BUILDING CROSS WORD BUZZLE GAME USING C PROGRAMMING

ARUL GNANAKUMAR.R
95072317016

Abstract:

This report outlines the development of a word search game implemented in C programming. The game provides an interactive platform for users to engage in a classic puzzle-solving activity. It allows players to find hidden words in a randomly generated grid of letters, enhancing cognitive skills and providing entertainment. The proposed system improves upon existing implementations by incorporating random word placement in multiple orientations and providing a user-friendly interface for gameplay.

Problem Statement:

Traditional word search games are often limited in functionality and can become monotonous due to predictable layouts and word placements. Existing systems may lack features that enhance user engagement and make the gameplay experience more enjoyable. Additionally, some implementations may not effectively handle user inputs or manage word placements, leading to a less immersive experience. This project aims to address these limitations by creating a more dynamic and interactive word search game.

Existing System

Current word search game implementations often focus on basic functionality, providing static grids with fixed words. Some common limitations include:

- Static Word Placement: Words are often placed in a fixed direction (horizontal or vertical) and lack variety.
- Limited User Interaction: Many implementations do not provide meaningful feedback to users or do not handle incorrect inputs effectively.
- Lack of Randomization: Some systems use pre-defined grids, leading to predictable gameplay.
- Poor Usability: Interfaces may not be intuitive, making it challenging for users to engage fully with the game.

Existing Systems:

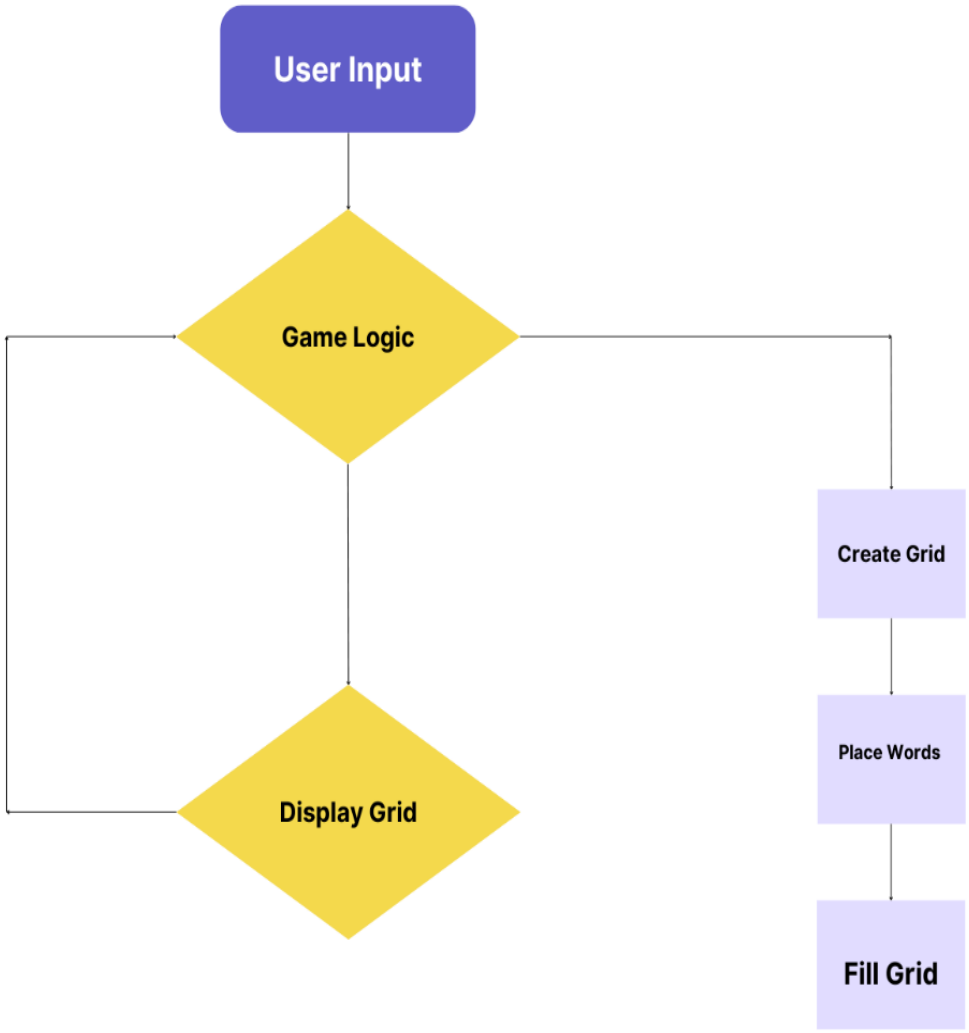
Existing word search game systems typically offer basic functionalities, enabling users to find words hidden in a grid of letters. These systems often use static grids with predefined layouts, limiting replay value as players can easily memorize word placements. Additionally, many implementations restrict word orientation to only horizontal or vertical placements, reducing the challenge of the puzzles. The user interfaces are often simplistic, relying on basic command-line prompts or minimal graphical representations, which can lead to an uninspiring experience. Feedback mechanisms are usually limited, providing only basic notifications on word discovery without tracking user progress or offering hints. Furthermore, most systems lack randomization, leading to predictable gameplay that diminishes user engagement over time. Overall, while existing systems fulfill the fundamental purpose of word search puzzles, they often fall short in enhancing user interaction and long-term enjoyment.

Proposed System:

The proposed word search game enhances user experience by introducing several features:

- Randomized Word Placement: Words can be placed horizontally, vertically, or diagonally, and the placement is randomized to create a unique experience each time the game is played.
- Dynamic User Input Handling: The system effectively manages user inputs, providing feedback for correct and incorrect guesses while allowing players to exit gracefully.
- Filling Empty Spaces: After word placement, the grid is filled with random letters to create a more challenging puzzle.
- Expandable Architecture: The system allows for the easy addition of new words or features in the future, enabling scalability**.**

BLOCK DIAGRAM:



PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#define SIZE 15
#define MAX_WORDS 10
#define MAX_WORD_LENGTH 20

// Function to create an empty grid
void create_grid(char grid[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            grid[i][j] = ' ';
        }
    }
}

// Function to display the grid
void display_grid(char grid[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%c ", grid[i][j]);
        }
        printf("\n");
    }
    printf("\n");  // Add a line break for better readability
}

// Function to fill empty spots in the grid with random letters
void fill_empty(char grid[SIZE][SIZE]) {
    char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (grid[i][j] == ' ') {
                grid[i][j] = letters[rand() % 26];
            }
        }
    }
}

// Function to check if a word can be placed without collision
int can_place_word_horizontally(char grid[SIZE][SIZE], char *word, int row, int col) {
    if (col + strlen(word) > SIZE) return 0;
    for (int i = 0; i < strlen(word); i++) {
        if (grid[row][col + i] != ' ' && grid[row][col + i] != word[i]) {
            return 0;
        }
```

```
 }
    return 1;
}

int can_place_word_vertically(char grid[SIZE][SIZE], char *word, int row, int col) {
    if (row + strlen(word) > SIZE) return 0;
    for (int i = 0; i < strlen(word); i++) {
        if (grid[row + i][col] != ' ' && grid[row + i][col] != word[i]) {
            return 0;
        }
    }
    return 1;
}

int can_place_word_diagonally(char grid[SIZE][SIZE], char *word, int row, int col) {
    if (row + strlen(word) > SIZE || col + strlen(word) > SIZE) return 0;
    for (int i = 0; i < strlen(word); i++) {
        if (grid[row + i][col + i] != ' ' && grid[row + i][col + i] != word[i]) {
            return 0;
        }
    }
    return 1;
}

// Function to place a word in the grid
void place_word_horizontally(char grid[SIZE][SIZE], char *word, int row, int col) {
    for (int i = 0; i < strlen(word); i++) {
        grid[row][col + i] = word[i];
    }
}

void place_word_vertically(char grid[SIZE][SIZE], char *word, int row, int col) {
    for (int i = 0; i < strlen(word); i++) {
        grid[row + i][col] = word[i];
    }
}

void place_word_diagonally(char grid[SIZE][SIZE], char *word, int row, int col) {
    for (int i = 0; i < strlen(word); i++) {
        grid[row + i][col + i] = word[i];
    }
}

// Function to create and populate the word search puzzle
void create_word_search(char words[MAX_WORDS][MAX_WORD_LENGTH], int
word_count, char grid[SIZE][SIZE]) {
    create_grid(grid);
    for (int i = 0; i < word_count; i++) {
        int placed = 0;
        while (!placed) {
```

```
            int direction = rand() % 3;  // 0: horizontal, 1: vertical, 2: diagonal
            int row = rand() % SIZE;
            int col = rand() % SIZE;

            if (direction == 0 && can_place_word_horizontally(grid, words[i], row, col)) {
                place_word_horizontally(grid, words[i], row, col);
                placed = 1;
            } else if (direction == 1 && can_place_word_vertically(grid, words[i], row, col))
{
                place_word_vertically(grid, words[i], row, col);
                placed = 1;
            } else if (direction == 2 && can_place_word_diagonally(grid, words[i], row,
col)) {
                place_word_diagonally(grid, words[i], row, col);
                placed = 1;
            }
        }
    }
    fill_empty(grid);
}
void play_word_search(char words[MAX_WORDS][MAX_WORD_LENGTH], int
word_count, char grid[SIZE][SIZE]) {
    char found_words[MAX_WORDS][MAX_WORD_LENGTH];
    int found_count = 0;

    // Remove the output of words to find
    // printf("Words to find: ");
    // for (int i = 0; i < word_count; i++) {
    //     printf("%s ", words[i]);
    // }
    // printf("\n");

    while (found_count < word_count) {
        printf("Current Grid:\n");
        display_grid(grid);

        // Get user input
        char guess[MAX_WORD_LENGTH];
        printf("Enter a word you found (or type 'quit' to exit): ");
        scanf("%s", guess);
        for (int j = 0; guess[j]; j++) guess[j] = toupper(guess[j]);  // Convert to uppercase

        if (strcmp(guess, "QUIT") == 0) {
            printf("Thanks for playing!\n");
            break;
        }

        int correct = 0;
        for (int j = 0; j < word_count; j++) {
            if (strcmp(guess, words[j]) == 0) {
```

```
            int already_found = 0;
            for (int k = 0; k < found_count; k++) {
                if (strcmp(found_words[k], guess) == 0) {
                    already_found = 1;
                    break;
                }
            }
            if (!already_found) {
                strcpy(found_words[found_count++], guess);
                printf("Correct!\n");
                correct = 1;
            } else {
                printf("You already found that word!\n");
                correct = 1;
            }
            break;
        }
    }
    if (!correct) {
        printf("Wrong,Try again!\n");
    }

    printf("\n");
  }
}

// Main function
int main() {
  srand(time(NULL));  // Seed random number generator

  char words[MAX_WORDS][MAX_WORD_LENGTH] = {
    "DATASTRUCTURE",
    "DATASCIENCE",
    "PHYSICS",
    "CHEMISTRY",
    "PYTHON",
    "KEDAR",
    "ADHIL",
    "AMRITHESH",
    "IKSHAAN",
    "DOLPHIN"
  };

  char grid[SIZE][SIZE];
  create_word_search(words, MAX_WORDS, grid);
  play_word_search(words, MAX_WORDS, grid);

  return 0;
}
```

OUTPUT:
Sample1:

```
Current Grid:
Y Y A V B Z T D P H Y S I C S
T S U S K N D G C V R H S D C
A M R I T H E S H G W T I A V
U D B D O L P H I N Z C F T B
Z A F Y C H S F A K B H U A W
O T Z C T V V A P X I E L S S
Q A J P O T I T U J Z M T T N
I S N Y T K Z U F V J I E R X
E C B T A D H I L U E S Q U S
M I E H K C N Z V X U T R C I
T E M O V D Z Q O Y L R K T T
V N W N Z U A B P G R Y E U H
S C X T W M U R J P K T D R K
Z E R I K S H A A N O A A E H
C B U X W Q W R S X I Y R R R

Enter a word you found (or type 'quit' to exit): KEDAR
Correct!
```

Sample2:

```
Current Grid:
Y Y A V B Z T D P H Y S I C S
T S U S K N D G C V R H S D C
A M R I T H E S H G W T I A V
U D B D O L P H I N Z C F T B
Z A F Y C H S F A K B H U A W
O T Z C T V V A P X I E L S S
Q A J P O T I T U J Z M T T N
I S N Y T K Z U F V J I E R X
E C B T A D H I L U E S Q U S
M I E H K C N Z V X U T R C I
T E M O V D Z Q O Y L R K T T
V N W N Z U A B P G R Y E U H
S C X T W M U R J P K T D R K
Z E R I K S H A A N O A A E H
C B U X W Q W R S X I Y R R R

Enter a word you found (or type 'quit' to exit): KEDAR
You already found that word!
```

Sample3:

```
Current Grid:
Y Y A V B Z T D P H Y S I C S
T S U S K N D G C V R H S D C
A M R I T H E S H G W T I A V
U D B D O L P H I N Z C F T B
Z A F Y C H S F A K B H U A W
O T Z C T V V A P X I E L S S
Q A J P O T I T U J Z M T T N
I S N Y T K Z U F V J I E R X
E C B T A D H I L U E S Q U S
M I E H K C N Z V X U T R C I
T E M O V D Z Q O Y L R K T T
V N W N Z U A B P G R Y E U H
S C X T W M U R J P K T D R K
Z E R I K S H A A N O A A E H
C B U X W Q W R S X I Y R R R

Enter a word you found (or type 'quit' to exit): KEDA
Wrong,Try again!
```

Sample4:

```
Current Grid:
Y Y A V B Z T D P H Y S I C S
T S U S K N D G C V R H S D C
A M R I T H E S H G W T I A V
U D B D O L P H I N Z C F T B
Z A F Y C H S F A K B H U A W
O T Z C T V V A P X I E L S S
Q A J P O T I T U J Z M T T N
I S N Y T K Z U F V J I E R X
E C B T A D H I L U E S Q U S
M I E H K C N Z V X U T R C I
T E M O V D Z Q O Y L R K T T
V N W N Z U A B P G R Y E U H
S C X T W M U R J P K T D R K
Z E R I K S H A A N O A A E H
C B U X W Q W R S X I Y R R R

Enter a word you found (or type 'quit' to exit): quit
Thanks for playing!
```

Conclusion:

The developed word search game successfully addresses the limitations of existing systems by incorporating randomization, dynamic user interaction, and effective gameplay mechanics. By providing a more engaging and interactive experience, the game not only serves as an entertaining pastime but also aids in developing cognitive skills such as pattern recognition and vocabulary enhancement. Future enhancements may include a graphical interface, advanced scoring systems, and multiplayer options, further enriching the user experience.