INSTITUTO POLITECNICO DE BEJA
Escola superior de Tecnologia e Gestão
Licenciatura de Engenharia Informática

Sistemas Operativos

Trabalho Prático

Relatório Técnico do
Jogo "***Hunter Hallow***"
com Threads

Rute Figueiredo, nº 15408
Tiago Campos, nº 13953

5 de Fevereiro de 2018

Índice

Introdução

O jogo “Hunter Hallow” é desenvolvido no âmbito da disciplina de Sistemas Operativos, o professor da disciplina propôs desenvolver um jogo em C em que o tema do jogo fica a cargo dos grupos.

O jogo é inspirado na série Monster Hunters, este trabalho tem vários monstros pré-definidos, assim que o jogador insere o seu nome são criadas Threads, para ser mais preciso uma Thread para o jogador que vai controlar os seus movimentos e outra para o monstro que também vai ter os seus movimentos controlados pela sua Thread. As Threads serão sincronizados para permitirem ao jogador ter uma experiencia agradável.

Nesta versão o jogador não tem de derrotar tantos monstros como na versão anterior apenas tem de derrotar o Monstro final e um esqueleto.

Mecânica do Jogo

1.1. História do Jogo

O jogo trata-se de um jogo de aventura e ação, o jogo é inspirado em alguns jogos do género, trata-se de um Soldado que anda em busca de glória, reputação e reconhecimento.

Este Soldado pertence a uma organização chamada Hunter Hallow, esta organização tem como objetivo salvar a humanidade dos monstros e caçar esses monstros para impedir que estes se espalhem pelo planeta.

1.2. Mapa do jogo

O mapa do jogo representa uma pequena vila que tem uma armadura e uma espada lendárias, esta vila foi tomada por monstros selvagens sendo um deles o rei dos monstros.

1.3. Objetivos do jogo

Os objetivos que o jogador vai ter de alcançar são eliminar os quatro monstros de elite e derrotar o rei dos monstros para poder recuperar os itens lendários e salvar a vila dos monstros.

1.4. Super User

O super user é um utilizador especial que tem todos os status ao máximo e não perde com nenhum dos monstros, este só pode ser ativado por um dos seguintes comandos “SU”, “su”, “Super User” ou “super user”.

Desenvolvimento do Jogo

STRUCTs

1.1. Quais as estruturas criadas

```
struct Threads {
	struct Monster monsters[MAX_MOSNTERS];
	struct Player Player;
	struct Map map;
};
```

A estrutura a cima foi criada para permitir que os dados que estão nas estruturas do Jogador, do Monstro e do Mapa para que possam ser usada nas Threads.

Threads / Mutexs

1.1. Threads Criadas

As Threads que foram criadas nesta versão do jogo que tinha sido feita previamente no TG1, sendo estas um requisito crucial nesta fase do TG2.

Thread do Jodador

```
DWORD WINAPI ThreadMovePlayer(LPVOID lpParam)
{
    //WaitForSingleObject(hMutex, INFINITE);
    do {
    PlayerWalk(&((struct Threads *) lpParam)->Player, &((struct Threads
    *) lpParam)->map, ((struct Threads *) lpParam)->monsters);
    } while (true);
    //ReleaseMutex(hMutex);

    return 0;
}
```

Thread do Monstro

```
DWORD WINAPI ThreadMoveMonsters(LPVOID lpParam)
{
    srand(time(NULL));
    do{
    MonstersWalk(&((struct Threads *) lpParam)->Player, &((struct
    Threads *) lpParam)->map, ((struct Threads *) lpParam)->monsters);
           Sleep(1000 + (rand() % 20000));
    } while (true);
    return 0;
}
```

1.2. Função Complementar

Durante o desenvolvimento das Threads foi necessário criar uma função auxiliar para poder passar os dados que estavam nas estruturas (Player, Monster e Map) para a estrutura que passa os dados para as Threads.

```
void UpdateThreads(struct Player *pPlayer, struct Monster monster[], struct Map
*pMap, struct Threads *pThreads, int controller) {

	if (controller == 0) {
		for (int i = 0; i < monster[0].nMonsters; i++) { // apenas passar
os monstros de 5 a 8
				strcpy(pThreads->monsters[i].nameMosnter, mon-
ster[i].nameMosnter);
				pThreads->monsters[i].cellMonster = mon-
ster[i].cellMonster;
				pThreads->monsters[i].criticMonster = monster[i].crit-
icMonster;
				pThreads->monsters[i].damageMonster = monster[i].dam-
ageMonster;
				pThreads->monsters[i].itemMonster = monster[i].item-
Monster;
				pThreads->monsters[i].lifeMonster = monster[i].lifeM-
onster;
				pThreads->monsters[i].treasureMonster = mon-
ster[i].treasureMonster;
				pThreads->monsters[i].nMonsters = monster[0].nMon-
sters;
		}

		for (int i = 0; i < pMap->nCells; i++) { // para passar os dados do
mapa para a estrutura de estruturas
			pThreads->map.cell[i].north = pMap->cell[i].north;
			pThreads->map.cell[i].south = pMap->cell[i].south;
			pThreads->map.cell[i].west = pMap->cell[i].west;
			pThreads->map.cell[i].east = pMap->cell[i].east;
			pThreads->map.cell[i].up = pMap->cell[i].up;
			pThreads->map.cell[i].down = pMap->cell[i].down;
			strcpy(pThreads->map.cell[i].descriptionCell, pMap-
>cell[i].descriptionCell);
		}

		//  add data from player
		pThreads->Player = *pPlayer;
	}
	else if (controller == 1) {
		for (int i = 0; i < monster[0].nMonsters; i++) {
				strcpy(monster[i].nameMosnter, pThreads->mon-
sters[i].nameMosnter);
				monster[i].cellMonster = pThreads->mon-
sters[i].cellMonster;
				monster[i].criticMonster = pThreads->monsters[i].crit-
icMonster;
				monster[i].damageMonster = pThreads->monsters[i].dam-
ageMonster;
				monster[i].itemMonster = pThreads->monsters[i].item-
Monster;
				monster[i].lifeMonster = pThreads->monsters[i].lifeM-
onster;
```

```
                        monster[i].treasureMonster = pThreads->mon-
sters[i].treasureMonster;
                        monster[0].nMonsters = pThreads->monsters[i].nMon-
sters;
            }

            for (int i = 0; i < pMap->nCells; i++) { // para passar os dados do
mapa para a estrutura de estruturas
                    pMap->cell[i].north = pThreads->map.cell[i].north;
                    pMap->cell[i].south = pThreads->map.cell[i].south;
                    pMap->cell[i].west = pThreads->map.cell[i].west;
                    pMap->cell[i].east = pThreads->map.cell[i].east;
                    pMap->cell[i].up = pThreads->map.cell[i].up;
                    pMap->cell[i].down = pThreads->map.cell[i].down;
                    strcpy(pMap->cell[i].descriptionCell, pThreads-
>map.cell[i].descriptionCell);
            }

            //  add data from player
            *pPlayer = pThreads->Player;
      }


}
```

1.3. Criação de Mutexs

Para puder sincronizar as threads é necessário usar mutex ou semáforos para controlar as threads neste caso foi optado pela criação dos mutexs.

Um dos mutex controla os prints no ecrã impedindo que as threads escrevam umas por cima umas das outras, e o outro controla as threads para as impedir de continuar a correr ou as libertar novamente.

```
      hMutex = CreateMutex(
            NULL,                       // default security attributes
            FALSE,                      // initially not owned
            NULL);                      // unnamed mutex

      hMutexEcran = CreateMutex(
            NULL,                       // default security attributes
            FALSE,                      // initially not owned
            NULL);                      // unnamed mutex
```

Conclusão

Ao realizar este jogo foi necessário dividir a sua realização por fases para facilitar a criação de métodos assim como a gestão de tempo e para facilitar a correção de erros que foram aparecendo ao longo do desenvolvimento do jogo.

Um dos objetivos que se pretendia era o jogador assim como o monstro terem uma thread própria em que a sua respetiva thread controla os seus movimentos, permitindo assim tanto o jogador como o monstro andarem em simultâneo pelo mapa.

Alguns dos problemas e/ou bugs surgiram durante a criação das threads pois os parâmetros não estavam a ser passados correntemente, assim como quando foi implantada a batalha e entre algumas funções sendo algumas alteradas drasticamente e outras ainda mesmo removidas para facilitar a gestão das threads, de uma forma gerar implementar as threads e fazer a gestão das mesmas foi um grande desafio.

Referencias

Toda a informação consultado foi do material fornecido pelo professor da disciplina.

Anexos

```
#include "stdafx.h"

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



#define MAX_NAME 12
#define MAX_MOSNTERS 100
#define MAX_ITEMS 1000
#define MAX_TREASURE 1000
#define MAX_CELLS 200
#define MAX_DESCRIPTION_CELL 1000
#define NCELLS 200
#define MAX_LINE 1000
#define MAX_FILENAME 30

#define EMPTY _T( ' ' )
#define MAX_THREADS 20

HANDLE hMutex;
HANDLE hMutexEcran;
```

```
/*
Estruturas begin
*/

struct Player {
	char namePlayer[MAX_NAME];
	int energyPlayer;
	int damage;
	int critic;
	int cellPlayer;
	int itemPlayer;
	int treasurePlayer;
};

struct Monster {
	char nameMosnter[MAX_NAME];
	int lifeMonster;
	int damageMonster;
	int criticMonster;
	int cellMonster;
	int itemMonster;
	int treasureMonster;
	int nMonsters;
};

struct Item {
	int CodItem;
	char NameItem[MAX_NAME];
	int DamageItem;
```

```
	int CriticItem;
	int PositionItem;
	int LifeItem;
};

struct Tresure {
	int CodTresure;
	char NameTreasure[MAX_NAME];
	int Gold;
	int PositionTresure;
};

struct Cell {
	int north;
	int south;
	int east;
	int west;
	int up;
	int down;

	int treasureCell;
	int itemCell;
	char descriptionCell[MAX_DESCRIPTION_CELL];
};

struct Map
{
	struct Cell cell[MAX_CELLS];
	struct Item item[MAX_ITEMS];
	struct Tresure treasure[MAX_TREASURE];
```

```
	int nCells;
};

struct SaveGame {
	struct Monster saveMonster;
};

struct Threads {
	struct Monster monsters[MAX_MOSNTERS];
	struct Player Player;
	struct Map map;
};
/*
Struct END
*/

void FunctionClear(); // esta função limpa o ecrã
void PrintToConsole(char text[]);
void InsertPlayer(Player *pPlayer);/*Funça que permite o utilizador inserir o seu avatar*/
void InicializeMonster(Monster monster[]);// inicializa varios monstros no jogo
void LoadMapFromFile(Map *pMap); // carrega o mapa do jogo de um ficheiro txt

void PlayerWalk(Player *pPlayer, Map *pMap, Monster monster[]);/*~Função que cria o mapa do jogo*/
void MonstersWalk(Player *pPlayer, Map *pMap, Monster monster[]);/*Função que permite o monstro se mover pelo mapa sozinho*/
void Battle(struct Player *pPlayer, struct Map *pMap, struct Monster monster[]);
void EndGame(struct Player *pPlayer, struct Monster monster[], struct Map *pMap);
void SaveGame(struct Player *pPlayer, struct Monster monster[]);
void LoadGame(struct Player *pPlayer, struct Monster monster[]);
void UpdateThreads(struct Player *pPlayer, struct Monster monster[], struct Map *pMap, struct Threads *pThreads, int controller);
```

```
DWORD WINAPI ThreadMovePlayer(LPVOID lpParam)
{
	do {
		PlayerWalk(&((struct Threads *) lpParam)->Player, &((struct Threads *) lpParam)->map, ((struct Threads *) lpParam)->monsters);
	} while (true);

	return 0;
}

DWORD WINAPI ThreadMoveMonsters(LPVOID lpParam)
{
	srand(time(NULL));
	do{
		MonstersWalk(&((struct Threads *) lpParam)->Player, &((struct Threads *) lpParam)->map, ((struct Threads *) lpParam)->monsters);
		Sleep(1000 + (rand() % 20000));
	} while (true);
	return 0;
}

/*THREADS END*/
/*
---------- Hunter Hallow --------
O main é onde as funções principais são chamados
*/
int main()
{
	int i;
```

```
int countThreads = 0;

hMutex = CreateMutex(
        NULL,           // default security attributes
        FALSE,          // initially not owned
        NULL);          // unnamed mutex

hMutexEcran = CreateMutex(
        NULL,           // default security attributes
        FALSE,          // initially not owned
        NULL);          // unnamed mutex


HANDLE hThreadPlayer;
HANDLE hThreadMonster;

struct Player player;
struct Monster monster[MAX_MOSNTERS];
struct Cell cells[MAX_CELLS];
struct Map map;
struct Threads threads;
int nCells;

printf("--------------------------------\n");
printf("        HUNTER HALLOW                   \n");
printf("        First Episode                   \n");
printf("--------------------------------\n");
printf("\n");


int option = 0;
```

```
	printf("Pretende iniciar um novo desafio soldado? \n");
	printf(" 1 - Novo Jogo     2 - Continuar Jogo     3 - Sair do Jogo\n");
	scanf("%d", &option);

	switch (option) {
	case 1:
		InsertPlayer(&player);
		//PrintPlayer(&player);
		InicializeMonster(monster);
		//PrintMonster(monster);

		break;

	case 2:
		LoadGame(&player, monster);
		//PrintPlayer(&player);
		break;

	case 3:
		exit(0);
		break;
	default:
		printf("Soldado estas a dormir e não percebeste as instruções vou explicar de
uma maneira mais facil se for possivel\n apenas pode iserir numeros de 1 a 3\n");
		break;
	}

	LoadMapFromFile(&map);
	//PrintMapFromFile(&map);

	UpdateThreads(&player, monster, &map, &threads, 0);
```

```
hThreadPlayer = CreateThread(
        NULL,           // default security attributes
        0,              // use default stack size
        ThreadMovePlayer,       // thread function
        &threads,         // argument to thread function
        0,              // use default creation flags
        NULL);   // returns the thread identifier

hThreadMonster = CreateThread(
        NULL,           // default security attributes
        0,              // use default stack size
        ThreadMoveMonsters,       // thread function
        &threads,           // argument to thread function
        0,              // use default creation flags
        NULL);   // returns the thread identifier



while (player.cellPlayer <= (map.nCells + 1)) {
        UpdateThreads(&player, monster, &map, &threads, 1);
        Battle(&player, &map, monster);
        EndGame(&player, monster, &map);
        UpdateThreads(&player, monster, &map, &threads, 0);
}

WaitForSingleObject(hThreadPlayer, INFINITE);
CloseHandle(hThreadPlayer);
WaitForSingleObject(hThreadMonster, INFINITE);
CloseHandle(hThreadMonster);

CloseHandle(hMutex);
CloseHandle(hMutexEcran);
```

```
	return 0;
}

/*
Função que tem como objetivo limpar a consola com se fosse um "System("cls")"
esta função foi retirado de um dos documentos de apoio do Professor Luis Garcia
*/
void FunctionClear() {
	_tsetlocale(LC_ALL, _T("Portuguese"));

	HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
	CONSOLE_SCREEN_BUFFER_INFO strConsoleInfo;

	GetConsoleScreenBufferInfo(hStdout, &strConsoleInfo);

	COORD Home = { 0, 0 };
	DWORD hWrittenChars;

	/*limpa os caracteres*/
	FillConsoleOutputCharacter(hStdout, EMPTY, strConsoleInfo.dwSize.X *
strConsoleInfo.dwSize.Y,
		Home, &hWrittenChars);
	/*limpa a formatação*/
	FillConsoleOutputAttribute(hStdout, strConsoleInfo.wAttributes,
		strConsoleInfo.dwSize.X * strConsoleInfo.dwSize.Y, Home, &hWrittenChars);

	SetConsoleCursorPosition(hStdout, Home);

}
```

```
void PrintToConsole(char text[]) {
	WaitForSingleObject(hMutexEcran, INFINITE);
	printf("%s", text);
	fflush(stdout);
	ReleaseMutex(hMutexEcran);
}

/*
Esta função inicializa o avatar do jogador no jogo,
pondendo tambem definir o modo de jogo e a dificuldade do jogo
*/
void InsertPlayer(struct Player *pPlayer) {
	printf("\n");
	printf("Soldado insere o teu nome! \n");
	scanf("%s", pPlayer->namePlayer);

	if ((strcmp(pPlayer->namePlayer, "SU") == 0) || (strcmp(pPlayer->namePlayer, "su") == 0) ||
		(strcmp(pPlayer->namePlayer, "super user") == 0) || (strcmp(pPlayer->namePlayer, "Super User") == 0)) {

		// selecionar modo de jogo / dificuldade
		pPlayer->energyPlayer = 1000000;
		pPlayer->damage = 10000;
		pPlayer->critic = 200;
		pPlayer->itemPlayer = 5;
		pPlayer->cellPlayer = 0;
		pPlayer->treasurePlayer = 5;
	}
	else {

		// selecionar modo de jogo / dificuldade
```

```
                pPlayer->energyPlayer = 900;
                pPlayer->damage = 300;
                pPlayer->critic = 200;
                pPlayer->itemPlayer = 0;
                pPlayer->cellPlayer = 0;
                pPlayer->treasurePlayer = 0;
        }
}

/*
Nesta função são inicializados os monstros do jogo
*/
void InicializeMonster(struct Monster monster[]) {
        int count = 0;
        //Monstro Principal - só depois de ser derrotado é que o jogador ganha
        monster[0].cellMonster = 13;
        strcpy(monster[0].nameMosnter, "LUCIFER");
        monster[0].damageMonster = 60;
        monster[0].lifeMonster = 1000;
        monster[0].criticMonster = 70;
        monster[0].itemMonster = 5;
        monster[0].treasureMonster = 5;
        count++;


        //Monstros lv1
        monster[1].cellMonster = 10;
        strcpy(monster[1].nameMosnter, "ESCLETO 1");
        monster[1].damageMonster = 10;
        monster[1].lifeMonster = 100;
        monster[1].criticMonster = 10;
```

```
	monster[1].itemMonster = 0;
	monster[1].treasureMonster = 1;
	count++;

	monster[0].nMonsters = count;
}

/*
Esta função carrega o mapa do jogo de um ficheiro com o nome "map.txt"
*/
void LoadMapFromFile(struct Map *pMap) {
	FILE *f;
	char l[MAX_LINE];
	int line = 1;
	int i = 0;
	int count = 0;

	int north;
	int south;
	int west;
	int east;
	int up;
	int down;
	int item;
	int treasure;

	f = fopen("map.txt", "r");


	while (fgets(l, MAX_LINE, f) != NULL) {
		if (line == 1 && strcmp(l, "\n") != 0) {
```

```
                sscanf(l, "%d %d %d %d %d %d %d %d", &north, &south, &west,
                        &east, &up, &down, &item, &treasure);

                pMap->cell[i].north = north;
                pMap->cell[i].south = south;
                pMap->cell[i].west = west;
                pMap->cell[i].east = east;
                pMap->cell[i].up = up;
                pMap->cell[i].down = down;
                pMap->cell[i].itemCell = item;
                pMap->cell[i].treasureCell = treasure;

                line++;

        }
        else if (strcmp(l, "\n") != 0 && l != " ") {
                strcpy(pMap->cell[i].descriptionCell, l);
                line = 1;
                i++;
                count++;
        }
        else {
                //do Nothing
        }

    }
    pMap->nCells = count;
    fclose(f);
}

/*
```

```
Esta função é o que permite o jogador navegar no mapa e usar algumas das funçoes
implementadas

*/

void PlayerWalk(struct Player *pPlayer, struct Map *pMap, struct Monster monster[]) {


	int option;

	//FunctionClear();

	WaitForSingleObject(hMutexEcran, INFINITE);

	printf("\n");

	printf(" ---------------------------------------------------------------------\n");

	printf(" Para andar na aldeia seleciona uma das opções abaixo Soldado\n");

	printf("1 - Norte  2 - Sul  3 - Oeste  4 - Este  5 - Subir 6 - Descer 7 - Save  8 - Menu
Principal\n");

	printf(" ---------------------------------------------------------------------\n");

	printf("\n");

	printf("Posiçao atual: %d\n", pPlayer->cellPlayer);

	printf("\n");

	printf("%s", pMap->cell[pPlayer->cellPlayer].descriptionCell);

	printf("\n");

	printf("Escolhe a direção que queres soldado: \n");

	ReleaseMutex(hMutexEcran);

	scanf("%d", &option);


	//FunctionClear();

	switch (option)

	{

	case 1:

		if (pMap->cell[pPlayer->cellPlayer].north == -1) {

			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");

		}

		else {

			pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].north;
```

```
		}
		break;
case 2:
		if (pMap->cell[pPlayer->cellPlayer].south == -1) {
			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");
		}
		else { pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].south; }
		break;
case 3:
		if (pMap->cell[pPlayer->cellPlayer].west == -1) {
			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");
		}
		else { pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].west; }
		break;
case 4:
		if (pMap->cell[pPlayer->cellPlayer].east == -1) {
			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");
		}
		else { pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].east; }
		break;
case 5:
		if (pMap->cell[pPlayer->cellPlayer].up == -1) {
			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");
		}
		else { pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].up; }
		break;
case 6:
		if (pMap->cell[pPlayer->cellPlayer].down == -1) {
			PrintToConsole("Lamento mas nao podes atravesar paredes !!!");
		}
		else { pPlayer->cellPlayer = pMap->cell[pPlayer->cellPlayer].down; }
```

```
                break;
        case 7:
                SaveGame(pPlayer, monster);
                printf("-----------------------------------\n");
                printf("O Jogo foi guardado com sucesso\n");

                break;

        case 8:
                main();
                break;
        default:
                printf("O valor introduzido é invalido!!! \n Insira novamente um numero de 1 a
8 \n");
                break;
        }
}

/*
Esta Função faz com que os monstros do indice 5 a 7 andem pelo mapa de forma aleatória e
sendo estes escolhidos de forma aleatória
*/
void MonstersWalk(struct Player *pPlayer, struct Map *pMap, struct Monster monster[]) {
        srand(time(NULL));
        int randMoveMonster = rand() % 6;
        int randMonster = rand() % monster[0].nMonsters;
        while ((randMonster < 5) && (randMonster > 8)) {
                randMonster = rand() % monster[0].nMonsters;
        }
```

```
	if ((strcmp(pPlayer->namePlayer, "SU") == 0) || (strcmp(pPlayer->namePlayer, "su") == 0) ||
		(strcmp(pPlayer->namePlayer, "super user") == 0) || (strcmp(pPlayer->namePlayer, "Super User") == 0)) {
		switch (randMoveMonster)
		{
		case 1:
			if (pMap->cell[monster[randMonster].cellMonster].north == -1) {
				PrintToConsole("Monstro não se moveu!\n");
			}
			else {
				monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].north;
				WaitForSingleObject(hMutexEcran, INFINITE);
				printf("\n");
				printf("Monstro: %s\n", monster[randMonster].nameMosnter);
				printf("Monstro foi para a sala: %d\n", monster[randMonster].cellMonster);
				ReleaseMutex(hMutexEcran);
			}
			break;
		case 2:
			if (pMap->cell[monster[randMonster].cellMonster].south == -1) {
				PrintToConsole("Monstro não se moveu!\n");
			}
			else {
				monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].south;
				WaitForSingleObject(hMutexEcran, INFINITE);
				printf("\n");
				printf("Monstro: %s\n", monster[randMonster].nameMosnter);
```

```
                    printf("Monstro foi para a sala: %d\n",
monster[randMonster].cellMonster);
                    ReleaseMutex(hMutexEcran);
                }
                break;
            case 3:
                if (pMap->cell[monster[randMonster].cellMonster].west == -1) {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                else {
                    monster[randMonster].cellMonster = pMap-
>cell[monster[randMonster].cellMonster].west;
                    WaitForSingleObject(hMutexEcran, INFINITE);
                    printf("\n");
                    printf("Monstro: %s\n",
monster[randMonster].nameMosnter);
                    printf("Monstro foi para a sala: %d\n",
monster[randMonster].cellMonster);
                    ReleaseMutex(hMutexEcran);
                }
                break;
            case 4:
                if (pMap->cell[monster[randMonster].cellMonster].east == -1) {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                else {
                    monster[randMonster].cellMonster = pMap-
>cell[monster[randMonster].cellMonster].east;
                    WaitForSingleObject(hMutexEcran, INFINITE);
                    printf("\n");
                    printf("Monstro: %s\n",
monster[randMonster].nameMosnter);
                    printf("Monstro foi para a sala: %d\n",
monster[randMonster].cellMonster);
```

```
                    ReleaseMutex(hMutexEcran);
                }
                break;
            case 5:
                if (pMap->cell[monster[randMonster].cellMonster].up == -1) {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                else {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                break;
            case 6:
                if (pMap->cell[monster[randMonster].cellMonster].down == -1) {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                else
                {
                    PrintToConsole("Monstro não se moveu!\n");
                }
                break;
            default:
                break;
            }
        }
        else {
            switch (randMoveMonster)
            {
            case 1:
                if (pMap->cell[monster[randMonster].cellMonster].north == -1) {

                }
```

```
                else {
                        monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].north;

                }
                break;
        case 2:
                if (pMap->cell[monster[randMonster].cellMonster].south == -1) {

                }
                else {
                        monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].south;

                }
                break;
        case 3:
                if (pMap->cell[monster[randMonster].cellMonster].west == -1) {

                }
                else {
                        monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].west;

                }
                break;
        case 4:
                if (pMap->cell[monster[randMonster].cellMonster].east == -1) {

                }
                else {
                        monster[randMonster].cellMonster = pMap->cell[monster[randMonster].cellMonster].east;
```

```
                }
                break;
            case 5:
                if (pMap->cell[monster[randMonster].cellMonster].up == -1) {

                }
                else {

                }
                break;
            case 6:
                if (pMap->cell[monster[randMonster].cellMonster].down == -1) {
                }
                else
                {

                }
                break;
            default:
                break;
            }
        }
}

/*
Nesta Função o jogador batalha com um monstro ou varios caso exista algum monstro na sua cela
*/
void Battle(struct Player *pPlayer, struct Map *pMap, struct Monster monster[]) {
        //WaitForSingleObject(hMutex, INFINITE);
```

```
srand(time(NULL));

int randCriticPlayer = 0; // é sorteado um valor de ataque critico
int randAtkPlayer = 0; //O Player acerta o ataque ou falha
int newatkPlayer = 0; //Novo valor de ataque do jogador

int randCriticMonster = 0; // é sorteado um valor de ataque critico
int randAtkMonster = 0; //O Monster acerta o ataque ou falha
int newatkMonster = 0; //Novo valor de ataque do monstro

 for (int i = 0; i < monster[0].nMonsters; i++) {// enquanto houver monstros na sala o
jodador vai lutando contra eles

 while (pPlayer->cellPlayer == monster[i].cellMonster && pPlayer-
>energyPlayer > 0 &&
 monster[i].lifeMonster > 0 && monster[i].cellMonster == pPlayer-
>cellPlayer) {

 randAtkPlayer = rand() % 6;
 randCriticPlayer = rand() % pPlayer->critic;
 newatkPlayer = ((pPlayer->damage * randCriticPlayer) / 100);
 newatkPlayer = newatkPlayer + pPlayer->damage;

 if (randAtkPlayer > 3 && pPlayer->energyPlayer > 0) {// o jogador ataca
 monster[i].lifeMonster = monster[i].lifeMonster -
newatkPlayer;
 WaitForSingleObject(hMutexEcran, INFINITE);
 printf("\n");
 printf("Monstro: %s\n", monster[i].nameMosnter);//o jogador
falha o ataque
 printf("Dano tirado ao Monstro: %d\n", newatkPlayer);
 printf("HP atual do Monstro: %d\n", monster[i].lifeMonster);
```

```
                ReleaseMutex(hMutexEcran);
        }
        else {
                if (pPlayer->energyPlayer > 0) {
                        PrintToConsole("Parabens acabaste de falar
completamente o ATAQUE!!!\n");
                }
                else {
                        //do nothing
                }
        }

        randAtkMonster = rand() % 6;
        randCriticMonster = rand() % monster[i].criticMonster;
        newatkMonster = ((monster[i].damageMonster * randCriticMonster) /
100);
        newatkMonster += monster[i].damageMonster;


        if (randAtkMonster > 3 && monster[i].lifeMonster > 0) { // o mosntro
ataca
                pPlayer->energyPlayer -= newatkMonster;
                WaitForSingleObject(hMutexEcran, INFINITE);
                printf("\n");
                printf("Soldado: %s\n", pPlayer->namePlayer);
                printf("Dano tirado pelo Monstro: %d\n", newatkMonster);
                printf("HP: %d\n", pPlayer->energyPlayer);
                ReleaseMutex(hMutexEcran);
        }
        else {
                if (monster[i].lifeMonster > 0) {// o monstro falhou o ataque
                        WaitForSingleObject(hMutexEcran, INFINITE);
```

```
                                        printf("Para tua sorte Soldado o Monstro %s",
monster[i].nameMosnter);
                                        printf(" acabou de falhar o ataque\n");
                                        ReleaseMutex(hMutexEcran);
                                }
                        }
                } // end while
        } // end for
        //ReleaseMutex(hMutex);
}

/*
Esta função verifica se o jogador ganha o jogo o se o jodador morre e mostra as mensagens
correspondentes
*/
void EndGame(struct Player *pPlayer, struct Monster monster[], struct Map *pMap) {
                if (strcmp(monster[0].nameMosnter, "LUCIFER") == 0 && pPlayer-
>energyPlayer > 0 && monster[0].lifeMonster <= 0) {
                        FunctionClear();
                        printf("Muitos Parabens soldado acabaste de derrotar rei dos
monstros e salvaste a vila da destruição\n");
                        printf("Agora que recuperaste a lendaria armadura mais a lendaria
espada das maos do %s\n", monster[0].nameMosnter);
                        printf("Por este feito decidimos-te prover a Class B soldado %s\n",
pPlayer->namePlayer);
                        printf("Aconselho-te a não tirar muito tempo de ferias, estou com
sentimento que em breve vamos precisar de ti ... \n");
                        printf("\n");
                        printf("----------------------------------------------------------------------------------
---------------------------------\n");
                        printf("\n");
                        printf("Jogo desenvolvido por: \n");
                        printf("Rute Figeiredo\n");
                        printf("Tiago Campos\n");
```

```
                printf("\n");
                PlayerWalk(pPlayer, pMap, monster);
        }

        if (pPlayer->energyPlayer <= 0) {
                WaitForSingleObject(ThreadMovePlayer, INFINITE);
                CloseHandle(ThreadMovePlayer);
                WaitForSingleObject(ThreadMoveMonsters, INFINITE);
                CloseHandle(ThreadMoveMonsters);
                FunctionClear();
                printf("O Soldado %s", pPlayer->namePlayer);
                printf(" foi um soldado exemplar, mas ficou demasiado convencido e
acabou por se descuidar\n");
                printf("e isso foi o seu fim -_-\n");
                printf("R.I.P : &s", pPlayer->namePlayer);
                main();
        }

        if (monster[1].lifeMonster <= 0 &&
strcmp(monster[1].nameMosnter,"ESCLETO 1") == 0 ) { // fechar a thread do walk monster
                WaitForSingleObject(ThreadMoveMonsters, INFINITE);
                CloseHandle(ThreadMoveMonsters);
        }



}

/*
Esta função guarda o jogo num ficheiro binário em que o nume é atribuido pelo utilizador
*/
void SaveGame(struct Player *pPlayer, struct Monster monster[]) {
        struct SaveGame save;
```

```
	FILE *f;
	char fileName[MAX_FILENAME];
	printf("Insere um nome para o ficheiro: \n");
	scanf("%s", fileName);
	//fgets(fileName, MAX_NAME, stdin);
	fileName[strlen(fileName) - 1] = '\0';
	strcat(fileName, ".bin");

	f = fopen(fileName, "wb");

	fwrite(pPlayer, sizeof(struct Player), 1, f);
	for (int i = 0; i < monster[0].nMonsters; i++) {
		save.saveMonster.cellMonster = monster[i].cellMonster;
		save.saveMonster.criticMonster = monster[i].criticMonster;
		save.saveMonster.damageMonster = monster[i].damageMonster;
		save.saveMonster.itemMonster = monster[i].itemMonster;
		save.saveMonster.lifeMonster = monster[i].lifeMonster;
		strcpy(save.saveMonster.nameMosnter, monster[i].nameMosnter);
		save.saveMonster.treasureMonster = monster[i].treasureMonster;
		fwrite(&save, sizeof(struct SaveGame), 1, f);
	}
	fclose(f);
}

/*
Esta função carrega o jogo de um ficheiro binário, o utilizador pode escolher qual é o save que quer carregar
*/
void LoadGame(struct Player *pPlayer, struct Monster monster[]) {
	struct SaveGame save;
	FILE *f;
	char fileName[MAX_FILENAME];
```

```
	printf("Insere o nome do savefile que pretendes continuar: \n");
	//fgets(fileName, MAX_NAME, stdin);
	scanf("%s", fileName);
	strcat(fileName, ".bin");

	f = fopen(fileName, "rb");

	if (f == NULL) {
		printf("Não existe nenhum jogo com esse nome!");
		LoadGame(pPlayer, monster);
	}
	else {
		fread(pPlayer, sizeof(struct Player), 1, f);
		for (int i = 0; i < monster[0].nMonsters; i++) {
			fread(&save, sizeof(struct SaveGame), 1, f);
			monster[i].cellMonster = save.saveMonster.cellMonster;
			monster[i].criticMonster = save.saveMonster.criticMonster;
			monster[i].damageMonster = save.saveMonster.damageMonster;
			monster[i].itemMonster = save.saveMonster.itemMonster;
			monster[i].lifeMonster = save.saveMonster.lifeMonster;
			strcpy(monster[i].nameMosnter, save.saveMonster.nameMosnter);
			monster[i].treasureMonster = save.saveMonster.treasureMonster;
		}
		fclose(f);
	}
}

/*
Esta função passa os dados da estrutura das threads para as estruturas do player, do monstro e do mapa
e vice verça
*/
```

```
void UpdateThreads(struct Player *pPlayer, struct Monster monster[], struct Map *pMap,
struct Threads *pThreads, int controller) {


	if (controller == 0) {

		for (int i = 0; i < monster[0].nMonsters; i++) { // apenas passar os monstros de
5 a 8

				strcpy(pThreads->monsters[i].nameMosnter,
monster[i].nameMosnter);

				pThreads->monsters[i].cellMonster = monster[i].cellMonster;

				pThreads->monsters[i].criticMonster =
monster[i].criticMonster;

				pThreads->monsters[i].damageMonster =
monster[i].damageMonster;

				pThreads->monsters[i].itemMonster =
monster[i].itemMonster;

				pThreads->monsters[i].lifeMonster = monster[i].lifeMonster;

				pThreads->monsters[i].treasureMonster =
monster[i].treasureMonster;

				pThreads->monsters[i].nMonsters = monster[0].nMonsters;

		}


		for (int i = 0; i < pMap->nCells; i++) { // para passar os dados do mapa para a
estrutura de estruturas

			pThreads->map.cell[i].north = pMap->cell[i].north;

			pThreads->map.cell[i].south = pMap->cell[i].south;

			pThreads->map.cell[i].west = pMap->cell[i].west;

			pThreads->map.cell[i].east = pMap->cell[i].east;

			pThreads->map.cell[i].up = pMap->cell[i].up;

			pThreads->map.cell[i].down = pMap->cell[i].down;

			strcpy(pThreads->map.cell[i].descriptionCell, pMap-
>cell[i].descriptionCell);

		}

		//  add data from player

		pThreads->Player = *pPlayer;
```

```
	}
	else if (controller == 1) {
		for (int i = 0; i < monster[0].nMonsters; i++) {
			strcpy(monster[i].nameMosnter, pThreads->monsters[i].nameMosnter);
			monster[i].cellMonster = pThreads->monsters[i].cellMonster;
			monster[i].criticMonster = pThreads->monsters[i].criticMonster;
			monster[i].damageMonster = pThreads->monsters[i].damageMonster;
			monster[i].itemMonster = pThreads->monsters[i].itemMonster;
			monster[i].lifeMonster = pThreads->monsters[i].lifeMonster;
			monster[i].treasureMonster = pThreads->monsters[i].treasureMonster;
			monster[0].nMonsters = pThreads->monsters[i].nMonsters;
		}
		for (int i = 0; i < pMap->nCells; i++) { // para passar os dados do mapa para a estrutura de estruturas
			pMap->cell[i].north = pThreads->map.cell[i].north;
			pMap->cell[i].south = pThreads->map.cell[i].south;
			pMap->cell[i].west = pThreads->map.cell[i].west;
			pMap->cell[i].east = pThreads->map.cell[i].east;
			pMap->cell[i].up = pThreads->map.cell[i].up;
			pMap->cell[i].down = pThreads->map.cell[i].down;
			strcpy(pMap->cell[i].descriptionCell, pThreads->map.cell[i].descriptionCell);
		}
		//  add data from player
		*pPlayer = pThreads->Player;
	}


}
```