COMPUTER PROGRAMMING (UTA003)

Submitted by:
Dev Mehta (102203073)
Barnika Sen (102255007)

Submitted to:
Dr. Saurabh Arora

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala.

June-July 2025
(Summer Semester)

Table of Contents

1. Introduction

This project is a terminal-based **Pac-Man Clone** developed in C, recreating the classic arcade game experience using ASCII characters. The game is built with the aim of providing a simple yet engaging gaming experience in a command-line interface. In the game, the player controls **Pac-Man**, navigating through a randomly generated maze while avoiding **demons** and collecting **food** (represented as .) to score points. The primary goal is to eat all the food while evading the demons. The game ends when all food is eaten or Pac-Man is caught by a demon.

This implementation features key game mechanics such as **movement** (using the W, A, S, D keys), **random maze generation**, **score tracking**, and **game over conditions**. The maze is created dynamically, with walls (#) and demons (X) scattered randomly throughout the map. As the player collects food, their score increases, and they advance further in the game. If Pac-Man encounters a demon, the game ends. This project showcases fundamental game development concepts, including **randomization**, **user input handling**, and **game state management**, all while maintaining a nostalgic retro feel.

This game serves as both a fun and educational exercise, highlighting basic programming principles such as **loops**, **arrays**, **functions**, and the use of **terminal I/O**. It's a simple yet effective way to understand core game mechanics while enjoying a classic game in a new, text-based format.

2. C programming concept used

1. Randomization

The maze layout, including walls and food, is generated randomly each time the game is played. This ensures that each session offers a unique experience. The use of the rand() function allows the random placement of walls, demons, and food, adding variety and challenge to the game. Randomization enhances replayability and keeps the game engaging.

2. Arrays

Arrays are used to store the game board, which is a 2D grid. Each cell in the grid represents a part of the maze, such as walls, food, or Pac-Man. This allows efficient storage and manipulation of game elements. The board is updated after every move, and the positions of different objects (Pac-Man, food, demons) are tracked using the array indices.

3. User Input Handling

The program handles real-time user input using the keyboard. Pac-Man's movement is controlled by the **W, A, S, D** keys (up, left, down, right). Input is processed using getchar(), and the program reacts instantly to player actions. The use of **raw mode** ensures smooth, non-blocking input handling, making the game more interactive.

4. Game State Management

The game continuously checks its state—whether Pac-Man is still alive, if the player has eaten all the food, or if the game has ended due to Pac-Man colliding with a demon. These states are tracked using variables like res and score, which help control the flow of the game and determine when to end the game and display the final result.

5. Control Structures

The game uses common control structures such as **loops** and **conditionals** to manage the flow of the game:

- **Loops** are used to repeatedly update the game board, process user input, and check for win/loss conditions.

- **Conditionals** are used to determine if Pac-Man has collided with walls, eaten food, or encountered a demon.

6. Score Tracking

The game tracks the player's score, which increases each time Pac-Man eats food. This is done by incrementing the score variable whenever food is consumed. The score is displayed on the screen, motivating the player to continue playing and reach higher scores.

3. User-defined Functions used

1. enableRawMode()

This function enables raw mode in the terminal, allowing for real-time user input without waiting for the Enter key to be pressed.

2. disableRawMode()

Restores the terminal to its default input settings after the game ends, re-enabling standard input buffering and echoing.

3. clearScreen()

Clears the terminal screen using an ANSI escape code, providing a fresh display for each game frame.

4. initialize()

Sets up the game by generating a random maze, placing walls, food, and demons, and positioning Pac-Man in the center of the board.

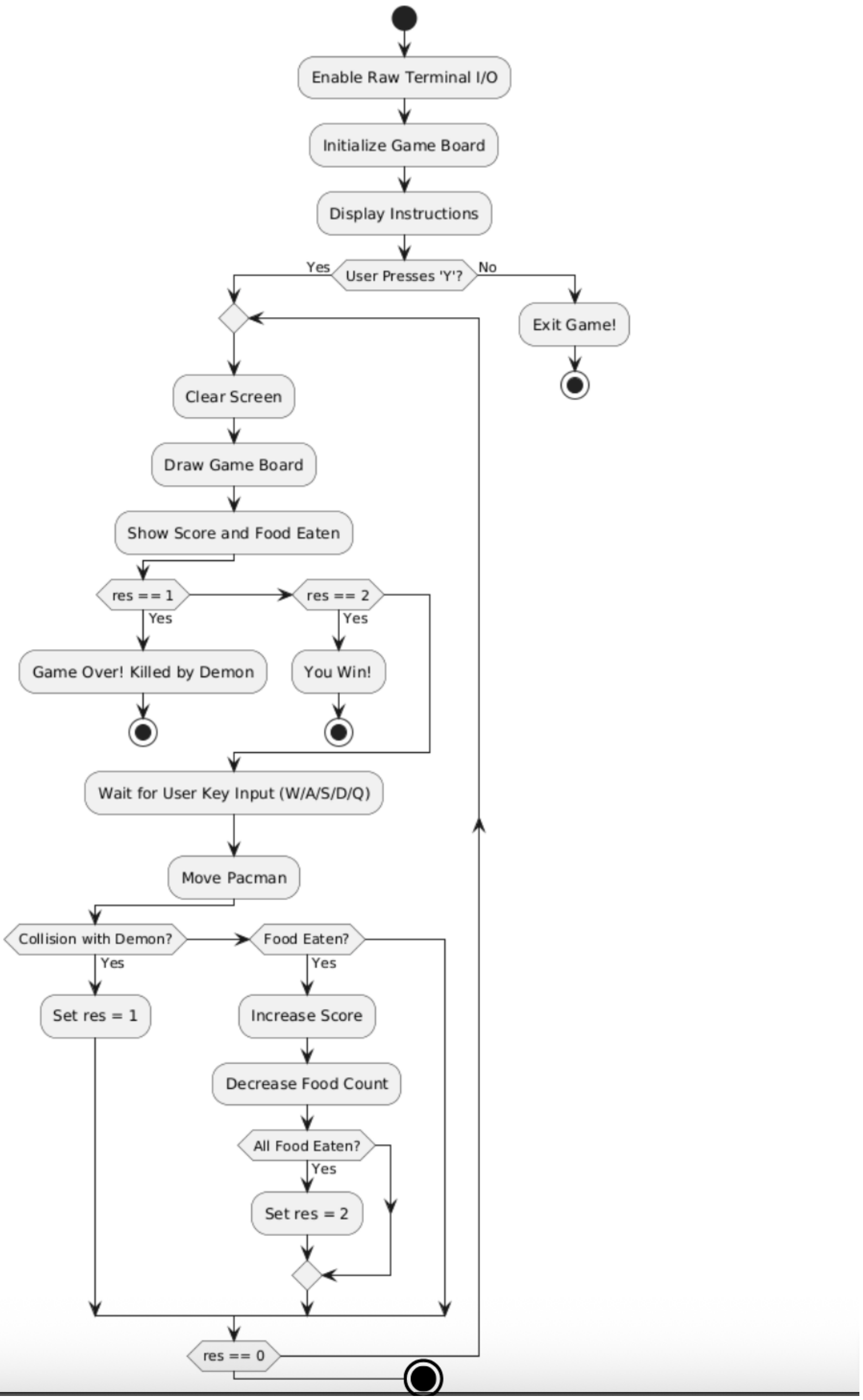
5. draw()

Displays the game board, including the maze, Pac-Man's position, food, demons, and the current score.

6. move_pacman(int move_x, int move_y)

Handles Pac-Man's movement based on user input, checks for collisions with walls, food, or demons, and updates the game state accordingly.

5. Flowchart


Enable Raw Terminal I/O
Initialize Game Board
Display Instructions
User Presses 'Y'?
Yes
No
Exit Game!
Clear Screen
Draw Game Board
Show Score and Food Eaten
res == 1
Yes
res == 2
Yes
Game Over! Killed by Demon
You Win!
Wait for User Key Input (W/A/S/D/Q)
Move Pacman
Collision with Demon?
Yes
Food Eaten?
Yes
Set res = 1
Increase Score
Decrease Food Count
All Food Eaten?
Yes
Set res = 2
res == 0

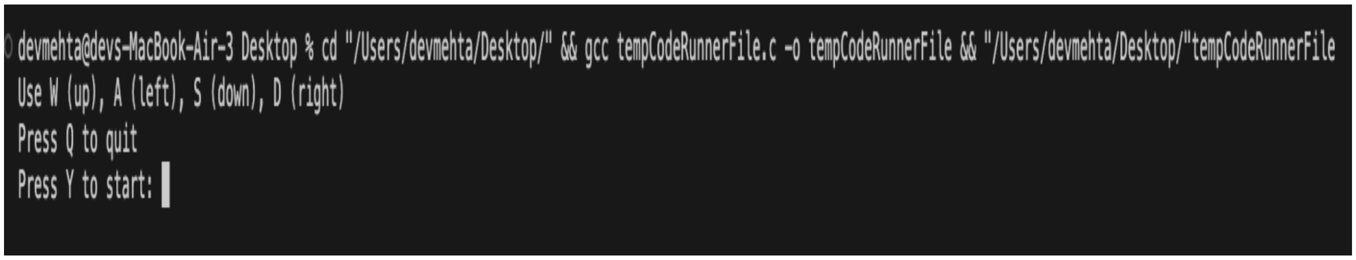
5. Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <time.h>

#define WIDTH 40
#define HEIGHT 20
#define PACMAN 'C'
#define WALL '#'
#define FOOD '.'
#define EMPTY ' '
#define DEMON 'X'

int res = 0;
int score = 0;
int pacman_x, pacman_y;
char board[HEIGHT][WIDTH];
int food = 0;
int curr = 0;

// Function to disable buffered input and echo
void enableRawMode() {
   struct termios term;
   tcgetattr(STDIN_FILENO, &term);
   term.c_lflag &= ~(ICANON | ECHO);
   tcsetattr(STDIN_FILENO, TCSANOW, &term);
}

// Function to reset terminal settings
void disableRawMode() {
    struct termios term;
    tcgetattr(STDIN_FILENO, &term);
    term.c_lflag |= (ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &term);
}

// Clear the screen using ANSI escape code
void clearScreen() {
    printf("\033[2J\033[1;1H");
}

void initialize() {
    srand(time(NULL));  // Seed the RNG

    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            if (i == 0 || j == 0 || i == HEIGHT - 1 || j == WIDTH - 1)
                board[i][j] = WALL;
            else
                board[i][j] = EMPTY;
        }
    }

    int count = 50;
    while (count--) {
        int i = rand() % HEIGHT;
        int j = rand() % WIDTH;
        if (board[i][j] == EMPTY)
            board[i][j] = WALL;
    }

    int val = 5;
    while (val--) {
        int row = rand() % HEIGHT;
        for (int j = 3; j < WIDTH - 3; j++) {
            if (board[row][j] == EMPTY)
```

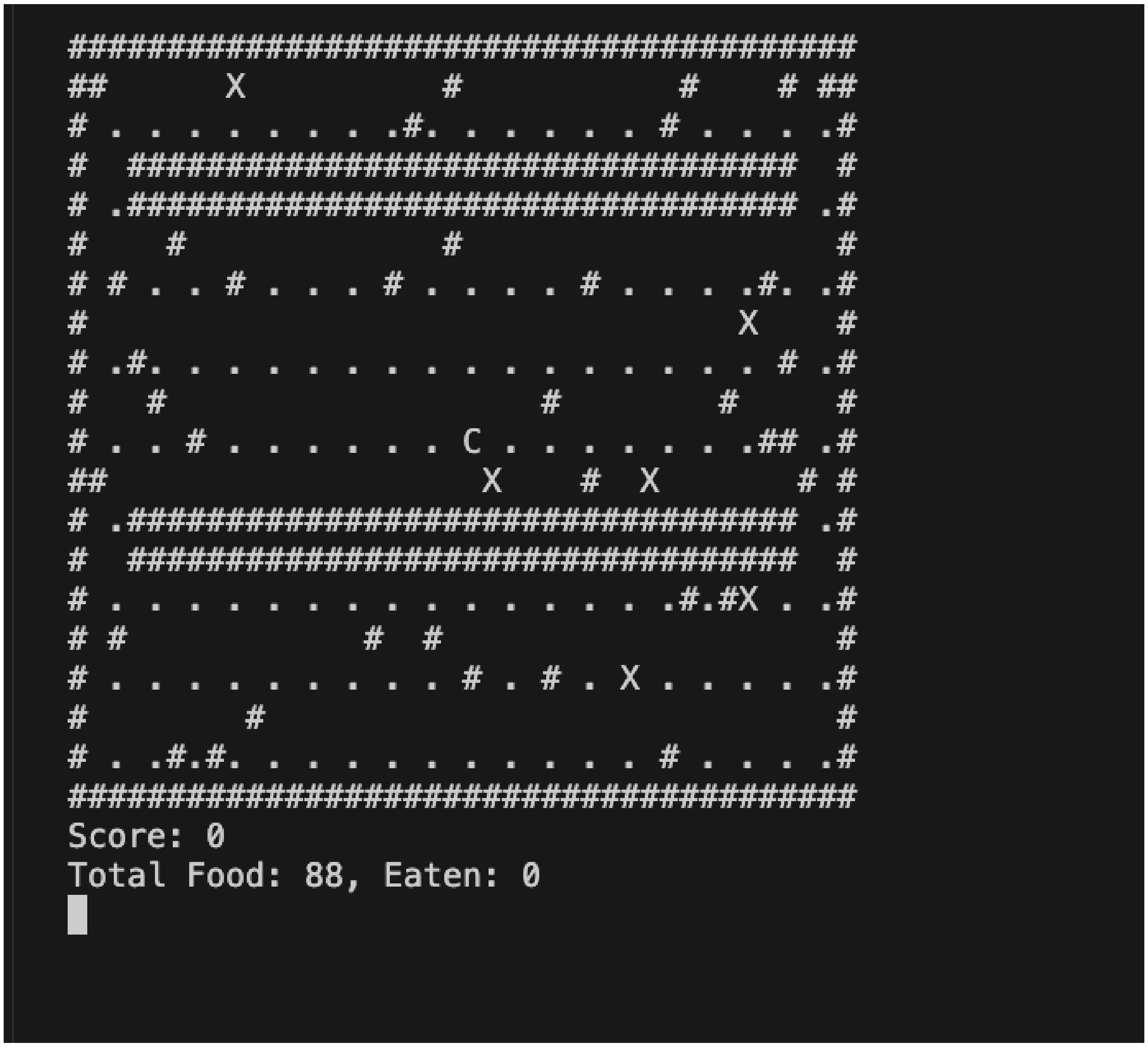
```
                board[row][j] = WALL;
            }
        }

        count = 10;
        while (count--) {
            int i = rand() % HEIGHT;
            int j = rand() % WIDTH;
            if (board[i][j] == EMPTY)
                board[i][j] = DEMON;
        }

        pacman_x = WIDTH / 2;
        pacman_y = HEIGHT / 2;
        board[pacman_y][pacman_x] = PACMAN;

        for (int i = 0; i < HEIGHT; i++) {
            for (int j = 0; j < WIDTH; j++) {
                if (i % 2 == 0 && j % 2 == 0 &&
                    board[i][j] == EMPTY) {
                    board[i][j] = FOOD;
                    food++;
                }
            }
        }
    }

    void draw() {
        clearScreen();
        for (int i = 0; i < HEIGHT; i++) {
            for (int j = 0; j < WIDTH; j++) {
                printf("%c", board[i][j]);
            }
            printf("\n");
        }
        printf("Score: %d\n", score);
```

```
}

void move_pacman(int move_x, int move_y) {
    int x = pacman_x + move_x;
    int y = pacman_y + move_y;

    if (board[y][x] != WALL) {
        if (board[y][x] == FOOD) {
            score++;
            food--;
            curr++;
            if (food == 0) {
                res = 2;
                return;
            }
        } else if (board[y][x] == DEMON) {
            res = 1;
        }

        board[pacman_y][pacman_x] = EMPTY;
        pacman_x = x;
        pacman_y = y;
        board[pacman_y][pacman_x] = PACMAN;
    }
}

int main() {
    enableRawMode();
    atexit(disableRawMode); // Reset terminal when program exits

    initialize();
    char ch;
    food -= 35;
    int totalFood = food;

    printf("Use W (up), A (left), S (down), D (right)\n");
    printf("Press Q to quit\nPress Y to start: ");
    ch = getchar();
    if (ch != 'Y' && ch != 'y') {
        printf("\nExit Game!\n");
        return 0;
    }

    while (1) {
        draw();
        printf("Total Food: %d, Eaten: %d\n", totalFood, curr);

        if (res == 1) {
            clearScreen();
            printf("Game Over! You were killed by a Demon.\nScore: %d\n", score);
            break;
        } else if (res == 2) {
            clearScreen();
            printf("You Win! Final Score: %d\n", score);
            break;
        }

        ch = getchar();

        switch (ch) {
            case 'w':
            case 'W':
                move_pacman(0, -1);
                break;
            case 's':
            case 'S':
                move_pacman(0, 1);
                break;
            case 'a':
            case 'A':
                move_pacman(-1, 0);
                break;
            case 'd':
            case 'D':
                move_pacman(1, 0);
                break;
            case 'q':
            case 'Q':
                printf("Game Over! Score: %d\n", score);
                return 0;
        }

        usleep(100000);  // Add a small delay to slow down the loop
    }

    return 0;
}
```

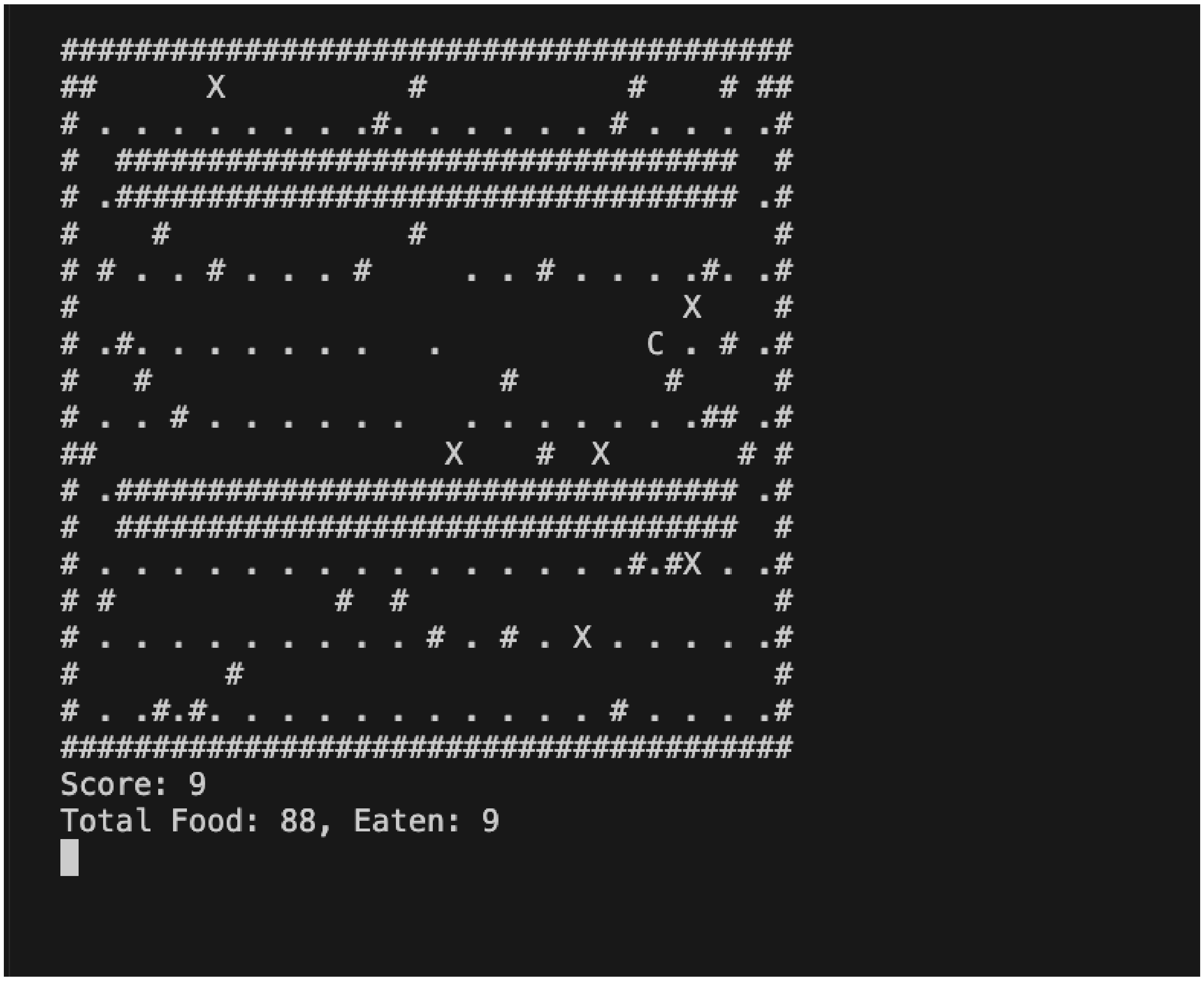
6. Output Screenshots



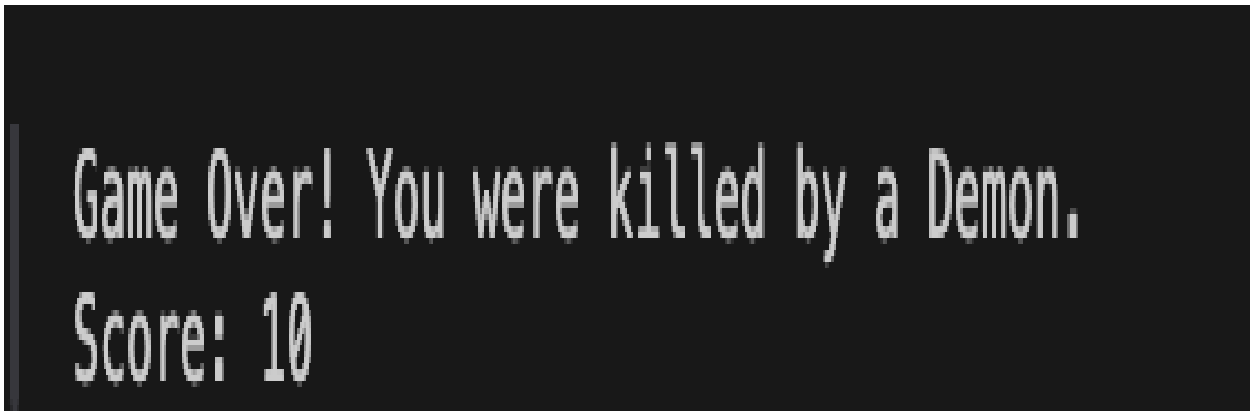
```
devmehta@devs-MacBook-Air-3 Desktop % cd "/Users/devmehta/Desktop/" && gcc tempCodeRunnerFile.c -o tempCodeRunnerFile && "/Users/devmehta/Desktop/"tempCodeRunnerFile
Use W (up), A (left), S (down), D (right)
Press Q to quit
Press Y to start:
```



```
########################################
##      X          #           #    # ##
# . . . . . . . .#. . . . . . # . . . .#
#  ##################################  #
# .################################## .#
#    #             #                   #
# # . . # . . . # . . . . # . . . .#. .#
#                                 X    #
# .#. . . . . . . . . . . . . . . . # .#
#   #                   #        #     #
# . . # . . . . . . C . . . . . . .## .#
##                   X    #  X       # #
# .################################## .#
#  ##################################  #
# . . . . . . . . . . . . . . .#.#X . .#
# #            #  #                    #
# . . . . . . . . . # . # . X . . . . .#
#        #                             #
# . .#.#. . . . . . . . . . # . . . .#
########################################
Score: 0
Total Food: 88, Eaten: 0
```



```
########################################
##      X          #           #    # ##
# . . . . . . . .#. . . . . . # . . . .#
#  ##################################  #
# .################################## .#
#    #             #                   #
# # . . # . . . #     . . # . . . .#. .#
#                                 X    #
# .#. . . . . . .   .           C . # .#
#   #                   #        #     #
# . . # . . . . . .   . . . . . . .## .#
##                   X    #  X       # #
# .################################## .#
#  ##################################  #
# . . . . . . . . . . . . . . .#.#X . .#
# #            #  #                    #
# . . . . . . . . . # . # . X . . . . .#
#        #                             #
# . .#.#. . . . . . . . . . . # . . . .#
########################################
Score: 9
Total Food: 88, Eaten: 9
```



```
Game Over! You were killed by a Demon.
Score: 10
```