
b)Wrong guess (red)
c)Warnings / invalid input (yellow)
d)Sound feedback using Beep ( )
  (right, wrong, and win tones)

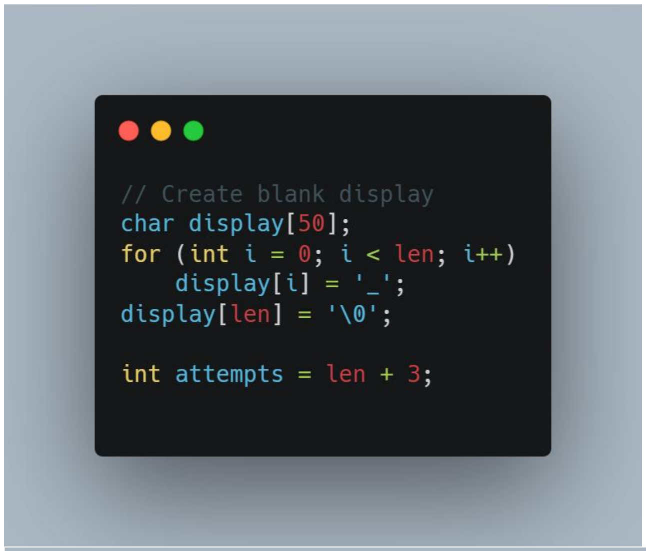
8)Win/Lose end screens with complete word reveal

Code Snippet

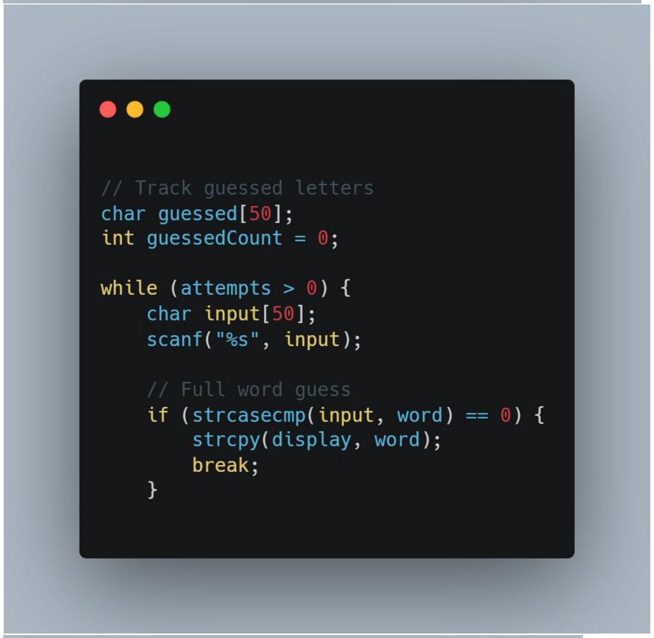
Following snippets cover the cover logic of the game (the comments explain the functionality of each)



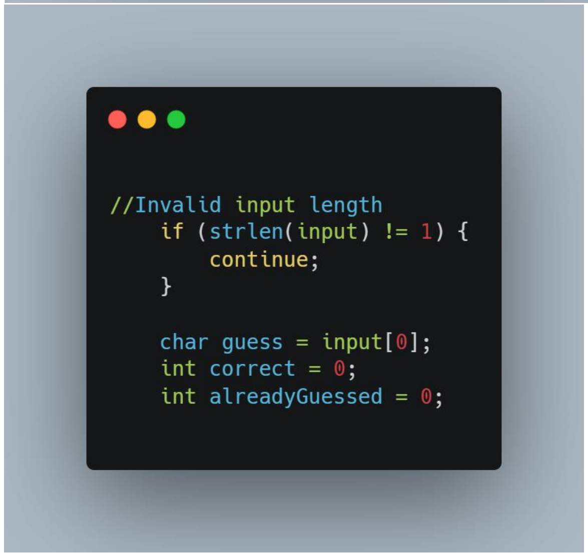
```c
// Select random word
int idx = rand() % total;
char *word = words[idx].word;
int len = strlen(word);
```



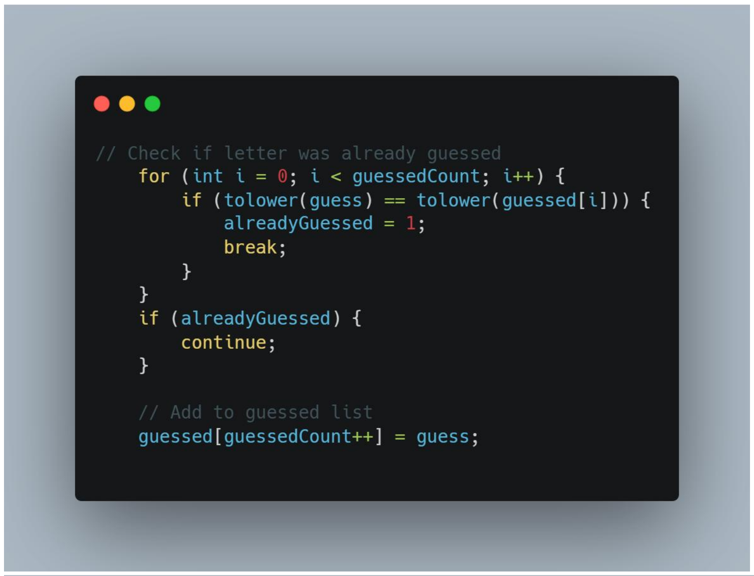
```c
// Create blank display
char display[50];
for (int i = 0; i < len; i++)
    display[i] = '_';
display[len] = '\0';

int attempts = len + 3;
```



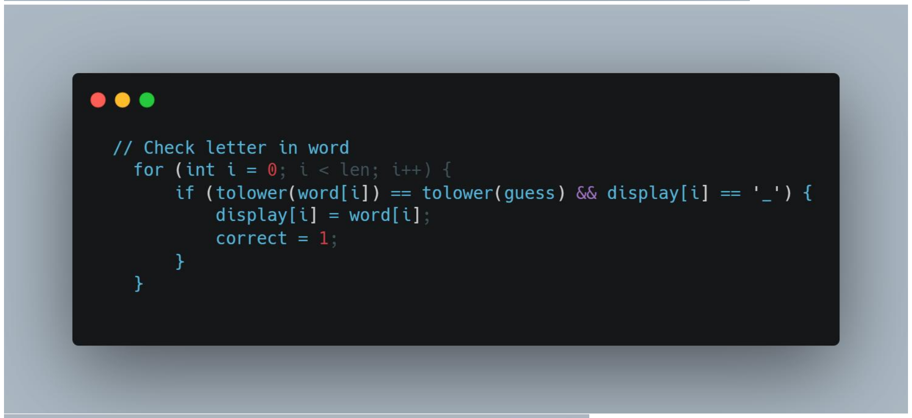
```c
// Track guessed letters
char guessed[50];
int guessedCount = 0;

while (attempts > 0) {
    char input[50];
    scanf("%s", input);

    // Full word guess
    if (strcasecmp(input, word) == 0) {
        strcpy(display, word);
        break;
    }
```



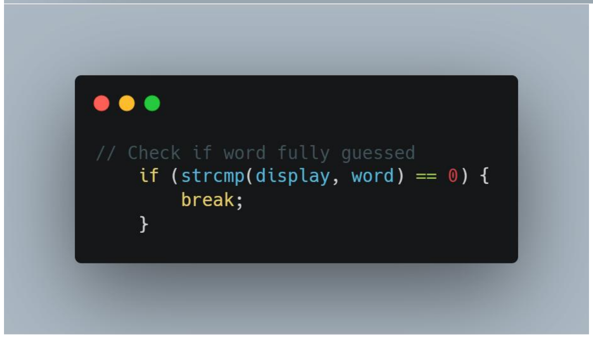
```c
//Invalid input length
    if (strlen(input) != 1) {
        continue;
    }

    char guess = input[0];
    int correct = 0;
    int alreadyGuessed = 0;
```



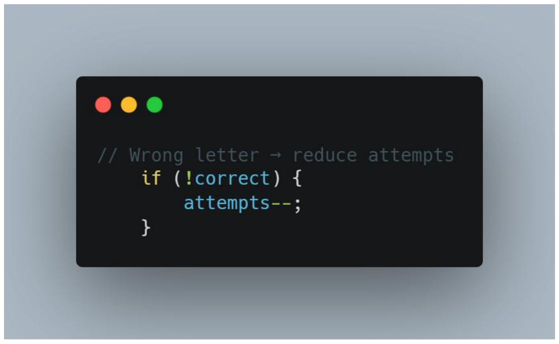
```c
// Check if letter was already guessed
    for (int i = 0; i < guessedCount; i++) {
        if (tolower(guess) == tolower(guessed[i])) {
            alreadyGuessed = 1;
            break;
        }
    }
    if (alreadyGuessed) {
        continue;
    }

    // Add to guessed list
    guessed[guessedCount++] = guess;
```



```c
  // Check letter in word
    for (int i = 0; i < len; i++) {
        if (tolower(word[i]) == tolower(guess) && display[i] == '_') {
            display[i] = word[i];
            correct = 1;
        }
    }
```

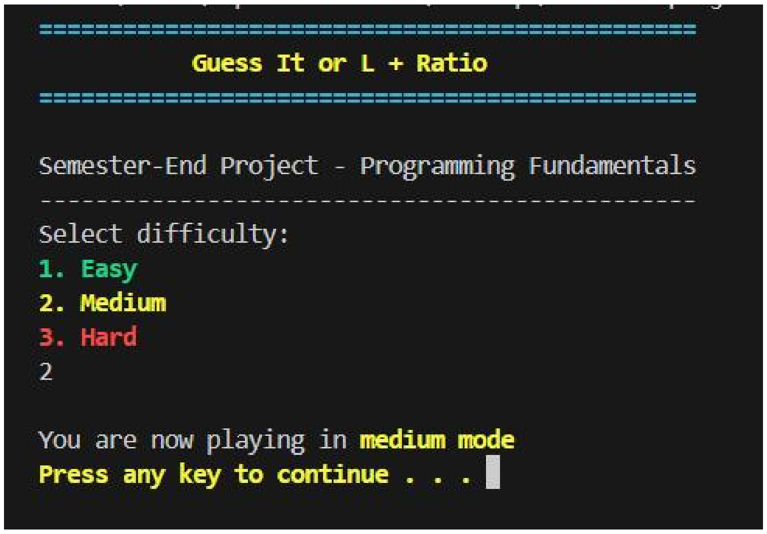


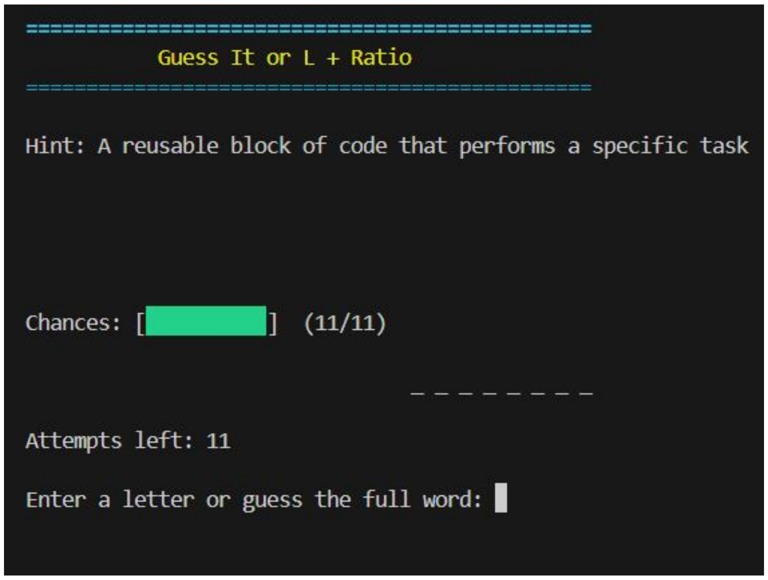
```c
// Check if word fully guessed
    if (strcmp(display, word) == 0) {
        break;
    }
```

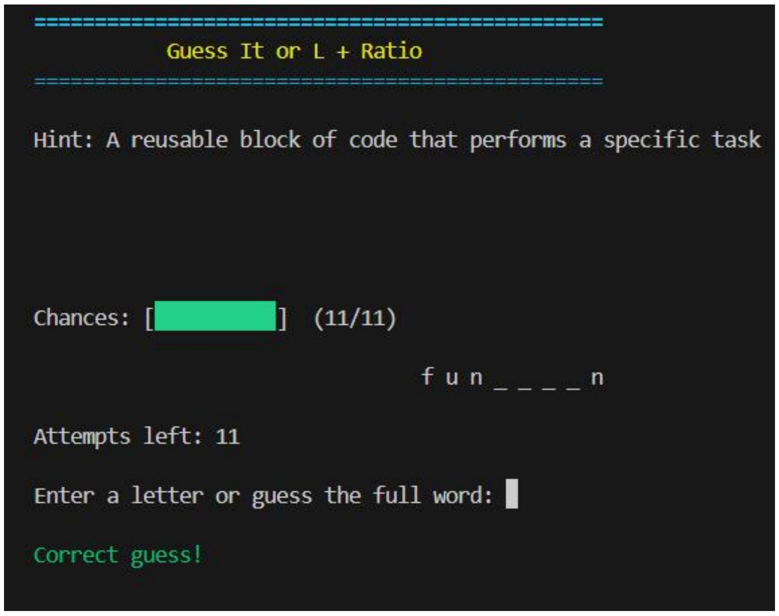


```c
// Wrong letter → reduce attempts
    if (!correct) {
        attempts--;
    }
```

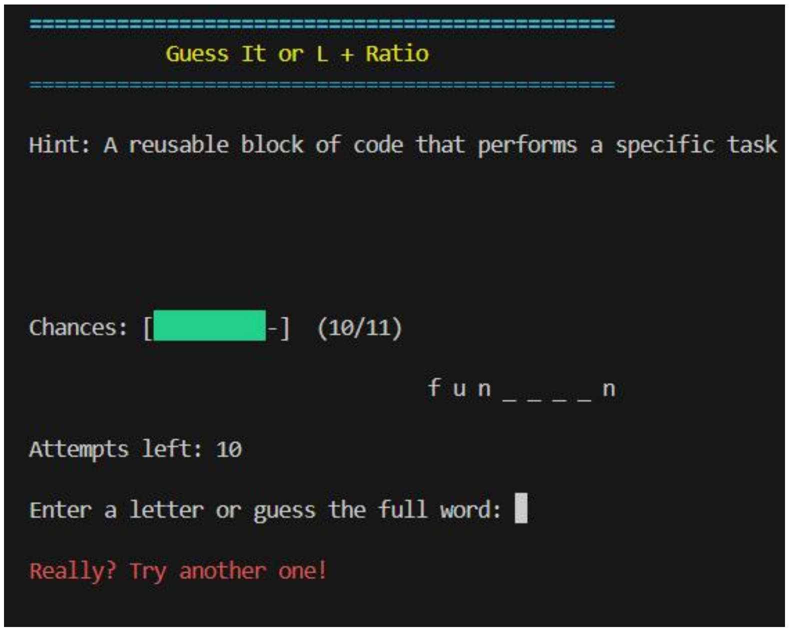
Sample Output


```
================================================================
              Guess It or L + Ratio
================================================================

Semester-End Project - Programming Fundamentals
----------------------------------------------
Select difficulty:
1. Easy
2. Medium
3. Hard
2

You are now playing in medium mode
Press any key to continue . . .
```

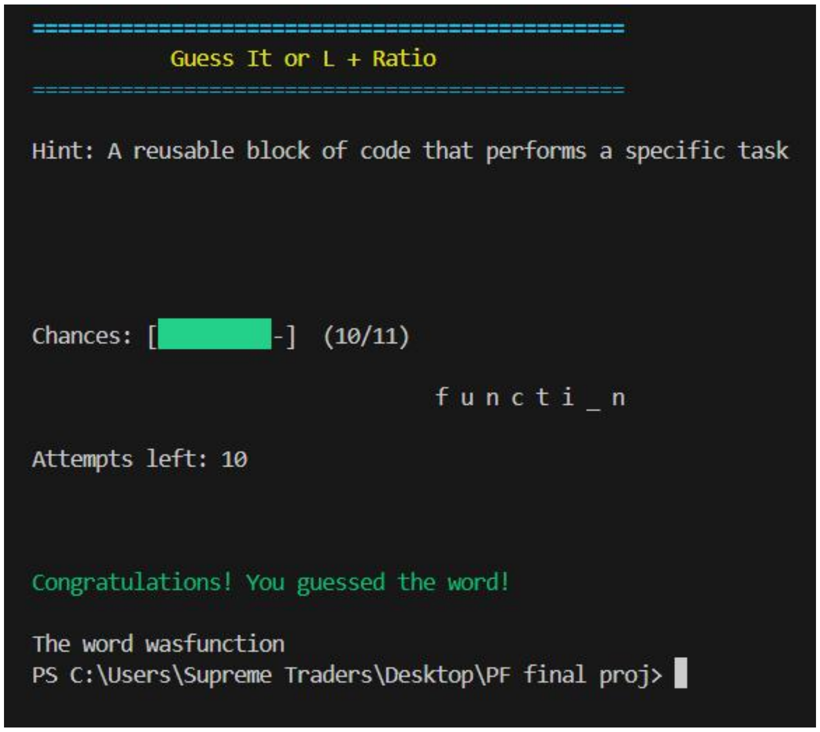

```
==================================================
           Guess It or L + Ratio
==================================================

Hint: A reusable block of code that performs a specific task




Chances: [███████████]  (11/11)

                                _ _ _ _ _ _ _ _

Attempts left: 11

Enter a letter or guess the full word:
```



```
================================================
          Guess It or L + Ratio
================================================

Hint: A reusable block of code that performs a specific task



Chances: [██████████]  (11/11)

                                fun____n

Attempts left: 11

Enter a letter or guess the full word: 

Correct guess!
```



```
================================================
          Guess It or L + Ratio
================================================

Hint: A reusable block of code that performs a specific task



Chances: [█████████-]  (10/11)

                                fun____n

Attempts left: 10

Enter a letter or guess the full word: 

Really? Try another one!
```



```
==============================================
          Guess It or L + Ratio
==============================================

Hint: A reusable block of code that performs a specific task



Chances: [█████████-]  (10/11)

                              f u n c t i _ n

Attempts left: 10


Congratulations! You guessed the word!

The word wasfunction
PS C:\Users\Supreme Traders\Desktop\PF final proj> █
```

5. Testing & Results

Test No	Input	Expected Output	Actual Output	Status
1	Guessing correct letters: a, r, t for word “art”	Letters revealed correctly; attempts remain unchanged	Works Correctly	✓
2	Entering already-guessed letter *(e.g., guessing “a” twice)*	Display warning: *"You already guessed that letter!"*; attempts should NOT decrease”	Warning displayed; attempts unchanged	✓
3	Wrong guess: letter not in word	Random wrong-message + attempts decrease by 1	Functions Correctly	✓
4	Full correct	Instant reveal full word + win	Works Correctly	✓

	word guess	message		
5	Full wrong word guess	Should reduce the attempts by 1 and show wrong message	Works correctly	✓
6	Entering more than one letter (valid)	Show warning: “Enter a single letter or guess the full word”	Message appears correctly	✓
7	Running game when file has N words	Random word selected only from loaded difficulty file	Correct behaviour	✓
8	Running out of attempts	Show “Out of attempts! The word was _”	Output matches expected	✓

The game performed successfully for all test cases.It handled correct/wrong guesses, repeated letters, and full-word guesses accurately. All feedback messages and attempt deductions were correct. Gameplay was smooth, and performance remained stable with instant execution.

6. Conclusion, Limitations & References

Conclusion

The Guess It or L + Ratio game successfully demonstrates the application of fundamental C programming concepts. It combines file handling, arrays, loops, conditional statements, functions, and console manipulation to create an interactive word-guessing game. The project strengthened understanding of user input validation, randomization, and basic UI/UX design using the console.

Limitations

- No graphical interface — purely console-based.
- Words must be preloaded in text-files
- Limited to single player mode only

Future Enhancements

- Implement a graphical interface using libraries like SDL or ncurses.
- Add multiplayer support or competitive mode with timers and scores.
- Include dynamic word packs with categories, hints, and difficulty scaling.
- Add sound effects, animations, and better visual representation of the game state.

References

- Let Us C by Yashavant P. Kanetkar
- https://www.programiz.com/c-programming
- https://www.geeksforgeeks.org/c-programming-language/