

Ankara Yıldırım Beyazıt University

Department of Computer Engineering

CENG 209 – System Programming  
Term Homework

Dungeon Game

**Student :** Özgür Atlı

**Student id:** 23050151029

**Instructor**: Çağdaş Evren Gerede

**Date:** 17/12/2024

**Table of Contents**

[1. Introduction 2](#_Toc120484213)

[2. Requirements 2](#_Toc120484214)

[3. Methodology 2](#_Toc120484217)

[4. Implementation 3](#_Toc120484222)

# Introduction

In this homework, we are required to create a simple, command-line-based dungeon exploration game. The game allows players to move through dungeon, collect items, attack creatures, save and upload progress. The goal is to use to programming concept for made basic interactive game.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure . Sample figure.

# Requirements

Functional requirements refer to what your system must do. The following list outlines the main functional requirements for the dungeon exploration game, detailing the interactions and tasks that the player should be able to perform during gameplay.

* Item collection: Player should get any item in the game for this we made an list.
* Player movement: The game allow the player to navigate through in the dungeon, move forward or move back.
* Combat system: Player can attack the creatures.
* Save-Load: Player should allow to take a save and load of this save.
* Exit: Player can quit on the game.

# Methodology

## Technologies and Tools Used

Game is programming in C language. In this code we used features of C like I/O, user interaction and structures.

## Data Structures and Algorithms

* **Structures**: Game has several structures for player, items, creatures and rooms.
* **Linked Lists**: For collect more than one item and move through the dungeon we use Lists.
* I **I/O Operations**: Room descriptions are read from the Room.txt file, and I/O operations are also used for saving and loading the game state to preserve progress.

## Logic of The Game

* **Combat with Creatures**: The player can fight creatures encountered in the rooms. Battles are carried out by both sides attacking in turn.
* **Items**: The player can collect items found in the rooms and add them to their inventory. Items can affect the character’s health or attack power.
* **Room Transition**: The player can move between rooms. Room transitions are managed through linked lists.
* **Save-Load Game**: The game state can be saved to a file, and the saved data can be loaded when the game is restarted.

# Implementation

**1-Structures**:  
typedef struct Items

{

char name[30];

char description[50];

int additionalHelat;

int additionalAttack;

struct Items\* next;

}Items;

typedef struct Creature{

char name[30];

int health;

int attack;

}Creature;

typedef struct Room {

int roomNumber;

Creature\* creature;

Items\* item;

struct Room\* nextRoom;

struct Room\* prevRoom;

}Room;

typedef struct Player {

char name[30];

int healt;

int attack;

Items\* item;

}Player;

* **Items:** Contains the properties of the items collected by the player.
* **Creature:** Contains the properties of the creatures encountered by the player.
* **Room:** Represents the rooms in the game world. Each room may contain a creature and an item.
* **Player:** Contains the information of the player's character.

**2-Funtions:**

**1-) attachRooms():**

**Description:** This function links a series of rooms in a doubly linked list. Each room points to the next room using nextRoom, and the next room points back to the current room using prevRoom. The rooms are arranged in sequence, forming a bidirectional chain of rooms.

void attachRooms(Room\* rooms[], int length) {

for (int i = 0; i < length-1; i++)

{

rooms[i]->nextRoom = rooms[i + 1];

rooms[i + 1]->prevRoom = rooms[i];

}

}

**2-) attachItems():**

**Description**: This function links the items in the room list. Each item in the list is connected to the next one.

void attachItems(Items\* items[], int length) {

for (int i = 0; i < length-1; i++)

{

items[i]->next = items[i + 1];

}

}

**3-) war():**

**Description:** This function initiates a battle between the player and a creature. The battle

proceeds with each side attacking in turns. The health of both the player and the creature is updated after each attack.

void war(Creature\* creature, Player\* player1) {

if (creature == NULL) {

printf("There is no monster.");

}

printf("You ara against %s\n", creature->name);

while (creature->health>0 && player1->healt>0) {

printf("You strike first... You gave the %d damega\n", player1->attack);

creature->health -= player1->attack;

if (creature->health <= 0) {

printf("You killed the monster.\n");

return;

}

printf("It attack you and you get %d damega\n", creature->attack);

player1->healt -= creature->attack;

if (player1->healt < 0) {

printf("You died\n");

exit(1);

}

}

}

**4-) pickUp():**

**Description:** This function allows the player to pick up an item from the room. If there is

an item in the room, it adds it to the player's inventory and removes it from the room.

void pickUp(Room\* room,Player\* p1) {

if (room->item == NULL) {

printf("There is no item.\n");

}

Items\* addItem = room->item;

room->item = NULL;

addItem->next = p1->item;

p1->item = addItem;

p1->attack += addItem->additionalAttack;

p1->healt += addItem->additionalHelat;

printf("You picked up %s\n", addItem->description);

}

**5-) lookUp():**

**Description:** This function read to Room.txt for room descriptions.

void lookUp(int i) {

FILE\* file = fopen("Room.txt", "r");

if (file == NULL) {

printf("There is nothing.\n");

return;

}

char line[100];

int lineNumber = 1;

while (fgets(line, sizeof(line), file) != NULL) {

if (lineNumber == i) {

printf("%s\n", line);

break;

}

lineNumber++;

}

fclose(file);

}

**6-) saveGame():**

**Description:** This function saves the current game state, including the player's health,

attack power. It also stores the current room the player is in.

void saveGame(Room\* room,Player\* p1) {

FILE\* file = fopen("saves.txt", "w");

if (file == NULL)

{

printf("Error!");

}

fprintf(file,"Charecter healt %d\n", p1->healt);

fprintf(file,"Charecter attack power %d\n", p1->attack);

fprintf(file,"Current room number: %d\n", room->roomNumber);

fclose(file);

printf("Save completed.");

}

**7-) loadGame():**

**Description:** This function loads a saved game. It retrieves the player's health, attack

power, and the room they were in from the saved file.

int loadGame(Room\* room, Player\* p1) {

FILE\* file = fopen("saves.txt", "r");

if (file == NULL)

{

printf("Error!");

}

fscanf\_s(file,"Charecter healt %d\n", &p1->healt);

fscanf\_s(file,"Charecter attack power %d\n", &p1->attack);

int a;

fscanf\_s(file, "Current room number: %d\n",&a);

fclose(file);

printf("Load completed.\n");

return a-1;

}

**8-) exitGame():**

**Description:** This function ends the game.  
 void exitGame() {

printf("Your game is closed");

exit(1);

}

**9-)main():**

**Description:** In this function firstly creats creatures and items after thad it attechs items.

Secondly creats a array for rooms and decleare that rooms.Thirdly creats array for room pointers than attechs rooms.Finnaly makes a while loop for game play.

int main() {

Creature goblin = { "goblin",50,5 };

Creature ork = { "ork",100,10 };

Creature golem = { "golem",100,20 };

Creature dragon = { "dragon",300,50 };

Items item1 = { "Sword","Old and rusty sword",0,25 };

Items item2 = { "Bread","Restore some healt",25,0 };

Items item3 = { "Book","Old book about elixir of life",0,0 };

Items item4 = { "Armor","Gives additional protection",50,0 };

Items item5 = { "Golden Sword","Gives additional Attack",0,100 };

Items item6 = { "Elixir","Gives immortality",0,0 };

Items item7 = { "Potion","Restor your full health",200,0 };

Items\* items[] = { &item1,&item2,&item3,&item4,&item5,&item6,&item7 };

attachItems(items, 7);

Room rooms[14];

for (int i = 0; i < 14; i++)

{

rooms[i].roomNumber = i + 1;

rooms[i].creature = NULL;

rooms[i].item = NULL;

rooms[i].nextRoom = NULL;

rooms[i].prevRoom = NULL;

}

rooms[2].creature = &goblin;

rooms[5].creature = &ork;

rooms[7].creature = &golem;

rooms[11].creature = &dragon;

rooms[1].item = &item1;

rooms[3].item = &item2;

rooms[6].item = &item4;

rooms[7].item = &item7;

rooms[8].item = &item5;

rooms[9].item = &item3;

rooms[12].item = &item6;

Room\* roomPtr[14];

for (int i = 0; i < 14; i++)

{

roomPtr[i] = &rooms[i];

}

attachRooms(roomPtr, 14);

char name[30];

printf("Enter your name: ");

scanf\_s("%s", name, sizeof(name));

Player p1 = { " ",100,50,NULL};

strcpy\_s(p1.name, sizeof(p1.name), name);

printf("Enter your command:\n");

char command[100];

int i = 0;

int loop = 0;

while (loop==0) {

rooms[i];

//printf("Player health %d\n", p1.attack);

scanf\_s("%s", command,sizeof(command));

if (strcmp(command,"MoveForward")==0) {

if (i < 13) {

i++;

printf("You are at room %d\n", i + 1);

}

else

{

printf("You completed the game.");

}

}

else if (strcmp(command, "MoveBack")==0) {

if (i == 0) {

printf("You cant go back");

}

else

{

i--;

printf("You are at room %d\n", i + 1);

}

}

else if (strcmp(command, "Pickup")==0) {

pickUp(&rooms[i], &p1);

rooms[i].item = NULL;

}

else if (strcmp(command, "LookUp")==0) {

lookUp(i+1);

}

else if (strcmp(command, "Attack")==0) {

war(rooms[i].creature, &p1);

rooms[i].creature = NULL;

}

else if (strcmp(command, "Save")==0) {

saveGame(&rooms[i], &p1);

}

else if (strcmp(command, "Load")==0) {

i=loadGame(& rooms[i], &p1);

}

else if (strcmp(command, "Exit")==0) {

exitGame();

}

else

{

printf("Invalid move!\n");

}

}

return 0;

}

# Conclusion

In conclusion, I have created a game in which are thirteen rooms which have different purposes in the game. Implements structures are used to create items, creatures, rooms, etc. I used functions to attach rooms, items and also I used them to create war between the player and the monsters. In addition to these, the player is able to load the saved game, pick up items, look up the descriptions of the rooms, save the game, and exit.