IC Project Report for Text-Based Escape Room/Murder mystery Game

Course Code: CSL1010 - Introduction to Computer Science

Group Members:

- Abhinav Kumar (B24CM1002)
- Dalwadi Yug Bharat (B24CM1018)
- Pranav H Nair (B24CS1056)
- Sadat Ul Rouf Wani (B24CM1053)

Project Title

Text-Based Murder Escape room/ Murder Mystery Game with Turn-Based Battle Elements and minigmes

---

Central Idea

This project involves the development of a story-driven, text-based escape room + murder mystery game featuring a variety of minigames and turn-based battle mechanics. Designed in the C programming language, the game aims to create an immersive and dynamic gaming experience. Players will navigate through a rich storyline while solving puzzles, engaging in probabilistic gameplay, and facing challenges influenced by their choices and character actions.

The project integrates classic minigames into the main gameplay loop, providing players with a variety of interactive challenges.

Key Features

1. Dynamic Storyline

- Player choices influence the fate of characters and the narrative's progression.

2. Sanity System

- Characters' sanity affects their behaviour and choices.
- Lower sanity levels may lead to unexpected or chaotic actions.

3. Gameplay Elements

- **Inventory System**: Collect and manage items/tools to solve puzzles and unlock story paths.
- **Turn-Based Combat**: Players engage enemies with probabilistic attack patterns and randomized parameters.

5. Minigames

Integrated into the main storyline:

- **Hangman**: A word-guessing game that utilizes character lore.
- **Maze**: Randomly generated mazes requiring navigation skills.
- **Russian Roulette**: A high-stakes probabilistic challenge.
- **Reaction-Based Game**: Time-sensitive typing challenges.
- **Wordle**: Vocabulary-based word-guessing.

6. Save System

- Players can save progress at specific checkpoints, ensuring smoother gameplay continuity.

7. Open-Source Approach

- The game will be hosted on GitHub, promoting collaboration and accessibility for future contributions.

Technical Implementation

Programming and Libraries

- **C Libraries**:
  - conio.h: Console formatting for a better text-based interface.
  - time.h: To implement time-based events and randomness.
  - windows.h: Provides access to the **Windows API**, a collection of functions, macros, and definitions that allow you to interact with the Windows operating system at a low level.
- **Structures and Pointers**: For managing reusable game elements like characters, items, and enemies.
- **Dynamic Memory Management**: Efficient handling of game states and resource allocation.

Core Minigame Algorithms

1. **Hangman**:
   - Stores a word in an array.
   - Allows player guesses to gradually reveal the word while tracking incorrect attempts.
2. **Russian Roulette**:
   - Simulates a probabilistic chance game with a randomized chamber.
   - Offers tension-filled gameplay with decisions to spin or fire.
3. **Maze**:

    - Randomized maze generation and player navigation logic.
4. **Reaction-Based Game**:
    - Tests typing speed and reflexes using time-sensitive input checks.
5. **Wordle**:
    - Players deduce a word within a set number of attempts, reinforcing vocabulary.

Motivation for the Project

The primary motivation is to learn the intricacies of software development in C, including algorithm design, debugging, and implementing complex gameplay mechanics. Developing a game from scratch serves as a challenging yet rewarding learning opportunity.

Additionally, incorporating multiple minigames and a compelling storyline ensures a rich and engaging experience for players. The open-source nature encourages collaboration and skill-sharing.

Skills and Knowledge to Acquire

- **Technical Skills**:
    - Structs, pointers, and dynamic memory management.
    - Effective use of libraries like conio.h and time.h.
- **Teamwork and Collaboration**:
    - Leveraging GitHub for version control and team collaboration.
    - Learning the software development life cycle (SDLC).
- **Game Development**:
    - Logical structuring of game loops and integrating diverse gameplay mechanics.

Game-Wise reports:

1) Wordle

Key Features of the Code

1. Dynamic Gameplay Logic

- **Random Word Selection**:
    - The program randomly selects a five-letter word from an external source (words.h), indexed using rand().

    - This randomization ensures that the gameplay feels fresh and unpredictable in every run.
- **Color Feedback**:
    - The functions green(), yellow(), and white() utilize ANSI escape codes to provide visual feedback for the guessed word.
    - Correctly placed letters turn green, misplaced but correct letters turn yellow, and incorrect letters remain white.

---

2. Word Checking Mechanism

- The check() function:
    - Iterates through the guessed word (entered) and compares it against the target word (given) using a **case-insensitive comparison** (tolower).
    - Tracks guessed letter states using an auxiliary array states[] to avoid highlighting duplicate letters multiple times incorrectly.
    - Provides immediate feedback to the player by printing the result with appropriate colors.

---

3. Sanity Feature

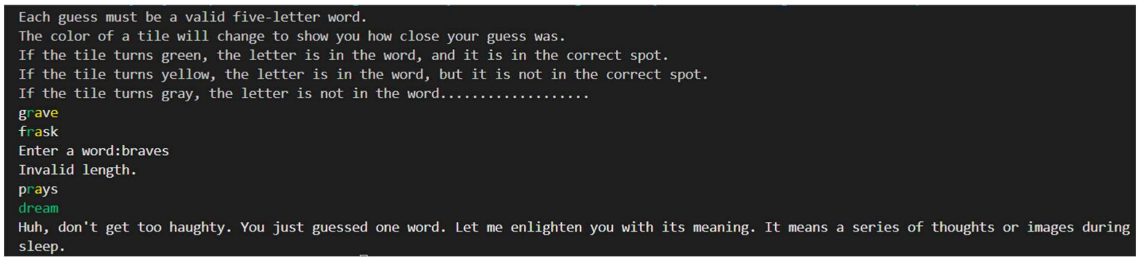
- **Dynamic Word Mutation**:
    - A randomized feature based on the "sanity" parameter alters one letter of the guessed word if the player's sanity drops below a certain threshold.
    - This is implemented using:
        - rand() for generating random indices and probabilities.
        - A probabilistic check (x > sanity) to decide when to mutate the word.

---

4. Input Validation

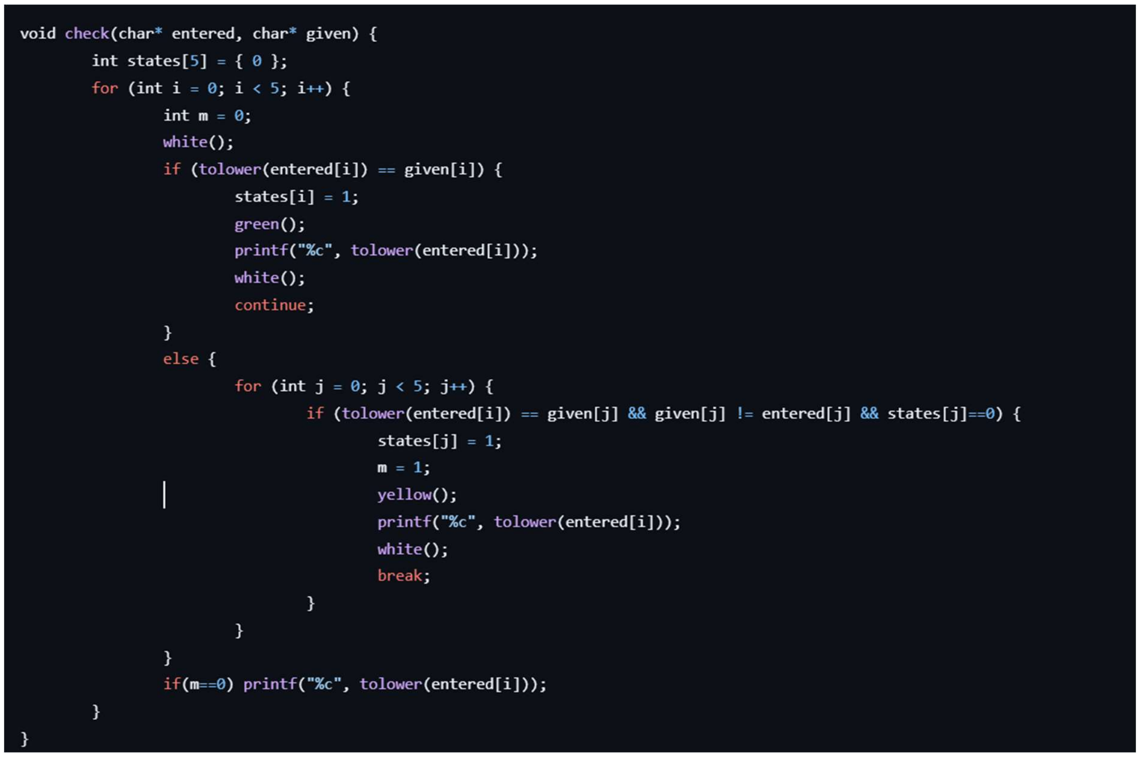
- **Input Length Check**:
    - Ensures that the player's input is exactly five characters long. If not, it prompts the user to re-enter a valid word.

- **Game snapshot**:

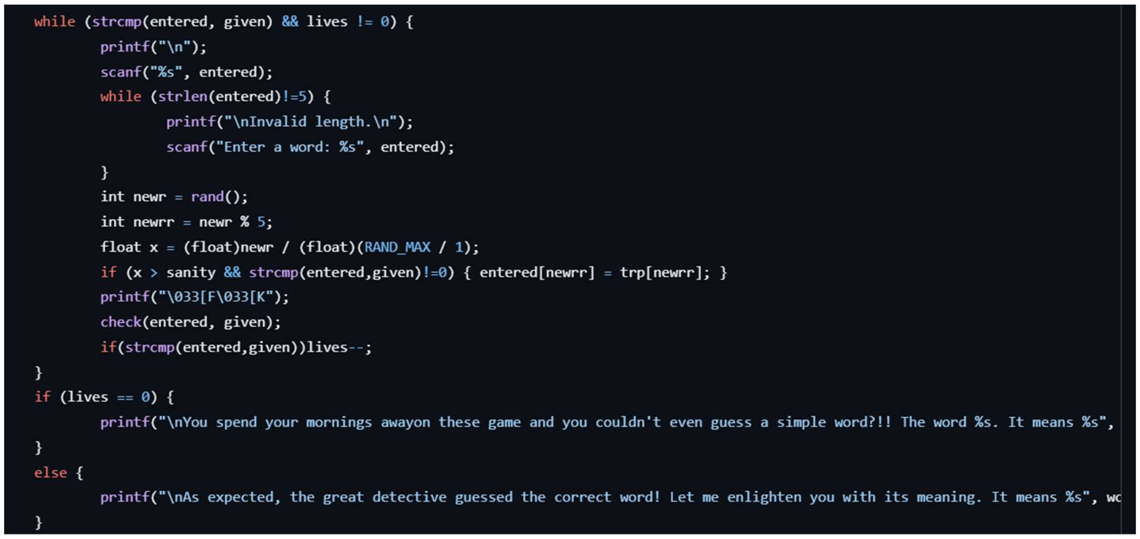


```
Each guess must be a valid five-letter word.
The color of a tile will change to show you how close your guess was.
If the tile turns green, the letter is in the word, and it is in the correct spot.
If the tile turns yellow, the letter is in the word, but it is not in the correct spot.
If the tile turns gray, the letter is not in the word..................
grave
frask
Enter a word:braves
Invalid length.
prays
dream
Huh, don't get too haughty. You just guessed one word. Let me enlighten you with its meaning. It means a series of thoughts or images during
sleep.
```

- **Code snapshots**:



```
void check(char* entered, char* given) {
        int states[5] = { 0 };
        for (int i = 0; i < 5; i++) {
                int m = 0;
                white();
                if (tolower(entered[i]) == given[i]) {
                        states[i] = 1;
                        green();
                        printf("%c", tolower(entered[i]));
                        white();
                        continue;
                }
                else {
                        for (int j = 0; j < 5; j++) {
                                if (tolower(entered[i]) == given[j] && given[j] != entered[j] && states[j]==0) {
                                        states[j] = 1;
                                        m = 1;
                                        yellow();
                                        printf("%c", tolower(entered[i]));
                                        white();
                                        break;
                                }
                        }
                }
                if(m==0) printf("%c", tolower(entered[i]));
        }
}
```



```
while (strcmp(entered, given) && lives != 0) {
        printf("\n");
        scanf("%s", entered);
        while (strlen(entered)!=5) {
                printf("\nInvalid length.\n");
                scanf("Enter a word: %s", entered);
        }
        int newr = rand();
        int newrr = newr % 5;
        float x = (float)newr / (float)(RAND_MAX / 1);
        if (x > sanity && strcmp(entered,given)!=0) { entered[newrr] = trp[newrr]; }
        printf("\033[F\033[K");
        check(entered, given);
        if(strcmp(entered,given))lives--;
}
if (lives == 0) {
        printf("\nYou spend your mornings awayon these game and you couldn't even guess a simple word?!! The word %s. It means %s",
}
else {
        printf("\nAs expected, the great detective guessed the correct word! Let me enlighten you with its meaning. It means %s", wo
}
```

---

2) Battle Elements:

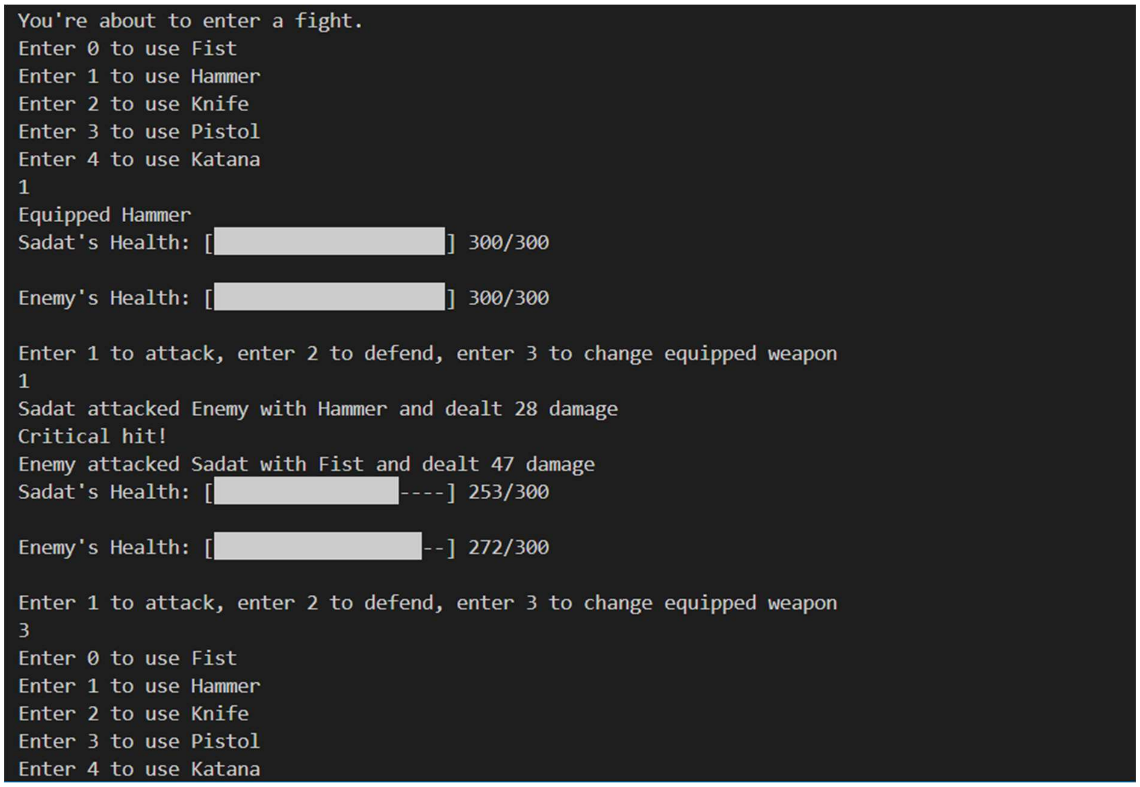
Key Features of the Code:

1. **Use of Pointers for Character Manipulation**:
   - The program uses pointers to pass the Character structures to functions like initialize, equip, atk, and battle. This allows for **direct manipulation of the character's attributes** (e.g., health, equipped weapon, and shield status) without creating unnecessary copies of the structure, which optimizes memory usage and improves performance.
2. **Dynamic Weapon Management**:
   - The Weapon and Character structures are tightly integrated, with a Weapon object assigned dynamically during gameplay using the equip function. The ownership of weapons is tracked with the isowned array, enabling or disabling access to specific weapons.
   - The program demonstrates how **structured data** is handled efficiently and dynamically updated during runtime.
3. **Health Bar Visualization**:
   - The showhealth function creates a **graphical health bar** that reflects the player's or enemy's current health status relative to their maximum health. This feature adds to the game's user experience by providing visual feedback about progress in battle.
4. **Battle Mechanics with Randomized Damage and Critical Hits**:
   - Damage calculation in the atk function incorporates randomization (rand()) to simulate realistic combat outcomes, including the possibility of critical hits. The use

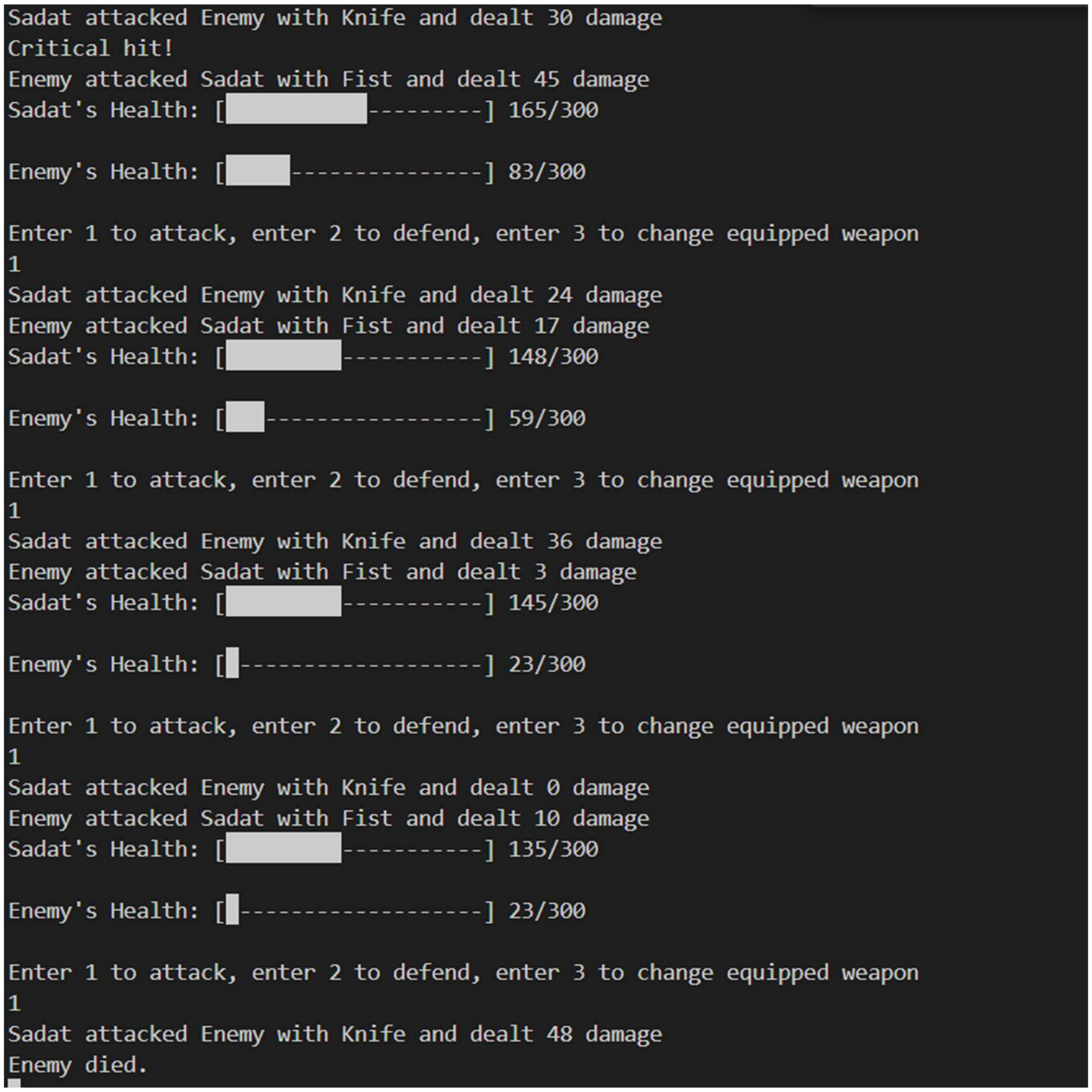
of rand() for critical hit probabilities and damage boosts dynamically influences the flow of battles.

5. **File Handling Potential for Weapon and Character Management**:
   - Although the current code does not directly involve file I/O, it has the potential to integrate **file handling** for loading or saving game states. For instance:
     - The Weapon data (e.g., mname and boost) and Character attributes (e.g., isowned and mhp) could be loaded from or written to files using fopen, fscanf, and fprintf.
     - A possible implementation could include saving and loading the player's progress, equipped weapons, and health stats for continuity between sessions.

Game snapshots:

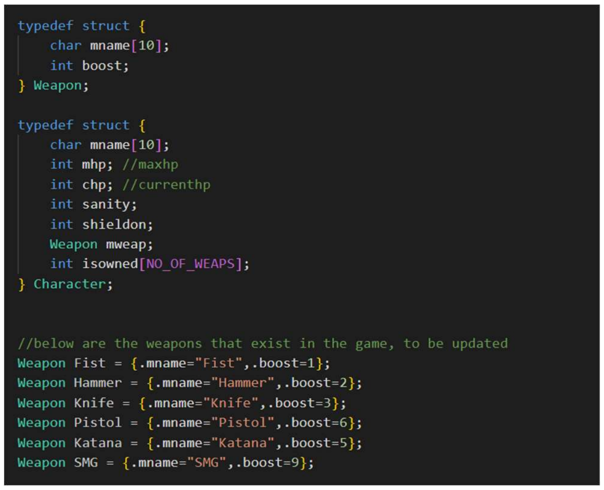


```
You're about to enter a fight.
Enter 0 to use Fist
Enter 1 to use Hammer
Enter 2 to use Knife
Enter 3 to use Pistol
Enter 4 to use Katana
1
Equipped Hammer
Sadat's Health: [████████████████████] 300/300

Enemy's Health: [████████████████████] 300/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
1
Sadat attacked Enemy with Hammer and dealt 28 damage
Critical hit!
Enemy attacked Sadat with Fist and dealt 47 damage
Sadat's Health: [████████████████----] 253/300

Enemy's Health: [██████████████████--] 272/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
3
Enter 0 to use Fist
Enter 1 to use Hammer
Enter 2 to use Knife
Enter 3 to use Pistol
Enter 4 to use Katana
```

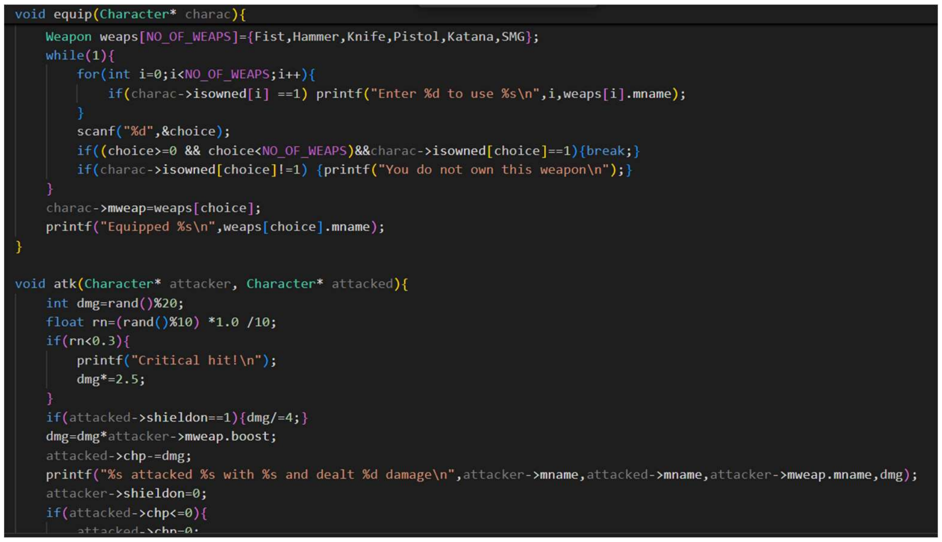


```
Sadat attacked Enemy with Knife and dealt 30 damage
Critical hit!
Enemy attacked Sadat with Fist and dealt 45 damage
Sadat's Health: [███████████---------] 165/300

Enemy's Health: [█████---------------] 83/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
1
Sadat attacked Enemy with Knife and dealt 24 damage
Enemy attacked Sadat with Fist and dealt 17 damage
Sadat's Health: [█████████-----------] 148/300

Enemy's Health: [███-----------------] 59/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
1
Sadat attacked Enemy with Knife and dealt 36 damage
Enemy attacked Sadat with Fist and dealt 3 damage
Sadat's Health: [█████████-----------] 145/300

Enemy's Health: [█-------------------] 23/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
1
Sadat attacked Enemy with Knife and dealt 0 damage
Enemy attacked Sadat with Fist and dealt 10 damage
Sadat's Health: [█████████-----------] 135/300

Enemy's Health: [█-------------------] 23/300

Enter 1 to attack, enter 2 to defend, enter 3 to change equipped weapon
1
Sadat attacked Enemy with Knife and dealt 48 damage
Enemy died.
```

Code snapshots:



```c
typedef struct {
    char mname[10];
    int boost;
} Weapon;

typedef struct {
    char mname[10];
    int mhp; //maxhp
    int chp; //currenthp
    int sanity;
    int shieldon;
    Weapon mweap;
    int isowned[NO_OF_WEAPS];
} Character;


//below are the weapons that exist in the game, to be updated
Weapon Fist = {.mname="Fist",.boost=1};
Weapon Hammer = {.mname="Hammer",.boost=2};
Weapon Knife = {.mname="Knife",.boost=3};
Weapon Pistol = {.mname="Pistol",.boost=6};
Weapon Katana = {.mname="Katana",.boost=5};
Weapon SMG = {.mname="SMG",.boost=9};
```



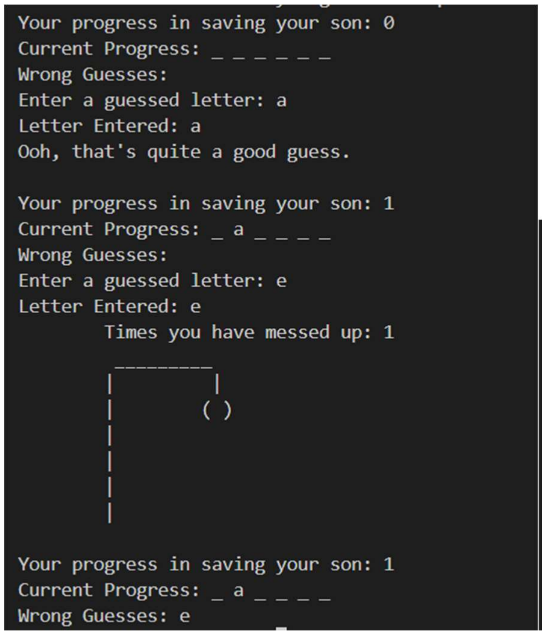
```c
void equip(Character* charac){
    Weapon weaps[NO_OF_WEAPS]={Fist,Hammer,Knife,Pistol,Katana,SMG};
    while(1){
        for(int i=0;i<NO_OF_WEAPS;i++){
            if(charac->isowned[i] ==1) printf("Enter %d to use %s\n",i,weaps[i].mname);
        }
        scanf("%d",&choice);
        if((choice>=0 && choice<NO_OF_WEAPS)&&charac->isowned[choice]==1){break;}
        if(charac->isowned[choice]!=1) {printf("You do not own this weapon\n");}
    }
    charac->mweap=weaps[choice];
    printf("Equipped %s\n",weaps[choice].mname);
}

void atk(Character* attacker, Character* attacked){
    int dmg=rand()%20;
    float rn=(rand()%10) *1.0 /10;
    if(rn<0.3){
        printf("Critical hit!\n");
        dmg*=2.5;
    }
    if(attacked->shieldon==1){dmg/=4;}
    dmg=dmg*attacker->mweap.boost;
    attacked->chp-=dmg;
    printf("%s attacked %s with %s and dealt %d damage\n",attacker->mname,attacked->mname,attacker->mweap.mname,dmg);
    attacker->shieldon=0;
    if(attacked->chp<=0){
        attacked->chp=0;
```

4) Hangman:

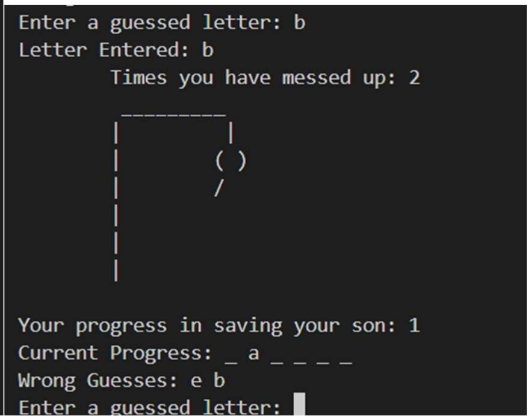
Key Features of the Hangman Code:

1. **Dynamic Word Selection:**
   - **The program selects a random word from a predefined word list (Hangman_Word.h), ensuring variety in each game session using srand and rand.**
2. **Interactive Gameplay:**
   - **Provides a clear visual representation of the hangman and tracks the player's progress by displaying correctly guessed letters and missed attempts.**
3. **Real-Time Feedback:**
   - **Continuously updates the player with the current word status, wrong guesses, and the evolving hangman drawing as mistakes are made.**
4. **User Interface with Styling:**
   - **Enhances the game experience using ANSI color codes (e.g., red for losing and green for winning messages), adding immersion and emphasis during critical moments.**
5. **Custom Hangman Drawing:**
   - **Dynamically builds the hangman figure as the player makes mistakes, with increasing severity until the game is lost.**

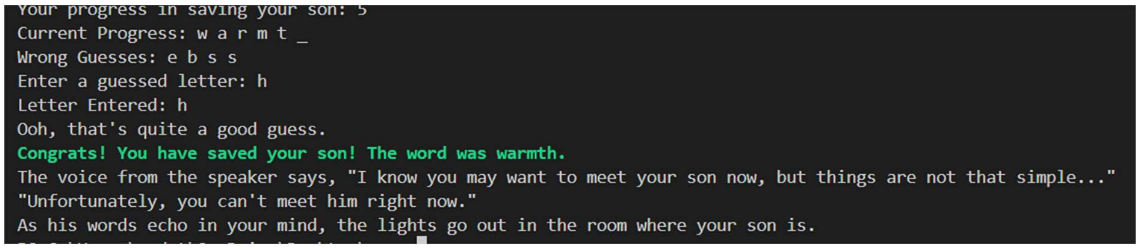
Game snapshots:



```
Your progress in saving your son: 0
Current Progress: _ _ _ _ _ _
Wrong Guesses:
Enter a guessed letter: a
Letter Entered: a
Ooh, that's quite a good guess.

Your progress in saving your son: 1
Current Progress: _ a _ _ _ _
Wrong Guesses:
Enter a guessed letter: e
Letter Entered: e
        Times you have messed up: 1
         _________
        |         |
        |        ( )
        |
        |
        |
        |

Your progress in saving your son: 1
Current Progress: _ a _ _ _ _
Wrong Guesses: e
```

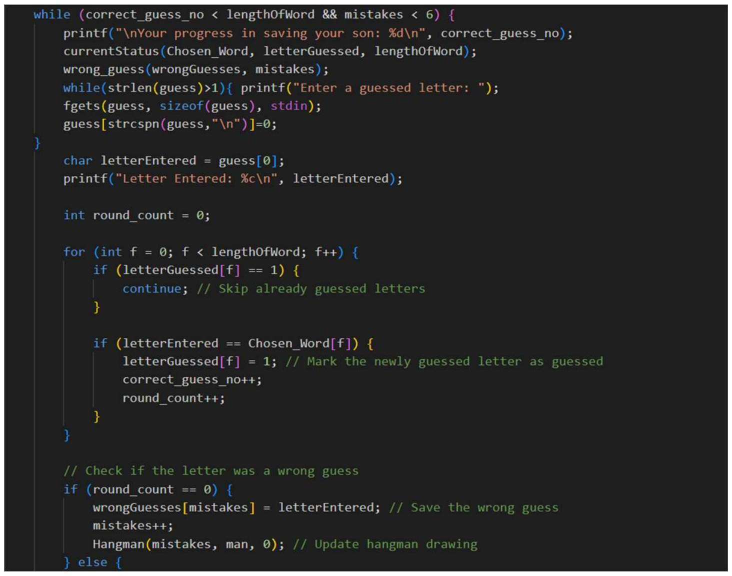


```
Enter a guessed letter: b
Letter Entered: b
        Times you have messed up: 2
         _________
        |         |
        |        ( )
        |         /
        |
        |
        |

Your progress in saving your son: 1
Current Progress: _ a _ _ _ _
Wrong Guesses: e b
Enter a guessed letter:
```

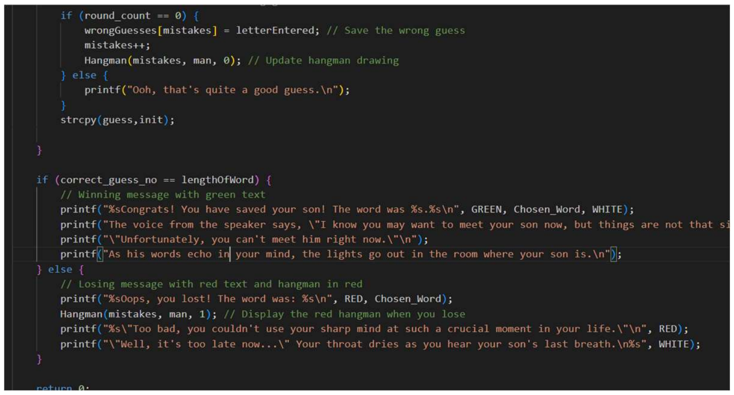


```
Your progress in saving your son: 5
Current Progress: w a r m t _
Wrong Guesses: e b s s
Enter a guessed letter: h
Letter Entered: h
Ooh, that's quite a good guess.
Congrats! You have saved your son! The word was warmth.
The voice from the speaker says, "I know you may want to meet your son now, but things are not that simple..."
"Unfortunately, you can't meet him right now."
As his words echo in your mind, the lights go out in the room where your son is.
```

Code snapshots:



```c
while (correct_guess_no < lengthOfWord && mistakes < 6) {
    printf("\nYour progress in saving your son: %d\n", correct_guess_no);
    currentStatus(Chosen_Word, letterGuessed, lengthOfWord);
    wrong_guess(wrongGuesses, mistakes);
    while(strlen(guess)>1){ printf("Enter a guessed letter: ");
    fgets(guess, sizeof(guess), stdin);
    guess[strcspn(guess,"\n")]=0;
}
    char letterEntered = guess[0];
    printf("Letter Entered: %c\n", letterEntered);

    int round_count = 0;

    for (int f = 0; f < lengthOfWord; f++) {
        if (letterGuessed[f] == 1) {
            continue; // Skip already guessed letters
        }

        if (letterEntered == Chosen_Word[f]) {
            letterGuessed[f] = 1; // Mark the newly guessed letter as guessed
            correct_guess_no++;
            round_count++;
        }
    }

    // Check if the letter was a wrong guess
    if (round_count == 0) {
        wrongGuesses[mistakes] = letterEntered; // Save the wrong guess
        mistakes++;
        Hangman(mistakes, man, 0); // Update hangman drawing
    } else {
```



```c
    if (round_count == 0) {
        wrongGuesses[mistakes] = letterEntered; // Save the wrong guess
        mistakes++;
        Hangman(mistakes, man, 0); // Update hangman drawing
    } else {
        printf("Ooh, that's quite a good guess.\n");
    }
    strcpy(guess,init);

}

if (correct_guess_no == lengthOfWord) {
    // Winning message with green text
    printf("%sCongrats! You have saved your son! The word was %s.%s\n", GREEN, Chosen_Word, WHITE);
    printf("The voice from the speaker says, \"I know you may want to meet your son now, but things are not that si
    printf("\"Unfortunately, you can't meet him right now.\"\n");
    printf("As his words echo in your mind, the lights go out in the room where your son is.\n");
} else {
    // Losing message with red text and hangman in red
    printf("%sOops, you lost! The word was: %s\n", RED, Chosen_Word);
    Hangman(mistakes, man, 1); // Display the red hangman when you lose
    printf("%s\"Too bad, you couldn't use your sharp mind at such a crucial moment in your life.\"\n", RED);
    printf("\"Well, it's too late now...\" Your throat dries as you hear your son's last breath.\n%s", WHITE);
}

return 0;
```

5) Maze:

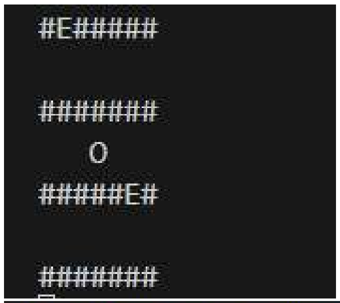
Key Features of the Code:

1. **Dynamic Map Generation:**
    - **A game grid (map) of HEIGHT x WIDTH is initialized with walls (#), spaces ( ), and a player (O), with randomly placed horizontal obstacles leaving gaps for navigation.**
2. **Player Movement:**
    - **The player (O) moves in response to keyboard inputs (W, A, S, D) while avoiding collisions with walls (#).**
3. **Real-Time Gameplay:**
    - **The game runs in a continuous loop, updating the grid and redrawing it after every move to provide real-time feedback.**
4. **Randomized Obstacles:**

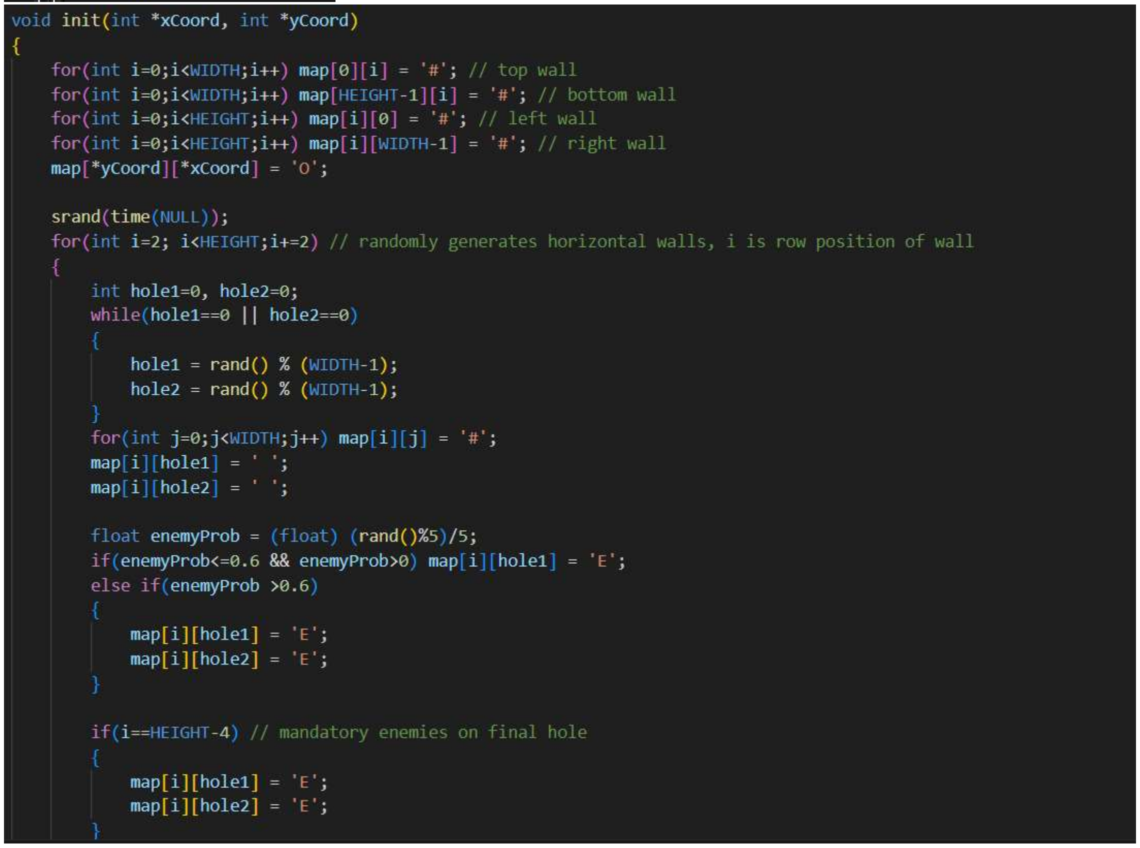
- Horizontal walls are generated randomly with alternating gaps at the left or right end, adding variability and challenge.

5. **Boundary Collision Handling:**
   - Ensures the player cannot move beyond the map boundaries or through walls by checking conditions before updating the player's position.

6. **Screen Clearing:**
   - Uses system("cls") to refresh the console display after every move, simulating a smooth animation for the player's movement.



```
#E#####

#######
   O
#####E#

#######
```



```c
void init(int *xCoord, int *yCoord)
{
    for(int i=0;i<WIDTH;i++) map[0][i] = '#'; // top wall
    for(int i=0;i<WIDTH;i++) map[HEIGHT-1][i] = '#'; // bottom wall
    for(int i=0;i<HEIGHT;i++) map[i][0] = '#'; // left wall
    for(int i=0;i<HEIGHT;i++) map[i][WIDTH-1] = '#'; // right wall
    map[*yCoord][*xCoord] = 'O';

    srand(time(NULL));
    for(int i=2; i<HEIGHT;i+=2) // randomly generates horizontal walls, i is row position of wall
    {
        int hole1=0, hole2=0;
        while(hole1==0 || hole2==0)
        {
            hole1 = rand() % (WIDTH-1);
            hole2 = rand() % (WIDTH-1);
        }
        for(int j=0;j<WIDTH;j++) map[i][j] = '#';
        map[i][hole1] = ' ';
        map[i][hole2] = ' ';

        float enemyProb = (float) (rand()%5)/5;
        if(enemyProb<=0.6 && enemyProb>0) map[i][hole1] = 'E';
        else if(enemyProb >0.6)
        {
            map[i][hole1] = 'E';
            map[i][hole2] = 'E';
        }

        if(i==HEIGHT-4) // mandatory enemies on final hole
        {
            map[i][hole1] = 'E';
            map[i][hole2] = 'E';
        }
```

6) Russian Roulette:

Key Features of the Code:

1. **Russian Roulette Game Logic:**
   - **Simulates the dangerous game where a revolver with six chambers contains two bullets. The player can choose to shoot themselves or another character.**
2. **Randomized Outcomes:**
   - **Uses rand() to randomly determine the locations of the two empty chambers and the chamber chosen for firing, ensuring unpredictability.**

4. **Choice-Based Gameplay:**
   - **Provides the player with two options: shoot themselves or the victim. The game proceeds based on their choice.**
5. **Outcome Messages:**
   - **Displays messages based on whether the chamber fired is empty or contains a bullet, determining the survival of the player or victim.**

7. **Probabilistic Mechanics:**
   - **Ensures realistic odds by generating chamber numbers between 1 to 6 and checking against the pre-determined empty chambers.**

Conclusion

This project represents an ambitious yet achievable goal of building a multifaceted game from scratch. By focusing on storytelling, gameplay mechanics, and technical proficiency, it promises to deliver a rewarding experience for both the developers and players. The team is committed to overcoming challenges and creating a high-quality, open-source game that showcases creativity and teamwork.