Programação

Ultimate Tic-Tac-Toe

Docente: Francisco Pereira

Nuno Alexandre Almeida Santos nº 2019110035 – LEI

Coimbra, 13 de junho de 2022

Índice

1. Introdução

Este trabalho surge no âmbito da Unidade Curricular de Programação, no 2º Semestre do ano letivo de 2021/2022. Feito em linguagem C standard, respeitando a norma C99, desenvolvido no Visual Studio Code, tem como objetivos principais:

1. Simular, um jogo de tabuleiro (jogo do galo) entre 2 pessoas que efetuam jogadas alternadas, até que uma delas vença ou que se verifique um empate;
2. Ser de fácil utilização, através da implementação de uma interface simples e amigável, e que esclareça o utilizador em tudo o que pode fazer num determinado momento.

Para alcançar estes objetivos, foram usados conceitos da linguagem C, como estruturas dinâmicas, estruturas ligadas, leitura de ficheiros binários e leitura e escrita de ficheiros de texto, entre outros.

2. Código Fonte

O código fonte do programa encontra-se dividido por oito ficheiros de código (extensão .c) e oito ficheiros do tipo *header* (extensão .h), separados em menus e preparação, utilidades gerais, utilidades de estruturas e utilidades de ficheiros:

- main.c
  - Contém a chamada da função *initRandom()* para inicializar o gerador de números aleatórios;
  - Contém a chamada da função *initializer()* que verifica se existe algum ficheiro em disco chamado "fich.bin" e identifica qual o modo de jogo que o utilizador irá pretender jogar.

- fileio.c/header.h
  - Contêm as funções relacionadas com o início do programa tais como o *menu() (*menu inicial*)*, *initializer()* e o *menuGame() (*menu que irá aparecer ao utilizador a cada jogada*)*.

- board.c/.h
  - Contêm todas as funções relacionadas com a manipulação do tabuleiro, tais como, *boardsInitializer(), globalBoardInitializer(), boardPrint(), globalBoardPrint(), verifyWinner(), verifyGlobalWinner(), convertPosition(), winnerSection(), convertPositionBoard() e freeBoards()* .

- plays.c/.h
    - Contêm todas as funções relacionadas com a manipulação da lista ligada, tais como, showPlays(), addNodePlays(), showKPlays(), removeList().

- file.c/.h
    - Contêm todas as funções relacionadas com a manipulação de ficheiros, tais como, *pause(), exportFile(), loadFich().*

- game.c/.h
    - Contêm todas as funções relacionadas com o funcionamento do jogo, tas como, *game()* (função principal), *choosePlays(), rules(), endGame().*

- globals.h
    - Contêm todos os includes necessários ao funcionamento do programa.

- util.c/.h e matdin.c/.h
    - Ficheiros disponibilizados pelo Professor, contendo vários funções, das quais, só utilizo a função *initRandom()* (obrigatório) e a função *setPos()* contida no ficheiro matdin.c que serve para introduzir um caracter numa determinada posição.

- Makefile
    - Permite a compilação do programa;

3. Estruturas de Dados

Neste trabalho, foram utilizadas, no total, duas estruturas diferentes:

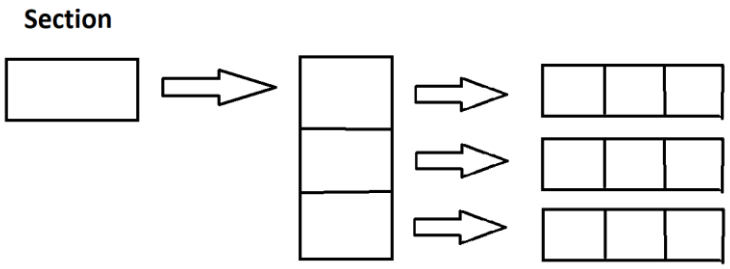
- Board - renomeada para Board;
- Plays - renomeada para Plays;

Estas estruturas foram declaradas no ficheiro board.h e no ficheiro plays.h. As suas declarações e usos vão ser apresentados nas próximas páginas.

3.1 Estrutura Board

```
typedef struct Board {
    char **section;
} Board;
```

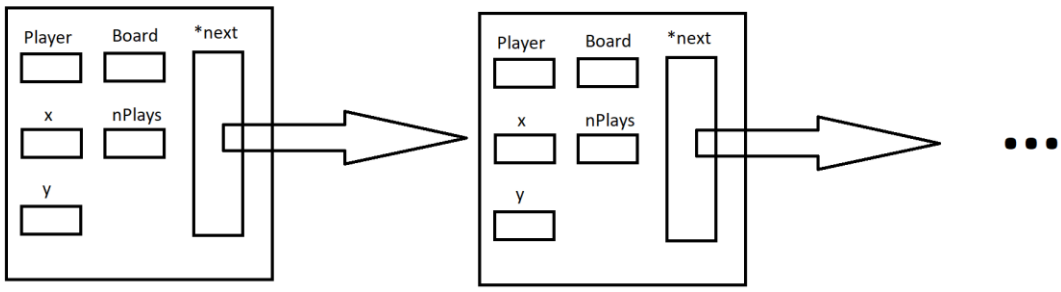
A estrutura Board é um array de arrays de caracteres, ou seja, é um array de colunas. Guarda a informação associada ao tabuleiro.



3.2 Estrutura Plays

```
typedef struct Plays {
        int player; //Para saber o jogador (1 ou 2)
        int x;
        int y;
        int Board; //Para saber o tabuleiro em que jogou
        int nPlays; //Número total das jogadas efetuadas
        struct Plays *next; //Para ligar ao próximo nó
} Plays;
```

A estrutura Plays é usada para guardar a informação relativa de cada jogada efetuada por um jogador.



4. Decisões

4.1 Estruturas Dinâmicas

Relativamente à estrutura **Board** decidi optar por criar 9 mini-tabuleiros ao invés de um tabuleiro 9x9, esta decisão deve-se ao facto de para mim, fazer mais sentido criar 9 mini-tabuleiros pois facilita na verificação das linhas/colunas e na dinâmica do jogo (posições, etc..).

Relativamente à estrutura **Plays** optei por guardar o jogador (1 ou 2), a posição onde foi inserida a peça, o tabuleiro onda a mesma foi inserida, o número de jogadas e o ponteiro "next" que liga à próxima jogada da lista.

4.1.1 Array de Arrays de caracteres

Criado a partir da estrutura Board, armazena a informação relativa a todos os tabuleiros.

4.1.3 Lista Ligada de Jogadas

Serve para guardar a informação de uma jogada, sendo criado um novo nó a cada nova jogada.

De modo a manipular esta estrutura, as seguintes funções foram implementadas:

- *showPlays()*: Mostra todo a informação que está armazenada na lista ligada;

- *addNodePlays()*: Insere a jogada no fim da lista ligada. Considero que que ao tomar a decisão de inserir o nó no fim da lista não tive em consideração que as jogadas deviam ser visualizadas da última para a primeira (Ex.: Se o utilizador pretender visualizar 2 jogadas, ele quer ver as últimas duas e não as primeiras duas), portanto se inserisse no início seria mais fácil para visualizar pela ordem pretendida. No entanto resolvi facilmente este contratempo usando funções recursivas (*showKPlays()*).

- *showKPlays()*: Mostra k jogadas anteriores inseridas na lista;

- *removeList()*: Liberta a lista de ligada.

4.2 Implementação

A implementação foi codificada da seguinte forma:

- Ao executar o programa este verifica se o ficheiro “fich.bin” existe e caso exista pergunta ao utilizador se este deseja continuar o jogo, caso o utilizador diga que sim o programa reconstrói a lista ligada e os diversos tabuleiros para que o jogo possa retomar na sua normalidade. A partir daí são pedidas sucessivas jogadas ao utilizador até que se verifique a vitória de um deles ou o empate, é apresentado um pequeno menu (ver anexo 2) com as opções que o utilizador pode escolher, são elas:

    - Ver jogadas anteriores;
    - Jogar;
    - Pausa.

  A opção “Ver jogadas anteriores” foi feita com recursa a funções recursivas, pois como foi referido acima, estou a inserir jogadas no fim da lista logo esta foi a solução mais simples que consegui implementar.

  A opção “Jogar” pede ao utilizador uma posição (0-8) que depois é convertida para linha x coluna (Ex.: posição = 1 -> linha = 0 x coluna = 1) e depois de validada a posição é inserido um “X” ou “O” consoante o jogador com recurso à função *setPos().*

  A opção “Pausa” guarda a informação necessária num ficheiro binário à qual me permite retomar o jogo mais tarde (será explicada ao detalhe na defesa).

  Caso o utilizador diga que não será apresentado um menu inicial (ver anexo 1) onde o utilizador escolhe se deseja jogar contra uma pessoa (física) ou contra um robô.

5. Manual de Utilização

5.1 Menu Inicial

Ao executarmos o programa, deparamo-nos com o primeiro menu (ver anexo 1): o menu inicial. Contém quatro opções que permite ao utilizador jogar (2 modos), obter ajuda sobre o funcionamento do jogo ou sair da aplicação.

Opções:

1. Multiplayer: O programa permite jogar simultaneamente com dois jogadores;
2. Singleplayer: O programa permite jogar contra um robô;
3. Help: Apresenta na consola toda a informação necessária que permita jogar;
4. Exit: Terminar o programa.

5.2 Menu Jogador

Na fase em que o jogo se encontra a decorrer é apresentado um segundo menu (ver anexo 2) que contém todas as opções que o jogador poderá escolher.

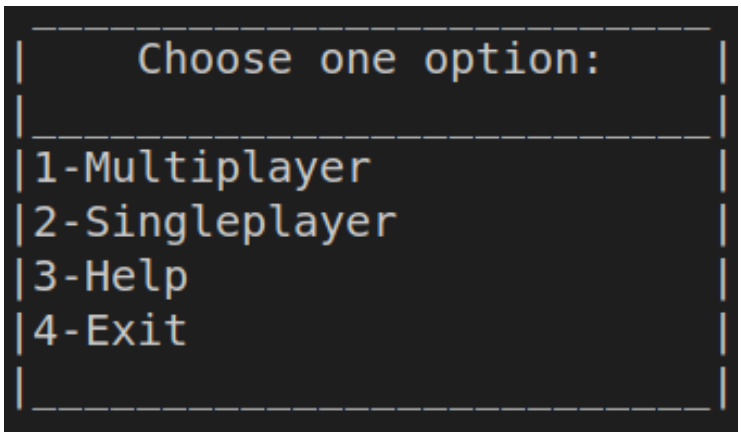
Opções:

1. Previous plays: Permite ao jogador ver k jogadas anteriores até um total de 10 jogadas;
2. Play: Permite ao jogador efetuar uma jogada;
3. Pause: Permite ao jogador colocar o jogo em pausa para ser retomado mais tarde.

6. Anexos

6.2 Anexo 1 – Menu Inicial

O Menu inicial tem o seguinte aspeto:



6.3 Anexo 2 – Menu Jogador

O Menu Jogador tem o seguinte aspeto: