UNIVERSIDADE DE ÉVORA

Othello

Relatório do trabalho

Programação 1

Trabalho realizado por:

- Henrique Rosa, EI, nº 51923

- Diogo de Matos, EM, nº 52710

22/01/2022

Introdução

→ O que é o Othello?

O Othello é um jogo de tabuleiro para dois jogadores. O tabuleiro tem 8 linhas e 8 colunas onde serão colocadas as peças. As peças são discos com uma face clara e outra escura, cada cor pertence a um jogador.

O objetivo do jogador é ter mais peças da sua cor voltadas para cima do que o seu adversário, quem tiver mais peças à mostra no final do jogo ganha.

O jogo tem início com 2 peças brancas e 2 pretas dispostas no tabuleiro, as peças brancas nas posições 4D e 5E e as peças pretas nas posições 4E e 5D.

Uma jogada tem como objetivo flanquear as peças do oponente trocando estas. Flanquear significa colocar uma peça no tabuleiro de modo a que a linha/s, coluna/s ou diagonais com peças do adversário seja limitada, em cada extremidade, por uma peça do jogador que está a jogar.

→ Regras

As pretas jogam sempre primeiro.

Se numa jogada não for possível flanquear o adversário e virar pelo menos uma peça do oponente, a vez é perdida e o oponente joga novamente. No entanto, se existir um movimento disponível o jogador é obrigado a jogar.

Todas as peças flanqueadas em qualquer movimento devem ser viradas (mesmo que seja vantajoso para o jogador não virá-las).

As peças só podem ser flanqueadas como resultado direto de um movimento e devem estar na linha direta da peça colocada.

Quando não houver mais jogadas válidas para ambos os jogadores, o jogo termina. As peças são contadas e o jogador com o maior número de peças ganha. É possível o jogo terminar sem que os 64 quadrados estejam preenchidos.

Lista de comandos

Para executar o jogo, deverá executar um dos seguintes comandos no terminal:

$ ./othello (Executar o jogo do início);

$ ./othello jogadas.txt (Executar o jogo iniciando o tabuleiro a partir de uma lista de jogadas consecutivas guardadas num ficheiro de texto indicado como parâmetro).

Descrição do código

Foi nos proposto a realização do código do jogo Othello em linguagem C. Para a realização do código utilizamos o Visual Studio. Nesta parte do relatório iremos descrever todo o nosso código, falando das bibliotecas incluídas no código e de cada função que foi feita, sendo estas as que nos foram apresentadas para fazer obrigatoriamente e ainda as que foram pensadas por nós para deixar o código ainda mais completo.

Antes de descrever cada função, queremos relembrar como será representado cada espaço livre do tabuleiro, as peças brancas e as peças pretas, sendo estas '.', 'o' e 'x', respectivamente.

→ Bibliotecas incluídas

`#include <stdio.h>` Biblioteca padrão de linguagem em C (utilizada para as funções mais conhecidas como `printf`, `scanf`, `feof`, `fopen`, `fgetc`).

`#include <stdlib.h>` Biblioteca incluída para utilizar a função `srand`, `rand`.

`#include <time.h>` Biblioteca incluída para utilizar a função `time`.

`#include <ctype.h>` Biblioteca incluída para utilizar a função `toupper`.

→ Structs

Foram criadas duas structs, a `struct jogada` e a `struct jogada_char`, a primeira comporta duas partes, ambas inteiras chamadas `line` e `col` às quais atribuímos os valores inteiros verdadeiros da linha e da coluna da jogada atual, respetivamente ao array board (usadas nas funções `computer_play_normal`, `computer_play_easy` e na `main`), a segunda comporta duas partes, ambas caracteres chamadas também `line` e `col`, às quais atribuímos caracteres alfanuméricos representantes da jogada no tabuleiro como seriam escritas pelos jogadores (usada principalmente para transportar as jogadas recebidas em formato caráter do ficheiro de texto “jogadas.txt” [ou qualquer outro nome de ficheiro de texto com jogadas, dado que o nosso programa não discrimina o nome do ficheiro]).

→ Função finish

Esta função é executada quando o jogo acaba (seja porque o tabuleiro está cheio ou porque já não há jogadas válidas para nenhuma das cores), conta as peças e determina quem ganhou, mostrando as mensagens do fim de jogo. Esta função `finish` é chamada na função `main`.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo [array de caracteres]), `int modo` (o modo de jogo selecionado no início do jogo [um jogador ou dois jogadores]), `int num` (número de jogadas realizadas no jogo) e `char comcolor` (cor das peças do computador [caso seja necessário]).

A função não retorna nada (tipo void).

Variáveis declaradas:

- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[count_brancas]`**, inteiro: serve para, depois da contagem, guardar o número de peças brancas no tabuleiro. Inicialmente toma o valor 0;
- **`[count_pretas]`**, inteiro: serve para, depois da contagem, guardar o número de peças pretas no tabuleiro. Inicialmente toma o valor 0;

Funcionamento da função:

Inicialmente a função irá imprimir a mensagem “O jogo acabou!”. De seguida irá executar a contagem das peças consoante o modo de jogo, para fazer a contagem das peças fez-se inicialmente dois ciclos `for` que iram contar linha a linha quantas peças brancas e quantas peças pretas existem, irá mostrar de seguida quem ganhou o jogo segundo as regras e irá mostrar também quantas peças existem da cor de quem ganhou e da cor de quem perdeu ou então se a contagem for igual irá mostrar que foi empate e mostrar quantas peças existem de cada cor. No fim a função pára o programa (exit(1)).

→ Função num_char_to_int

Esta função só é utilizada no caso de o jogo ser inicializado com um ficheiro de texto com jogadas válidas, serve para converter caracteres numéricos em valores inteiros, para estes poderem ser utilizados na função `play` e na função `flanked`.

Como argumentos da função temos a variável `char var` (é o caracter (numérico) que se quer converter para inteiro).

Funcionamento da função:

A função irá retornar o valor inteiro correspondente à posição do tabuleiro indicado pelo caractere que recebe como argumento. Por exemplo: "1" vai retornar 0.

→ Função let_to_num

Esta função só é utilizada para converter caracteres numéricos em valores inteiros, para estes poderem ser utilizados na função `play` e na função `flanked`.

Como argumentos da função temos a variável `char var` (aquilo que se quer converter de caractere (letra) para o seu inteiro correspondente(no array do tabuleiro)).

Funcionamento da função:

A função inicialmente irá garantir que o caractere (letra) da coluna inserido será sempre maiúscula (usando a função `toupper`) e de seguida irá retornar o valor inteiro correspondente à posição do tabuleiro indicado pelo caractere (letra) que recebe como argumento. Por exemplo: "A" vai retornar 0.

→ Função num_to_let

Esta função só é utilizada para converter números inteiros para letras (usado para conversão entre os números das colunas no array board e as suas respectivas letras), para estes serem mostrados quando o jogo mostra a jogada do computador no modo de um jogador.

Como argumentos da função temos a variável `int var` (aquilo que se quer converter de inteiro para o seu caracter(letra) correspondente(no array do tabuleiro)).

Funcionamento da função:

A função irá retornar o caractere(letra) correspondente à posição do tabuleiro indicado pelo número inteiro que recebe como argumento. Por exemplo: 0 vai retornar "A".

→ Função verify_full

Esta função é utilizada para percorrer o tabuleiro em busca de espaços “vazios” linha a linha com os ciclos `for`, retorna 0 se houver espaços e 1 se o tabuleiro estiver cheio, esta função é chamada na função `main`. Como argumentos da função temos a variável `char board` (tabuleiro de jogo(array de caracteres)).

Variáveis declaradas:

- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;

→ Função verify

Esta função é utilizada para verificar se há jogadas para o jogador atual em qualquer uma das direções, retornando 1 se houver jogadas válidas e 0 se não. Esta função é chamada na função `main`.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)), `char color` (cor das peças do jogador atual).

Variáveis declaradas:

- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[i1]`**, inteiro: vai tomar o valor do i quando é usado, e utiliza `i1` como nova variável que se mete como localizador de array, para continuar a utilizar o `i`;
- **`[j1]`**, inteiro: vai tomar o valor do j quando é usado, e utiliza `j1` como nova variável que se mete como localizador de array, para continuar a utilizar o `j`;
- **`[color2]`**, char: cor oposta à cor que está a jogar de momento, quando a função é executada;

Funcionamento da função:

A função inicialmente irá definir o valor para a variável `color2` consoante o valor da variável `color`, ou seja, se `color`=”X” então `color2`=”O” e vice-versa. Em seguida irá percorrer o tabuleiro linha a linha com os dois ciclos `for` até encontrar uma peça da cor do jogador atual. Depois irá verificar para qualquer uma das direções (esquerda, direita, cima, baixo, cima-esquerda, cima-direita, baixo-esquerda, baixo-direita) se há jogadas válidas, para isto verificando para cada direção a existência de peças oponentes (`color2`), no caso de haver peças oponentes numa dada direção (potencial jogada válida) a função vai percorrer essa direção no tabuleiro até encontrar ou um espaço vazio ou uma peça da cor do jogador atual. Se for encontrado um espaço vazio, a função continua a verificar as restantes direções das restantes posições até encontrar uma jogada válida (retornando 1) ou encontrar chegar ao fim e retornar 0.

→ Função count_flips_dir

Esta função é a parte "operacional" da contagem das peças viradas na jogada atual. Conta as peças numa certa direção do tabuleiro. Aliada à função `flanked`, conta as peças viradas na jogada atual.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)), `int line` (linhas(do array) da jogada atual), `int col` (coluna(do array) da jogada atual), `int delta_line` (direção referente às linhas, na qual se quer que a função conte as peças viradas), `int delta_col` (direção referente às colunas, na qual se quer que a função conte as peças viradas) e `char color` (cor das peças do jogador atual).

Variáveis declaradas:

- **`[color2]`**, char: cor oposta à cor que está a jogar de momento, quando a função é executada;
- **`[count]`**, inteiro: variável de contagem. Inicialmente toma o valor 0;

Funcionamento da função:

A função inicialmente irá definir o valor para a variável `color2` consoante o valor da variável `color`, ou seja, se `color`=”X” então `color2`=”O” e vice-versa. A contagem é feita com o ciclo `while` que vai, enquanto a posição especificada do board for peça branca, percorrer uma certa direção dada pelo `flanked` até encontrar ou um espaço vazio (retornando 0) ou uma peça do jogador atual. Se esta última acontecer, a função vai voltar para trás na mesma direção, incrementando a variável `count` por 1 a cada posição que passa, fazendo assim a contagem das peças que vão virar naquela direção. No fim retorna a variável `count`.

→ Função flanked

Esta função é a parte "logística" da contagem das peças viradas na jogada atual. Depois de mandar todas as direções possíveis para a função `count_flips_dir`, soma os valores por ela retornados. Aliada à função `count_flips_dir`, conta as peças viradas na jogada atual. Esta função é chamada nas funções `computer_play_easy`, `computer_play_normal` e na `main`.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)), `int line` (linhas(do array) da jogada atual), `int col` (coluna(do array) da jogada atual), `char color` (cor das peças do jogador atual).

Variáveis declaradas:

- **`[flips]`**, inteiro: variável da soma do número de peças que vai virar.

Funcionamento da função:

A função vai chamar a função `count_flips_dir` uma vez para cada uma das direções para as quais é possível haver jogada, mandando como parâmetro de cada uma das funções os valores correspondentes às direções acima referidas (exemplo: deslocar a posição na direção cima-direita, a coluna tem de aumentar e a linha tem de diminuir, logo a função é chamada com os parâmetros `delta_line=-1` e `delta_col=1`, pois são estes os valores que vão ser adicionados aos valores da linha e da coluna para nos movermos naquela direção, e assim por diante). Em cada uma das linhas, é adicionado o retorno da função `count_flips_dir` ao valor da variável `flips`, fazendo assim com que a variável `flips` comporte o número total de peças que vão ser viradas naquela jogada específica para todas as direções possíveis.

→ Função delta_play

Esta função é a parte "operacional" da execução das jogadas. Percorre todas as direções, virando as peças de acordo com as regras do jogo. Aliada à função `play`, executa as jogadas. Esta função apenas é chamada na função `play`.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo (array de caracteres)), `int line` (linhas(do array) da jogada atual), `int col` (coluna(do array) da jogada atual), `char color` (cor das peças do jogador atual), `int delta_line` (direção relativamente às linhas, na qual se quer que a função vire as peças), `int delta_col` (direção relativamente às colunas, na qual se quer que a função vire as peças) e `char color2` (cor das peças adversárias ao jogador atual).

Variáveis declaradas:

- **`[i]`**, inteiro: variável para guardar os valores recebidos do parâmetro `line` (linha da posição inicial). Serve para marcar a linha da posição inicial quando a função tiver que a ela voltar, para virar as peças daquela direção.
- **`[j]`**, inteiro: variável para guardar os valores recebidos do parâmetro `col` (coluna da posição inicial). Serve para marcar a coluna da posição inicial quando a função tiver que a ela voltar, para virar as peças daquela direção.

Funcionamento da função:

A função vai percorrer o tabuleiro na direção indicada pelos índices `delta_line` e `delta_col`, usando para isso um ciclo `while` que vai repetir enquanto a próxima posição dessa direção for da cor adversária ao jogador atual (dentro do ciclo há também um `if` que verifica se a próxima posição está fora do tabuleiro, verificando se a linha e coluna estão ambas entre 0 e 7. Caso a próxima posição esteja fora do tabuleiro, a função da return. Isto funciona como um failsafe para não haver bugs relacionados com a stack). A posição avança pela adição dos índices `delta_line` e `delta_col` às variáveis `line` e `col` a cada repetição do ciclo `while`, efetivamente movendo a posição que vamos verificar na próxima repetição do ciclo.

Em cada repetição do ciclo `while` há também a verificação (feita pelo `if`) da próxima posição: se for detectada uma peça da cor do jogador atual na próxima posição (ou seja, naquela direção,as peças têm de ser viradas), a função entra num novo ciclo `while`, que vai repetir até que a posição atual não seja diferente da posição inicial (até que os valores da linha e da coluna do array sejam iguais aos das variáveis `i` e `j`, variáveis estas discutidas acima). Este ciclo vai “andar para trás” na mesma direção (fazendo praticamente o contrário do que o outro ciclo fazia), substituindo a cor que estava antes no tabuleiro pela cor do jogador atual, efetivamente “ virando” as peças do adversário.

→ Função play

Esta função é a parte "logística" da execução das jogadas. Coloca uma peça da cor do jogador atual na posição(do array) designada e manda todas as direções possíveis para a função `delta_play`. Aliada à função `delta_play` , executa as jogadas. Esta função é chamada nas funções `computer_play_easy` , `computer_play_normal` e na função `main`.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)), `int line` (linhas(do array) da jogada atual), `int col` (coluna(do array) da jogada atual) e `char color` (cor das peças do jogador atual).

Variáveis declaradas:

- **`[color2]`**, char: cor oposta à cor que está a jogar de momento, quando a função é executada.

Funcionamento da função:

A função inicialmente irá definir o valor para a variável `color2 (cor do adversário)` consoante o valor da variável `color`, ou seja, se `color`=”X” então `color2`=”O” e vice-versa, colocando depois uma peça da cor do jogador atual (`color`) na posição indicada pelos parâmetros `line col` (o que efetivamente seria a própria jogada)

De modo similar à função flanked com a função `count_flips_dir`, a função `play` vai chamar a função `delta_play` uma vez para cada uma das direções onde seja possível virar peças, mandando como parâmetros em cada uma dessas vezes os valores de `delta_line` e `delta_col` associados àquela direção específica.(ver exemplo no funcionamento da função `flanked`).

→ Função computer_play_easy

Esta função escolhe randomicamente uma das possíveis jogadas que o computador pode fazer e aplica-a ao tabuleiro. Esta função é chamada na função `main` quando esta inicia o modo de jogo de um jogador na dificuldade fácil.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)) e `char color` (cor das peças do computador).

Variáveis declaradas:

- **`[i]`**, inteiro: variável que serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: variável que serve para o funcionamento dos ciclos `for`;
- **`[pass]`**, inteiro: variável que vai guardar um número randómico entre 0 e 5. Vai ter a função de definir quantas jogadas válidas vão ser ignoradas pela função até ser escolhida uma jogada para ser executada e retornada.
- **`[jogada_pc]`**, tipo composto `struct jogada`: variável que irá ter o valor de uma jogada qualquer para o computador. Inicialmente toma o valor 0.

Funcionamento da função:

Inicialmente a função define ambas as componentes da variável `jogada_pc` como -1 (Isto é feito para proteger o funcionamento da função, para assegurar que o ciclo `while` repete até que ambas as componentes da variável guardem um valor válido para os índices de jogada do computador e para que a função não retorne valores invalidos para a jogada, como referido abaixo.) e define a variável pass com um valor randómico entre 0 e 4. De seguida a função entra num loop `while` que repete enquanto pelo menos uma das componentes da variável `jogada_pc` for igual a -1 (isto assegura que nenhuma das componentes da variável jogada retorna sem um valor de jogada válido.).

Dentro do ciclo `while` há mais dois ciclos `for` encadeados, um que incrementa `i` de 0 até 7 e outro que incrementa `j` de zero até 7, usados para percorrer todas as linhas e colunas do tabuleiro (array `board`). A cada repetição dos ciclos, vai ser feita uma verificação (pelo primeiro `if`) se aquela posição é simultaneamente um espaço vazio e uma jogada válida para o computador (esta última condição é retirada do valor da função `flanked` para aquela posição e para a cor do computador). Caso a condição anterior se verifique, as componentes da variável `jogada_pc.line` e `jogada.col` vão tomar (respetivamente) os valores da linha e da coluna da posição atual, e retira-se 1 ao valor da variável `pass`.

Ainda na mesma repetição, é verificado (pelo segundo `if`) se o valor da variável `pass` é menor ou igual que zero e se, simultaneamente, ambas as componentes da variável `jogadas_pc` são diferentes de -1 (esta última serve para garantir que a função não retorna valores inválidos para a jogada do pc). Se este for o caso, executa-se a função `play` para jogada agora guardada na variável `jogada_pc` e retorna-se a variável `jogada_pc`.

→ Função computer_play_normal

Esta função escolhe, das possíveis jogadas que o computador pode fazer, aquela que vira mais peças, e aplica-a ao tabuleiro. Esta função é chamada na função `main` quando esta inicia o modo de jogo de um jogador na dificuldade normal.

Como argumentos da função temos as variáveis `char board[8][8]` (tabuleiro do jogo(array de caracteres)) e `char color` (cor das peças do computador).

Variáveis declaradas:

- **`[i]`**, inteiro: variável que serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: variável que serve para o funcionamento dos ciclos `for`;
- **`[bestplay]`**, tipo composto `struct jogada`: variável que irá comportar os valores da linha e da coluna da jogada que vire mais peças a ser executada pelo computador. Inicialmente toma o valor 0.

Funcionamento da função:

Inicialmente função vai atriubir 0 a ambas as componentes da variável `bestplay` (inicializando a variável com valores já válidos da posição [0][0] do tabuleiro). De seguida temos dois ciclos `for` encadeados: um incrementa `i` de 0 até 7 e o outro incrementa `j` de 0 até 7. Similarmente ao funcionamento da função acima, estes ciclos vão percorrer as linhas e as colunas do tabuleiro (array `board`) até que se encontre uma posição que seja simultâneamente um espaço vazio e que, se for aplicada à função `flanked`, tenha um maior número de peças viradas do que a jogada anteriormente guardada na variável (Sendo assim considerada uma melhor jogada, porque vira mais peças). Se ambas as condições forem cumpridas, as componentes `bestplay.line` e `bestplay.col` da variável `bestplay` vão tomar os valores de `i` e `j` (respetivamente), valores estes que representam a linha e a coluna da jogada. Quando todas as posições do tabuleiro tiverem sido testadas, a função sai do loop `for`, aplica a jogada agora guardada na variável `bestplay` à função `play`(realizando a jogada), e retorna a variável `bestplay`.

→ Função init_board

Esta função inicializa o tabuleiro na posição inicial, com as quatro peças alternadas no centro do tabuleiro e é chamada na função `main`.

Como argumento da função temos a variável `char board[8][8]`(tabuleiro do jogo(array de caracteres)).

Variáveis declaradas:

- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;

Funcionamento da função:

A função inicialmente irá substituir todas as posições do tabuleiro por “.”. Em seguida irá trocar as posições board[3][3] e board[4][4] por “O” e as posições board[3][4] e board[4][3] por “X”.

→ Função print_board

Esta função mostra o tabuleiro com a legenda das linhas e das colunas depois de cada jogada, e esta é chamada na função `main`.

Como argumento da função temos a variável `char board[8][8]`(tabuleiro do jogo(array de caracteres)).

Variáveis declaradas:

- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[linhas[8]]`**, char: array dos números que vão ser utilizados na exibição das linhas do tabuleiro.

Funcionamento da função:

A função inicialmente irá imprimir a letra de cada coluna. Em seguida, irá imprimir cada linha com um número à frente imprimindo assim o tabuleiro inteiro atual.

→ Função main

Atuando como a estrutura principal do progama, esta função usa todas as outras funções apresentadas anteriormente (exceto certas funções que vão ser usadas por outras funções) para fazer o jogo funcionar.

Como argumentos da função temos as variáveis `int argn` (número de "palavras"(strings, (separadas por espaços) presentes quando o jogo é inicializado (./othello)) e `char *args[]` (array de strings que guarda as strings anteriormente referidas).

Variáveis declaradas:

- **`[a]`**, inteiro**:** esta variável irá guardar um valor randómico entre 0 e 2, o qual vai ser usado na atribuição das cores das peças ao jogador e ao computador no modo modo de um jogador;
- **`[flk]`**, inteiro: variável que toma o valor retornado pelo flanked. Inicialmente toma o valor 0;
- **`[inicio]`**, inteiro: variável que é utilizada para a escolha do modo de jogo no menu inicial;
- **`[line]`**, inteiro: variável que guarda a linha da jogada do jogador atual;
- **`[vez]`**, inteiro: variável que toma alternadamente os valores 0 e 1. utilizada como solução à escolha randómica da cor das peças, pois, como esta é randómica, não podemos depender dela para decidir de quem é a vez. Inicialmente toma o valor 0;
- **`[count_fim]`**, inteiro: a variável vai contar o número vezes que não se acham jogadas válidas. Se alguma vez esta variável chegar a dois, o jogo acaba;
- **`[play_count]`**, inteiro: uma variável que vai ser incrementada a cada jogada efetuada. Guarda o número de jogadas que foram feitas até ao final do jogo. Inicialmente toma o valor 0;
- **`[ficheiro_ver]`**, inteiro: variável que toma valores referentes à abertura (`ficheiro_ver=1`) ou não abertura (`ficheiro_ver=0`) do ficheiro;
- **`[dific]`**, inteiro: variável que é utilizada para a escolha da dificuldade no modo de um jogador;
- **`[i]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[j]`**, inteiro: serve para o funcionamento dos ciclos `for`;
- **`[board[8][8]]`**, char: tabuleiro do jogo (array de caracteres);
- **`[plcolor]`**, char: variável que guarda o caractere que corresponde à cor das peças do jogador;
- **`[color]`**, char: esta variável vai primeiramente alternar entre 'X' e 'O' para marcar a cor das jogadas realizadas a partir de um ficheiro (a primeira vai ser jogada das pretas, a segunda das brancas, a terceira das pretas…etc). Vai também, depois da abertura e da realização das jogadas a partir do ficheiro, servir para decidir de quem é próxima jogada no modo de um jogador(contra computador), dado que agora não podemos usar nem a cor das peças do

jogador/computador(dado que estas são atribuídas randómicamente) nem a variável vez (pois, depois da leitura do ficheiro, esta perde o efeito);

- **`[comcolor]`**, char: variável que guarda o caractere que corresponde à cor das peças do computador;
- **`[col]`**, char: esta variável guarda a coluna da jogada do jogador atual;0
- **`[jogadas_char[60]]`**, tipo composto `struct jogada_char`: esta variável guarda as jogadas obtidas a partir do ficheiro de texto;
- **`[jogada_pc]`**, tipo composto `struct jogada`: variável que guarda a jogada efetuado pelo computador para posteriormente ser mostrada ao utilizador;
- **`[fp]`**, apontador do tipo FILE: serve para, no caso de haver ficheiro, ajudar na leitura do ficheiro que contém as jogadas.

Funcionamento da função:

Depois da declaração das variáveis, temos a **fase de leitura do ficheiro de texto:**

O primeiro `if` vai servir de failsafe caso o utilizador introduza demasiados argumentos na inicialização do código. Entrando no `if` seguinte(se tivermos dois argumentos na inicialização do código), vamos ter a atribuição do valor da localização do ficheiro (no caso o parametro `args`) ao pointer `fp`. Se `fp` for NULL (o ficheiro não existir) o código dá a mensagem de erro e pára a execução. Caso o ficheiro exista, continuamos para a atribuição do valor 1 à variável `ficheiro_ver` (útil mais tarde para o código saber se foi ou não aberto um ficheiro) e para a inicialização do tabuleiro com a função `init_board` com o parametro `board` (o nosso array de caracteres que serve de tabuleiro para o jogo). De seguida temos um ciclo `for` que repete até que a função `feof` retorne 0 (o que quer dizer que chegamos ao fim do ficheiro), incrementando a variável `i` (variável esta que nos vai dar o número de linhas - e, subsequentemente, o número de jogadas retiradas do ficheiro, e também nos vai dar valores consecutivos para usarmos como índice do array `jogadas_char`, no qual vamos armazenar as nossas jogadas em formato caractere), e por cada repetição do ciclo, temos a atribuição do primeiro e segundo caracteres (obtido a partir das primeiras instâncias da função `fgec`, que lê o ficheiro caractere a caractere) às componentes `line` e `col` do array composto `jogadas_char` no índice `i`, e de seguida temos duas instâncias da função `fgetc` para avançarmos o cursor de modo a podermos ler a próxima linha (no caso, a próxima jogada).

Depois de chegar ao fim do ficheiro, (sair do ciclo `for`), temos outro ciclo `for` que repete enquanto `j` for menor que `i` e incrementa `j` a cada repetição do ciclo, dentro do qual temos um `if` que verifica cada uma das jogadas quanto à sua validade no jogo (usando a função `flanked` - se o valor retornado da `flanked` para aquela jogada for 0, a jogada é inválida), e caso a jogada seja a invalida o código mostra uma mensagem de erro e para a execução. Dentro do ciclo `for` temos também um pequeno sistema que faz com que o valor da variável `color` alterne entre 'X' e 'O' (sendo inicialmente X, porque a primeira jogada é sempre das pretas), de modo a fazermos com que as

jogadas lidas no ficheiro sejam aplicadas ao tabuleiro com a cor alternada, como mandam as regras do jogo.

Continuando pela função `main`, temos a atribuição da seed à função `srand`, para possibilitar a obtenção de números randómicos. De seguida temos uma verificação do valor da variável `ficheiro_ver` - se o valor for 0, é mostrada a mensagem de início normal e é chamada a função `init_board` para inicializar o tabuleiro na posição inicial. Se o valor for 1, a mensagem de texto é diferente (dando ao utilizador a confirmação do sucesso da leitura das jogadas no ficheiro introduzido) e a função `init_board` já não é chamada, visto que o tabuleiro já foi inicializado na fase da leitura do ficheiro (e caso o inicializassemos agora, iriamos desfazer todas as jogadas lidas a partir do ficheiro). Depois da mensagem temos o menu inicial do jogo, onde, com ajuda das funções `scanf`, vamos escolher o modo de jogo (atribuindo valor à variável `inicio`) (também temos um pequeno failsafe: se o utilizador tentar escolher uma opção que não existe, o programa continua a pedir uma opção válida até que o utilizador introduza uma). Dependendo da escolha do utilizador, vamos ter, agora na fase de jogo, o **modo de um jogador(caso `inicio` = 2)** ou o **modo de dois jogadores(caso `inicio` = 1);**

Modo de dois jogadores: vai ser exibido o tabuleiro atual ao utilizador, e de seguida entramos num ciclo `for` que repete até que seja maior que 60, pois só existem no máximo 60 possíveis jogadas no tabuleiro de othello. Dentro do ciclo vamos ter a verificação da vez (dada pela variável vez, que no início é zero (pois, num jogo normal, seriam as pretas a começar), mas com a leitura do ficheiro podemos ficar na situação em que a próxima jogada é das peças brancas). Se a vez for das pretas,primeiramente temos a verificação da existência de jogadas válidas para as peças pretas (pois se não houver jogadas válidas para um jogador, é passada a vez ao próximo jogador.), com a ajuda da função `verify` - se `verify` retornar 0 não há jogadas válidas e é mostrada uma mensagem a indicar isso ao utilizador, incrementando a variável `count_fim`.

De seguida, é dito ao utilizador que é a vez das pretas, e entramos num ciclo `while`, que vai repetir enquanto a variável `flk` for 0 (variável esta inicializada com o valor 0 para possibilitar a entrada garantida no ciclo `while`), e dentro do ciclo vamos ter uma mensagem a pedir a jogada (um número para a linha e um caractere para a coluna, atribuídas às variáveis `line` e `col` respectivamente) e vamos fazer com que a variável `flk` tome o valor de retorno da função `flanked` para a jogada (das peças pretas) que foi armazenada nas variáveis `line` `col`. Se `flk` continuar a ser 0 porque a função retornou 0 (o que representa uma jogada inválida), é apresentada uma mensagem a indicar que a jogada é inválida e o ciclo repete. Caso a jogada seja válida, a função sai do ciclo e aplica a jogada ao tabuleiro, usando a função `play` com os valores de `line` e `col`.

De seguida, incrementa-se a variável `play_count`, mostra-se o tabuleiro ao utilizador e, por fim, passa a vez ao próximo jogador, mudando a variável vez para 1. O código da parte da jogada das peças brancas é exatamente igual ao das peças pretas, com exceção de certos argumentos das funções, onde, em vez de se mandar como argumento 'X' (cor das pretas), manda-se 'O'(cor das brancas),

Se alguma vez a variável `count_fim` for igual 2 (o que representa o caso de não haver jogadas válidas para nenhum dos jogadores) ou o valor de retorno da função `verify_full` for igual a 1 (o que representa o caso do tabuleiro estar cheio), é chamada a função `finish`, que vai acabar o jogo e parar a execução do código.

Modo de dois jogadores:

Se a variável `inicio` tiver tomado o valor 1 mais atrás no código, vamos entrar no modo de um jogador. Depois de declaradas as variáveis, o código pede ao utilizador que escolha a dificuldade do jogo (valor armazenado variável `dific`). Se for introduzido um valor que não seja nem 1 nem 2, o código entra no loop `while` seguinte, que repete a mensagem "Escolha invalida" , até que uma escolha válida seja introduzida. Continuando, temos a parte da escolha das peças começando a atribuição de um valor randómico (entre 0 e 1) à variável `ran`, seguindo com uma verificação da mesma variável: se `a` variável for 0, o jogador fica com as peças brancas e o computador com as pretas (`plcolor` passa a guardar 'O', e com `comcolor` passa a guardar 'X') e é mostrada uma mensagem de texto correspondente; caso `ran` seja 1, o contrário acontece: o jogador fica com as peças pretas e o computador com as brancas (`plcolor` passa a guardar 'X', e com `comcolor` passa a guardar 'O').

Continuando, temos condições `if` encadeadas que vão servir o propósito de verificar o valor das variáveis `ficheiro_ver` e `color` e, conforme esse valor, atribuir um valor à variável `vez` - como falámos antes, é possível que, depois da leitura do ficheiro, a próxima jogada tenha de ser feita pelas brancas, pelo que necessitamos de mudar o valor da variável `vez` conforme.

De seguida temos uma verificação do valor da variável `ficheiro_ver` em que, no caso de não se ter lido nenhum ficheiro (`ficherio_ver` = 0), o tabuleiro é inicializado com a posição inicial, sendo este depois mostrado ao utilizador (função `print_board`). Entramos agora no loop das jogadas, feito por um ciclo `while` que repete enquanto o tabuleiro não estiver cheio (enquanto o retorno da variável `verify_full` for diferente de 1), e dentro do ciclo temos, primeiramente (caso a variável `vez` seja 1) a vez do jogador: Depois de verificada a existência de jogadas válidas para a cor do jogador (de modo similar ao anteriormente citado), o código é bastante similar ao código presente na parte do modo de dois jogadores, pelo que não carece de explicação.

Depois de passada a vez ao computador, verifica-se a existência de jogadas válidas para a cor do computador, e depois verifica-se o modo da dificuldade: se a dificuldade for fácil (`dific` = 1), executa-se a função `computer_play_easy`, guardando o retorno desta na variável `jogada_pc`, para depois ser mostrada ao utilizador; caso a dificuldade esteja no normal, o mesmo acontece mas é executa a função `computer_play_normal`

em vez da `computer_play_easy`. De seguida passa-se a vez para o jogador mostra-se ao utilizador o tabuleiro atual já com as jogada do computador aplicada (usando a função `print_board`) e também a jogada do computador (depois dos valores convertidos para os números apresentados no tabuleiro para a linhas, e para as colunas a letra correspondente dada pela função `num_to_let`), incrementando por fim a variável `play_count`.

Caso, na verificação da existência de jogadas válidas para o computador, não se tenham encontrado jogadas, é mostrado uma mensagem referente e de seguida, o código verifica a existência de jogadas válidas para a cor do jogador, mostrando a mensagem “É a sua vez”. Similarmente ao que acontece anteriormente no código, se a variável `count_fim` chegar a 2 (o que representa não haver jogadas válidas para nenhuma das cores), a função `finish` é executada e o jogo acaba, e caso o tabuleiro esteja cheio, o jogo acaba também.