T.C.

YALOVA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ALGORITHMS AND PROGRAMMING II ASSIGNMENT 1

TREASURE HUNT GAME IN C

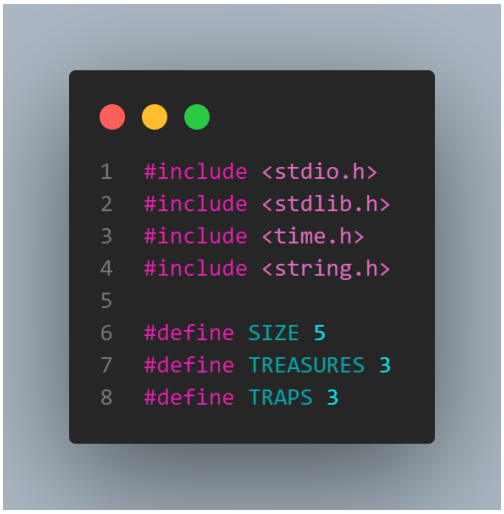
PREPARED BY:

240101107 FİRDEVS SATİCİ

ADVISOR: MURAT OKKALIOĞLU

YALOVA-2025

1- CODE



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

#define SIZE 5
#define TREASURES 3
#define TRAPS 3
```

The codes includes four standard libraries. For example:

- **<time.h>:** Used for random number generator
- **<stdlib.h>**: Provides functions like rand(), srand(), and exit().

Then, three macros are defined:

- **SIZE 5**: Defines the grid size as a 5x5 matrix.
- **TREASURES 3:** Specifies that three treasures will be placed in the grid.
- **TRAPS 3**: Specifies that three traps will also be placed in the grid.

This part of the code defines global variables and function types for the game:

```c
char grid[SIZE][SIZE];
char visibleGrid[SIZE][SIZE];
int playerX = 0, playerY = 0;
int treasuresCollected = 0;
int moveCount = 0;

void initializeGrid();
void displayGrid();
void placeTreasuresAndTraps();
void movePlayer(char direction);
void saveGame(const char *filename);
void loadGame(const char *filename);
void updateLeaderboard(const char *playerName);
void displayLeaderboard();
```

• **Grids:**

- grid[SIZE][SIZE]: Stores the actual game board with treasures and traps.
- visibleGrid[SIZE][SIZE]: Represents what the player can see (initially hidden).

• **Player Info:**

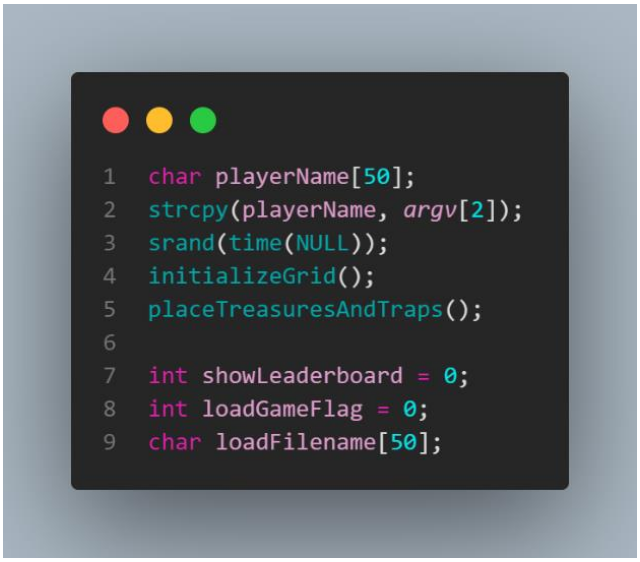
- playerX, playerY: Track the player's position (starts at (0,0)).
- treasuresCollected: Keeps count of collected treasures.
- moveCount: Tracks the number of moves taken.

• **Function Prototypes:** Define the main game functionalities, including initialization, display, movement, saving/loading, and leaderboard updates.

```c
int main(int argc, char *argv[]) {
    if (argc < 3 || strcmp(argv[1], "p") != 0) {
        printf("Usage: ./treasureHunt.exe p <player_name> [load <filename>] [leaders]\n");
        return 1;
    }
```

This part defines the main function, handling command-line arguments:

- argc and argv[]: Used for reading input arguments when running the program.
- **Argument Check:**
  - The program requires at least three arguments:
    - "p": Indicates the game is being played.
    - <player_name>: Stores the player's name.
    - (Optional) "load <filename>": Loads a saved game.
    - (Optional) "leaders": Displays the leaderboard.
  - If the conditions aren't met, it prints the correct usage format and exits.



```c
char playerName[50];
strcpy(playerName, argv[2]);
srand(time(NULL));
initializeGrid();
placeTreasuresAndTraps();

int showLeaderboard = 0;
int loadGameFlag = 0;
char loadFilename[50];
```

This section initializes key variables and settings for the game:

- char playerName[50]; → Stores the player's name from command-line input.
- strcpy(playerName, argv[2]); → Copies the player's name from argv[2].
- srand(time(NULL)); → Seeds the random number generator for random treasure/trap placement.
- initializeGrid(); → Prepares the game grid.
- placeTreasuresAndTraps(); → Randomly places treasures and traps on the grid.

- **Optional Features:**
  - showLeaderboard: If set to 1, the leaderboard will be displayed.
  - loadGameFlag: If set to 1, a saved game will be loaded.
  - char loadFilename[50]; → Stores the name of the file to load if needed.

```c
for (int i = 3; i < argc; i++) {
    if (strcmp(argv[i], "leaders") == 0) {
        showLeaderboard = 1;
    }
    if (strcmp(argv[i], "load") == 0 && i + 1 < argc) {
        loadGameFlag = 1;
        strcpy(loadFilename, argv[i + 1]);
    }
}
if (showLeaderboard) {
    displayLeaderboard();
}
if (loadGameFlag) {
    loadGame(loadFilename);
}
```

This section processes additional command-line arguments:

- **Loop through arguments (for loop):**
    - Starts from argv[3] because argv[1] and argv[2] are already used for "p" and <player_name>.
    - If "leaders" is found, showLeaderboard is set to 1.
    - If "load" is found and followed by a filename, loadGameFlag is set to 1, and the filename is copied into loadFilename.

- **Handling leaderboard and game loading:**
    - If showLeaderboard == 1, it calls displayLeaderboard(); to show scores.
    - If loadGameFlag == 1, it calls loadGame(loadFilename); to restore a saved game.

```c
char command;
while (treasuresCollected < TREASURES) {
    displayGrid();
    printf("Enter move (u/d/l/r) or 's' to save: ");
    scanf(" %c", &command);
    if (command == 's') {
        char saveFile[50];
        printf("Enter save file name: ");
        scanf("%s", saveFile);
        saveGame(saveFile);
        continue;
    }
    movePlayer(command);
    moveCount++;
}

printf("Congratulations, %s! You collected all treasures in %d moves!\n", playerName, moveCount);
updateLeaderboard(playerName);
return 0;
}
```

This is the main game loop, where the player moves until all treasures are collected:

- char command; → Stores the player's move input.
- **Loop (while):**
  - Runs **until** treasuresCollected == TREASURES.
  - Calls displayGrid(); to show the current state of the game.
  - Asks the player to enter a move:
    - 'u' (up), 'd' (down), 'l' (left), 'r' (right) → Moves the player.
    - 's' → Saves the game:
      - Asks for a filename and calls saveGame(saveFile);.
      - continue; prevents increasing moveCount when saving.
  - Calls movePlayer(command); to process movement.
  - Increases moveCount++;.
- **End of Game:**
  - When all treasures are collected, prints a congratulatory message with the player's move count.
  - Calls updateLeaderboard(playerName); to record the player's score.
  - return 0; → Ends the program.

```c
void initializeGrid() {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            grid[i][j] = ' ';
            visibleGrid[i][j] = '?';
        }
    }
    visibleGrid[0][0] = 'P';
}
```

The initializeGrid() function sets up the game board:

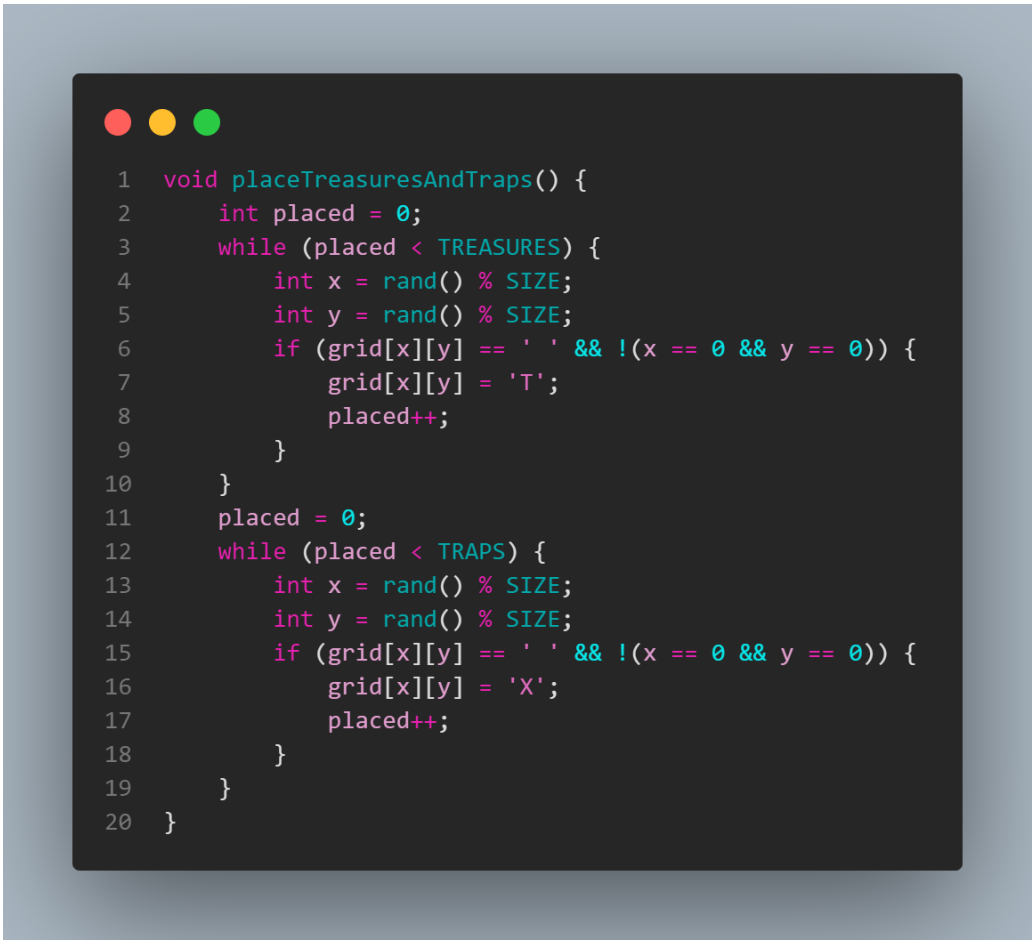
- **Double for loop:**
  - Iterates over all positions in the SIZE × SIZE grid.
  - Initializes grid[i][j] with ' ' (empty space) → This is the hidden game grid.
  - Initializes visibleGrid[i][j] with '?' → This represents the fog of war, meaning unexplored areas.

- **Player's starting position (visibleGrid[0][0] = 'P';)**
  - The player always starts at the top-left corner (0,0).
  - The P character is visible to the player, indicating their position

```c
void displayGrid() {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%c ", visibleGrid[i][j]);
        }
        printf("\n");
    }
    printf("Treasures collected: %d\n", treasuresCollected);
}
```

The displayGrid() function prints the current state of the game board:

- **Double for loop:**
  - Iterates over the SIZE × SIZE visible grid.
  - Uses printf("%c ", visibleGrid[i][j]); to print each cell with a space between them.
  - After printing a row, printf("\n"); moves to the next line.
- **Displays the treasure count:**
  - printf("Treasures collected: %d\n", treasuresCollected); → Shows how many treasures the player has found so far.



```
void placeTreasuresAndTraps() {
    int placed = 0;
    while (placed < TREASURES) {
        int x = rand() % SIZE;
        int y = rand() % SIZE;
        if (grid[x][y] == ' ' && !(x == 0 && y == 0)) {
            grid[x][y] = 'T';
            placed++;
        }
    }
    placed = 0;
    while (placed < TRAPS) {
        int x = rand() % SIZE;
        int y = rand() % SIZE;
        if (grid[x][y] == ' ' && !(x == 0 && y == 0)) {
            grid[x][y] = 'X';
            placed++;
        }
    }
}
```

The placeTreasuresAndTraps() function places treasures and traps randomly on the grid:

- **Treasure Placement:**
  - while (placed < TREASURES) loop runs until all treasures are placed.
  - Random coordinates (x and y) are generated using rand() % SIZE; for both rows and columns.
  - The condition if (grid[x][y] == ' ' && !(x == 0 && y == 0)) ensures:
    - The selected cell is empty (' ').
    - The starting position (0,0) is never selected for treasures (to prevent overwriting the player's starting point).
  - grid[x][y] = 'T'; places a treasure at the random location.
  - placed++ increases the count of placed treasures.

- **Trap Placement:**
  - A similar process occurs for traps, but with the condition grid[x][y] = 'X'; to place traps instead of treasures.
  - The traps are also randomly placed but follow the same rules (no overwrite of starting position).

```c
void movePlayer(char direction) {
    int newX = playerX, newY = playerY;
    switch (direction) {
        case 'u': newX--; break;
        case 'd': newX++; break;
        case 'l': newY--; break;
        case 'r': newY++; break;
        default: printf("Invalid move!\n"); return;
    }

    if (newX < 0 || newX >= SIZE || newY < 0 || newY >= SIZE) {
        printf("Move out of bounds!\n");
        return;
    }
```

• **Code Purpose:** Moves the player based on the direction input ('u', 'd', 'l', 'r' for up, down, left, right).

• **newX, newY Variables:** Initially set to the player's current position. These variables will update according to the player's chosen move.

• **Switch Statement:**

- Each case checks for one direction and updates newX or newY accordingly.
- If the input doesn't match any valid direction, printf("Invalid move!\n"); is called.

• **Bounds Check:**

- The if statement checks if the new position (newX, newY) is outside the grid's boundaries. If it's out of bounds, the function prints "Move out of bounds!" and exits without making any move.

```
if (visibleGrid[playerX][playerY] != 'T') {
        visibleGrid[playerX][playerY] = '=';
    }

    playerX = newX;
    playerY = newY;

    if (grid[playerX][playerY] == 'T') {
        treasuresCollected++;
        grid[playerX][playerY] = ' ';
        visibleGrid[playerX][playerY] = 'T';
    } else if (grid[playerX][playerY] == 'X') {
        printf("Game Over! You hit a trap.\n");
        exit(0);
    } else {
        visibleGrid[playerX][playerY] = 'P';
    }
}
```

- **Visible Grid Update:**
  - If statement checks if the player is on a treasure ('T'). If not, it marks the previous position with '=' (visited but not a treasure).
- **Position Update:**
  - Player's position is updated to the new coordinates (newX, newY).
- **Treasure Check:**
  - If player lands on a treasure (grid[playerX][playerY] == 'T'), the treasure count increases (treasuresCollected++).
  - The visible grid is updated to show the treasure (visibleGrid[playerX][playerY] = 'T').
- **Trap Check:**
  - If player lands on a trap (grid[playerX][playerY] == 'X'), the game ends with the message "Game Over! You hit a trap.", and the program exits.

    And if player is on an empty space, the visible grid is updated to show the player's new position (visibleGrid[playerX][playerY] = 'P').

```c
void saveGame(const char *filename) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        printf("Error saving game!\n");
        return;
    }
    fprintf(file, "%d %d %d %d\n", playerX, playerY, treasuresCollected, moveCount);
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            fprintf(file, "%c", grid[i][j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("Game saved!\n");
}
```

- **Opening the File:**
  - fopen(filename, "w") opens the file in write mode ("w"), meaning it will overwrite any existing content.
  - If the file cannot be opened (file == NULL), an error message is displayed, and the function returns without saving.
- **Saving Game State:**
  - fprintf() writes the player's X and Y position, the number of treasures collected, and the move count into the file as space-separated values.
- **Saving the Grid:**
  - for loop goes through each cell in the grid array.
  - fprintf(file, "%c", grid[i][j]); writes the character (' ', 'T', or 'X') to the file.
  - After writing an entire row, fprintf(file, "\n"); moves to the next line.
- **Closing the File:**
  - fclose(file); ensures that the file is properly saved and closed.
- **Confirmation Message:**
  - printf("Game saved!\n"); lets the player know that the game was successfully saved.

```c
void loadGame(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        printf("Error loading game!\n");
        return;
    }
    fscanf(file, "%d %d %d %d", &playerX, &playerY, &treasuresCollected, &moveCount);
    for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        fscanf(file, " %c", &grid[i][j]);
    }
}
    fclose(file);
    printf("Game loaded!\n");
}
```

1. **Opening the File:**
   - It attempts to open the specified file (filename) in read mode ("r").
   - If the file cannot be opened (e.g., it does not exist), an error message is printed, and the function returns without making any changes.

2. **Reading Game Data:**
   - It reads four integer values from the file using fscanf, which correspond to:
     - playerX and playerY: The player's current position.
     - treasuresCollected: The number of treasures the player has found.
     - moveCount: The number of moves made so far.

3. **Reading the Game Grid:**
   - A nested loop iterates over the grid (SIZE × SIZE) and reads each character from the file, storing it in the grid array.

4. **Closing the File**

5. **Displaying a Success Message:**
   - Finally, a message "Game loaded!" is printed to indicate that the game state was successfully restored.

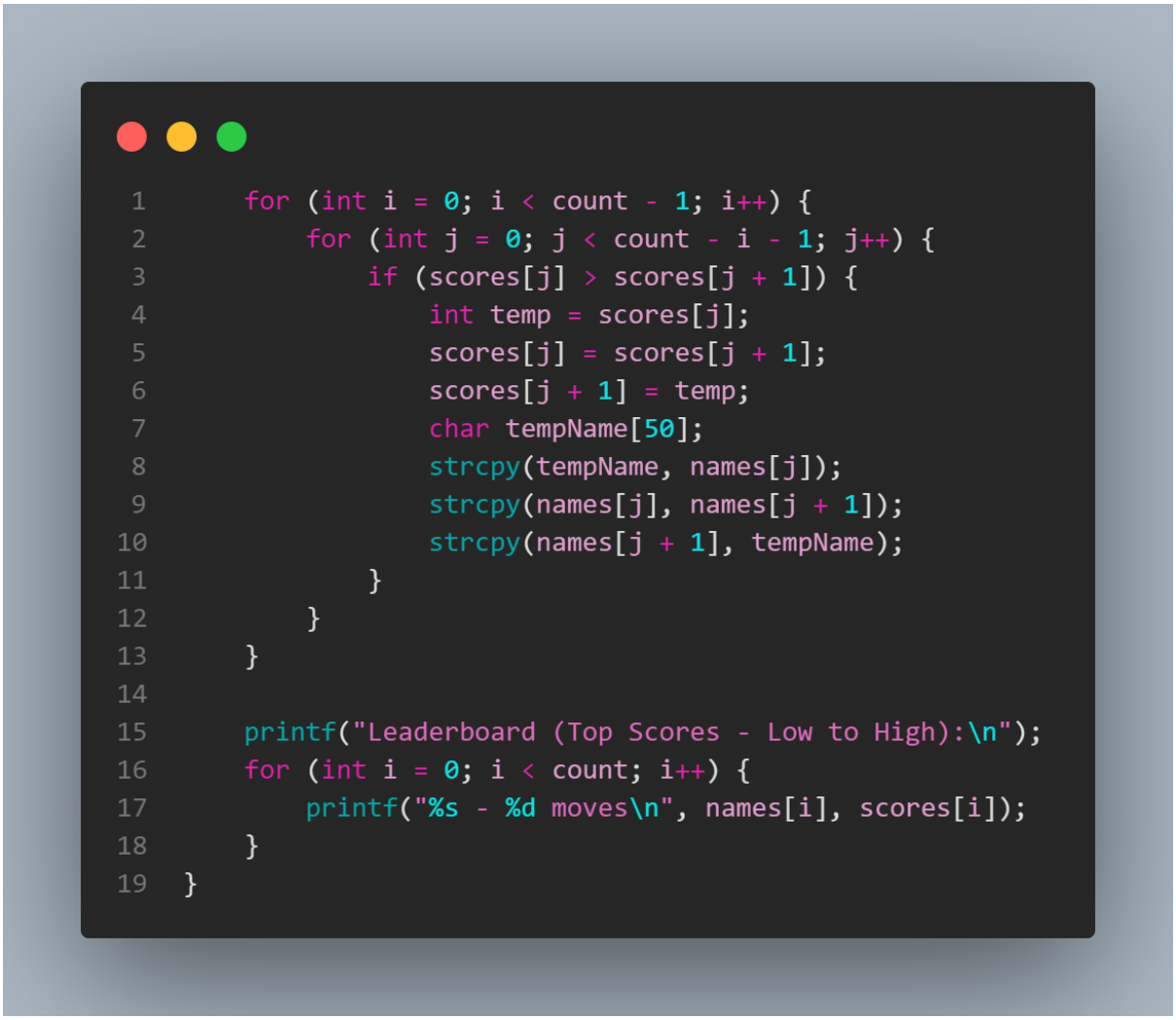
```c
void updateLeaderboard(const char *playerName) {
    FILE *file = fopen("leaderboard.txt", "a");
    if (!file) {
        printf("Error updating leaderboard!\n");
        return;
    }
    fprintf(file, "%s %d\n", playerName, moveCount);
    fclose(file);
}
```

- **Opening the File in Append Mode:**
  - fopen("leaderboard.txt", "a") opens (or creates if it doesn't exist) the leaderboard.txt file in append mode ("a").
  - Append mode ensures that new data is added at the end of the file without overwriting previous records.
- **Checking for File Opening Errors:**
  - If the file cannot be opened, an error message "Error updating leaderboard!" is printed, and the function returns without making changes.
- **Writing to the File:**
  - The function writes the player's name (playerName) and the number of moves (moveCount) to the file in the format:

    PlayerName MoveCount
- **Closing the File.**

```c
void displayLeaderboard() {
    FILE *file = fopen("leaderboard.txt", "r");
    if (!file) {
        printf("No leaderboard available.\n");
        return;
    }

    char names[100][50];
    int scores[100];
    int count = 0;

    while (fscanf(file, "%s %d", names[count], &scores[count]) != EOF) {
        count++;
    }
    fclose(file);
```

- **Opening the File in Read Mode:**
  - fopen("leaderboard.txt", "r") opens the file in read mode ("r").
  - If the file does not exist, it prints "No leaderboard available." and exits the function.
- **Reading Player Data:**
  - char names[100][50]; → Stores up to 100 player names, each with a maximum length of 50 characters.
  - int scores[100]; → Stores the corresponding move counts for each player.
  - int count = 0; → Keeps track of how many records are read.
- **Reading Entries from the File:**
  - fscanf(file, "%s %d", names[count], &scores[count]) reads player names and scores.
  - The while loop continues until the end of the file (EOF).
  - After reading a record, count is incremented to store the next entry.
- **Closing the File.**



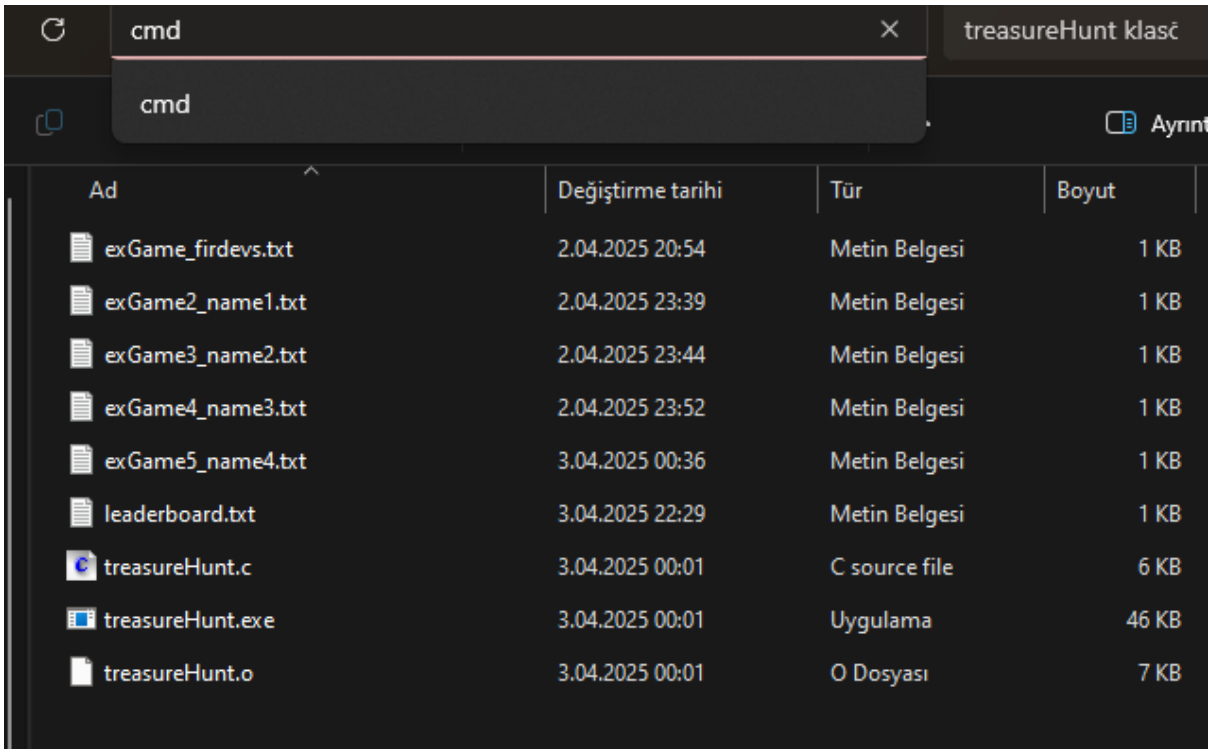
```
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (scores[j] > scores[j + 1]) {
                int temp = scores[j];
                scores[j] = scores[j + 1];
                scores[j + 1] = temp;
                char tempName[50];
                strcpy(tempName, names[j]);
                strcpy(names[j], names[j + 1]);
                strcpy(names[j + 1], tempName);
            }
        }
    }

    printf("Leaderboard (Top Scores - Low to High):\n");
    for (int i = 0; i < count; i++) {
        printf("%s - %d moves\n", names[i], scores[i]);
    }
}
```

- **Sorting the Scores:**

  - A nested loop is used to sort players based on their move count in ascending order (lowest to highest).
  - If scores[j]>scores [j + 1], the values are swapped:
    - Move count (scores[j] and scores[j+1])
    - Player names (names[j] and names[j+1])

2-CMD

First we will open cmd in our folder treasureHunt.



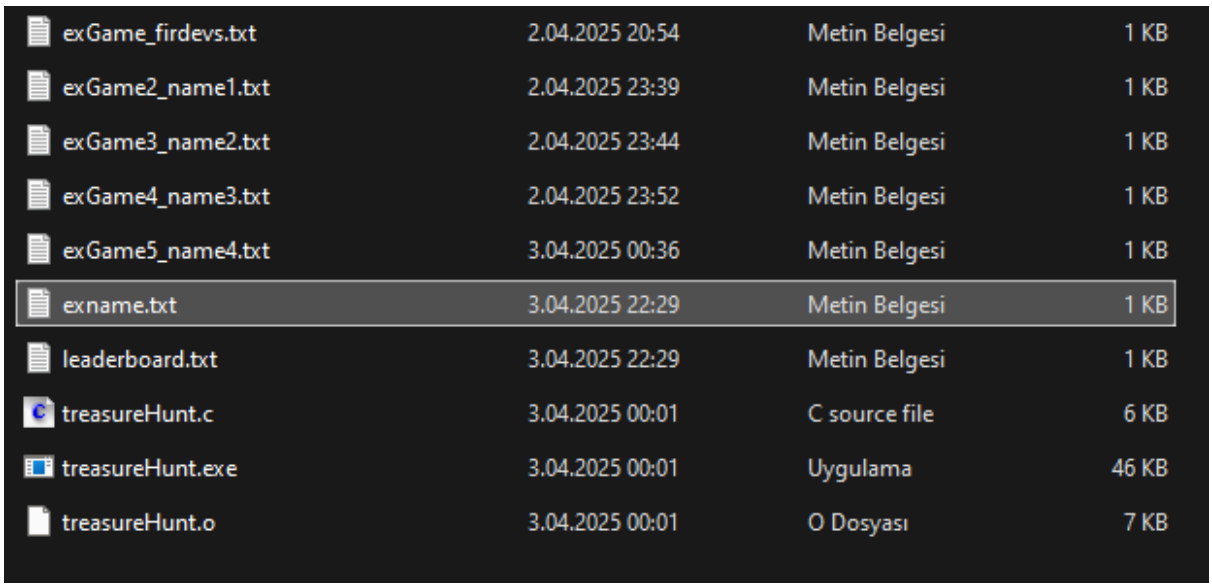
Before we start the game, we should supply a player's name via the command line argument, p. Our example name be exname.

```
C:\Users\User\Desktop\treasureHunt>treasureHunt.exe p exname
```

```
P ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
Treasures collected: 0
Enter move (u/d/l/r) or 's' to save: d
= ? ? ? ?
P ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
Treasures collected: 0
Enter move (u/d/l/r) or 's' to save: d
= ? ? ? ?
= ? ? ? ?
T ? ? ? ?
? ? ? ? ?
? ? ? ? ?
Treasures collected: 1
Enter move (u/d/l/r) or 's' to save: d
= ? ? ? ?
= ? ? ? ?
T ? ? ? ?
P ? ? ? ?
? ? ? ? ?
Treasures collected: 1
Enter move (u/d/l/r) or 's' to save: d
```

After a few moves i will save the game. In the treasureHunt folder our saved text file exname.txt will be appear.

```
Enter move (u/d/l/r) or 's' to save: s
Enter save file name: exname.txt
Game saved!
```



```
= T = ? ?
= = = ? ?
T = P ? ?
= = ? ? ?
= = ? ? ?
Treasures collected: 2
Enter move (u/d/l/r) or 's' to save: d
Congratulations, exname! You collected all treasures in 13 moves!
```

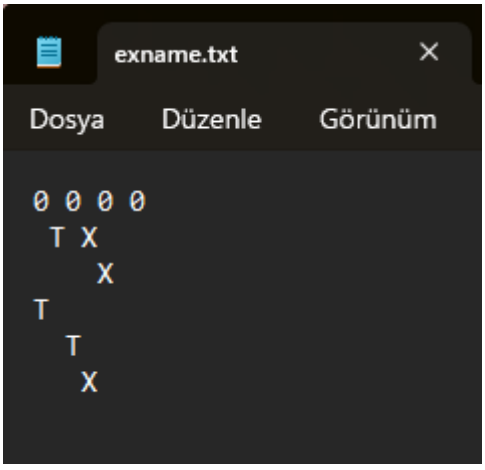
Then i finished the game after 13 moves. Now we can check leaderboard, and our name -exname- will be there.

```
C:\Users\User\Desktop\treasureHunt>treasureHunt.exe p exname leaders
Leaderboard (Top Scores - Low to High):
exname - 13 moves
name2 - 15 moves
name3 - 15 moves
firdevs - 18 moves
name1 - 21 moves
```

C:\Users\User\Desktop\treasureHunt>treasureHunt.exe p exname load exname.txt leaders

```
C:\Users\User\Desktop\treasureHunt>treasureHunt.exe p exname load exname.txt leaders
Leaderboard (Top Scores - Low to High):
exname - 13 moves
name2 - 15 moves
name3 - 15 moves
firdevs - 18 moves
name1 - 21 moves
Game loaded!
```

And this is the exname.txt. We can see all treasures and trapes in exname.txt file in trasureHunt folder.



Thanks for your attention… 😊