PROJECT NAME

ARCADE OF MAZE GAME

&

WORD PUZZLE GAME

GROUP MEMBERS:-

1. SAWAIRA ARSHAD (CT-057)
2. HAIQA ANIS KHAN (CT-060)
3. AHMAD RAZA (CT-086)
4. M KAZIM ABBAS LASHARI (CT-095)

Table of Content:

CHAPTER 1

INTRODUCTION:

This C application offers users an enjoyable and engaging experience by combining two interactive games: a word puzzle game and a maze game. Users are asked to pick one of the two games, and the application starts the matching game based on their choice. Users of the Word Puzzle Game are faced with unscrambled words to solve, while those playing the Maze Game must negotiate a maze to reach the destination.

PURPOSE:

This initiative aims to provide players with a fun and varied gaming experience. It accommodates users with diverse interests by combining two separate games. In the Maze Game, players navigate a maze to lead a snail over obstacles while testing their navigation and decision-making skills. However, when players unscramble words, the Word Puzzle Game tests their language and mental abilities.

PROJECT’S OBJECTIVE:

This code aims to combine two different games, a word puzzle game and a maze game, to provide an easy-to-use and engaging gaming experience. Users of the software can select from a variety of games and participate in various challenges. The overall goal is to provide players with a fun and varied gaming experience that motivates them to take on word unscrambling and labyrinth navigation tasks. With the help of the code, users may play two distinct games in one software.

MAZE GAME:

In the Maze Game, the goal is to guide a snail through a maze and arrive at a predetermined spot. Users control the snail's movement with arrow keys ('w' for upward, 'a' for leftward, 's' for downward, 'd' for rightward). To navigate the snail through the maze and avoid obstacles (designated with a #) while arriving at the endpoint (designated with a @), players must make strategic judgments. To correctly navigate the snail to its destination and finish the maze is the goal.

WORD PUZZLE GAME:

The objective of the Word Puzzle Game is to unscramble words that are displayed to users. In a certain number of tries, players must correctly guess the unscrambled words. As players sort through the mixed-up words, the game assesses their vocabulary and mental abilities. The goal is to predict as many words accurately as you can in the allotted time in order to get a high score.

CHAPTER 2

PRODUCT FEATURES OF ARCADE:

1. GAME SELECTION MENU:

The player can decide between the Maze Game and the Word Puzzle Game at the arcade's initial game selection menu.

2. **MAZE GAME:**
   The maze pattern of The Maze Game is a 16x16 grid.
   The user controls a character ('0') in the maze with 'w', 'a', 's', and 'd' keys. There are choices to escape ('m') and to get instructions ('i') while playing. Winning condition: A winning message is displayed after navigating the maze to reach the terminus.
3. **WORD PUZZLE GAME:**
   In the Word Puzzle Game, players must unscramble words that are given in an erratic manner. Gamers can choose a maximum number of attempts (MAX_ATTEMPTS) that they can use to guess the right phrase. Accurate estimations yield points. The final score is shown when the game is over.
4. **GAME LOOP:**
   Because the arcade uses an unlimited loop, users may play more than one game at a time without having to restart the application.Users are given the option to select another game or leave the arcade after finishing one.
5. **RANDOMIZATION:**
   In the Word Puzzle Game, randomization is utilized to choose a word at random and to mix up the letters inside each word.
6. **INPUT HEADINGS:**
   The getch() and kbhit() routines are used by the application to handle keyboard input effectively.The game that is currently running determines how the input is handled.

7. **USER INTERFACE:**

The console-based UI is clean and tidy. The player receives direction and feedback from messages and prompts.

8. **DELAY FUNCTIONALITY:**

To improve user experience, a delay function (delay()) is introduced to construct time intervals.

9. **FUNCTION ABSTRACTION:**

The code is structured into functions (playMazeGame(), playWordPuzzleGame(), etc.) for clarity and modularity. **10. DYNAMIC GAME SELECTION:**

Without having to restart the application, users may select and play different games repeatedly.

11. INTRODUCTION AND PURPOSE:

The code provides introduction messages and prompts to instruct users and create the scene for each game.

12. CLEAN EXIT:

By opting to leave the arcade after finishing a game, users may do so in a polite manner.

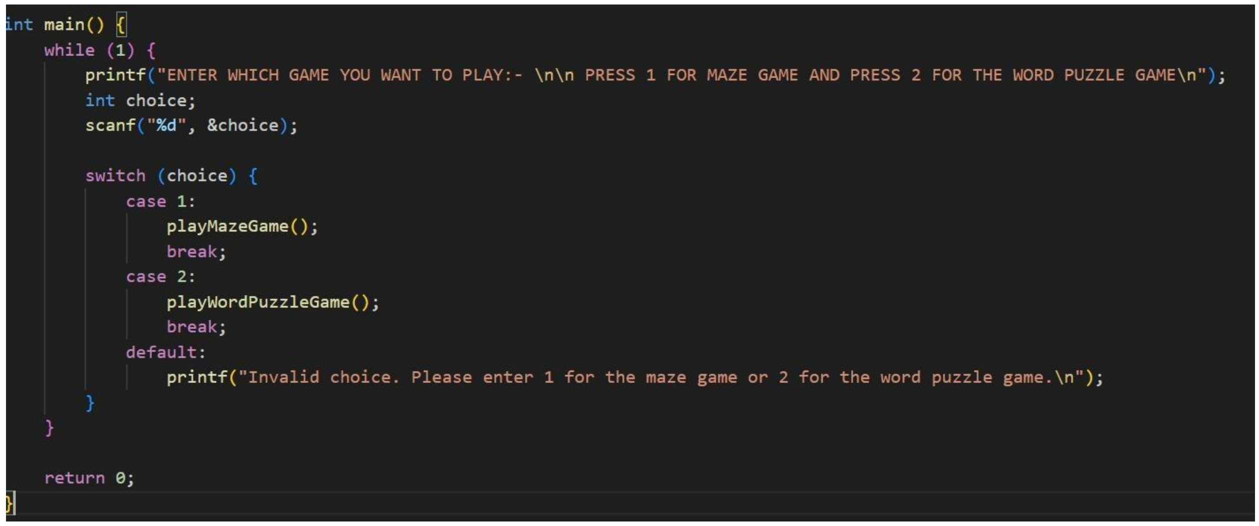
Together, these elements offer a multitude of game alternatives in an entertaining and participatory arcade experience.

CHAPTER 3

DESIGN AND LAYOUT:

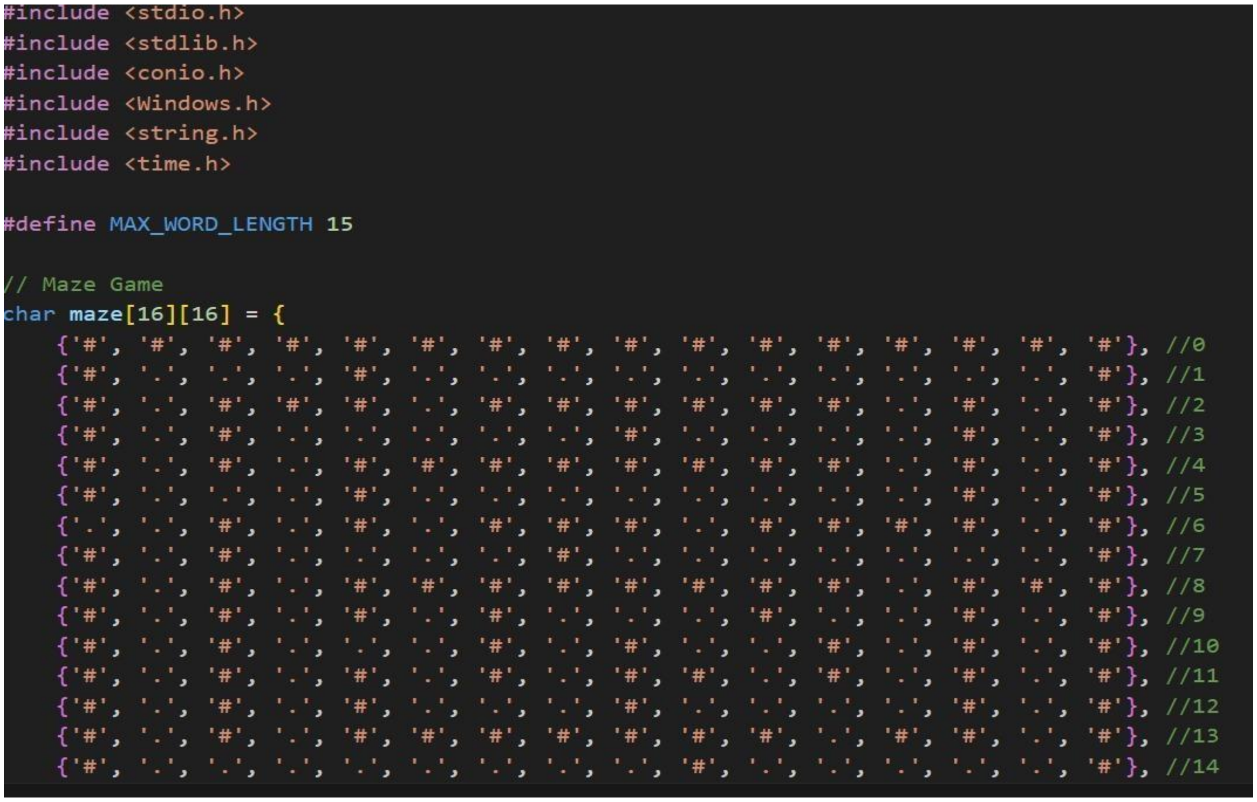
ARCADE

The main loop of an arcade-style game selection menu is included in this piece of code. The user is continuously prompted to select between playing the Word Puzzle Game (option 2) and the Maze Game (option 1). The user's selection is used as input, and depending on the option chosen, a switch statement tells the program to go to the appropriate game function. The loop keeps going, giving the user the chance to select a game again, and a notification is shown if they enter an incorrect selection. The user may play numerous rounds of either game without having to restart the application since the loop continues continuously unless deliberately stopped.

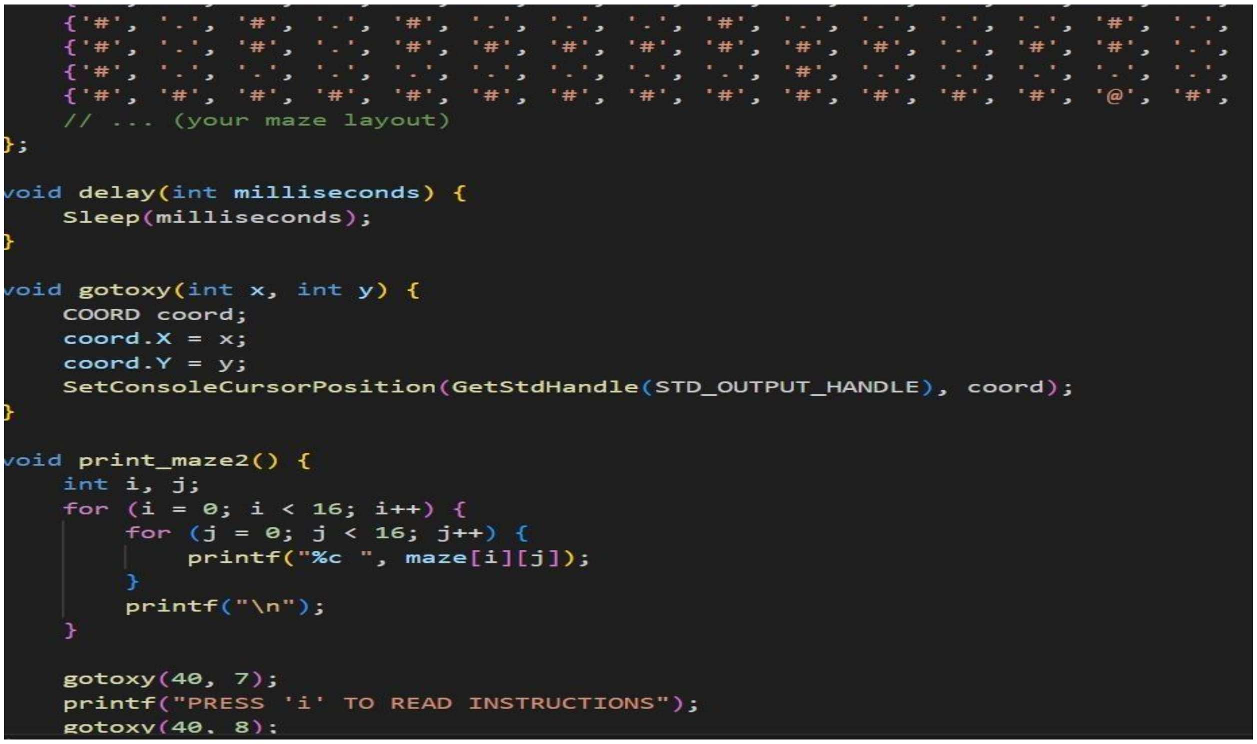


```
int main() {
    while (1) {
        printf("ENTER WHICH GAME YOU WANT TO PLAY:- \n\n PRESS 1 FOR MAZE GAME AND PRESS 2 FOR THE WORD PUZZLE GAME\n");
        int choice;
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                playMazeGame();
                break;
            case 2:
                playWordPuzzleGame();
                break;
            default:
                printf("Invalid choice. Please enter 1 for the maze game or 2 for the word puzzle game.\n");
        }
    }

    return 0;
}
```

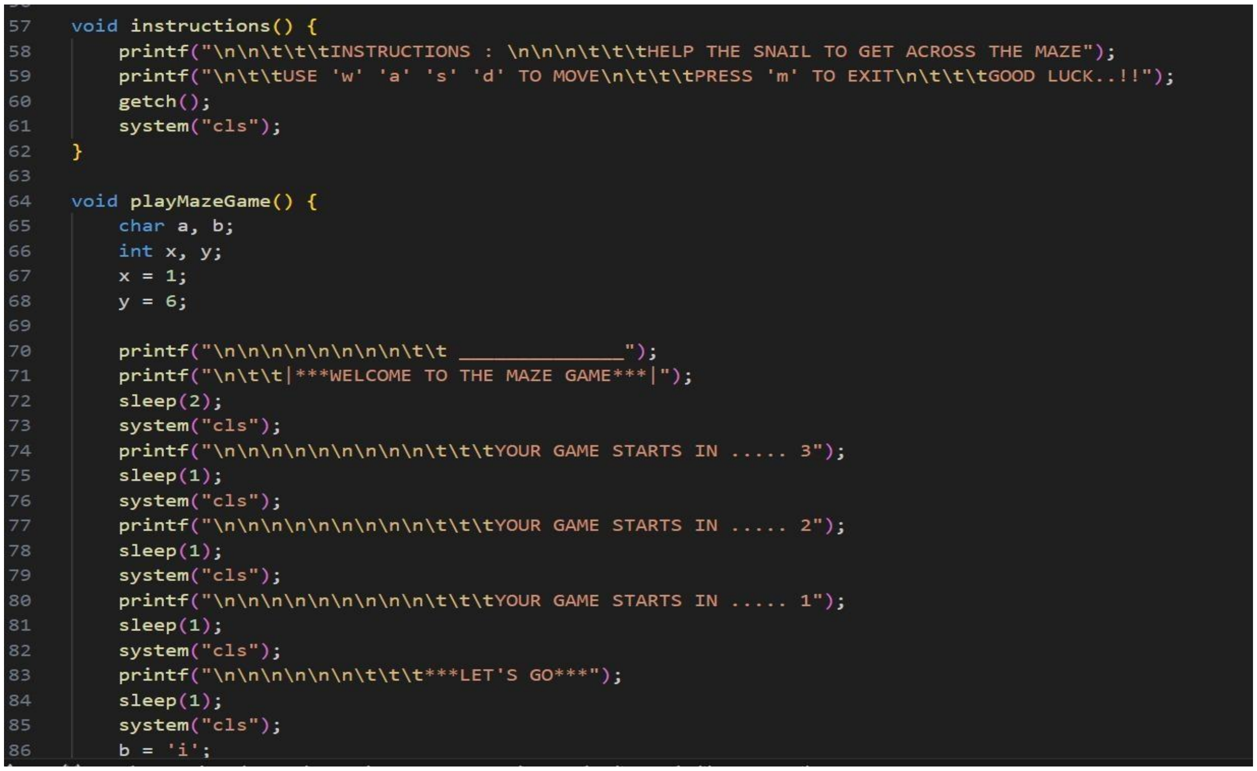
MAZE GAME



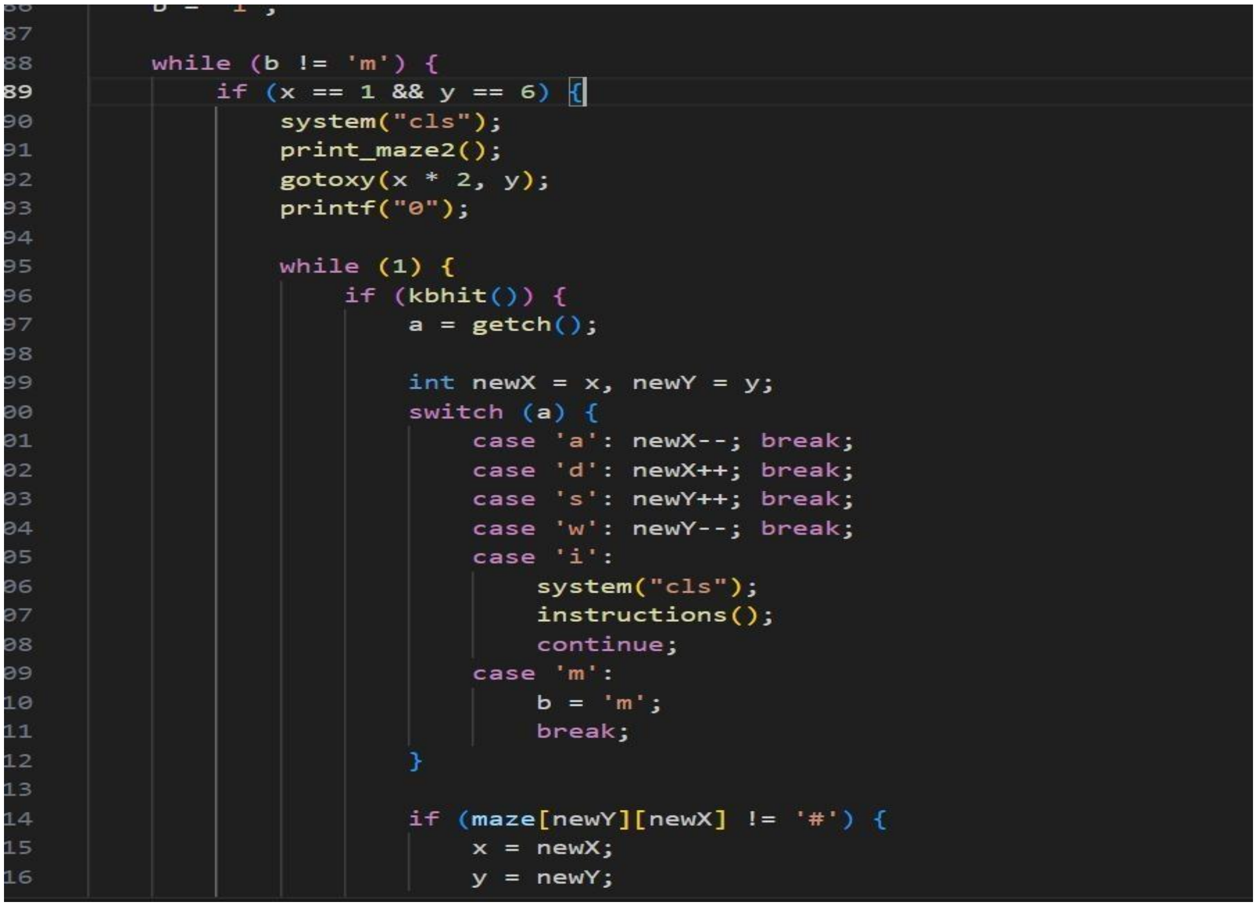
```c
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <Windows.h>
#include <string.h>
#include <time.h>

#define MAX_WORD_LENGTH 15

// Maze Game
char maze[16][16] = {
    {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'}, //0
    {'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#'}, //1
    {'#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#'}, //2
    {'#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '#', '.', '#'}, //3
    {'#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#'}, //4
    {'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#'}, //5
    {'.', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '.', '#'}, //6
    {'#', '.', '#', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#'}, //7
    {'#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#'}, //8
    {'#', '.', '#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '#', '.', '#'}, //9
    {'#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '#', '.', '#', '.', '#'}, //10
    {'#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '.', '#', '.', '#', '.', '#'}, //11
    {'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '#', '.', '#'}, //12
    {'#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '.', '#'}, //13
    {'#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#'}, //14
```



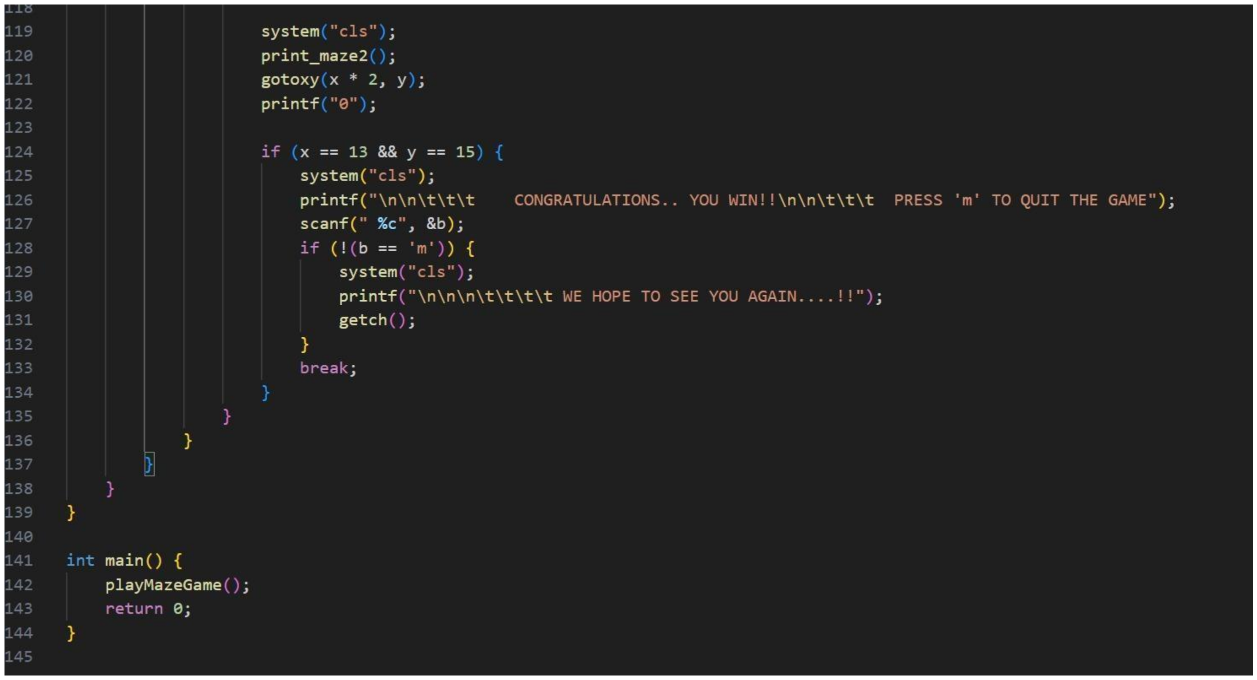
```c
    {'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '#', '.',
    {'#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '.',
    {'#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.',
    {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '@', '#',
    // ... (your maze layout)
};

void delay(int milliseconds) {
    Sleep(milliseconds);
}

void gotoxy(int x, int y) {
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

void print_maze2() {
    int i, j;
    for (i = 0; i < 16; i++) {
        for (j = 0; j < 16; j++) {
            printf("%c ", maze[i][j]);
        }
        printf("\n");
    }

    gotoxy(40, 7);
    printf("PRESS 'i' TO READ INSTRUCTIONS");
    gotoxy(40, 8);
```



```c
void instructions() {
    printf("\n\n\t\t\tINSTRUCTIONS : \n\n\n\t\t\tHELP THE SNAIL TO GET ACROSS THE MAZE");
    printf("\n\t\tUSE 'w' 'a' 's' 'd' TO MOVE\n\t\t\tPRESS 'm' TO EXIT\n\t\t\tGOOD LUCK..!!");
    getch();
    system("cls");
}

void playMazeGame() {
    char a, b;
    int x, y;
    x = 1;
    y = 6;

    printf("\n\n\n\n\n\n\n\n\t\t _______________");
    printf("\n\t\t|***WELCOME TO THE MAZE GAME***|");
    sleep(2);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 3");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 2");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 1");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\t\t\t***LET'S GO***");
    sleep(1);
    system("cls");
    b = 'i';
```



```c

    while (b != 'm') {
        if (x == 1 && y == 6) {
            system("cls");
            print_maze2();
            gotoxy(x * 2, y);
            printf("0");

            while (1) {
                if (kbhit()) {
                    a = getch();

                    int newX = x, newY = y;
                    switch (a) {
                        case 'a': newX--; break;
                        case 'd': newX++; break;
                        case 's': newY++; break;
                        case 'w': newY--; break;
                        case 'i':
                            system("cls");
                            instructions();
                            continue;
                        case 'm':
                            b = 'm';
                            break;
                    }

                    if (maze[newY][newX] != '#') {
                        x = newX;
                        y = newY;
```



```c
                    system("cls");
                    print_maze2();
                    gotoxy(x * 2, y);
                    printf("0");

                    if (x == 13 && y == 15) {
                        system("cls");
                        printf("\n\n\t\t\t    CONGRATULATIONS.. YOU WIN!!\n\n\t\t\t  PRESS 'm' TO QUIT THE GAME");
                        scanf(" %c", &b);
                        if (!(b == 'm')) {
                            system("cls");
                            printf("\n\n\n\t\t\t\t WE HOPE TO SEE YOU AGAIN....!!");
                            getch();
                        }
                        break;
                    }
                }
            }
        }
    }
}

int main() {
    playMazeGame();
    return 0;
}
```

1. Character Illustration:

'#': Represents walls or obstacles in the maze.
'.': Stands for unoccupied areas or open routes.
'@': Denotes the intended location or endpoint.

2. Maze Arrangement:

Using a 2D array, the maze is shown as a 16x16 grid (char maze[16][16]).
Walls ('#') encircle the arrangement on all sides.
The goal point, open routes, and walls abound in the actual maze design.

3. Features of the Maze:

The player may explore the maze by using the walls as a path.
In the maze, the player's starting position is often (1, 6).

4. Personalization:

There is space for personalization and extension because the maze pattern is just partially established.
You may alter the maze by modifying the order of barriers ('#'), open routes ('.'), and the destination ('@').

5. Possible Enhancements:

There are presently no intricate elements in the maze, such as branching pathways, dead ends, or extra difficulties. You may want to add more complex maze constructions to improve the gameplay. **6. Logic of Gameplay:**

This maze design is meant to be used in a game where players solve mazes. Walls will limit player movement throughout the maze, and one of the possible goals is to get to the target point ('@').

7. FUNCTION OF “delay”:

The delay time, expressed in milliseconds, is one of the parameters. Its Functionality pauses the running of the application for the predetermined amount of time. It’s
Implementation is when a delay is introduced by using the Windows API's Sleep function. This feature is frequently used to adjust how quickly animations or game loops play.

8. FUNCTION OF “gotoxy”:

These features are intended to make console-based user interfaces easier to use, especially in situations when accurate cursor positioning or real-time updates are necessary.
Delay is a helpful tool for coordinating events or achieving seamless transitions.
With gotoxy, you can create console outputs that are more aesthetically pleasing by allowing the cursor to be moved to certain places.

9. FUNTION OF “instructions”:

The function gives the player instructions by printing formatted text to the console using printf commands.
To produce new lines and tabs in formatting, escape sequences like \n are utilized.

The instructions explain how to use the 'w', 'a', 's', and 'd' keys to guide the snail through the maze and how to hit 'm' to end the game.
The getch() method is invoked to wait for a key press before proceeding once the instructions have been shown. This is probably done to make sure the gamer has adequate time to comprehend the directions.
Lastly, the console screen is cleared using system("cls"), making it ready for the game to launch.

10. FUNCTION OF "playMazeGame":

Initialization: Sets additional variables and the beginning point (x, y). shows a countdown and a greeting.
Main Game Loop: Continues until the user chooses to end it.
Verifies if the player is in the starting position. shows
the player's position and the labyrinth.
Motion Loop: Manages movement while it waits for keyboard input. changes the player's location and verifies that their motions are correct. It shows and updates the maze appropriately. Winning Condition: Checks if the player achieves the winning position. It shows a congrats message and requests that you end the session.

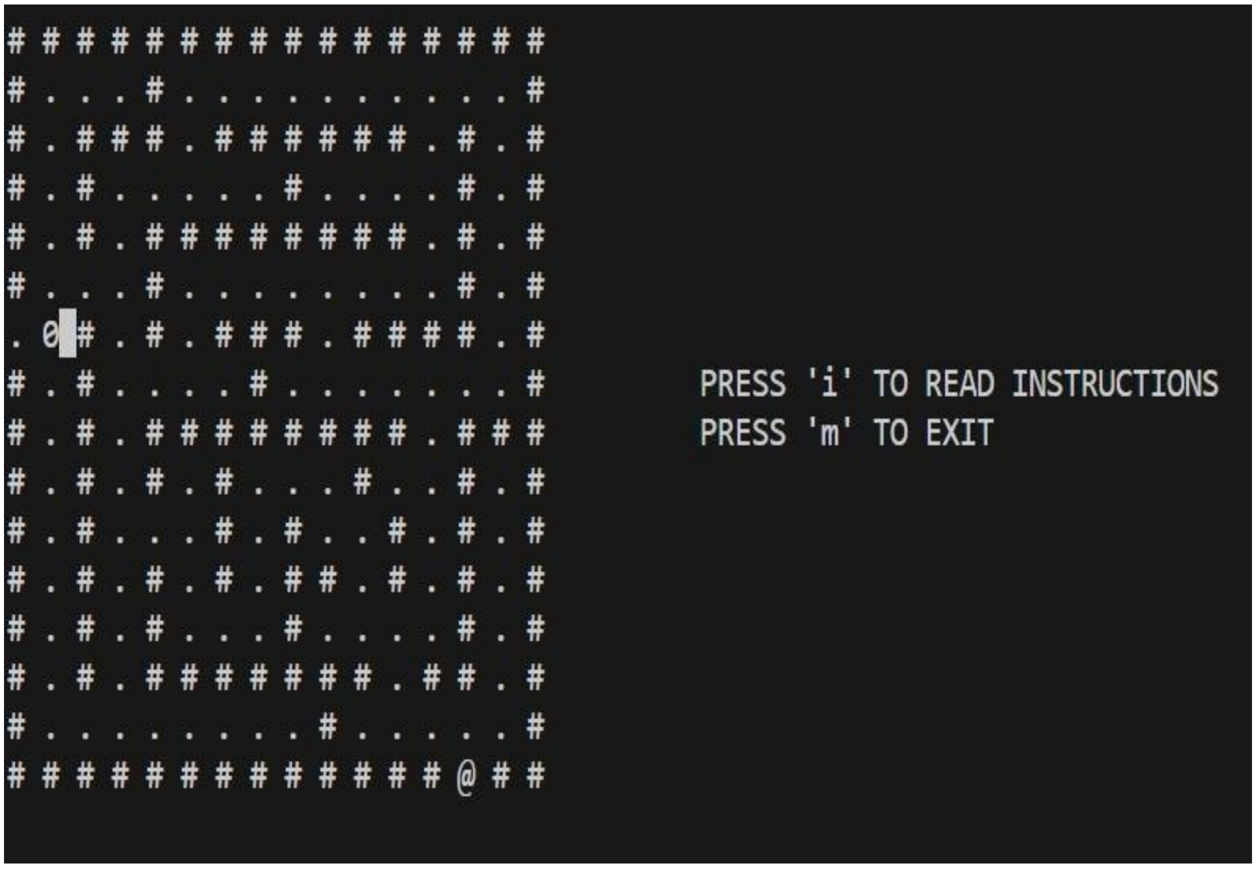
For the description of this function: Using the 'w', 'a','s', and 'd' keys, the player must make their way through a maze as the code launches a maze game. It offers a welcome message and a countdown. The player begins at a predetermined spot (1, 6).The player's position is updated on the labyrinth display by the game. A triumph message appears if the player achieves the winning position (13, 15).

11. FUNCTION OF "main":

Signature of Function: int main(): The main function, the entry point of the program.
Body Function: PlayMazeGame() is the method that is called when the Maze Game has to be run.

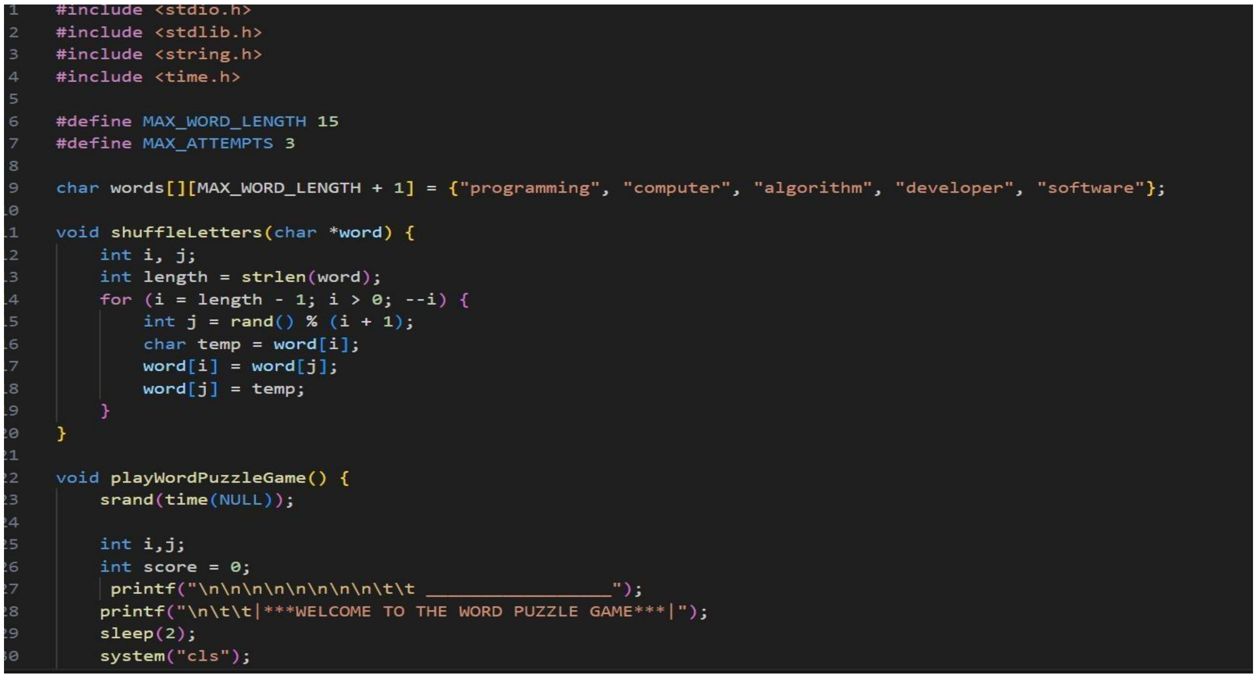
Statement of Return: return 0; Notifies the operating system that the execution was successful.

<u>OUTPUT:-</u>

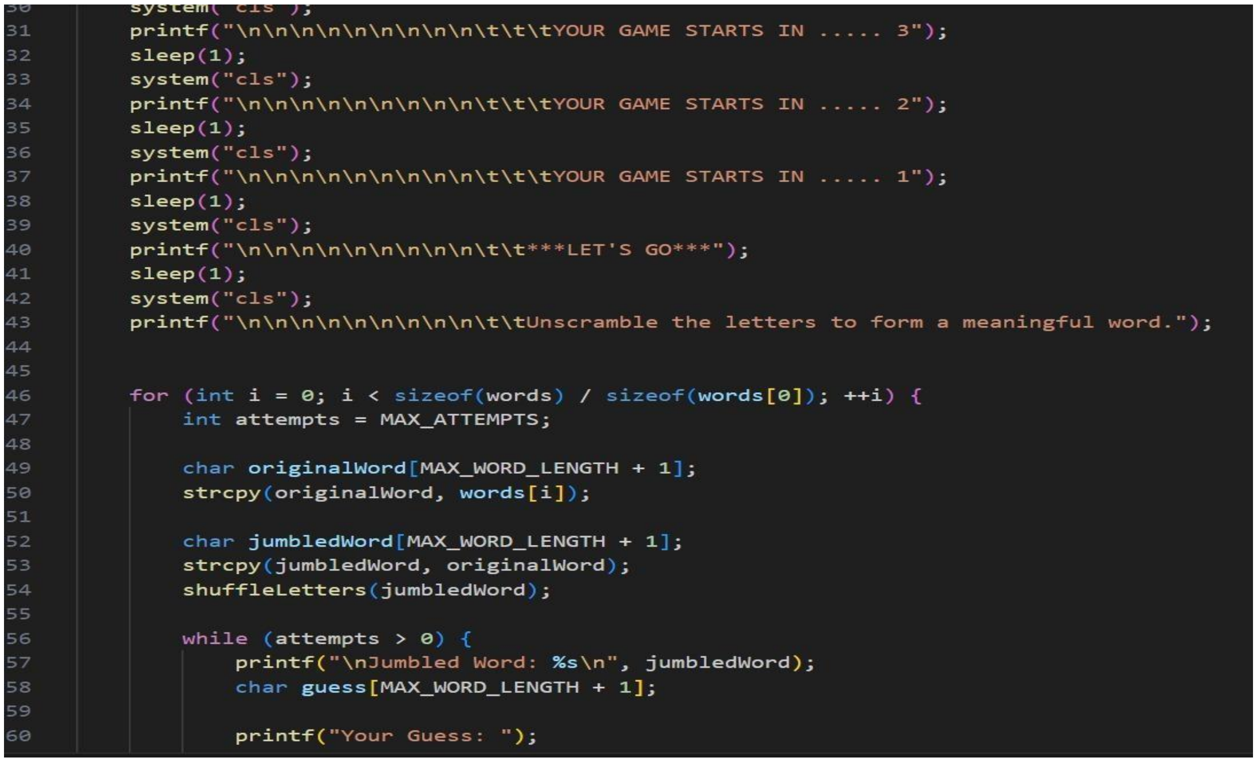


```
# # # # # # # # # # # # # # # #
# . . . # . . . . . . . . . . #
# . # # # . # # # # # # . # . #
# . # . . . . . # . . . . # . #
# . # . # # # # # # # # . # . #
# . . . # . . . . . . . . # . #
. 0 # . # . # # # . # # # # . #
# . # . . . . # . . . . . . . #          PRESS 'i' TO READ INSTRUCTIONS
# . # . # # # # # # # # . # # #          PRESS 'm' TO EXIT
# . # . # . # . . . # . . # . #
# . # . . . # . # . . # . # . #
# . # . # . # . # # . # . # . #
# . # . # . . . # . . . . # . #
# . # . # # # # # # # . # # . #
# . . . . . . . . # . . . . . #
# # # # # # # # # # # # # @ # #
```

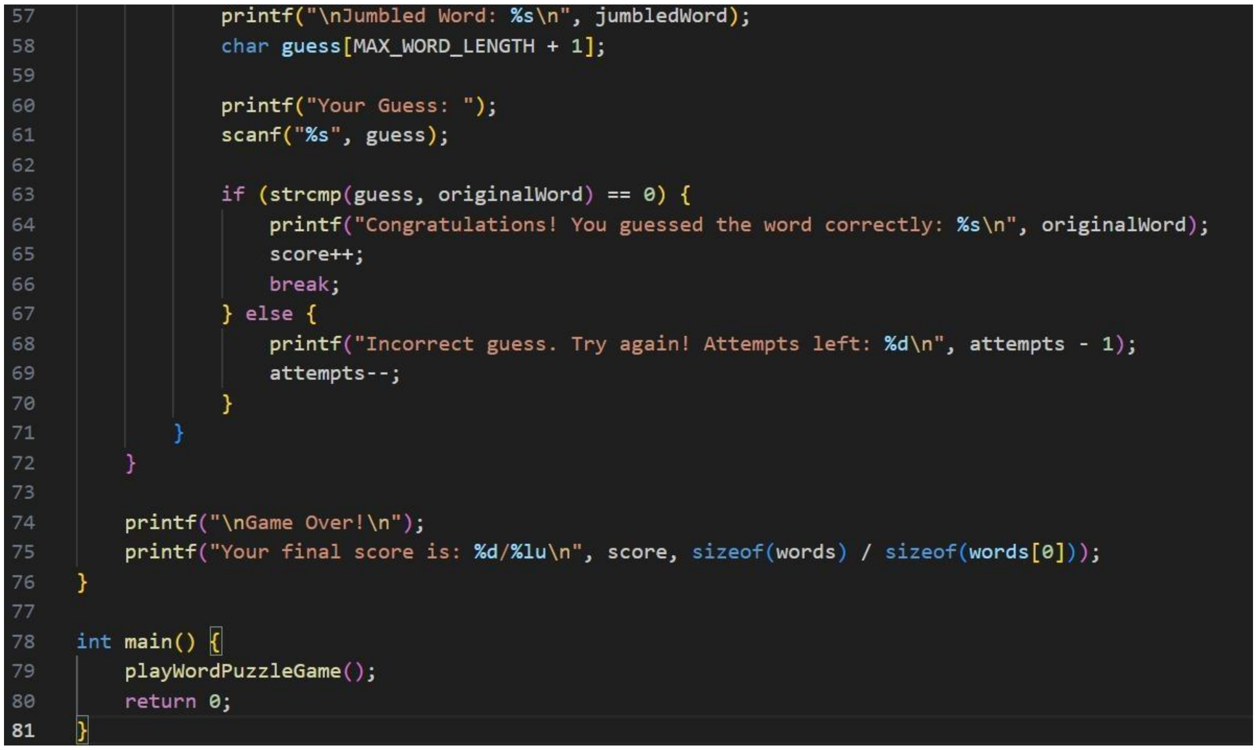
WORD PUZZLE GAME



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX_WORD_LENGTH 15
#define MAX_ATTEMPTS 3

char words[][MAX_WORD_LENGTH + 1] = {"programming", "computer", "algorithm", "developer", "software"};

void shuffleLetters(char *word) {
    int i, j;
    int length = strlen(word);
    for (i = length - 1; i > 0; --i) {
        int j = rand() % (i + 1);
        char temp = word[i];
        word[i] = word[j];
        word[j] = temp;
    }
}

void playWordPuzzleGame() {
    srand(time(NULL));

    int i,j;
    int score = 0;
     printf("\n\n\n\n\n\n\n\n\t\t _________________");
    printf("\n\t\t|***WELCOME TO THE WORD PUZZLE GAME***|");
    sleep(2);
    system("cls");
```



```c
    printf("\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 3");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 2");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\t\t\tYOUR GAME STARTS IN ..... 1");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\t\t***LET'S GO***");
    sleep(1);
    system("cls");
    printf("\n\n\n\n\n\n\n\n\n\t\tUnscramble the letters to form a meaningful word.");


    for (int i = 0; i < sizeof(words) / sizeof(words[0]); ++i) {
        int attempts = MAX_ATTEMPTS;

        char originalWord[MAX_WORD_LENGTH + 1];
        strcpy(originalWord, words[i]);

        char jumbledWord[MAX_WORD_LENGTH + 1];
        strcpy(jumbledWord, originalWord);
        shuffleLetters(jumbledWord);

        while (attempts > 0) {
            printf("\nJumbled Word: %s\n", jumbledWord);
            char guess[MAX_WORD_LENGTH + 1];

            printf("Your Guess: ");
```



```
            printf("\nJumbled Word: %s\n", jumbledWord);
            char guess[MAX_WORD_LENGTH + 1];

            printf("Your Guess: ");
            scanf("%s", guess);

            if (strcmp(guess, originalWord) == 0) {
                printf("Congratulations! You guessed the word correctly: %s\n", originalWord);
                score++;
                break;
            } else {
                printf("Incorrect guess. Try again! Attempts left: %d\n", attempts - 1);
                attempts--;
            }
        }
    }

    printf("\nGame Over!\n");
    printf("Your final score is: %d/%lu\n", score, sizeof(words) / sizeof(words[0]));
}

int main() {
    playWordPuzzleGame();
    return 0;
}
```

This code block presents an opening screen with a countdown and welcome message to start the Word Puzzle Game. Below is an outline of the key components: **STEP 1:** int i, j;: Declaration of loop variables. int score = 0;: Initialization of the player's score to 0.

A countdown sequence and a graphically prepared welcome message are shown by the sequence of printf commands. The introduction screen and the user-friendly interface are printed by these statements. sleep(2);: Gives the user a 2-second execution pause so they may read the first message.
system("cls");: After every sleep, clears the console screen to appear to be showing updated information.
Before the game starts, a countdown sequence ("YOUR GAME STARTS IN...") is shown to build excitement.
printf("**LET'S GO**", printf("...");: provides the consumer with a visual cue that the game has begun.In general, this block introduces the user to the Word Puzzle Game in a visually engaging and dynamic way. Before

the game begins, interest and engagement are increased by the countdown and "LET'S GO" message.

STEP 2:

The fundamental reasoning behind the Word Puzzle Game is represented by this code block. Let's dissect the essential elements: ++i) {: for (int i = 0; i < sizeof(words) / sizeof(words[0]); In this loop, every word in the 'words' array is iterated over. For every word: int attempts = MAX_ATTEMPTS;: Initializes the variable 'attempts' to the maximum permitted attempts for guessing the word.char originalWord[MAX_WORD_LENGTH + 1]; strcpy(originalWord, words[i]);: Copies the original unscrambled word from the 'words' array to 'originalWord'. **STEP 3:**

To create a jumbled version of the word, duplicate the original and use the'shuffleLetters' function to shuffle the letters. char jumbledWord[MAX_WORD_LENGTH + 1]; strcpy(jumbledWord, originalWord); shuffleLetters(jumbledWord);

The while (attempts > 0) loop enables the player numerous attempts to guess the unscrambled word.

printf("Jumbled Word: %s", jumbledWord);: Provides the player with the jumbled word to sort through. char guess[MAX_WORD_LENGTH + 1];

printf("Your Guess: "); scanf("%s", guess);: Takes the player's guess as input.

STEP 4:

The guess is compared to the original word using the if (strcmp(guess, originalWord) == 0) expression.

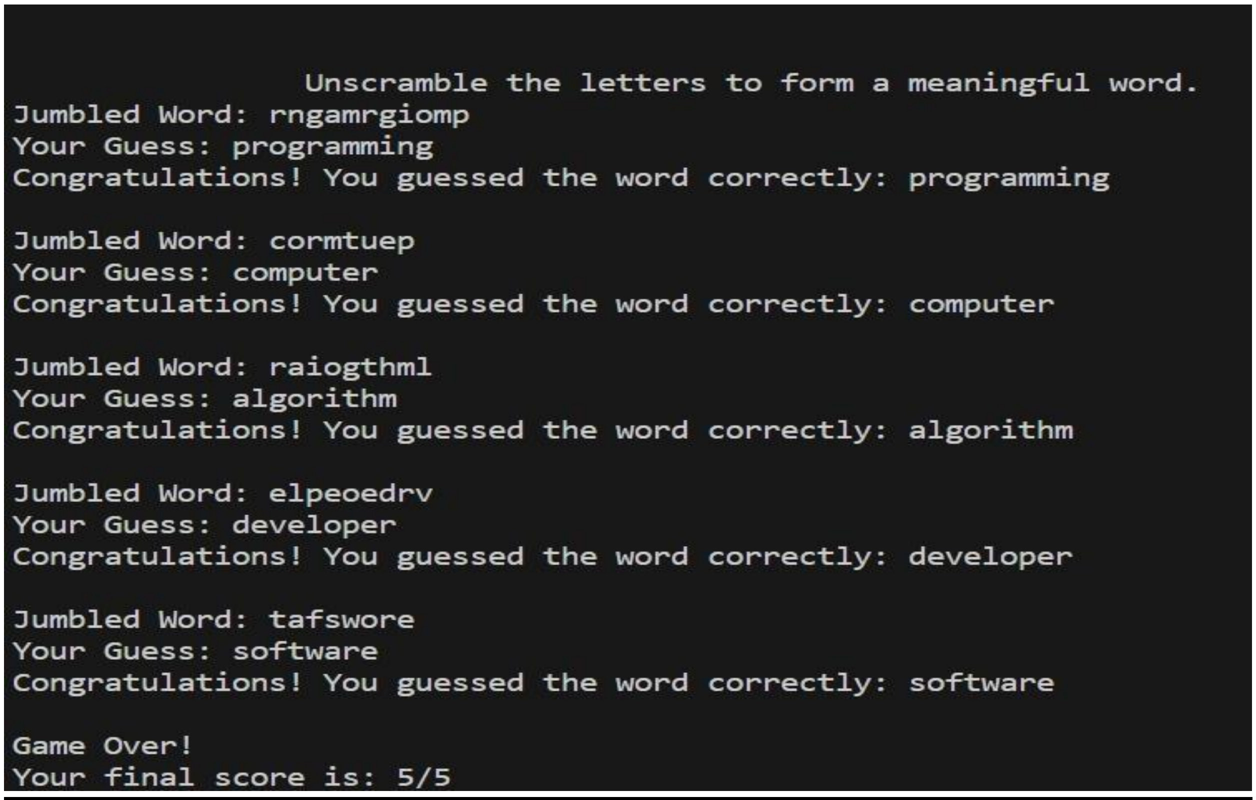
In the event that it is true, the loop is broken, a success message is printed, and the player's score is increased.

If the answer is false, the loop continues until all tries have been made, produces an error message, and reduces the number of attempts. The game ends when printf("\nGame Over! Your final score is: %d/%lu\n", score, sizeof(words) / sizeof(words[0])); shows the score.

STEP 5:

int main(){ this is the compulsory function of every C program, in this main function the word puzzle game is called, The Word Puzzle Game is started by using the playWordPuzzleGame function. The logic specified in the playWordPuzzleGame function will be carried out by the program. The return 0;: This line of code indicates that the program has run successfully. The operating system receives a result of 0, signifying that the program ended successfully.

<u>OUTPUT:-</u>



```
                Unscramble the letters to form a meaningful word.
Jumbled Word: rngamrgiomp
Your Guess: programming
Congratulations! You guessed the word correctly: programming

Jumbled Word: cormtuep
Your Guess: computer
Congratulations! You guessed the word correctly: computer

Jumbled Word: raiogthml
Your Guess: algorithm
Congratulations! You guessed the word correctly: algorithm

Jumbled Word: elpeoedrv
Your Guess: developer
Congratulations! You guessed the word correctly: developer

Jumbled Word: tafswore
Your Guess: software
Congratulations! You guessed the word correctly: software

Game Over!
Your final score is: 5/5
```

CONCEPTS COVERED IN COURSE USED IN THIS CODE:-

1. FUNCTIONS.
2. ARRAYS.
3. HOW TO GENERATE RANDOM NUMBERS.
4. HEADER FILES LIKE stdlib.b, windows.h, time.h, unistd.h.
5. STRINGS.
6. SLEEP FUNCTION.
7. LOOPS.

-:LIMITATIONS AND FUTURE ENHANCEMENT:-

LIMITAIONS:-

1. A hardcoded labyrinth reduces adaptability.
2. Dependence on features unique to Windows.
3. Inadequate handling of incorrect input errors.
4. Magic numbers make it harder to interpret code.
5. Few elements and interactions in the labyrinth.

FUTURE ENHANCEMENT:-

1. Dynamic creation of mazes or loading of other files.
2. Maze size is adjustable for variation.
3. Interoperability across platforms for increased utilization.
4. Strong error management to avoid crashes.
5. Modularization to improve the structure of the code.
6. Extra features (score, levels) in the game.
7. Enhanced UI and visuals for a more appealing look.
8. Support for multiplayer for more interaction.

-:CONTRIBUTION OF EACH MEMBER:-

CONTRIBUTOR	TASK DESCRIPTION
Sawaira Arshad	Code of maze game, making header files, integration of code.
	Generation of report
Haiqa Anis Khan	Code of maze game, word puzzle, modification of maze game.
	Generation of report.
Ahmad Raza	Code of maze game, word puzzle modification of maze game.
	Generation of report.
M Kazim Abbas Lashari	Modification of word puzzle game, making header files.
	Uploaded and pushed on GIT.

CONCLUSION

The main goal of this code is to provide a gaming platform for consoles that has two different games: word puzzles and mazes. While the Word Puzzle Game requires the player to unscramble letters and guess hidden words, the Maze Game has the player guide a snail through a maze to achieve the destination. With a choice of these two games, the code seeks to provide users a dynamic and captivating experience that combines language and strategic thinking. To improve the entire gaming experience, the games come with user-friendly interfaces and clear instructions.