



Sistemas Operativos

RELATÓRIO DO TRABALHO PRÁTICO

META 2



Guilherme de Sousa Camacho – 2021138502
Maria Tavares Reis – 2021144186
2023/2024

Índice

Introdução	3
Estrutura do trabalho	4
bot.....	4
jogoUI	4
map	5
motor	5
utils	6
Estrutura de dados	7
jogoUI	7
motor	8
Estrutura de Threads	9
jogoUI - 2 threads	9
motor - 4 threads	9
bot - 1 thread	10
Variável Ambiente	11
Lista de Comandos.....	12
jogoUI	12
motor	12
Comunicação.....	14
JogoUI -> Motor	14
Motor -> JogoUI	15
JogoUI -> JogoUI	15
Sinal.....	16
Sequência de Lógica	17
JogoUI.....	17
Motor	17

Introdução

O trabalho prático de Sistemas Operativos consiste num jogo de labirinto multijogador, com vários níveis. O objetivo dos jogadores é mover-se de um ponto inicial para um ponto final num labirinto em que poderão surgir obstáculos em posições aleatórias. Quando o jogador completa um labirinto, avança de nível no jogo e o novo mapa será mais complexo.

Antes do início do jogo há um período durante o qual os jogadores podem associar-se ao jogo. Terminado esse período, o jogo inicia e outros jogadores que surjam podem ver, mas não participar. A plataforma apenas terá a decorrer um jogo em simultâneo.

Os jogadores competem entre si num labirinto, deslocando-se de um ponto de partida até um ponto de chegada. O labirinto é gerido centralmente, mas é dado a conhecer a todos os jogadores e é representado visualmente como uma grelha de caracteres.

Estrutura do trabalho

O trabalho está organizado por pasta, para uma melhor organização do mesmo. Sendo essas pastas as seguintes:

- bot
- jogoUI
- map
- motor
- utils

bot

A pasta “bot” contem:

- bot.c

jogoUI

A pasta “jogoUI” contem:

- NamePipe
 - NamePipe.c
 - NamePipe.h
- UI
 - UI.c
 - UI.h
- jogoUI.c
- jogoUI.h

map

A pasta “map” contem:

- level1.txt
- level2.txt

motor

A pasta “motor” contem:

- Consola
 - Consola.c
 - Consola.h
- MapManager
 - MapManager.c
 - MapManager.h
- NamePipe
 - NamePipe.c
 - NamePipe.h
- UserManager
 - UserManager.c
 - UserManager.h
- motor.c
- motor.h

utils

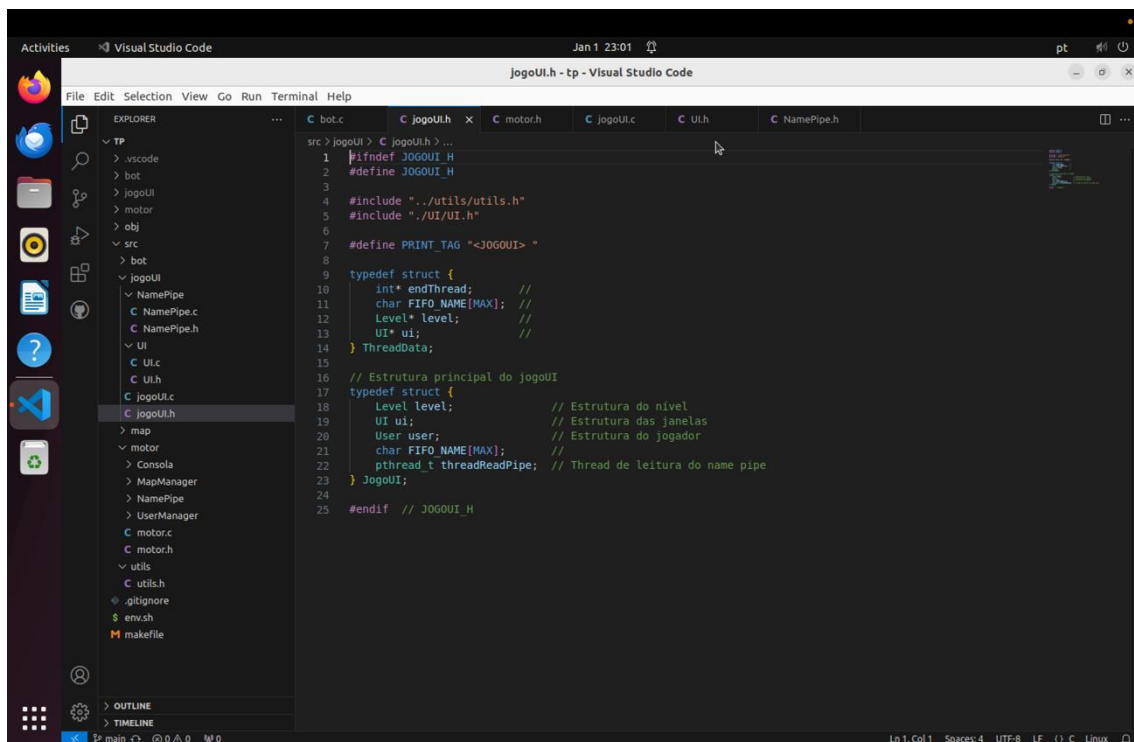
A pasta “utils” contem:

- utils.h

Estrutura de dados

jogoUI

O **jogoUI** envia ao **motor** o nome do jogador. Caso a validação seja bem-sucedida (não exista outro jogador com o mesmo nome, esteja no período de inscrições e existam vagas), o programa **jogoUI** fica a aguardar pelo início do jogo. Caso contrário, o motor informa o jogador do sucedido e ignora todos os pedidos desse jogador, mas ainda permite a este jogador ver o desenrolar do jogo. Em ambos os casos, o **jogoUI** é informado do resultado.



```
1 #ifndef JOGQUI_H
2 #define JOGQUI_H
3
4 #include "../utils/utils.h"
5 #include "../UI/UI.h"
6
7 #define PRINT_TAG "<JOGQUI> "
8
9 typedef struct {
10     int* endThread; //
11     char FIFO_NAME[MAX]; //
12     Level* level; //
13     UI* ui; //
14 } ThreadData;
15
16 // Estrutura principal do jogoUI
17 typedef struct {
18     Level level; // Estrutura do nível
19     UI ui; // Estrutura das janelas
20     User user; // Estrutura do jogador
21     char FIFO_NAME[MAX]; //
22     pthread_t threadReadPipe; // Thread de leitura do name pipe
23 } JogoUI;
24
25 #endif // JOGQUI_H
```

motor

Na implementação **pode assumir que existem máximos** para as seguintes entidades:

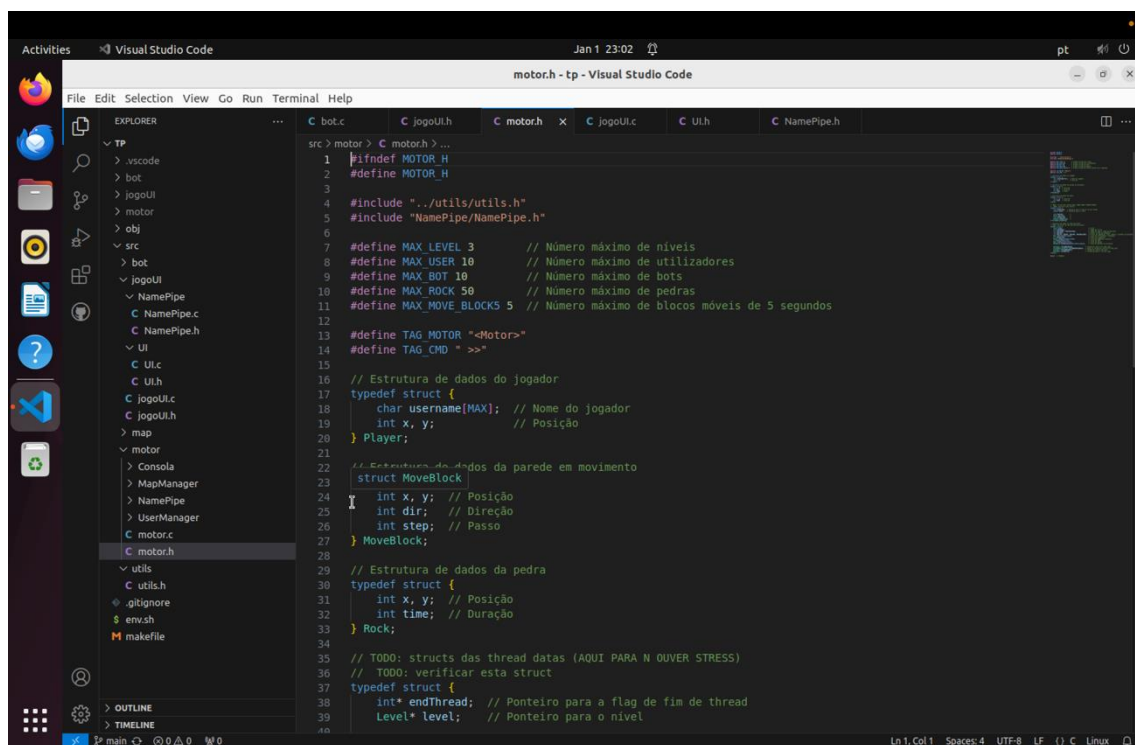
Utilizadores: máximo 5

Processos bot: máximo 10

Pedras no labirinto (definidas pelos **bots**) **em simultâneo:** máximo 50 (mais do que os **bots** porque as pedras podem permanecer por algum tempo e acumular com outras que vão surgindo).

Número de níveis: máximo 3

Bloqueios móveis: máximo 5



The screenshot shows the Visual Studio Code editor with the 'motor.h' file open. The file is a C header file defining constants and structures for a game engine. The Explorer sidebar on the left shows the project structure, including folders like 'src', 'bot', 'jogoUI', 'map', 'motor', 'utils', and 'env.sh'. The main editor area displays the following code:

```
1 #ifndef MOTOR_H
2 #define MOTOR_H
3
4 #include "../utils/utils.h"
5 #include "NamePipe/NamePipe.h"
6
7 #define MAX_LEVEL 3 // Número máximo de níveis
8 #define MAX_USER 10 // Número máximo de utilizadores
9 #define MAX_BOT 10 // Número máximo de bots
10 #define MAX_ROCK 50 // Número máximo de pedras
11 #define MAX_MOVE_BLOCKS 5 // Número máximo de blocos móveis de 5 segundos
12
13 #define TAG_MOTOR "<Motor>"
14 #define TAG_CMD " >>"
15
16 // Estrutura de dados do jogador
17 typedef struct {
18     char username[MAX]; // Nome do jogador
19     int x, y; // Posição
20 } Player;
21
22 // Estrutura de dados da parede em movimento
23 struct MoveBlock {
24     int x, y; // Posição
25     int dir; // Direção
26     int step; // Passo
27 } MoveBlock;
28
29 // Estrutura de dados da pedra
30 typedef struct {
31     int x, y; // Posição
32     int time; // Duração
33 } Rock;
34
35 // TODO: structs das thread datas (AQUI PARA N OUVIR STRESS)
36 // TODO: verificar esta struct
37 typedef struct {
38     int* endThread; // Ponteiro para a flag de fim de thread
39     Level* level; // Ponteiro para o nível
40 }
```


Estrutura de Threads

jogoUI - 2 threads

- Main
 - Input/Consola do jogador
 - Escrita para o FIFO do motor/outro jogador
- Leitura do FIFO
 - Receção do feedback do motor
 - Receber mensagem de um outro jogador
 - Receber “pacote de dados” do motor (level + infoes)
 - Tratamento de dados de receção
 - Atualiza o mapa/informações

motor - 4 threads

- Main
 - Consola do jogador
- Leitura do FIFO
 - Receção do comando do jogador
 - Enviar o feedback ao jogador caso seja um comando
 - Receção do input do jogador
 - Valida/atualiza a posição do jogador
 - Processamento geral do motor (servidor)
- Leitura do unnamed pipe (Bot)
 - Receção das coordenadas da pedra
 - Validar/criar a pedra
- Temporizador/Cronómetro/Relógio

bot - 1 thread

- Main
 - Logica do bot

Variável Ambiente

A duração do período de inscrições é dada pela **variável de ambiente INSCRICAO**, em segundos. O jogo inicia automaticamente findo esse período se tiver o número de jogadores mínimo, estipulado na **variável de ambiente NPLAYERS**. Caso não haja esse número de jogadores, o jogo só terá início quando estiverem reunidos os jogadores necessários, ou então, por ordem do administrador.

O primeiro nível demora uma quantidade de segundos indicada na **variável de ambiente DURACAO**. Cada nível seguinte demora menos **DECREMENTO** (outra variável de ambiente) segundos que o anterior.

Lista de Comandos

jogoUI

- **player**
 - lista todos os jogadores ativos
- **msg** <nome_utilizador> <mensagem>
 - manda uma mensagem para o utilizador desejado [nome_utilizador]
- **exit**
 - sair do jogo (e notifica o motor)

motor

- **users**
 - lista todos os jogadores atualmente a usar a plataforma
- **kick** <nome_utilizador>
 - retirar o utilizador desejado [nome_utilizador] da plataforma (basicamente um exit mas de seleção)
- **bots**
 - lista todos os bots ativos e as suas configurações
- **bmov**
 - inserir o bloqueio móvel

Um bloqueio móvel é um pedaço de parede que se vai movimentando a cada segundo para uma posição adjacente. Apenas se move para posições livres, não atropelando jogadores nem outros obstáculos que haja no labirinto. Serve para complicar a vida aos jogadores.

- **rbm**
 - remove o bloqueio móvel (caso de vários, remove o primeiro criado)
- **begin**
 - inicia, manualmente, o jogo
- **end**
 - encerra o jogo (havia todos os utilizadores que irá terminal, termina os jogosUI todos e termina os bot todos)
- **test_bot**
 - Comando temporário para a meta 1 para lançar um bot, receber os valores do bot e mostrar os valores no ecrã

Comunicação

JogoUI -> Motor

Tipo de mensagens

- Comando
 - move <direction>
 - msg <username>
 - plist
- Mensagem de conexão
 - *CMD_CONNECT e username*
- Mensagem de desconexão
 - *CMD_DISCONNECT e username*

```
#define CMD_CONNECT "connect"      // TODO: docs
#define CMD_DISCONNECT "disconnect" // TODO: docs
#define CMD_MOVE "move"           // TODO: docs
#define CMD_MSG "msg"             // TODO: docs
#define CMD_PLIST "plist"         // TODO: docs

#define ARG_UP "up"               // TODO: docs
#define ARG_DOWN "down"          // TODO: docs
#define ARG_LEFT "left"          // TODO: docs
#define ARG_RIGHT "right"        // TODO: docs

struct {
    pid_t PID;    // PID do processo do jogador
    char cmd[10]; // Comando - Identificar do tipo de comando (move, msg, plist)
    char arg[20]; // Argumentos do comando (move: up, down, left, right; msg: usern
} CommandToServer;
```

Motor -> JogoUI

Tipos de mensagens

- Level
- Feedback ou resposta de comando
- Aviso de encerramento

```
#define DATA_LEVEL 1      // Data do tipo Level
#define DATA_FEEDBACK 2  // Data do tipo feedback do motor
#define DATA_MSG 3       // Data do tipo mensagem de um jogoUI

#define FEEDBACK_CLOSE "close" // TODO: docs
#define FEEDBACK_INVALID "invalid" // TODO: docs

struct {
    int numPlayers;          // Numero de jogadores
    char players[5][MAX];    // Lista de jogadores
    User user;               // Dados do jogador que o jogoUI pediu
} FeedBack;

struct {
    int dataType;            // Tipo de dados (Level, FeedBack, MSG)
    Level level;             // Dados do nível
    FeedBack feedBack;       // Dados do feedback do motor
    MSG msg;                 // Dados da mensagem
} DataRecive;
```

JogoUI -> JogoUI

Tipo de mensagem

- Mensagem

```
struct {
    char username[MAX];      // Nome do jogador
    char msg[MAX];           // Mensagem
} MSG;
```

Sinal

Mecanismos para sair do programa. Cada vez que se desejamos sair do programa, é usado o sigqueue com SIGINT.

Com o SIGINT, todos os System Calls são interrompidos (scanf, read, write, etc)

```
int endFlag;

void signalHandler(int sig, siginfo_t* info, void* context) {
    endFlag = 1;
}

/* resto das funções e inicio na main */

struct sigaction sa;
sa.sa_handler = signalHandler;
sa.sa_flags = SA_SIGINFO;
sigaction(SIGINT, &sa, NULL);
```


Sequência de Lógica

JogoUI

- Main
 - Manda pedido de conexão
 - Se for valido
 - Leitura do comando
 - Validação do comando
 - Envio do comando para o motor
 - Se for para sair
 - Manda mensagem de saída
- Thread leitura do pipe
 - Ler o pacote de dados
 - Identificar o tipo de dados
 - Se for level
 - Atualizar mapa no ecrã
 - Se for feedback/resposta do comando
 - Se for lista
 - Mostrar no ecrã
 - Se for msg
 - Imprime mensagem do jogador

Motor

- Main
 - Leitura do comando
- Thread de leitura do pipe
 - Ler pacote de dados (comando)
 - Processa comando
 - Se for movimento
 - Lógica de movimento

- Atualiza mapa
 - Envia atualização a todos utilizadores conectados
- Se não
 - Verifica o tipo de comando
 - Retorna a resposta ao jogador
- Thread de leitura dos bots
 - Select na lista dos pipes dos bots
- Thread de ticks
 - A cada segundo fica a decrementar os segundos dos objetos ou a calcular o movimento de objetos