University of Science &Technology Chittagong (USTC)

FACULTY of SCIENCE ENGINEERING & TECHNOLOGY (FSET)

Department of Computer Science & Engineering(CSE)

Submitted to:

Prianka Das

Lecturer ,

Department of Computer Science & Engineering

Group Name: Cyber Squad

Course: CSE 114(Structured Programming)

Lab Report: 01

Topic Name: Treasuse Hunt Game

Name	Roll No
Md. Kamruzzaman	24070101
S.M. Ali Akbar khaled	24070103
Md. Razaul Karim	24070127
Naimul Karim	24070128
Md. Jane Alam Jony	24070136

Semester: 1st

Batch: 43rd

Section: A

Department: Computer Science & Engineering

PROJECT NAME:- TREASURE HUNT GAME.

Introduction:

Welcome to the **Treasure Hunt Game**! This is an interactive grid-based adventure game where you explore a grid in search of treasures while avoiding traps and collecting power-ups. The game is turn-based, with both the player and a computer opponent moving around the grid. The goal is to collect as many treasures as possible while managing your health and utilizing power-ups to survive traps. You'll have a limited number of moves, and your final score depends on how well you navigate the grid, find treasures, and avoid dangers.

Game Explanation:

- ❖ **Grid**: The game is played on a 5x5 grid, where each cell can contain different items such as:

    - ♦ **Treasures (T)**: Valuable items you need to find.
    - ♦ **Traps (X)**: Dangerous locations that reduce your health.
    - ♦ **Power-ups (+)**: Useful items that help you survive traps and maintain health.

- ❖ **Player and Computer**: You control the player's movement on the grid, while the computer also moves randomly, searching for treasures and encountering traps along the way.
- ❖ **Movement**: You can move the player in four directions:
    - ♦ **W**: Move up
    - ♦ **A**: Move left
    - ♦ **S**: Move down
    - ♦ **D**: Move right
- ❖ **Health and Power-ups**:
    - ♦ You start with **3 health points**. If you hit a trap and have no power-ups, you lose health.
    - ♦ **Power-ups** allow you to survive a trap without losing health. You can collect power-ups by landing on cells marked with +.
- ❖ **Objective**:
    - ♦ Your goal is to find all the **3 treasures** (T) on the grid.
    - ♦ Avoid **traps** (X) and manage your **health**.
    - ♦ You have a limited number of **20 moves**, so you need to act quickly!
- ❖ **Scoring**:
    - ♦ Each treasure you find is worth **10 points**.
    - ♦ Each power-up you collect adds **5 points** to your score.
    - ♦ Remaining health at the end of the game adds **10 points** per health point.
- ❖ **End Game**: The game ends when you either:
    - ♦ Find all the treasures and achieve a high score.
    - ♦ Run out of moves.
    - ♦ Lose all your health.

Key Features:

1. **Grid Exploration**: Navigate a 5x5 grid and uncover hidden items.
2. **Strategy**: Make smart decisions on when to use power-ups or avoid traps.

3. **Turn-based**: Play against the computer, which moves randomly, adding a layer of unpredictability.

Good luck on your treasure hunt, and may you find all the treasures while avoiding the traps!

Code:

```c
#include<stdio.h>

#include <stdlib.h>

#include <time.h>


#define GRID_SIZE 5

#define TREASURES 3

#define TRAPS 3

#define POWER_UPS 2

#define MAX_MOVES 20

#define INITIAL_HEALTH 3


void initializeGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]);

void placeItems(char grid[GRID_SIZE][GRID_SIZE], char item, int count);

void displayGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int playerX,
int playerY, int computerX, int computerY);

int isValidMove(int x, int y);

void computerMove(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int
*computerX, int *computerY, int *treasuresFound);

int playGame(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]);


int main() {

    char grid[GRID_SIZE][GRID_SIZE];
```

```c
    int revealed[GRID_SIZE][GRID_SIZE] = {0};

    srand(time(NULL));


    initializeGrid(grid, revealed);

    placeItems(grid, 'T', TREASURES); // Place treasures

    placeItems(grid, 'X', TRAPS);    // Place traps

    placeItems(grid, '+', POWER_UPS); // Place power-ups


    printf("Welcome to the Treasure Hunt Game!\n");

    printf("Navigate the grid to find treasures (T), avoid traps (X), and collect power-ups (+).\n");

    printf("You have %d moves to find treasures and avoid traps!\n\n", MAX_MOVES);


    int score = playGame(grid, revealed);


    printf("Game Over! Your final score is: %d\n", score);

    return 0;

}


void initializeGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]) {

    for (int i = 0; i < GRID_SIZE; i++) {

        for (int j = 0; j < GRID_SIZE; j++) {

            grid[i][j] = '.';

            revealed[i][j] = 0;

        }

    }

}
```

```c
void placeItems(char grid[GRID_SIZE][GRID_SIZE], char item, int count) {

    while (count > 0) {

        int x = rand() % GRID_SIZE;

        int y = rand() % GRID_SIZE;

        if (grid[x][y] == '.') {

            grid[x][y] = item;

            count--;

        }

    }

}


void displayGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int playerX,
int playerY, int computerX, int computerY) {

    for (int i = 0; i < GRID_SIZE; i++) {

        for (int j = 0; j < GRID_SIZE; j++) {

            if (i == playerX && j == playerY && i == computerX && j == computerY) {

                printf("PC "); // Both player and computer at the same position

            } else if (i == playerX && j == playerY) {

                printf("P "); // Player's position

            } else if (i == computerX && j == computerY) {

                printf("C "); // Computer's position

            } else if (revealed[i][j]) {

                printf("%c ", grid[i][j]);

            } else {

                printf("? ");
```

```c
        }

      }

      printf("\n");

    }

}


int isValidMove(int x, int y) {

    return x >= 0 && x < GRID_SIZE && y >= 0 && y < GRID_SIZE;

}


void computerMove(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int
*computerX, int *computerY, int *treasuresFound) {

    int newX, newY;

    do {

        newX = *computerX + (rand() % 3 - 1); // Random move: -1, 0, or +1

        newY = *computerY + (rand() % 3 - 1);

    } while (!isValidMove(newX, newY));


    revealed[newX][newY] = 1; // Mark the new position as revealed


    char cell = grid[newX][newY];

    printf("Computer moved to (%d, %d): ", newX, newY);

    if (cell == 'T') {

        printf("found a treasure!\n");

        (*treasuresFound)++;

        initializeGrid(grid, revealed); // Reset the grid visibility
```

```c
    } else if (cell == 'X') {

       printf("hit a trap!\n");

    } else if (cell == '+') {

       printf("found a power-up!\n");

    } else {

       printf("nothing here.\n");

    }



    *computerX = newX;

    *computerY = newY;

}


int playGame(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]) {

    int treasuresFound = 0;

    int powerUps = 0;

    int health = INITIAL_HEALTH;

    int moves = MAX_MOVES;

    int playerX = 0, playerY = 0;

    int computerX = GRID_SIZE - 1, computerY = GRID_SIZE - 1;

    char move;


    revealed[playerX][playerY] = 1; // Player's starting position is revealed

    revealed[computerX][computerY] = 1; // Computer's starting position is revealed


    while (moves > 0 && health > 0) {

       printf("\nCurrent Grid:\n");
```

```c
        displayGrid(grid, revealed, playerX, playerY, computerX, computerY);

        printf("\nTreasures found: %d, Power-ups: %d, Health: %d, Remaining moves: %d\n", treasuresFound,
powerUps, health, moves);

        printf("Enter move (W/A/S/D): ");

        scanf(" %c", &move);


        // Update player's position

        int newX = playerX;

        int newY = playerY;

        if (move == 'W' || move == 'w') newX--;

        else if (move == 'A' || move == 'a') newY--;

        else if (move == 'S' || move == 's') newX++;

        else if (move == 'D' || move == 'd') newY++;

        else {

            printf("Invalid move!\n");

            continue;

        }


        if (!isValidMove(newX, newY)) {

            printf("Out of bounds!\n");

            continue;

        }


        revealed[newX][newY] = 1; // Mark the new position as revealed


        char cell = grid[newX][newY];
```

```c
        if (cell == 'T') {

            printf("You found a treasure!\n");

            treasuresFound++;

            initializeGrid(grid, revealed); // Reset the grid visibility

        } else if (cell == 'X') {

            if (powerUps > 0) {

                printf("You hit a trap but used a power-up to survive!\n");

                powerUps--;

            } else {

                printf("You hit a trap! Health reduced by 1.\n");

                health--;

            }

        } else if (cell == '+') {

            printf("You found a power-up!\n");

            powerUps++;

        } else {

            printf("Nothing here.\n");

        }


        playerX = newX;

        playerY = newY;


        // Computer's turn

        computerMove(grid, revealed, &computerX, &computerY, &treasuresFound);


        moves--;
```

```c
    if (treasuresFound == TREASURES) {

        printf("Congratulations! You found all the treasures!\n");

        return treasuresFound * 10 + powerUps * 5 + health * 10;

    }

  }


  if (health <= 0) {

    printf("You ran out of health!\n");

  } else {

    printf("You ran out of moves!\n");

  }


  return treasuresFound * 10 + powerUps * 5 + health * 10;

}
```

Explanation:-

Key Terms and Concepts:

1. **Grid**: The game board is a 5x5 grid represented as a two-dimensional array. Each cell in this grid can contain different items like treasures, traps, or power-ups. The grid is displayed to the player and updated with each move.
2. **Treasures (T)**: These are the valuable items you need to find. The game places three treasures on the grid. Every treasure you find adds **10 points** to your score.
3. **Traps (X)**: These are dangerous locations that reduce your health. The game places three traps on the grid. If you step on a trap and don't have any power-ups, you lose 1 health point.
4. **Power-ups (+)**: Power-ups are helpful items that allow you to survive traps without losing health. There are two power-ups placed randomly on the grid. Each power-up you collect adds **5 points** to your score.
5. **Player Movement**: The player can move in four directions:
   - **W** (Up)
   - **A** (Left)
   - **S** (Down)
   - **D** (Right)
6. **Computer Movement**: The computer also moves randomly on the grid. It behaves similarly to the player but is not controlled by the user.
7. **Health**: The player starts with **3 health points**. If the player hits a trap and has no power-ups, health is reduced. If the player's health reaches zero, the game ends.
8. **Moves**: The player has a maximum of **20 moves**. The game ends if the player runs out of moves, or if the player finds all the treasures or loses all health.
9. **Score**: The score is based on the number of treasures found, power-ups collected, and remaining health. The final score is calculated as:
   - Treasures: **10 points each**
   - Power-ups: **5 points each**
   - Health: **10 points per remaining health point**

Function Breakdown:

1. **`initializeGrid`**:
   - **Purpose**: Initializes the grid by filling all cells with `.` (empty spaces) and sets all cells as "not revealed" (false).
   - **Parameters**:
     - `grid`: A 2D array representing the grid.
     - `revealed`: A 2D array of booleans that tracks whether a cell has been revealed or not.
   - **Action**: Iterates over each cell of the grid and sets the grid cell to `.` (empty) and marks all cells as `false` in the `revealed` array.

2. **`placeItems`**:
   - **Purpose**: Places a given item (like a treasure, trap, or power-up) randomly on the grid.
   - **Parameters**:
     - `grid`: A 2D array representing the game grid.
     - `item`: The character representing the item to place (e.g., `'T'` for treasures).
     - `count`: The number of items to place on the grid.
   - **Action**: It loops to randomly place the specified number of items on the grid, making sure no item is placed in a cell that is already occupied.

3. **`displayGrid`**:
   - **Purpose**: Displays the current state of the grid to the player.

- **Parameters**:
  - ➢ `grid`: The current grid with items (`T`, `X`, `+`).
  - ➢ `revealed`: A 2D boolean array that tracks which cells have been revealed to the player.
  - ➢ `playerX`, `playerY`: The current position of the player on the grid.
  - ➢ `computerX`, `computerY`: The current position of the computer on the grid.
- **Action**: Prints the grid, showing `?` for unrevealed cells, `P` for the player's position, `C` for the computer's position, and actual items (`T`, `X`, `+`) for revealed cells. If the player and the computer are on the same cell, it shows `PC`.

4. **`isValidMove`**:
   - **Purpose**: Checks if a move is within the grid boundaries.
   - **Parameters**:
     - ➢ `x`, `y`: The coordinates to check.
   - **Action**: Returns `true` if the coordinates are within the bounds of the grid (0 to `GRID_SIZE-1` for both `x` and `y`), otherwise returns `false`.

5. **`computerMove`**:
   - **Purpose**: Moves the computer to a new random position on the grid.
   - **Parameters**:
     - ➢ `grid`: The game grid.
     - ➢ `revealed`: The array that tracks which cells have been revealed.
     - ➢ `computerX`, `computerY`: The current position of the computer on the grid.
   - **Action**: The computer attempts to move randomly by selecting a new position based on its current position. It checks if the new position is valid, then updates the revealed array, and prints the result of the computer's move (whether it found a treasure, a trap, or nothing).

6. **`playGame`**:
   - **Purpose**: The main game loop where the player and computer take turns moving and interacting with the grid.
   - **Parameters**:
     - ➢ `grid`: The game grid.
     - ➢ `revealed`: The array tracking which cells are revealed.
   - **Action**: This function controls the flow of the game:
     - ➢ It initializes the player's position, sets up the grid, and enters a loop that continues as long as the player has health and moves left.
     - ➢ The player enters a move (W/A/S/D), and their position is updated.
     - ➢ After the player moves, the computer also moves randomly.
     - ➢ The game ends if the player finds all the treasures, runs out of moves, or loses all health.
     - ➢ The function returns the player's score based on the number of treasures found, power-ups collected, and remaining health.

Let's go through the key terms and explain them in the context of the program:

1. `#include` Directives:

- **`#include <stdio.h>`**: This line tells the compiler to include the standard input/output library, which is used for input and output functions such as `printf` and `scanf`.
- **`#include <stdlib.h>`**: This line includes the standard library, which provides functions for memory allocation, random number generation (`rand()`), and others like `srand()`.
- **`#include <time.h>`**: This header defines functions related to time. It is used in this program to generate random numbers based on the current time (`srand(time(NULL))`).
- **`#include <stdbool.h>`**: This header file allows the use of boolean values (`true` and `false`) in C programs.

2. `#define` Macros:

These macros define constants that will be used throughout the program:

- 🔸 `GRID_SIZE 5`: Defines the size of the grid as 5x5 (i.e., the grid is 5 rows and 5 columns).
- 🔸 `TREASURES 3`: Specifies that there will be 3 treasures (`'T'`) placed on the grid.
- 🔸 `TRAPS 3`: Specifies that there will be 3 traps (`'X'`) placed on the grid.
- 🔸 `POWER_UPS 2`: Specifies that there will be 2 power-ups (`'+'`) placed on the grid.
- 🔸 `MAX_MOVES 20`: Defines the maximum number of moves the player can make in the game.
- 🔸 `INITIAL_HEALTH 3`: Specifies that the player starts with 3 health points.

3. Function Prototypes:

These are declarations of functions that will be defined later in the code. They help the compiler know about the functions before they are used in the program.

- 🔸 `initializeGrid()`: Initializes the grid by setting all the cells to . (empty space) and marking them as unrevealed.
- 🔸 `placeItems()`: Randomly places treasures (`'T'`), traps (`'X'`), and power-ups (`'+'`) on the grid.
- 🔸 `displayGrid()`: Displays the current grid, showing the player's position, the computer's position, and revealing cells based on the game progress.
- 🔸 `isValidMove()`: Checks if a move (to a new grid position) is valid (i.e., within the grid boundaries).
- 🔸 `playGame()`: The main game loop that controls the interaction between the player and the computer, processing moves and updating the game state.
- 🔸 `computerMove()`: Makes the computer move randomly on the grid.

4. Variables:

- 🔸 `grid[GRID_SIZE][GRID_SIZE]`: A 2D array of characters that represents the game grid (5x5). Each cell can contain items like treasures (`'T'`), traps (`'X'`), power-ups (`'+'`), or empty spaces (`'.'`).
- 🔸 `revealed[GRID_SIZE][GRID_SIZE]`: A 2D array of booleans that tracks whether each cell on the grid has been revealed to the player. Initially, all cells are marked as `false` (unrevealed).
- 🔸 `treasuresFound`: Tracks the number of treasures the player has found.
- 🔸 `powerUps`: Tracks the number of power-ups the player has collected.
- 🔸 `health`: Tracks the player's remaining health.
- 🔸 `moves`: Tracks the remaining number of moves the player can make.

  - 🔸 `playerX, playerY`: The player's current position on the grid (row and column).
  - 🔸 `computerX, computerY`: The computer's current position on the grid (row and column).
  - 🔸 `move`: A variable used to store the player's input for the next move (W/A/S/D).

5. Control Flow Keywords:

- 🔸 `if`: A conditional statement that checks if a certain condition is true and executes a block of code accordingly. Example:

  ```
  if (move == 'W' || move == 'w') newX--;
  ```

  This checks if the player entered a 'W' or 'w' to move up on the grid, and if so, decreases the `newX` value (moves the player up).

- **else if**: Used to specify additional conditions to check if the initial `if` condition is false. Example:

```
else if (move == 'A' || move == 'a') newY--;
```

This checks if the player entered an 'A' or 'a' to move left on the grid.

- **else**: Executes a block of code if none of the previous `if` or `else if` conditions were true. Example:

```
else {
    printf("Invalid move!\n");
    continue;
}
```

This block is executed when the player enters an invalid move.

- **while**: A loop that continues executing as long as a specified condition is true. Example:

```
while (count > 0) {
    int x = rand() % GRID_SIZE;
    int y = rand() % GRID_SIZE;
    if (grid[x][y] == '.') {
        grid[x][y] = item;
        count--;
    }
}
```

This loop places the specified number of items (`item`) on the grid until the count reaches zero.

- **continue**: Skips the remaining code in the current iteration of a loop and proceeds to the next iteration. Example:

```
continue;
```
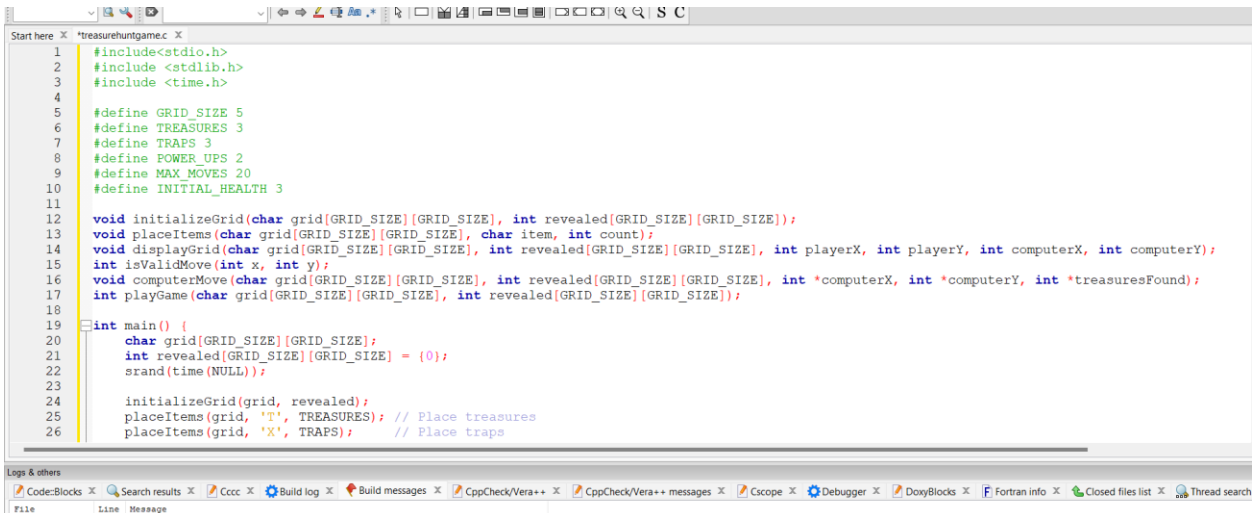
This is used to skip the rest of the code in a loop when the move is invalid or out of bounds.

- **return**: Exits a function and optionally returns a value. Example:
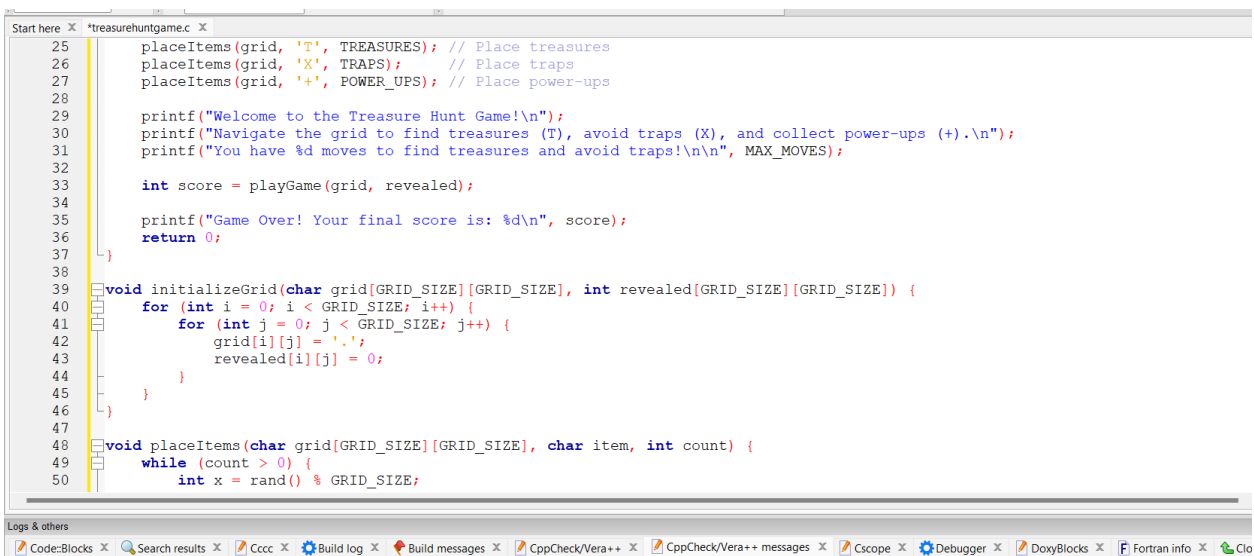
```
return treasuresFound * 10 + powerUps * 5 + health * 10;
```

This returns the calculated score when the game ends.

Screecnhots:-


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

#define GRID_SIZE 5
#define TREASURES 3
#define TRAPS 3
#define POWER_UPS 2
#define MAX_MOVES 20
#define INITIAL_HEALTH 3

void initializeGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]);
void placeItems(char grid[GRID_SIZE][GRID_SIZE], char item, int count);
void displayGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int playerX, int playerY, int computerX, int computerY);
int isValidMove(int x, int y);
void computerMove(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int *computerX, int *computerY, int *treasuresFound);
int playGame(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]);

int main() {
    char grid[GRID_SIZE][GRID_SIZE];
    int revealed[GRID_SIZE][GRID_SIZE] = {0};
    srand(time(NULL));

    initializeGrid(grid, revealed);
    placeItems(grid, 'T', TREASURES); // Place treasures
    placeItems(grid, 'X', TRAPS);     // Place traps
```



```c
        placeItems(grid, 'T', TREASURES); // Place treasures
        placeItems(grid, 'X', TRAPS);     // Place traps
        placeItems(grid, '+', POWER_UPS); // Place power-ups

        printf("Welcome to the Treasure Hunt Game!\n");
        printf("Navigate the grid to find treasures (T), avoid traps (X), and collect power-ups (+).\n");
        printf("You have %d moves to find treasures and avoid traps!\n\n", MAX_MOVES);

        int score = playGame(grid, revealed);

        printf("Game Over! Your final score is: %d\n", score);
        return 0;
}

void initializeGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]) {
    for (int i = 0; i < GRID_SIZE; i++) {
        for (int j = 0; j < GRID_SIZE; j++) {
            grid[i][j] = '.';
            revealed[i][j] = 0;
        }
    }
}

void placeItems(char grid[GRID_SIZE][GRID_SIZE], char item, int count) {
    while (count > 0) {
        int x = rand() % GRID_SIZE;
```

```c
49         while (count > 0) {
50             int x = rand() % GRID_SIZE;
51             int y = rand() % GRID_SIZE;
52             if (grid[x][y] == '.') {
53                 grid[x][y] = item;
54                 count--;
55             }
56         }
57     }
58
59 void displayGrid(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int playerX, int playerY, int computerX, int computerY) {
60     for (int i = 0; i < GRID_SIZE; i++) {
61         for (int j = 0; j < GRID_SIZE; j++) {
62             if (i == playerX && j == playerY && i == computerX && j == computerY) {
63                 printf("PC "); // Both player and computer at the same position
64             } else if (i == playerX && j == playerY) {
65                 printf("P "); // Player's position
66             } else if (i == computerX && j == computerY) {
67                 printf("C "); // Computer's position
68             } else if (revealed[i][j]) {
69                 printf("%c ", grid[i][j]);
70             } else {
71                 printf("? ");
72             }
73         }
74         printf("\n");
```

```c
73         }
74         printf("\n");
75     }
76 }
77
78 int isValidMove(int x, int y) {
79     return x >= 0 && x < GRID_SIZE && y >= 0 && y < GRID_SIZE;
80 }
81
82 void computerMove(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE], int *computerX, int *computerY, int *treasuresFound) {
83     int newX, newY;
84     do {
85         newX = *computerX + (rand() % 3 - 1); // Random move: -1, 0, or +1
86         newY = *computerY + (rand() % 3 - 1);
87     } while (!isValidMove(newX, newY));
88
89     revealed[newX][newY] = 1; // Mark the new position as revealed
90
91     char cell = grid[newX][newY];
92     printf("Computer moved to (%d, %d): ", newX, newY);
93     if (cell == 'T') {
94         printf("found a treasure!\n");
95         (*treasuresFound)++;
96         initializeGrid(grid, revealed); // Reset the grid visibility
97     } else if (cell == 'X') {
98         printf("hit a trap!\n");
```

```c
97     } else if (cell == 'X') {
98         printf("hit a trap!\n");
99     } else if (cell == '+') {
100        printf("found a power-up!\n");
101    } else {
102        printf("nothing here.\n");
103    }
104
105    *computerX = newX;
106    *computerY = newY;
107 }
108
109 int playGame(char grid[GRID_SIZE][GRID_SIZE], int revealed[GRID_SIZE][GRID_SIZE]) {
110    int treasuresFound = 0;
111    int powerUps = 0;
112    int health = INITIAL_HEALTH;
113    int moves = MAX_MOVES;
114    int playerX = 0, playerY = 0;
115    int computerX = GRID_SIZE - 1, computerY = GRID_SIZE - 1;
116    char move;
117
118    revealed[playerX][playerY] = 1; // Player's starting position is revealed
119    revealed[computerX][computerY] = 1; // Computer's starting position is revealed
120
121    while (moves > 0 && health > 0) {
122        printf("\nCurrent Grid:\n");
```

```c
        while (moves > 0 && health > 0) {
            printf("\nCurrent Grid:\n");
            displayGrid(grid, revealed, playerX, playerY, computerX, computerY);
            printf("\nTreasures found: %d, Power-ups: %d, Health: %d, Remaining moves: %d\n", treasuresFound, powerUps, health, moves);
            printf("Enter move (W/A/S/D): ");
            scanf(" %c", &move);

            // Update player's position
            int newX = playerX;
            int newY = playerY;
            if (move == 'W' || move == 'w') newX--;
            else if (move == 'A' || move == 'a') newY--;
            else if (move == 'S' || move == 's') newX++;
            else if (move == 'D' || move == 'd') newY++;
            else {
                printf("Invalid move!\n");
                continue;
            }

            if (!isValidMove(newX, newY)) {
                printf("Out of bounds!\n");
                continue;
            }

            revealed[newX][newY] = 1; // Mark the new position as revealed
```

```c
            char cell = grid[newX][newY];
            if (cell == 'T') {
                printf("You found a treasure!\n");
                treasuresFound++;
                initializeGrid(grid, revealed); // Reset the grid visibility
            } else if (cell == 'X') {
                if (powerUps > 0) {
                    printf("You hit a trap but used a power-up to survive!\n");
                    powerUps--;
                } else {
                    printf("You hit a trap! Health reduced by 1.\n");
                    health--;
                }
            } else if (cell == '+') {
                printf("You found a power-up!\n");
                powerUps++;
            } else {
                printf("Nothing here.\n");
            }

            playerX = newX;
            playerY = newY;

            // Computer's turn
            computerMove(grid, revealed, &computerX, &computerY, &treasuresFound);
```

```c
                printf("Nothing here.\n");
            }

            playerX = newX;
            playerY = newY;

            // Computer's turn
            computerMove(grid, revealed, &computerX, &computerY, &treasuresFound);

            moves--;

            if (treasuresFound == TREASURES) {
                printf("Congratulations! You found all the treasures!\n");
                return treasuresFound * 10 + powerUps * 5 + health * 10;
            }
        }

    if (health <= 0) {
        printf("You ran out of health!\n");
    } else {
        printf("You ran out of moves!\n");
    }

    return treasuresFound * 10 + powerUps * 5 + health * 10;
}
```

Output:


```
Welcome to the Treasure Hunt Game!
Navigate the grid to find treasures (T), avoid traps (X), and collect power-ups (+).
You have 20 moves to find treasures and avoid traps!

Current Grid:
P ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? C

Treasures found: 0, Power-ups: 0, Health: 3, Remaining moves: 20
Enter move (W/A/S/D):
```



```
Welcome to the Treasure Hunt Game!
Navigate the grid to find treasures (T), avoid traps (X), and collect power-ups (+).
You have 20 moves to find treasures and avoid traps!

Current Grid:
P ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? C

Treasures found: 0, Power-ups: 0, Health: 3, Remaining moves: 20
Enter move (W/A/S/D): S
Nothing here.
Computer moved to (3, 3): found a power-up!

Current Grid:
. ? ? ? ?
P ? ? ? ?
? ? ? ? ?
? ? ? C ?
? ? ? ? .

Treasures found: 0, Power-ups: 0, Health: 3, Remaining moves: 19
Enter move (W/A/S/D): |
```

```
? ? ? ? ?
? ? ? C ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 3, Remaining moves: 19
Enter move (W/A/S/D): D
Nothing here.
Computer moved to (2, 2): found a power-up!

Current Grid:
. ? ? ? ?
. P ? ? ?
? ? C ? ?
? ? ? . ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 3, Remaining moves: 18
Enter move (W/A/S/D): W
You hit a trap! Health reduced by 1.
Computer moved to (1, 2): nothing here.

Current Grid:
. P ? ? ?
. . C ? ?
? ? + ? ?
? ? ? . ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 2, Remaining moves: 17
Enter move (W/A/S/D):
```

```
? ? + ? ?
? ? ? . ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 2, Remaining moves: 17
Enter move (W/A/S/D): A
Nothing here.
Computer moved to (0, 1): hit a trap!

Current Grid:
P C ? ? ?
. . . ? ?
? ? + ? ?
? ? ? . ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 2, Remaining moves: 16
Enter move (W/A/S/D): D
You hit a trap! Health reduced by 1.
Computer moved to (1, 2): nothing here.

Current Grid:
. P ? ? ?
. . C ? ?
? ? + ? ?
? ? ? . ?
? ? ? ? +

Treasures found: 0, Power-ups: 0, Health: 1, Remaining moves: 15
Enter move (W/A/S/D):
```

Conclusion:

This C program simulates a turn-based "Treasure Hunt Game" where the player navigates a grid, searching for treasures while avoiding traps and collecting power-ups. The game is designed to provide an engaging

challenge where the player must manage moves, health, and power-ups while also competing against a computer-controlled character.