Ceng 209(System Programming) Assignment

Emirhan ASLANKARAYİĞİT
20050111059

Here are some screenshots and explanations that belong to my assignment:

```
struct Room {
    char* information;
    char* gun;
    char* score;
    struct Room* left;
    struct Room* right;
    struct Room* up;
    struct Room* down;
};
```

-Code 01-

In this code, I have designed the struct **Room** to traverse my *graph data structure* path. To pass through this path that we create on the main function, I used graph data structure to go on the path more effectively, instead of a singly linked list. (one directional linked list)

```
void connectRooms(struct Room* room1, struct Room* room2, char
direction) {
    switch (direction) {
        case 'U':
            room1->up = room2;
            room2->down = room1;
            break;
        case 'D':
            room1->down = room2;
            room2->up = room1;
            break;
        case 'L':
            room1->left = room2;
            room2->right = room1;
            break;
        case 'R':
            room1->right = room2;
            room2->left = room1;
            break;
        default:
            printf("Invalid direction!\n");
    }
}
```

-Code02-

In this code block, I used switch case depending on the direction case accepted as a function argument. Two rooms can be connected with direction by this function.

```
connectRooms(startingRoom, room1, 'R');
connectRooms(startingRoom, room2, 'D');
connectRooms(room1, room3, 'D');
```

-Code 03-

On the main part, the map will be created automatically according to this function.

```
struct Gamer {
    struct Room* currentPosition;
    int totalScores;
    char* gunList;
};
```

-Code04-

Gamer struct provides us to be agents on game paths. **Player** that creates from *Gamer,* has this attributes:

- *currentPosition*
- *totalScores*
- *gunList*

currentPosition is used to store the current address / location of an agent that plays the game.
totalScores is used to store the scores collected by the agent for each room visit.
gunList is used to store all the guns that the agent collects.

```
void navigateRooms(struct Gamer* gamer) {
    if (!gamer || !gamer->currentPosition) {
        printf("Gamer or current position is not initialized!\n");
        return;
    }
    char command;
    while (true) {
                    printf("\nYou are in the room: %s\n",
gamer->currentPosition->information);
                    printf("Your Current Total Score: %d\n",
gamer->totalScores);
                printf("Navigate: (U)p, (D)own, (L)eft, (R)ight,
(E)xit\n");
        scanf(" %c", &command);

        struct Room* nextRoom = NULL;
        switch (command) {
            case 'U': nextRoom = gamer->currentPosition->up; break;
                case 'D': nextRoom = gamer->currentPosition->down;
break;
                case 'L': nextRoom = gamer->currentPosition->left;
break;
               case 'R': nextRoom = gamer->currentPosition->right;
break;
            case 'E':
                printf("Exiting navigation.\n");
                return;
            default:
                printf("Invalid command!\n");
                continue;
        }

        if (nextRoom) {
            appendToGamerList(gamer, nextRoom);
            gamer->currentPosition = nextRoom;
                      printf("Moved to the next room: %s\n",
nextRoom->information);
        } else {
            printf("No room in that direction!\n");
        }
    }
}
```

-Code 05-

This code block tells us that The **navigateRooms** function allows a **Gamer** to navigate through a grid-like structure of rooms. Here's how it works:

1. **Input Validation** says that:
    - The function first checks if the Gamer object or their currentPosition (the room they are currently in) is properly initialized.
    - If not, it prints an error message and exits the function.

2. **Navigation Loop**:
    - The function enters an infinite loop where the gamer can repeatedly choose navigation commands.
    - Inside the loop:
        - It displays the information about the current room (gamer->currentPosition->information) and the total score (gamer->totalScores).
        - It prompts the gamer to enter a navigation command:
          U (Up), **D** (Down), **L** (Left), **R** (Right), or **E** (Exit).
3. **Command Handling**:
    - Based on the user's input (command), the function determines the next room to move to by accessing one of the pointers (up, down, left, or right) of the current room.
    - If the command is **E**, the loop breaks, and the function ends, allowing the gamer to exit navigation.
4. **Room Transition**:
    - If there is a valid room in the chosen direction, the function:
        - Calls appendToGamerList (likely to record the gamer's visited rooms or scores).
        - Updates gamer->currentPosition to point to the next room.
        - Prints a message showing that the gamer successfully moved to the new room.
    - If the chosen direction leads to NULL (no room exists), the function prints a warning message: *"No room in that direction!"*
5. **Invalid Input Handling**:
    - If the gamer enters an unrecognized command, the function prints an error message and prompts for another input without exiting the loop.

```
int main() {
     struct Room* startingRoom = createRoom("This is the Starting
Room.", "Pistol", "10");
       struct Room* room1 = createRoom("This is the Room A.",
"Shotgun", "20");
       struct Room* room2 = createRoom("This is the Room B.",
"Keyblade", "15");
       struct Room* room3 = createRoom("This is the Room C.",
"Rifle", "50");

    connectRooms(startingRoom, room1, 'R');
    connectRooms(startingRoom, room2, 'D');
    connectRooms(room1, room3, 'D');

       struct Gamer* gamer = (struct Gamer*)malloc(sizeof(struct
Gamer));
    gamer->currentPosition = startingRoom;
    gamer->totalScores = 0;
    gamer->gunList = NULL;

    printf("Welcome to the Room Navigation System!\n");
    navigateRooms(gamer);

    free(gamer->gunList);
    free(gamer);

    return 0;
}
```

-Code 06-

The main function sets up the environment for the Room System and initializes the necessary structures for the program to function properly. Here's a step-by-step breakdown:

1. **Room Creation:**
   - Four rooms are created using the createRoom function:
     - startingRoom: The initial room where the gamer begins. It contains a "Pistol" and a score of 10.
     - room1: Room A, which has a "Shotgun" and a score of 20.
     - room2: Room B, which has a "Keyblade" and a score of 15.
     - room3: Room C, which has a "Rifle" and a score of 50.
2. **Connecting Rooms:**
   - Rooms are linked together using the connectRooms function to form a navigable grid:
     - startingRoom is connected to room1 on the right (R) and to room2 below (D).
     - room1 is connected to room3 below (D).
3. **Gamer Initialization:**
   - A Gamer object is dynamically allocated using malloc.
   - The gamer's starting position is set to the startingRoom.
   - The gamer's totalScores is initialized to 0, and their gunList (likely an inventory of collected weapons) is set to NULL.
4. **Starting the System:**
   - A welcome message is printed: *"Welcome to the Room Navigation System!"*
   - The *navigateRooms* function is called, passing the gamer as an argument, allowing them to start exploring the connected rooms.
5. **Memory Cleanup:(crucial part for memory allocation)**
   - After the gamer exits the navigation system:
     - The gamer->gunList is freed to release any dynamically allocated memory associated with the gun inventory.
     - The gamer object itself is freed to avoid memory leaks.