INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistemas Operativos

Trabalho de Grupo N.º1

Elaborado por:

Ricardo Madeira Luís, 16147

Docente:

Luís Garcia

26 de Novembro de 2017

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

A temática do jogo apresentado foi desenvolvida tendo por base os jogos de aventura point and click dos anos 80 e 90 (Leisure Suit Larry, Broken Sword, Day of the Tentacle, etc…). Assim seria necessária uma localização, um jogado, um objectivo e um adversário.

Depois de muita criatividade e de alguma ponderação, optei por uma abordagem mais séria, tentando tornar o jogo um pouco mais escuro e tentando focar-me mais na história e narrativa, uma vez que estaria muito limitado em termos de programação. Assim a escolha final recaiu sobre uma fuga de uma prisão em pleno motim dos prisioneiros, onde encarnamos um prisioneiro, temos como objectivo abrir o portão de saída e corremos o risco de encontrar um guarda prisional.

A escolha de IDE recaiu sobre o Code:Blocks. Depois de ter iniciado o código em Visual Studio 2013, devido à complexidade deste IDE e de os exemplos da aula terem começados a ser trabalhados em Code:Blocks decidi efectuar a mudança, sabendo que as implementações da API do Windows me obrigariam a uma nova mudança do código para o Visual Studio 2013.

Infelizmente o trabalho não progrediu a um ritmo desejável e muitos dos pontos pedido não chegaram e ser iniciados. Assim todo o código foi efectuado no Code:Blocks.

2. Versão simples do jogo

A versão simples do jogo está muito próxima do código fornecido pelo professor. Existem 4 estruturas principais que definem o jogador, a célula do mapa, o objecto e o monstro.
A struct Player, representa um jogador e é composta por:

Elemento	Tipo	Descrição
Name	string	Representa o nome do jogador
Health	int	Representa a quantidade de vida do jogador
Offence	int	Representa a força de ataque do jogador
Defence	int	Representa a força de defesa do jogador
Visibility	Int (flag)	Representa se o jogador pode ser visto, ou não pelo monstro
Location	int	Representa a cell em que o jogador se encontra
Item 1	int	Representa o primeiro objecto recolhido pelo jogador
Item 2	int	Representa o segundo objecto recolhido pelo jogador
Item 3	int	Representa o terceiro objecto recolhido pelo jogador
tresaure	int	Representa a posse do tesouro pelo jogador

Tabela 1 - estrutura player

A struct Cell, representa uma das divisões do mapa e é composta por:

Elemento	Tipo	Descrição
North	int	Representa uma porta para outra cell
West	int	Representa uma porta para outra cell
South	int	Representa uma porta para outra cell
Up	int	Representa uma porta para outra cell
Down	int	Representa uma porta para outra cell

Celldescription	string	Representa descrição da cell e fornece a narrativa do jogo
Object	int	Representa a existência do objecto na célula
tresaure	int	Representa a existência do tesouro na célula

Tabela 2 - estrutura cell

A struct Object, representa um objecto e é composta por:

Elemento	Tipo	Descrição
Name	string	Representa o nome do objecto.
healthBonus	int	Representa o bónus de energia que o jogador recebe.
offenceBonus	int	Representa o bónus de força de ataque que o jogador recebe.
defenceBonus	int	Representa o bónus de força de defesa que o jogador recebe.
visibilityBonus	int	Representa o bónus de visibilidade que o jogador recebe.

Tabela 3 - estrutura object

A struct Monster, representa um monstro e é composta por:

Elemento	Tipo	Descrição
Name	string	Representa o nome do monstro
Health	int	Representa a quantidade de vida do monstro
Offence	int	Representa a força de ataque do monstro
Defence	int	Representa a força de defesa do monstro
Location	int	Representa a cell em que o monstro se encontra

Tabela 4 - estrutura monster

De seguida todos os elementos são inicializados com valores definidos no código e alguns pedidos ao utilizador.

O mapa é composto por 19 células, numeradas de 0 a a18 que correspondem aos espaços da prisão. De seguida apresenta-se um mapa simples.

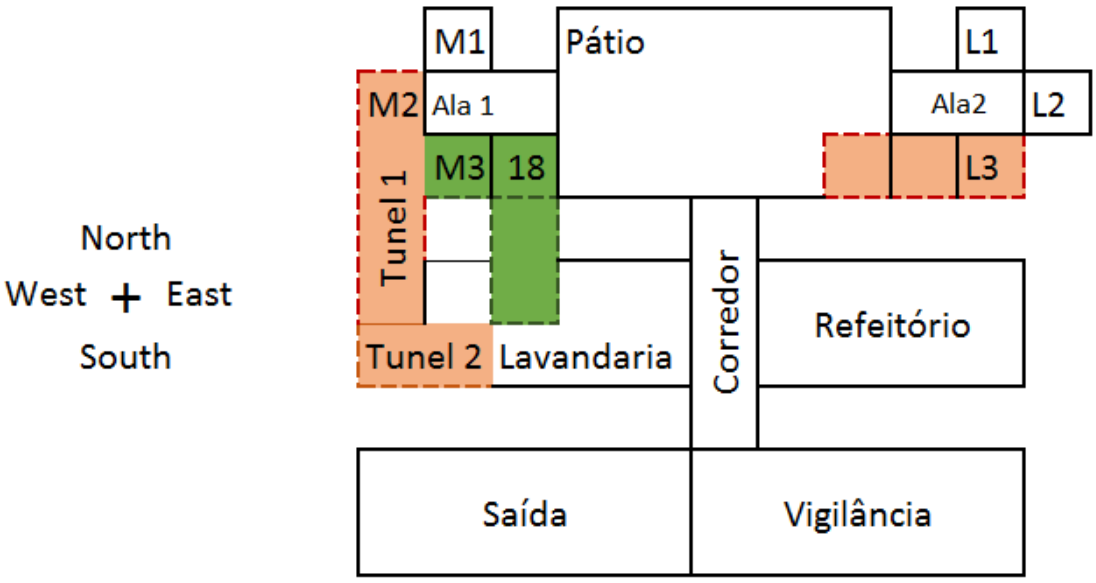

Figura 1 - mapa utilizado

Os tuneis (passagens inferiores) estão indicados a castanho claro e as condutas (passagens superiores) a verde. Quer os tuneis, quer as condutas são compostas por duas salas ligadas, permitindo assim a ligar duas salas entre elas, uma vez que as subidas dão-se em locais

diferentes. O mesmo se passa para as condutas de ventilação que permitem a descida em locais diferentes.

n.º da célula na estrutura	Célula do mapa
0	M1
1	M2
2	M3
3	Ala1
4	Pátio
5	Ala2
6	L1
7	L2
8	L3
9	Corredor

n.º da célula na estrutura	Célula do mapa
10	Lavandaria
11	Refeitório
12	Vigilância
13	Saída
14	Túnel 1
15	Túnel 2
16	Túnel 3
17	Túnel 3
18	Ventilação 1
19	Ventilação 2

Tabela 5 - representação do mapa

Em termos de objectos foram introduzidos 3 objectos, uma chave de fendas que dá um bónus de ataque de 50 ao jogador, um tabuleiro de metal que dá um bónus de defesa de 50 e uma farda de guarda que dá um bónus de visibilidade de 1 que significa que o monstro deixará de ver o jogador.

O jogo decorre através de um loop while na função main. Esse loop tem como condição que a função EndGame devolva o valor 0, o que assinala que as condições de final de jogo ainda não foram alcançadas.

Assim em primeiro lugar é verificado pela função EndGame se:

- O player possui o tesouro, e se encontra na cell 13 (saída), o que representa a condição de vitória do jogo, a função devolve 1
- Se o player possui uma health menos que 0, o que representa que foi derrotado em combate, ou seja uma condição de derrota do jogo, a função devolve 1
- Caso contrário será sempre devolvido 0, o que significa que o jogo continua.

De seguida é chamada a função movePlayer que irá criar uma string que representa as possíveis escolhas de movimentação, dentro da cell onde o jogador se encontra.

É pedido ao utilizador para fazer o input de uma escolha e caso a mesma não seja válida o utilizador é avisado. Caso a escolha seja válida o jogador é movimentado para essa cell e a descrição da cell é mostrada, avançamos assim na narrativa.

Caso existam um objecto na cell para onde o jodador se deslocou é chamado a função GrabObject, que informa da existência de um objecto na cell e o coloca num dos “espaços” disponíveis criados na struct Player. A função GrabOBject chama uma função AddObjectTo Player que actualiza a Health, offense, defense e visibility , baseada nos bónus do objecto recolhido.

A função MovePlayer chama ainda a função GrabTresaure que verifica se estamos numa cell com tesouro e caso estejamos chama a função HackComputer que nos apresenta um mini-jogo que permite desbloquear o portão de saída. Podemos considerar que esta função HackComputer que nos vais pedir um código que é alterado com base na data em que o jogo é corrido é o verdadeiro tesouro que o jogador deve reclamar.

Depois de executar o MovePlayer, executa-se o Move Monster que funciona de modo muito similar ao move player, mas com a diferença que em vez de pedir um input ao jogador, vamos gerar (através de uma função auxiliar chamada RandomNumber) um numero aleatório entre 0 e a lenght da string que representa as opções de portas existentes na cell. Esta aproximação apesar de mais complicada, permite que não existam tentativas para a saída da cell, uma vez que o random number será sempre correspondente a uma porta existente. Outro efeito é que o monstro nunca fica parado pois encontra sempre uma saída na chamada do MoveMonster.

Finalmente é chamada a função Combat, que verifica se o jogador e o monstro se encontram na mesma cell e caso estejam fornece uma série de frases que indicam o resultado dos ataques. Os ataque são realizado por uma função auxiliar chamada Attackmove.

A AttackMove basea-se em dois conceitos, os ataques são realizados à vez e o damage é dado por uma pequena expressão matemática.

Assim ficou determinado que o ataque inicial será sempre do monstro (uma vez que é o ultimo a entrar na cell) e que atacará sempre em rondas pares. A formula do dano feito ao jogador é :

damage = ((*pmonster).offence / 10) * multiplier - (((*pplayer).defence/100)) * (multiplier + 5);

Em que o multiplicador é um numero aleatório entre 1 e 5. Um exemplo de um ataque do monstro, em que o multiplicador é 4, será:

Damage = (100 / 10) * 4 – [(100/100) * (4 + 5)] = 40 – 9 = 31

A função Combat efectua assim a subtracção do valo à health do jogador e avança o roundCounter de modo a ser a vez do jogador atacar. É então perguntado ao jogador se pretende defender e a ronda é avançada para um número impar, impedindo assim o monstro de atacar. No caso de escolher atacar a função AttackMove é chamada novamente, com a informação de que é o jogador a atacar, e a formula de ataque é:

damage = ((*pplayer).offence / 10) * multiplier - (((*pmonster).defense/100)) * (multiplier + 5);

Considerando que o multiplicador é 3, o damage seria então:

Damage = (100/10) * 3 – [(100/100) * (3 + 5)] = 30 – 8 = 32

Dos exemplos pode-se entender que caso o jogador não transporte objectos, que permitem bónus, o combate será muito equilibrado, durando sempre entre 3 a 10 rondas.

3. Funcionamento do modo Super User

O funcionamento do super user depende da inserção do código 1234 quando o jogo for chamado por consola. Para tal foi criada um IF no initialize player que caso seja detectado que existem o argv 1234 e que o numero de argc é igual a 5 o player será inicializado com os valores fornceidos na linha de comandos.

É também ligado uma flag global (superFlag) que quando está igual a 1 permite 2 prints na função move monster que indicam a cell de origem do monstro e a cell de destino.

4. Inicialização do mapa com ficheiro de texto

A inicialização do mapa faz-se fazendo a leitura do ficheiro map.txt. Que está formatado para conter uma linha com os valores north, south, wes, east, up, down, object e treasure e por baixo da mesma a descrição da célula em diversas linhas.

A informação de uma célula é separada da outra por uma linha em branco.

```
1  -1 3 -1 -1 -1 -1  -1 -1
2  You are in the familiar confines of your cell.
3
4  After nine years watching the sky through those bars,you still ask yourself:
5  "Why do i have to pay for another men's crime.
6
7  -1 -1 -1 3 -1 14  -1 -1
8  You are in maximum security cell n.2.
9
10 The cell organization of the bank robber, whom you consider your only friend in this hell hole, fell's upon you.
11 But you notice something strange!
```

Figura 2 - configuração do ficheiro map.txt

O ficheiro é aberto com a função fopen e é inciado um ciclo while. A condição desse ciclo é que o fscanf da linha deve ser diferente do caracter EOF. No entanto a candição do while vai também fazer scan dos valores decimais para os campos cell.north, cell.south, etc …

Dentro do while existe outro while cuja condição é que o resultado do fgets dessa linha deve ser diferente de null, ou seja, uma linha com algo.

Nesse caso se a linha diferente de um caracter \n, o conteúdo é concatenado para cells[].cellDescription. Caso encontremos uma linha que contenha só o caracter \n, então o bloco que descreve a célula acabou e voltamos ao primeiro while que vais buscar uma linha com valores decimais que representam as portas, objecto e tesouro.

Para retirar o \n que se encontra no final da descrição, simplesmente dizemos que o ultimo caracter da descripção passa a null.

Finalmente incrementamos o counter nCells.

5. Inicialização dos objectos com ficheiro binário

Não implementado

6. Funcionalidade de gravação do jogo

Não implementado

7. Melhorias da interface com a API da consola

Não implementado

8. Funcionalidades extra

Como funcionalidade extra apenas posso apontar o facto de carregar o título do jogo em ASCII de um ficheiro txt e a função de HackComputer que exige um code que é formado pelo dia e mês, ou seja na presente data de entrega deste relatório o jogo gera o código 2611 que o jogador deve descobrir através da narrativa, pois é informado que o guarda da sala de vigilância está morto ao lado do seu bolo de aniversário e ainda dito que "people will allways be people" como uma clara alusão à tendeência de utilizar sempre passwords familiares.

9. Conclusões

Com a realização deste trabalho tomei contacto com a linguagem C. De facto, esta é sem a menor de dúvidas uma linguagem que origina programas bastante rápido, no entanto isto é alcançado muito à base de simplicidade e facilidade de utilização. A maior parte da implementação das funções de C parece um trabalho de correcção de erros e a existências de várias funções com nomes similares que foram criadas ao longo do tempo, mas com outputs diferentes não ajuda muito. Trabalhar com strings (algo banal em outras linguagens) é algo complicado em C, é sempre necessário ter em atenção o espaço que se definiu para a "string" e fazer o flushs nas alturas correctas, para evitar erros.

No entanto o estado do trabalho não se deve às dificuldades com a linguagem, mas sim a má gestão de tempo. Penso que pelo menos a funcionalidade de gravação do jogo deveria estar ao meu alcance e não a consegui implementar por falta de tempo.

Quanto a criticas ao enunciado do trabalho, penso que o meso é algo confuso, transmitindo a ideia de que o jogo simples é extremamente fácil de fazer quando na verdade este envolve o grosso do código e necessita de ser posteriormente alterado para implementar as funcionalidades pedidas mais à frente.

Assim concluo que este trabalho foi muito problemático para mim e que poderia ter conseguido um código melhor, se tivesse gerido melhor o meu tempo e enfrentado o trabalho como algo mais complexo do que aquilo que parecia inicialmente.

10. ANEXO – Código completo

```
//#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <windows.h>
#include <time.h>

#define MAX_PLAYER_NAME 100
#define PLAYER_INITIAL_HEALTH 150
#define PLAYER_INITIAL_OFFENCE 100
#define PLAYER_INITIAL_DEFENSE 100
#define PLAYER_INITIAL_VISIBILITY 1
#define PLAYER_INITIAL_CELL 0
#define PLAYER_INITIAL_OBJECT -1
#define PLAYER_INITIAL_TREASURE -1

#define MAX_CELL_DESCRIPTION 500
#define MAX_CELLS 25

#define MAX_OBJECT_NAME 20
#define MAX_OBJECTS 3

#define MAX_NAME_MONSTER 100
#define MONSTER_INITIAL_HEALTH 100
#define MONSTER_INITIAL_OFFENCE 100
#define MONSTER_INITIAL_DEFENSE 100
#define MONSTER_INITIAL_CELL 12

struct Player {
    char name[MAX_PLAYER_NAME];
    int health;
    int offence;
    int defence;
    int visibility;
    int location;
    int item1;
    int item2;
    int item3;
    int treasure;
};

struct Cell {
    int north;
    int west;
    int south;
```

```
    int east;
    int up;
    int down;
    char cellDescription[MAX_CELL_DESCRIPTION];
    int object;
    int treasure;
};

struct Object {
    char name[MAX_OBJECT_NAME];
    int healthBonus;
    int offenceBonus;
    int defenceBonus;
    int visibilityBonus;
};

struct Monster {
    char name[MAX_NAME_MONSTER];
    int health;
    int offence;
    int defense;
    int location;
};

//Debugging methods that should be commented
void PrintPlayer(struct Player player);
void PrintMap(struct Cell cells[], int nCells);
void PrintObject(struct Object object[], int nObjects);
void PrintMonster(struct Monster monster);
//End of Debugging methods

// Initialization
void InitializePlayer(struct Player *pplayer, char *argv[], int
argc);
int InitializeMap(struct Cell cells[]); //não é preciso ponteiro,
porque o vector já é um ponteiro
int InitializeObject(struct Object object[]); //, int *pnObjects);
void InitializeMonster(struct Monster *pmonster);

// main
void PrintTitle();
void PrintTitleText();
void MovePlayer(struct Player *pplayer, struct Cell cells[], struct
Object objects[]);
void MoveMonster(struct Monster *pmonster, struct Cell cells[]);
void GrabObject(struct Player *pplayer, struct Cell cells[], struct
Object object[]);
void addObjectToPlayer(struct Player *pPlayer, struct Object
object[], int idObject);
```

```
void Combat(struct Player *pPlayer, struct Monster *pmonster);
int EndGame(struct Player *pplayer, struct Cell cells[]);

//AUXILIAR METHODS
int RandomNumber(int numberOfRandoms);
void GrabTreasure(struct Player *pplayer, struct Cell cells[]);
void HackComputer(struct Cell cells[], struct Player *pPlayer);
int AttackMove(struct Player *pplayer, struct Monster *pmonster, int
attacker);

int superFlag = 0; // flags the use of superuser mode , or was i
like to call it the Andy Dufresne mode -
http://www.imdb.com/title/tt0111161/

int main(int argc, char *argv[]) {
    system("MODE 103,60"); //This is only useed to get a correct
console screen width, otherwise the ascii gets messed up

    struct Player player;
    struct Cell cells[MAX_CELLS];
    struct Object objects[MAX_OBJECTS];
    struct Monster monster;

    int nCells, nObjects;

    PrintTitle();
    PrintTitleText();

    nCells = InitializeMap(cells);
    //PrintMap(cells, nCells);

    InitializePlayer(&player, argv, argc);
    //PrintPlayer(player);

    nObjects = InitializeObject(objects); //, &nObjects);
    //PrintObject(objects, nObjects);
    //addObjectToPlayer(&player, objects, atoi(argv[4])); //DEBUGGIN

    InitializeMonster(&monster);
    //PrintMonster(monster);

    // loop that makes the black magic work
    while (EndGame(&player, cells) == 0){
        MovePlayer(&player, cells, objects);
        MoveMonster(&monster, cells);
        Combat(&player, &monster);
    }
    return 0;
}
```

```c
void PrintTitle() {
    FILE *f;
    f = fopen("title.txt", "r");
    char line[103];
    while(fgets(line, sizeof(line),f) != NULL){
        printf("%s", line);
    }
}

void PrintTitleText() {
    FILE *f;
    f = fopen("title_text.txt", "r");
    char line[103];
    while(fgets(line, sizeof(line),f) != NULL){
        printf("%s", line);
    }
}

void InitializePlayer(struct Player *pplayer, char *argv[], int
argc) {
    printf("\n\n  Enter your convict name: ");
    fflush(stdin);
    scanf("%s", (*pplayer).name);

    (*pplayer).offence = PLAYER_INITIAL_OFFENCE;
    (*pplayer).defence = PLAYER_INITIAL_DEFENSE;
    (*pplayer).visibility = PLAYER_INITIAL_VISIBILITY;
    (*pplayer).item2 = PLAYER_INITIAL_OBJECT;
    (*pplayer).item3 = PLAYER_INITIAL_OBJECT;
    (*pplayer).treasure = PLAYER_INITIAL_TREASURE;

    if(argv[1]!= '\0' && atoi(argv[1]) == 1234 && argc == 5){
        (*pplayer).health = atoi(argv[2]);
        (*pplayer).location = atoi(argv[3]);
        (*pplayer).item1 = atoi(argv[4]);
        PrintPlayer(*pplayer);
        superFlag =1;

    }else{
        (*pplayer).health = PLAYER_INITIAL_HEALTH;
        (*pplayer).location = PLAYER_INITIAL_CELL;
        (*pplayer).item1 = PLAYER_INITIAL_OBJECT;
    }

}

void PrintPlayer(struct Player player) {
    printf("\n==== PLAYER ====");
```

```
    printf("\nName: %s", player.name);
    printf("\nHealth: %d", player.health);
    printf("\nOffence: %d", player.offence);
    printf("\nDefense: %d", player.defence);
    printf("\nVisibility: %d", player.visibility);
    printf("\nLocation: %d", player.location);
    printf("\nObject1: %d", player.item1);
    printf("\nObject2: %d", player.item2);
    printf("\nObject3: %d", player.item3);
    printf("\nTreasure: %d", player.treasure);
    printf("\n\n");
}

int InitializeMap(struct Cell cells[]) {
    FILE *f;

    f = fopen("map.txt", "r");
    int nCells = 0;
    char line[500];

    while(fscanf(f, "%d %d %d %d %d %d %d %d\n",
                 &cells[nCells].north,
                 &cells[nCells].south,
                 &cells[nCells].west,
                 &cells[nCells].east,
                 &cells[nCells].up,
                 &cells[nCells].down,
                 &cells[nCells].object,
                 &cells[nCells].treasure) != EOF){

        strcpy(cells[nCells].cellDescription, "");
        while(fgets(line, sizeof(line),f) != NULL){
            if(strcmp(line, "\n")!=0){
                strcat(cells[nCells].cellDescription, line);
            }else{
                break;
            }
        }

cells[nCells].cellDescription[strlen(cells[nCells].cellDescription)
- 1] = 0;
        nCells++;
    }

    fclose(f);
    return nCells;
}

void PrintMap(struct Cell cells[], int nCells) {
```

```
    int i;
    for (i = 0; i < nCells; i++) {
        printf("\n==== CELL %i ====", i);
        printf("\nDescription: %s", cells[i].cellDescription);
        printf("\nNorth: %i", cells[i].north);
        printf("\nEast: %i", cells[i].east);
        printf("\nSouth: %i", cells[i].south);
        printf("\nWest: %i", cells[i].west);
        printf("\nUp: %i", cells[i].up);
        printf("\nDown: %i", cells[i].down);
        printf("\nObject: %i", cells[i].object);
        printf("\nTresaure: %i", cells[i].treasure);
        printf("\n\n");
    }
}

int InitializeObject(struct Object object[]){ //, int *pnObjects) {
    int nObjects = 0;

    // Object 0
    strcpy(object[0].name, "Sharp screwdriver");
    object[0].healthBonus = 0;
    object[0].offenceBonus = 50;
    object[0].defenceBonus = 0;
    object[0].visibilityBonus = 0;
    nObjects++;

    // Object 1
    strcpy(object[1].name, "Metal tray");
    object[1].healthBonus = 0;
    object[1].offenceBonus = 0;
    object[1].defenceBonus = 50;
    object[1].visibilityBonus = 0;
    nObjects++;

    // Object 2
    strcpy(object[2].name, "prision guard uniform");
    object[2].healthBonus = 0;
    object[2].offenceBonus = 0;
    object[2].defenceBonus = 0;
    object[2].visibilityBonus = -1;
    nObjects++;

    return nObjects;;
}

void PrintObject(struct Object object[], int nObjects) {
    int i;
    for (i = 0; i < nObjects; i++) {
```

```
        printf("\n==== OBJECT %i ====", i);
        printf("\nName: %s", object[i].name);
        printf("\nHealth Bonus: %i", object[i].healthBonus);
        printf("\nOffence Bonus: %i", object[i].offenceBonus);
        printf("\nDefense Bonus: %i", object[i].defenceBonus);
        printf("\nVisibility Bonus: %i", object[i].visibilityBonus);
        printf("\n\n");
    }
}

void InitializeMonster(struct Monster *pmonster) {
    strcpy((*pmonster).name, "The Jailer!");
    (*pmonster).health = MONSTER_INITIAL_HEALTH;
    (*pmonster).offence = MONSTER_INITIAL_OFFENCE;
    (*pmonster).defense = MONSTER_INITIAL_DEFENSE;
    (*pmonster).location = MONSTER_INITIAL_CELL;
}

void PrintMonster(struct Monster monster) {
    printf("\n==== MONSTER ====");
    printf("\nName: %s", monster.name);
    printf("\nHealth: %d", monster.health);
    printf("\nOffence: %d", monster.offence);
    printf("\nDefence: %d", monster.defense);
    printf("\nCell: %d", monster.location);
    printf("\n\n");
}

void MovePlayer(struct Player *pplayer, struct Cell cells[], struct
Object objects[]) {

    printf("\n
##########################################################################
##############################  \n");
    printf("\n%s\n\n", cells[(*pplayer).location].cellDescription);

    char moveOptions[60] = "What door do you want to cross?";
    char moveNorth[5] = " [N]";
    char moveEast[5] = " [E]";
    char moveSouth[5] = " [S]";
    char moveWest[5] = " [W]";
    char moveUp[5] = " [U]";
    char moveDown[5] = " [D]";

    char possibleChoices[12] = ""; // Door choices in current cell
    char choosenDoor;

    // >Concatenates a string with the available door options to
move
```

```
    if(cells[(*pplayer).location].north != -1) {
        strcat(moveOptions, moveNorth);
        strcat(possibleChoices, "nN");
    }
    if(cells[(*pplayer).location].east != -1) {
        strcat(moveOptions, moveEast);
        strcat(possibleChoices, "eE");
    }
    if(cells[(*pplayer).location].south != -1) {
        strcat(moveOptions, moveSouth);
        strcat(possibleChoices, "sS");
    }
    if(cells[(*pplayer).location].west != -1) {
        strcat(moveOptions, moveWest);
        strcat(possibleChoices, "wW");
    }
    if(cells[(*pplayer).location].up != -1) {
        strcat(moveOptions, moveUp);
        strcat(possibleChoices, "uU");
    }
    if(cells[(*pplayer).location].down != -1) {
        strcat(moveOptions, moveDown);
        strcat(possibleChoices, "dD");
    }

    fflush(stdin);
    printf("\n%s : ", moveOptions);
    scanf("%c", &choosenDoor);

    if(strchr(possibleChoices, choosenDoor) == 0) {
        printf("There is no door in that direction!\n");
    } else if (choosenDoor == 'N' || choosenDoor == 'n') {
        (*pplayer).location = cells[(*pplayer).location].north;
    } else if (choosenDoor == 'E' || choosenDoor == 'e') {
        (*pplayer).location = cells[(*pplayer).location].east;
    } else if (choosenDoor == 'S' || choosenDoor == 's') {
        (*pplayer).location = cells[(*pplayer).location].south;
    } else if (choosenDoor == 'W' || choosenDoor == 'w') {
        (*pplayer).location = cells[(*pplayer).location].west;
    } else if (choosenDoor == 'U' || choosenDoor == 'u') {
        (*pplayer).location = cells[(*pplayer).location].up;
    } else if (choosenDoor == 'D' || choosenDoor == 'd') {
        (*pplayer).location = cells[(*pplayer).location].down;
    }

    GrabObject(&*pplayer,cells, objects); //Checks for objects at
enterring a new cell
    GrabTreasure(&*pplayer,cells); //Checks for tresaure at
enterring a new cell
```

```
}

void GrabObject(struct Player *pplayer, struct Cell cells[], struct
Object object[]) {
    if(cells[(*pplayer).location].object != -1) {
        printf("You have found a %s\n",
object[cells[(*pplayer).location].object].name );

        addObjectToPlayer(&*pplayer, object,
cells[(*pplayer).location].object);

        printf("Has you grab the item you feel a new strenght\n");
        printf("Offence: %d\tDefence: %d\t Visibility:%d\n",
(*pplayer).offence, (*pplayer).defence, (*pplayer).visibility);

        if((*pplayer).item1 == -1) {
            (*pplayer).item1 = cells[(*pplayer).location].object;
        } else if((*pplayer).item2 == -1) {
            (*pplayer).item2 = cells[(*pplayer).location].object;
        } else if((*pplayer).item3 == -1) {
            (*pplayer).item3 = cells[(*pplayer).location].object;
        }
        //PrintPlayer(*pplayer);  //DEBUGGING
    }
}

void GrabTreasure(struct Player *pplayer, struct Cell cells[]) {
    if(cells[(*pplayer).location].treasure == 1) {
        printf("\n%s\n\n",
cells[(*pplayer).location].cellDescription);
        HackComputer(cells, &*pplayer);
    }
}

void MoveMonster(struct Monster *pmonster, struct Cell cells[]) {

    if(superFlag == 1){
    printf("\n%s was in cell number:%d\n", (*pmonster).name,
(*pmonster).location);
    }

    char possibleChoices[6] = "";
    int numberOfExits;
    char choosenDoor[1] = "";

    if(cells[(*pmonster).location].north != -1)
{strcat(possibleChoices, "N");}
    if(cells[(*pmonster).location].east != -1)
{strcat(possibleChoices, "E");}
```

```
    if(cells[(*pmonster).location].south != -1)
{strcat(possibleChoices, "S");}
    if(cells[(*pmonster).location].west != -1)
{strcat(possibleChoices, "W");}
    if(cells[(*pmonster).location].up != -1)
{strcat(possibleChoices, "U");}
    if(cells[(*pmonster).location].down != -1)
{strcat(possibleChoices, "D");}

    numberOfExits = strlen(possibleChoices);

    //printf("\nNumber of exits:%d\n", numberOfExits); //DEBUGGING
    int choosenExit = RandomNumber(numberOfExits);
    //printf("\nChoosen exit:%d\n", choosenExit); //DEBUGGING

    fflush(stdin);
    strncpy(choosenDoor, possibleChoices+choosenExit, 1);
    //printf("\nChoosen Door:%s\n", choosenDoor);  //DEBUGGING

    if (*choosenDoor == 'N') {
        (*pmonster).location = cells[(*pmonster).location].north;
    } else if (*choosenDoor == 'E') {
        (*pmonster).location = cells[(*pmonster).location].east;
    } else if (*choosenDoor == 'S') {
        (*pmonster).location = cells[(*pmonster).location].south;
    } else if (*choosenDoor == 'W') {
        (*pmonster).location = cells[(*pmonster).location].west;
    } else if (*choosenDoor == 'U') {
        (*pmonster).location = cells[(*pmonster).location].up;
    } else if(*choosenDoor == 'D') {
        (*pmonster).location = cells[(*pmonster).location].down;
    }

    if(superFlag == 1){
    printf("\n%s is now in cell number:%d\n", (*pmonster).name,
(*pmonster).location);
    }
}

void Combat(struct Player *pplayer, struct Monster *pmonster) {
    if((*pplayer).location==(*pmonster).location &&
(*pplayer).visibility > 0) {
        printf("In the middle of the confusion your bitter enemy %s
has found you.\nA fight between the two of you is inevitable\n",
(*pmonster).name);
        int damage = 0;
        int roundCounter = 0;
        char playerChoice;
        do{
```

```
            printf("\n  ----------------------------------------
----------------------------------------------------  \n");
            damage = AttackMove(&*pplayer, &*pmonster,
roundCounter%2);
            //printf("\nDamage is %d\n", damage); //DEBUGGING
            if(roundCounter%2 == 0){ // roundCounter % 2 = 0,
monster attacks
                printf("\n%s throws at you. You prepare to receive
the blow.", (*pmonster).name);
                printf("\n%s does an attack move and makes a damage
of %d\n", (*pmonster).name, damage);
                (*pplayer).health = (*pplayer).health - damage;
                printf("\nYou get a hard blow! \nYou feel %d of your
health leave you. Now you only have %d health\n", damage,
(*pplayer).health);
            }else if(roundCounter%2 == 1){ // roundCounter % 2 = 1,
player attacks
                fflush(stdin);
                printf("Do you want to attack(A) or defende(D)? ");
                scanf("%c", &playerChoice);
                if (playerChoice == 'A' || playerChoice == 'a') {
                    printf("\n%s does an attack move and makes a
damage of %d\n", (*pplayer).name, damage);
                    (*pmonster).health = (*pmonster).health -
damage;
                    printf("\nYou hit him with a well-placed blow!
\nHe looks more tired, like %d of is health left him. Now the Jailer
only has %d health", damage, (*pplayer).health);
                }else if (playerChoice == 'D' || playerChoice ==
'd'){
                    roundCounter++; //If player defends then we skip
monster attack round
                    printf("\n%s tries to hit you whit all of his
strength, but you are able to dogde it. You still have %d",
(*pmonster).name, (*pplayer).health);
                }else{
                    printf("That is not a choice, you must attack or
defend, don't make an arse of yourself!");
                }
            }
            roundCounter++;
        }while ((*pplayer).health > 0 && (*pmonster).health > 0);

        if((*pmonster).health <= 0){printf("\nThe fight was brutal!
But %s lies on the floor in a pool of is own blood. Your attempt to
escape has just become a little more easier. \nYou start to feel
that maybe you'll the little of day!", (*pmonster).name);}
```

```
    } else if((*pplayer).location==(*pmonster).location &&
(*pplayer).visibility == 0) {
        printf("In the middle of the confusion your bitter enemy %s
has found you.\nBut your disguise has a prison guard, allows you to
slip his wrath.", (*pmonster).name);
    }
}

int EndGame(struct Player *pplayer, struct Cell cells[]) {
    if( (*pplayer).treasure == 1 && (*pplayer).location == 13 ) {
        printf("\nYou reach the exit gate and they are wide open,
now nothing will stopp your escape.");
        printf("\n\n\nTHE END");
        return 1;
    }
    if ((*pplayer).health <=0){
        printf("\nThe fight was brutal! And in the end you lie back
in your cell in a pool of your own blood. Your attempt to escape has
been brought to an end. \nBut at least you live to fight another
day.");
        printf("\n\n\nTHE END");
        return 1;
    }
    else {
        return 0;
    }
}

/*AUXILIAR METHODOS*/

int RandomNumber(int numberOfRandoms) {
    srand(time(0)); //use current time as seed for random generator
    int randomNumber = 0;
    randomNumber = rand() % numberOfRandoms;
    //printf("Random number is:%d\n", randomNumber);  //DEBUGGING
    return randomNumber;
}

void HackComputer(struct Cell cells[], struct Player *pPlayer) {
    char date[10];
    char buffer[20];
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);

    int i, n;
    i = tm.tm_mon + 1;
    n = tm.tm_mday;

    itoa(n,buffer,10); //converts int to string
```

```
    strcpy(date,buffer); //copies converteds int to a string
    itoa(i,buffer,10); //converts int to string
    strcat(date,buffer); //concatenates int to string
    //printf("\n\nDATE: %s", date);  //DEBUGGING

    int counter = 3;

    do {
        char answer[255];
        printf("Insert personal code: \n");
        scanf("%s", answer);
        if (strcmp(answer, date)!=0) {
            printf("INCORRECT CODE\n");
            counter--;
            printf("\nYou have %d tries\n", counter);
        } else if(strcmp(answer, date)==0) {
            printf("CORRECT CODE\n");
            pPlayer->treasure = 1;
            counter=0;
            printf("\nYou ear a loud unocking sound from outside the
guard's room. You've just got the code right and the exit gate his
opening.");
        } else {
            printf("\nHACK FALIED!");
        }
    } while (counter>0);

}

int AttackMove(struct Player *pplayer, struct Monster *pmonster, int
attacker){
    int multiplier = RandomNumber(5) +1;
    printf("\nMULTIPLIER IS:%d\n", multiplier);  //DEBUGGING
    int damage = 0;
    if (attacker == 1){
        damage = ((*pplayer).offence / 10) * multiplier -
(((*pmonster).defense/100)) * (multiplier + 5); // Player does
damage
    }else{
        damage = ((*pmonster).offence / 10) * multiplier -
(((*pplayer).defence/100)) * (multiplier + 5); // Monster does
damage
    }
    return damage;
}

void addObjectToPlayer(struct Player *pplayer, struct Object
object[], int idObject){
        if(superFlag == 1){
```

```
            (*pplayer).offence += object[idObject].offenceBonus;
//In super user mode adds the object bonus to the player
            (*pplayer).defence += object[idObject].defenceBonus;
            (*pplayer).visibility +=
object[idObject].visibilityBonus;
        }
}
```