Relatório
Trabalho Prático

Programação

2021-2022

Docente Responsável:
Francisco Pereira
xico@isec.pt

Tiago Figueiredo
a2020122664@isec.pt

Neste trabalho, utilizei o Visual Studio Code como IDE com recurso ao compilador GCC na minha máquina com processador ARM.

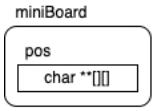
Organização de tabuleiros

Organizei o programa de forma a ter 9 tabuleiros internos e um tabuleiro externo. do tipo miniB.

```
typedef struct miniBoard{
    char **pos;
}miniB;
```

O tabuleiro externo simboliza as vitórias/empates do tabuleiro externo, ou seja, sempre que um tabuleiro interno é fechado, seja por vitória ou por empate, o respetivo tabuleiro externo relativamente à posição do interno recebe também um código de vitória (respetivo ao jogador que o vence) ou um código de empate.

Os tabuleiros internos têm a seguinte estrutura:



Estrutura da lista ligada

A estrutura ligada é do tipo pMove e é bastante simples.

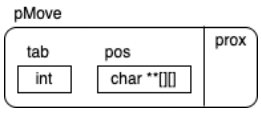
```
typedef struct move no, *pMove;

typedef struct {
    int y, x;
} coord;

struct move {
    int tab;
    coord pos;
    pMove prox;
};
```

Uma vez que conseguimos saber qual o jogador que fez a jogada tendo em conta a quantidade de jogadas, apenas guardei o tabuleiro e a posição (x,y) dentro da mesma. Ou seja, a primeira jogada (que fazendo mod 2 à posição da jogada me dá 0 ou 1, jogador 1 ou jogador 2 respetivamente) corresponde ao jogador 1, ou seja, insere um X.

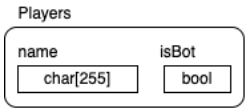
A lista ligada tem a seguinte estrutura:



Estrutura dos jogadores

Cada jogador tem uma estrutura chamada Player que contém o nome e uma varíavel booleana para definir se é ou não bot.

```
typedef struct Players{
    char name[TAMNOME];
    bool isBot;
} Player;
```



Estrutura do jogo

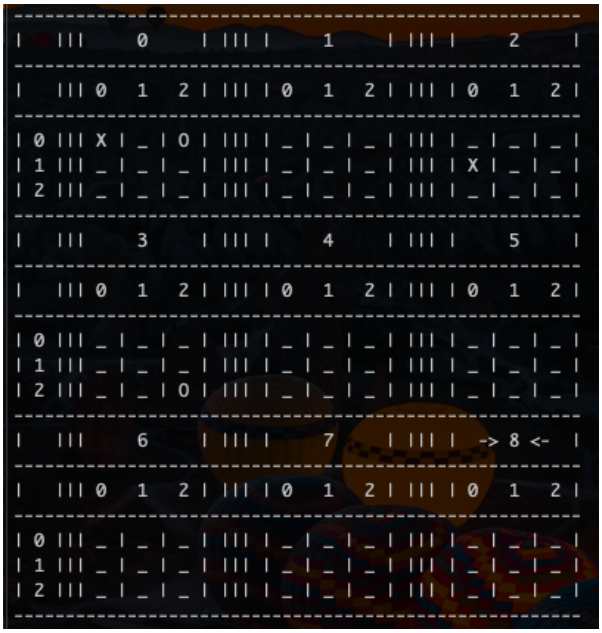
O atual jogo trata-se de um Tic-Tac-Toe, que é um jogo do galo, com 9 tabuleiros e onde esses 9 tabuleiros também fazem um jogo do galo. Para criar esta estrutura, comecei por apresentar um menu onde apresenta qual o tipo de jogo que queremos jogar. Este tipo de jogo corresponde se é para continuar um jogo começado anteriormente, jogar a 2 ou jogar sozinho (com o bot). Após essa validação, criei uma função chamada **newGame** onde esta recebe um valor **inteiro** de qual o tipo de jogo a jogar. A função começa com a criação de todos os tabuleiros envolvidos, duas estruturas para jogadores e a lista ligada que irá guardar as jogadas.

Quando é iniciado um jogo novo, são adicionadas às estruturas dos jogadores os seus nomes (se for jogo contra o bot, a estrutura do jogador2, que é o bot, já está previamente criada). De seguida, é dado um **tabuleiro aleatório** para ser jogado e **começa sempre o jogador 1**.

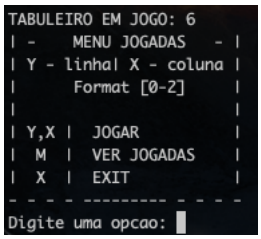
A cada jogada, é mostrado ao utilizador qual a ronda em questão e qual a peça que o jogador atual pode colocar.

```
- - Ronda 1 - -
Jogador 1: tiago -> X
- - - - - - - - -
```

De seguida, são mostrados os 9 tabuleiros existentes identificando as linhas, colunas e qual o nº do tabuleiro. Para facilitar o jogador para identificar qual o tabuleiro que está em jogo, este será identificado através de setas. (exemplo: -> 8 <-).



Depois, é mostrado o menu de jogadas, que tem **3** hipóteses possíveis, jogar, pedir para ver as jogadas anteriores ou para sair.



Verificação e inserção de tabuleiros

A função **moveRequest** é uma função que verifica o que o utilizador escolheu.
Se escolheu jogar, foi necessário inserir a posição do y (linha) e do x (coluna), separado por uma vírgula. Assim, é verificada a posição inserida pelo utilizador através da função **moveCheck**.

A função **moveCheck** é **booleana**, recebe os tabuleiros externos, o tabuleiro em jogo e a posição da jogada (y,x) e devolve **falso** se a posição do tabuleiro em questão for diferente de vazia (_), ou seja, não pode jogar porque já está ocupada ou, devolve **verdadeiro** se estiver a posição livre para jogar. Se a posição for impossível de jogar, imprime uma mensagem de erro e volta a pedir novas informações ao utilizador. Se for possível, é enviada nova informação para a função **moveSet**.

A função **moveSet** é das funções mais importantes na organização do jogo. É a única forma de alterar o conteúdo dos tabuleiros e só se chega a ela através de um pedido (**moveRequest**) e de uma verificação (**moveCheck**). Esta função recebe os tabuleiros, o nº de rondas, o tabuleiro em jogo e a posição da jogada (y,x). Dentro dela, é inserido no respetivo tabuleiro e posição, o carater respetivo do jogador que

está a jogar. Conseguimos descobrir qual é este jogador através do nº de rondas (utilizado mod 2 e verificando o resto, se 0 -> jogador 1 (X), se 1 -> jogador 2(O). Dentro desta função, atualizamos qual o tabuleiro a jogar a seguir convertendo a posição (y,x) para um tabuleiro (0-8).

Ao fim de cada jogada válida, é adicionada a mesma à lista ligada. Depois, é verificado se há uma **vitória** ou se há um **empate**. Caso não haja nenhum, aumenta o nº de rondas, verifica se o **tabuleiro** (inserido pelo utilizador em forma de posição no tabuleiro interno y,x) **seguinte** é possível, **se não, escolhe um aleatório**, verificando até ser possível.

Verificação de vitória

A verificação de vitória é feita através da função **endGame**. Esta função **booleana** recebe os tabuleiros internos, um ponteiro para o tabuleiro externo, o nº de rondas atuais e o tabuleiro jogado anteriormente. Se houver alguma linha, coluna ou diagonal preenchida no tabuleiro interno, ativa uma variável de controlo. Com esta variável de controlo ativa, vai recorrer a uma função chamada **moveSetExt** que vai colocar um carater de vitória do respetivo jogador (de acordo com o nº de rondas e através do mod 2 e verificando o resto sabemos qual o jogador) no tabuleiro externo correspondente ao interno que estamos a verificar e ainda vai verificar se aconteceu alguma vitória nos tabuleiros externos através da função **endGameExtern**. Esta função booleana, verifica linha, coluna e diagonal, mas apenas do tabuleiro externo e devolve verdadeiro ou falso. Caso seja verdadeiro, é sinal que o jogo acabou e devolve verdadeiro também na função **endGame**. Caso não haja nenhuma vitória no tabuleiro interno, vai verificar se há um empate no tabuleiro em questão através da função **checkDraw**, se existir, vai recorrer a uma função chamada **moveSetDraw**. Esta função serve para colocar um carater de empate (.) na posição do tabuleiro externo correspondente ao interno que estamos a verificar. Se nada disto se verificar, devolve falso.

Verificação de empate

Após verificar se existiu vitória, temos uma função que verifica se há um empate no tabuleiro externo chamada **checkDrawExt**. Uma vez que só chegamos a esta função se tivermos uma não vitória, esta verifica se o tabuleiro externo está todo preenchido. Se estiver, é sinal de que o jogo ficou empatado. Avança para o procedimento de fim de jogo com o pedido do nome do ficheiro TXT.

Organização de ficheiros

Neste trabalho, para além do main, usei 3 ficheiros: game, menus e utils.
O ficheiro game serviu-me para guardar estruturas e funções necessárias para a criação do jogo. Estruturas de dados, criação de jogadores, criação de tabuleiros, etc...

O ficheiro menus serviu-me para guardar todo o tipo de menus que apresento na consola, bem como a apresentação dos tabuleiros.

O ficheiro utils foi fornecido pelo professor Francisco Pereira e contei o necessário para a correta utilização de uma função random.

No ficheiro main, guardei todas as verificações de vitoria e empate e tudo o que está relacionado com a criação/leitura de ficheiros.