Guião da aula 5

Laboratório de Algoritmia I Laboratórios de Informática II
Ano letivo 2019/20

Tarefas a executar esta semana

Devido ao despacho RT-23/2020, não há aulas esta semana. Mas pretende-se que os alunos trabalhem em regime de ensino à distância. Assim, pretende-se que:

- Os alunos que ainda não o fizeram, deverão enviar um email ao docente do seu turno prático com a constituição do grupo até ao dia **10 de Março**;
- Será publicada a lista dos números dos grupos;
- Até ao dia **13 de Março às 20:00**, os alunos deverão submeter na plataforma **Blackboard** o resultado do seu trabalho da semana na forma de um link para o **Github** com o código correspondente ao trabalho que desenvolveram ao longo da semana. O projeto deverá ter não só o código mas também um pequeno resumo do trabalho efetuado e das dificuldades encontradas;
- Este link deverá ser mantido com as alterações que cada grupo faça ao seu projeto ao longo do semestre;
- A avaliação do acompanhamento será efetuada através da submissão do link do Github;
- Foi criado um fórum de discussão na plataforma **Blackboard** para dúvidas que possam surgir.

Esta semana pretende-se que:

1. Se criem os vários módulos;
2. Se implemente a função que inicializa o estado do jogo;
3. Se implemente a função que mostra o tabuleiro com a jogada efetuada;
4. Se implemente a função que coloca uma peça branca na casa jogada (ainda sem validar se a jogada é válida ou colocar uma peça preta no local atual da peça branca).

A Figura 1 mostra a informação que se deve submeter no **Blackboard** no link **Submissão aula 5**.

Figure 1: Eis que informação se deve submeter no link Submissão aula 5 no Blackboard

Github

Há muitos tutoriais na net, por exemplo: https://guides.github.com/activities/hello-world/. O objetivo da equipa docente é que cada grupo crie o repositório para o seu projeto e que faça *commits*. Como pretendemos avaliar a vossa prestação, incluimos as seguintes regras:

- Cada elemento deverá criar um utilizador com o **nome completo**.
- A página de rosto do projeto deve indicar:
  - o nome da **UC**;
  - o turno prático;
  - o número do grupo;
  - o número de aluno e nome completo de cada elemento do grupo;
- Cada vez que o código seja modificado, o **autor dessa modificação** deverá fazer um *commit* dessas alterações;
- A avaliação de cada elemento do grupo será de acordo com os *commits* efetuados;

Plágio

Ao colocar o código num repositório público, é sempre possível que alguém possa copiar o código de outro projeto. De acordo com o código de ética da Universidade do Minho, tal ação é considerada plágio. Caso haja dúvida em relação a plágio, a equipa docente irá utilizar informação sobre os *commits* para decidir quem produziu o código e quem o copiou. Caso a equipa docente decida que houve plágio, os grupos que sejam considerados autores de plágio terão **ZERO** na componente de projeto.

Estruturação

Pretende-se que o projeto seja estruturado em três camadas:

- Camada de dados
- Lógica do programa
- Camada de interface

Cada camada deverá corresponder a um ou mais módulos. Relembra-se que um módulo implica a existência de um ficheiro .h e outro .c com o mesmo nome, devendo o .h conter os tipos de dados e protótipos das funções que serão utilizadas pelos outros módulos e que o ficheiro .c deverá conter o código correspondente a essas funções e outras funções auxiliares que sejam necessárias.

Camada de dados

Esta camada deverá ter a estrutura de dados do programa que deverá ser algo do género:

```
typedef enum {VAZIO, BRANCA, PRETA} CASA;
typedef struct {
    int coluna;
    int linha;
} COORDENADA;
typedef struct {
    COORDENADA jogador1;
    COORDENADA jogador2;
} JOGADA;
typedef JOGADA JOGADAS[32];
typedef struct {
    CASA tab[8][8];
    COORDENADA ultima_jogada;
    JOGADAS jogadas;
    int num_jogadas;
    int jogador_atual;
} ESTADO;
```

Em que:

tab armazena informação sobre o tabuleiro;
ultima_jogada a coordenada da última jogada
jogadas armazena informação sobre as jogadas
num_jogadas indica quantas jogadas foram efetuadas
jogador_atual indica qual é o jogador a jogar

Pretende-se também que crie funções que alterem a estrutura de dados. Todas estas funções devem receber um apontador para ESTADO e modificá-lo. **Só estas funções deverão alterar o estado!** Todo o resto do programa só deverá modificar o estado através destas funções. Segue-se um exemplo dos protótipos das funções que deverão ser criadas (podendo existir outras):

```
ESTADO *inicializar_estado();
int obter_jogador_atual(ESTADO *estado);
```

```
int obter_numero_de_jogadas(ESTADO *estado);
CASA obter_estado_casa(ESTADO *e, COORDENADA c);
```

inicializar_estado Esta função deverá criar um estado vazio (com o tabuleiro inicializado)

obter_jogador_atual Esta função permite obter o número do jogador atual

obter_estado_casa Esta função permite obter o estado atual da casa

obter_numero_de_jogadas Esta função permite obter quantas jogadas foram efetuadas (cada jogada tem o movimento de dois jogadores)

Lógica do programa

Segue-se o exemplo de uma função que trata da lógica do programa:

```
int jogar(ESTADO *estado, COORDENADA c);
```

jogar esta função deverá receber o estado atual e uma coordenada e modificar o estado ao jogar na casa correta se a jogada for válida. A função devolve *verdadeiro* (valor diferente de zero) se for possível jogar e *falso* (zero) caso não seja possível.

Interface

Pretende-se nesta camada ter as funções que tratam do interface:

- O interpretador de comandos
- A parte que mostra o tabuleiro

Mostrar o tabuleiro

Deverá criar-se uma função:

```
void mostrar_tabuleiro(ESTADO estado);
```

Que deverá imprimir o tabuleiro.

Interpretador de comandos

O interpretador de comandos deverá funcionar da seguinte maneira:

1. Ler uma linha (usando fgets)
2. Separar a linha por espaços (por exemplo utilizando sscanf ou strtok)
3. Conforme o comando a executar, chamar a função correspondente do interface ou da lógica do programa e depois do interface.

Segue-se um exemplo para poder ser adaptado:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#define BUF_SIZE 1024

// Estruturas de dados (devem ser colocadas no módulo correto da camada dos dados)
    typedef enum {VAZIO, BRANCA, PRETA} CASA;
    typedef struct {
        int coluna;
        int linha;
    } COORDENADA;
    typedef struct {
        COORDENADA jogador1;
        COORDENADA jogador2;
    } JOGADA;
    typedef JOGADA JOGADAS[32];
    typedef struct {
        CASA tab[8][8];
        JOGADAS jogadas;
        int num_jogadas;
        int jogador_atual;
    } ESTADO;

// Função que deve ser completada e colocada na camada da lógica do programa
int jogar(ESTADO *e, COORDENADA c) {
  printf("jogar %d %d\n", c.coluna, c.linha);
  return 1;
}

// Função que deve ser completada e colocada na camada de interface
void mostrar_tabuleiro(ESTADO *e) {
}

// Função que deve ser completada e colocada na camada de dados
ESTADO *inicializar_estado() {
    ESTADO *e = (ESTADO *) malloc(sizeof(ESTADO));
    e->jogador_atual = 1;
    e->num_jogadas = 0;
    // Falta a resto da inicialização.
    return e;
}

// Função que deve ser completada e colocada na camada de interface
int interpretador(ESTADO *e) {
        char linha[BUF_SIZE];
        char col[2], lin[2];
```

```
        if(fgets(linha, BUF_SIZE, stdin) == NULL)
            return 0;

        if(strlen(linha) == 3 && sscanf(linha, "%[a-h]%[1-8]", col, lin) == 2) {
            COORDENADA coord = {*col - 'a', *lin - '1'};
            jogar(e, coord);
            mostrar_tabuleiro(e);
        }
    return 1;
}

// Função que deve ser colocada no ficheiro main.c
int main()
{
    ESTADO *e = inicializar_estado();
    interpretador(e);

    return 0;
}
```