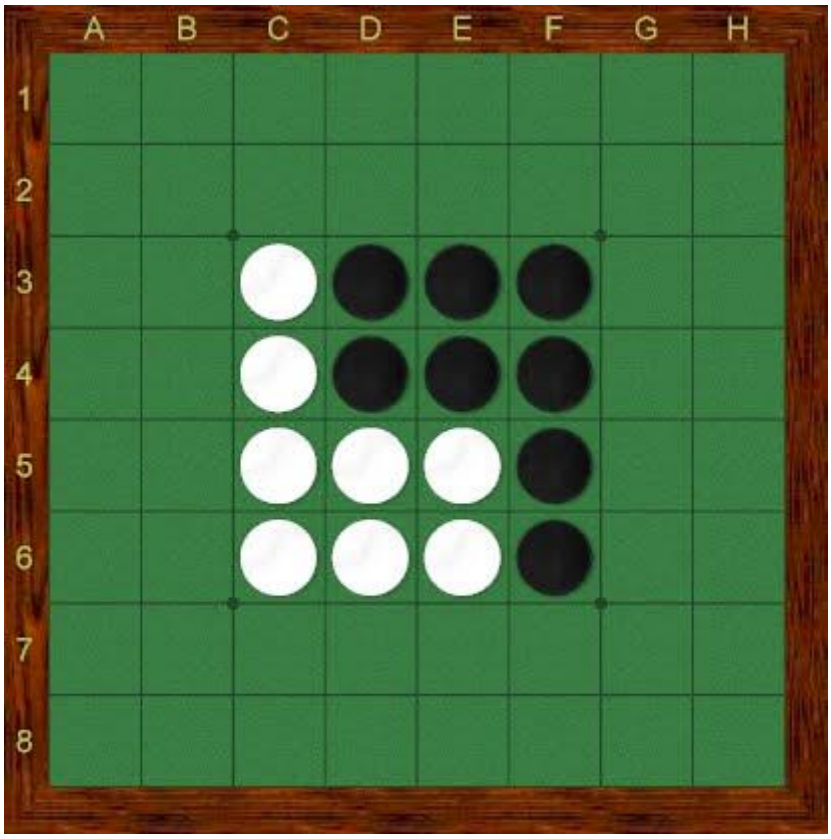
Programação I

Trabalho Prático 2021/2022

→ Othello ←



Trabalho realizado:

-António Nanita 48407

-Rodrigo Alves 48681

Introdução

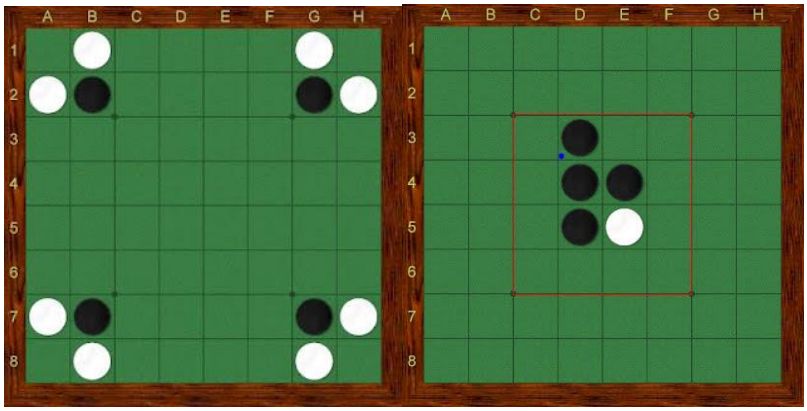
No âmbito da cadeira de Programação I, foi pedido a criação de um jogo de tabuleiro “Othello” também conhecido como “Reversi”, numa versão virtual.

Este jogo é para dois jogadores, e é jogado num tabuleiro tradicional de 8x8 onde inicialmente são colocadas 4 peças ao centro do tabuleiro, sendo duas de cada cor, isto é, uma das peças apresenta uma face clara e outra escura, que são colocadas em diagonal. Depois de estarem criadas as condições de se iniciar este jogo, a peça preta faz sempre o primeiro movimento e neste caso o jogador e o computador jogam alternadamente. O jogo só acaba quando não há mais jogadas possíveis tanto para o jogador como também para o computador.

Neste jogo de tabuleiro “Othello”, tem como objetivo de virar o maior número de peças do adversário para que consequentemente obter um maior número de peças presentes no tabuleiro no final do jogo, pois vence quem tiver mais peças no final da partida.

Neste trabalho o jogo pode ser jogado de forma interativa, entre um jogador e o computador, ou de forma não interativa, que consiste a partir da leitura de um ficheiro.

No jogo interativo foi desenvolvido um computador inteligente, que dependente do estado do jogo, o mesmo vai escolher a melhor jogada. Para dar “inteligência” ao computador foram lhe dados diferentes truques muito utilizados neste tipo de jogo de tabuleiro. As técnicas utilizadas são a do quadrado imaginário que consiste em jogar todas as jogadas possíveis ao centro do tabuleiro nas primeiras jogadas, a explicação para tal focasse em concentrar as peças da mesma cor, porque caso dispersarmos as peças temos menos probabilidade de ganhar o jogo, o outro truque utilizado é tentar ao máximo o oponente não ficar com a posse dos cantos do tabuleiro, pois são importantes pelo facto de que nessa posição do tabuleiro a peça não ser possível de ser virada e também por ser um ponto estratégico para virar numa só jogada um elevado número de peças viradas em diferentes direções.



Descrição dos diferentes modos de jogo

- Jogo interativo

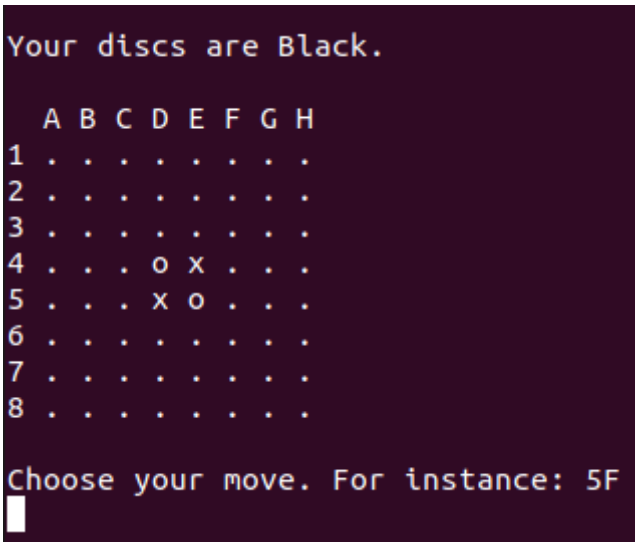
Antes do jogo ser gerado são primeiro criadas condições para guardar as jogadas realizadas no jogo num ficheiro, isto é, cada vez que é compilado o programa, o ficheiro "jogadas.txt" terá só as jogadas realizadas de uma compilação. Deste modo este ficheiro tem todas as jogadas de um único jogo realizado. A função **clean_last_game** é a função responsável por atualizar o conteúdo do ficheiro anteriormente referido. O nosso grupo decidiu que o conteúdo do ficheiro seria da seguinte forma, a primeira linha do ficheiro teria a seguinte mensagem "Jogadas efetuadas:" e nas restantes linhas abaixo desta são introduzidas cada jogada realizada no jogo. Caso um dos jogadores não ter jogadas possíveis num determinado momento, linha correspondente à suposta jogada é deixada em branco, e a jogada realizada pelo adversário na vez do jogador é escrita na linha abaixo. Assim o conteúdo do ficheiro seria só este, simples e preparado para receber a nova ordem de jogadas efetuadas durante o novo jogo.

Em primeiro lugar damos a conhecer o estado do nosso tabuleiro, para isso utilizamos a função **init_board** que é responsável pela criação do tabuleiro inicial, isto é, a introdução das quatro peças, duas de cada cor, no centro do tabuleiro (posições: 4D, 4E, 5D e 5E) e as restantes posições encontram se vazias, para definirmos todos os diferentes estado de ocupação é lhes atribuído um número inteiro que representa um dos diferentes estado: "-1" indica que a posição se encontra vazia, "0" indica que nessa posição encontra-se uma peça branca e o "1" indica que nessa posição encontra-se uma peça preta. O grupo decidiu que ao distinguir os diferentes estados do tabuleiro a partir de diferentes números inteiros seria uma vantagem pois na nossa opinião é mais fácil comparar dois inteiros do que duas strings como também é mais "leve" guardar um número em cada posição do que uma string.

Depois de o tabuleiro ser criado, é necessário atribuir aleatoriamente uma cor ao jogador, a função responsável pela a atribuição da cor é a função **player_color**. No corpo desta função é gerado um número aleatório entre 0 e 1, e depois de se conhecer o número este é guardado na estrutura do tipo color que armazena a cor do jogador como também a cor do computador.

De seguida a função **your_color** apresenta no terminal a cor da peça que foi atribuída ao jogador, esta informação advém da estrutura do tipo color que foi retornada pela função referida no paragrafo anterior.

Seguidamente, é mostrado o tabuleiro inicial, que é dado pela função **print_board**. Esta função está encarregada de mostrar tanto a numeração das linhas do tabuleiro como também das colunas que neste caso são representadas por letras (“A” a “H”). Além disso a **print_board** também apresenta cada posição do tabuleiro, que depende do estado da de cada posição, caso numa certa posição conter o inteiro “-1” a função mostra no terminal “.”, caso for “1” mostra “x” e se for “0” mostra “o”. Por decisão do grupo a numeração das linhas como também das colunas não devia de fazer parte da constituição do array bidimensional pois iriamos ocupar mais memória e seria uma desvantagem estar a misturar os representantes das linhas e das colunas com as posições do tabuleiro.



Ao ser mostrado o estado inicial do tabuleiro no terminal, é pedido a primeira jogada, que será sempre do jogador que possuir a peça preta. Depois de se realizar a jogada da peça preta irá ser pedido a jogada da peça branca, ocorrendo este processo de forma alternada. Para indicar a qual é a vez da peça a jogar foi criada a função **cor_a_jogar** que retorna o inteiro (0 ou 1), representantes da peça branca e preta respetivamente, que indica qual a cor que vai jogar. Este processo realiza-se enquanto não houver mais jogadas possíveis para ambos os jogadores, por isso foi criada a função **end_game** que tem como responsabilidade de averiguar após cada jogada efetuada de cada peça, se ambos os jogadores possuem jogadas possíveis. Caso não haja mais jogadas possíveis para ambos, o programa termina.

Supondo que o computador fique com as peças pretas e o jogador com as peças brancas. O computador será o primeiro a jogar, e a jogada declarada e efetuada é resultado de diferentes etapas.

Em primeiro lugar, para se descobrir a melhor jogada dado um certo momento de jogo é necessário revelar ao computador todas as jogadas possíveis. Nesse sentido, foi criado a função **jogadas_possiveis** que tem como objetivo guardar todas as jogadas que respeitam as regras do jogo. Para avaliarmos cada posição utilizamos as funções **jogadaH**, **jogadaV** e **jogadaD**, estas funções averiguam se uma dada posição do tabuleiro pode ser ou não uma jogada possível para uma determinada cor, caso for uma posição válida a mesma é guardada numa estrutura do tipo jogadaP que é onde são armazenadas as informações de todas as jogadas possíveis, isto é o número da linha que consta na numeração das linhas do tabuleiro e o caracter da coluna.

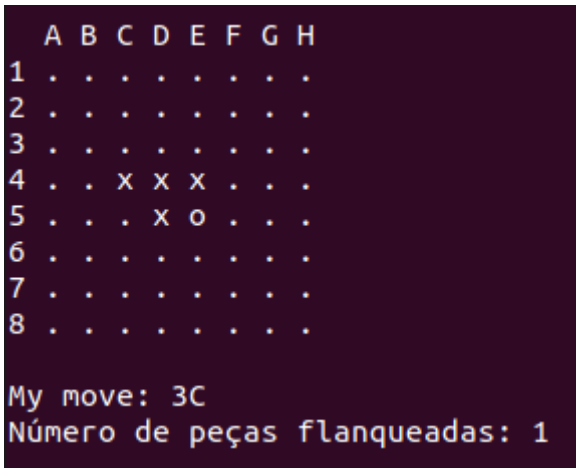
Ao serem conhecidas todas as jogadas que respeitam as regras deste jogo, é avaliado entre todas qual a melhor jogada. Para ajudar a detetar a melhor jogada foi criada a função **best_play**. Como diz o nome, vamos descobrir a melhor jogada para o computador, a mais inteligente, para isso foram utilizadas as técnicas referidas na introdução.

Primeiro é analisado se entre as jogadas possíveis existe/m alguma jogada que seja dentro do quadrado imaginário, caso houvesse dentro das jogadas possíveis uma que respeitava a condição anterior era escolhida aquela que iria virar o maior número de peças, esta informação é dada pela função **flanked**. A função **flanked** tem como responsabilidade de devolver o número de peças que são viradas caso efetuássemos uma certa jogada, este processo consiste na comparação entre os estados de cada posição do tabuleiro antes de se efetuar a jogada e o tabuleiro depois de se efetuar a jogada.

Se não houver nenhuma jogada possível que esteja dentro do quadrado imaginário, é testado se alguma jogada representava um dos quatro cantos do tabuleiro. No caso de haver, é escolhido o canto que iria virar um maior número de peças, esta informação é dada pela função **flanked**. Na eventualidade de não existir nenhuma jogada possível nas condições anteriores, então é realizado a escolha da jogada possível que iria virar um maior número de peças.

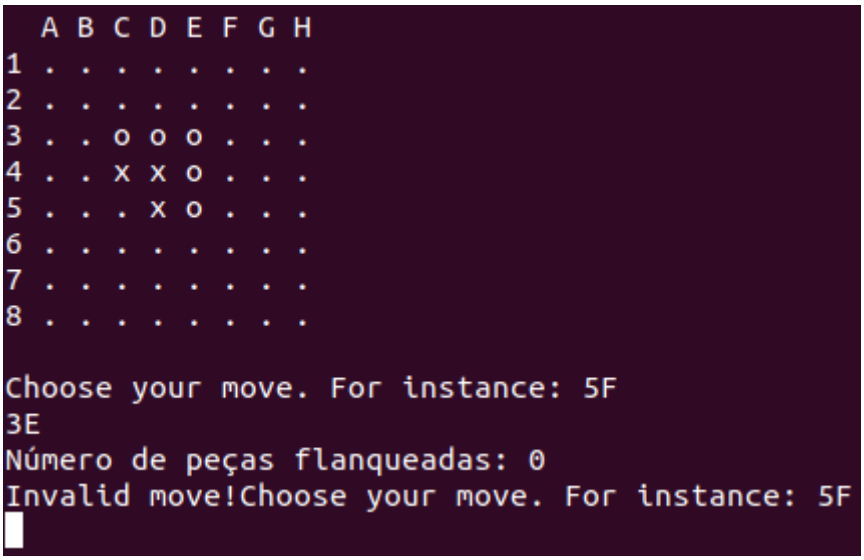
A melhor jogada é guardada na estrutura do tipo best que guarda a linha da jogada e o caracter da jogada. Depois de definida a melhor jogada para o computador, é chamada a função **play_cpu** para ser jogada a melhor jogada (utilizamos a função **play** para efetuar a jogada no tabuleiro) e de seguida mostra a jogada realizada no terminal.

A seguir de ser mostrado o movimento do computador, é apresentado no terminal o número de peças que foram viradas (foi utilizado a função **flanked**) e é feito a escrita da jogada efetuada no ficheiro “jogadas.txt”, este processo e realizado pela função **safe_file**, que acrescenta ao ficheiro referido a jogada efetuada. Depois de terminar a escrita no ficheiro é mostrado o novo estado do tabuleiro no terminal.



Depois de todas as etapas referidas anteriormente é a vez do jogador realizar a sua jogada, por isso o programa pede ao utilizador que escreva no terminal a jogada que pretende efetuar. Esta jogada é guardada numa string, sendo que a primeira posição da string (índice 0), contem o número da linha, mas como esta guardado num caracter o inteiro do mesmo será o seu código ASCII, por isso antes de passado o número da linha para a variável “l” é chamada a função **char2int** que converte o código ASCII do número ao seu inteiro, assim este mesmo valor já pode ser guardado pela variável “l” e posteriormente utilizado nas funções principais como por exemplo na função **play**. Esta função vai primeiro avaliar se a jogada respeita as regras do jogo, tais como se a posição da jogada esta vazia e se existe alguma peça por virar numa certa direção por isso o grupo decidiu criar 6 funções auxiliares para realizar todas as funções que são destinadas a função play, 3 delas averiguam se há condições para aceitar a jogada inserida pelo utilizador, **jogadaH**, **jogadaV** e **jogadaD**, cada uma delas avalia se existe condições para três direções possíveis, caso exista uma delas pelo menos que averigue que a jogada é possível a função play chama as outras 3 funções (**play_jogadaH**, **play_jogadaV** e **play_jogadaD**), que são responsáveis pela colocação da peça(da jogada inserida no terminal) e virar as peças possíveis. Este último passo necessita de certas variáveis para haver uma correta atualização do tabuleiro (explicado na parte das funções).

Caso a função play aceite a jogada, a mesma é efetuada, caso contrário é mostrado no terminal “Invalid Move” e pede novamente outra jogada até uma ser aceite pelo programa. Depois de acontecer esta aceitação é mostrado no terminal o número de peças viradas, que é dada pela função **flanked**, que já foi anteriormente explicada.



Por último é introduzida a jogada no ficheiro já mencionado antes, esta operação e realizada pela função **safe_file**. No entanto, caso não haja jogadas possíveis para uma certa cor, no momento da sua jogada, é dada a vez ao adversário, e posteriormente o sistema de jogo alternado é retomada, se não acontecer novamente a impossibilidade de jogar por parte de um dos jogadores.

Quando o jogo é terminado é chamada a função **score** que indica o número de peças brancas e pretas que estão no tabuleiro e se o jogador venceu, empatou ou perdeu contra o computador.

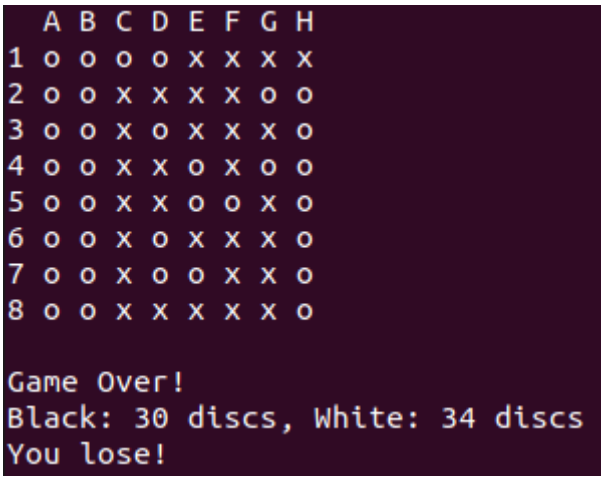
- Jogo não interativo

Este modo de jogo depende de um ficheiro mencionado no terminal, pois contem todas as jogadas efetuadas num determinado jogo. Assim o grupo decidiu que seria necessário criar uma função que guardasse todas as jogadas mencionadas no ficheiro, essa função é **game_inFile**.

Esta função vai abrir o ficheiro, em modo de leitura, e vai ler linha a linha do ficheiro, até chegar ao final do mesmo. Nesta função é só retirada informação que está de acordo com os dados procurados, isto é, em cada linha que é lida a função verifica se o primeiro caracter é um representante de um número com um código que seja maior ou igual a “1” e menor ou igual a “8”, e se o segundo caracter dessa linha seja um caracter entre “A” e “H”. Caso não respeite estas condições essa linha, não é passada nenhuma informação dessa mesma linha, por isso assim só é guardado na estrutura, que guarda as jogadas do ficheiro, jogadas que são possíveis de se jogar neste tabuleiro, mas que podem não respeitar as regras do jogo.

Depois de ter terminado a leitura do ficheiro é escolhida aleatoriamente a cor do jogador, por isso utilizamos as funções **player_color** e **your_color** (já foram faladas a sua função no jogo interativo). De seguida é definido o tabuleiro inicial (função **init_board**).

Após todos os passos descritos anteriormente é efetuado a introdução das jogadas uma a uma que estão guardadas numa estrutura como o nome de jogadaFile. A colocação da jogada no tabuleiro é feita pela função play, e caso esta ao ler uma jogada que não respeitasse as regras do jogo o programa termina automaticamente e apresenta no terminal um alerta a indicar qual o número da jogada que não respeita as regras do jogo. Enquanto houver jogadas aceites pela função **play** o programa procede mostrando o movimento que o jogador ou o computador efetuou e o número de peças viradas, até à última jogada. Este processo decorre até ser feito a introdução da última jogada que indica o fim do jogo e posteriormente é mostrado os dados acerca do tabuleiro indicando o vencedor, dado pela função score.



- Funções

Init_board - Esta função tem como parâmetro um array bidimensional definido como “board” que representa o tabuleiro. Com ajuda de dois ciclos for e com as variáveis “lines” e “coluns” que têm como valor inicial 0, vão percorrer todas as posições do tabuleiro até atingirem os valores max_lines (valor=8) e max_col (valor=8). Quando as variáveis “lines” e “coluns” representarem as posições {3,3} ou {4,4} da “board” vão ser colocadas nas mesmas, o inteiro “0” que representa as peças brancas. Quando as variáveis “lines” e “coluns” representarem as posições {3,4} ou {4,3} da “board” vão ser colocadas nas mesmas, o inteiro “1” que representa as peças pretas. As restantes posições do tabuleiro que não foram mencionadas são representadas pelo inteiro “-1” que indica que a posição está vazia.

Print_board - Esta função tem como parâmetro um array bidimensional definido como “board”. No primeiro for da função, a variável inteira “letra” vai ser inicializada com o valor 65, pois este valor representa a letra “A” em código ASCII, e esta vai ser incrementada até ser igual a 72, que é o código ASCII da letra “H”, enquanto a variável letra estiver dentro do intervalo definido, a variável vai ser convertida para um caracter e de logo após é escrita no terminal, assim obtemos todas as letras que representam as colunas do tabuleiro. No segundo ciclo for inicializamos a variável “nline” com o valor 1, que é o número representativo da primeira linha, esta variável vai ser incrementada até ser menor do que 9. Enquanto a variável “nline” estiver dentro do intervalo mencionado a cima, vai ser mostrado o seu valor no terminal, mas ao mesmo tempo é mostrado o estado da linha “l = nline - 1” do tabuleiro, para percorrer todas as posições dessa linha, isto é, todas as colunas foi utilizado um ciclo for inicializado por ”c” que corresponde à coluna no tabuleiro, e se caso numa das posições for encontrado um número 1 é mostrado no terminal “x”, caso for o número 0 é mostrado “o” e caso for igual a -1 é mostrado “.” .

Play - Esta função tem como parâmetros o array bidimensional que é representado pela variável board, as variáveis “line” e “col” que são a linha e a coluna respetivamente da jogada e a variável “color” que é o inteiro que representa a cor da peça que está a ser jogada. Esta função primeiro vai averiguar se a posição da jogada estava vazia, por isso convertemos tanto a linha da jogada para a respetiva linha no tabuleiro, que é representada pela variável “l”, como também convertemos a letra da jogada para a respetiva coluna do tabuleiro, que é representada pela variável “c”, caso na posição “board[l][c]” tivesse vazia a função vai constatar se a jogada introduzida respeita as regras do jogo (utilizando as funções **jogadaH**, **jogadaV** e **jogadaD**), caso um dos value_f de uma das estruturas retornadas pelas funções anteriores for igual a 1, a função play iria proceder à atualização do estado do tabuleiro procedendo à colocação da jogada. Se isto não se verificar, a jogada é dada como inválida, retornando o valor 0.

Flanked - Esta função tem como parâmetros dois arrays bidimensionais, o primeiro representado pela variável board que representa o tabuleiro depois de ter sido efetuada a jogada e o segundo pela variável board_s que representa o tabuleiro antes de ter sido efetuada a jogada, a função tem como parâmetros também as variáveis “line” e “col” que são a linha e a coluna respetivamente da jogada e por último a variável color que é o inteiro que representa a cor que está a ser jogada. Para comparar a mesma posição dos dois tabuleiros é utilizado o uso de dois ciclos for, um que vai percorrer as linhas do tabuleiro e outro as colunas com o uso das variáveis “l” e “c”. Caso numa certa posição haja uma diferença de estados, isto é, houve a mudança do inteiro de um tabuleiro para o outro, assim adicionamos mais um à variável que possui o número de peças

viradas contagem “count”, mas caso a posição no tabuleiro seja igual a posição da jogada introduzida a variável vai ser decrementada, pois nessa posição não houve a viragem de uma peça, mas sim a colocação da mesma no tabuleiro.

Count_flips_dir - Esta função tem como parâmetros dois arrays bidimensionais, o primeiro representado pela variável board_p que representa o tabuleiro depois de ter sido efetuada a jogada e o segundo pela variável board_s que representa o tabuleiro antes de ter sido efetuada a jogada, a função tem como parâmetros também as variáveis “line” e “col” que são a linha e a coluna respetivamente da jogada, de seguida temos duas variáveis que representam a direção imposta, "delta_line" e “delta_col” e por último a variável color que representa a cor que está a ser jogada. No corpo desta função existem várias condições todas dependentes das variáveis que indicam a direção a ser tomada para a contagem do número de peças viradas que são viradas. Se o delta_line for igualado a 1 e delta_col for igualado a 1 estes representam a direção baixo-direita, com isto podemos perceber que o nosso caminho que iremos ter que percorrer será feito a partir da incrementação das linhas e das colunas em simultâneo. Se as duas variáveis anteriores tiverem igualadas a zero, a direção que vai ser tomada irá ser o contrário da mencionada anteriormente, logo caminho que iremos ter que percorrer será feito a partir da decrementação das linhas e das colunas em simultâneo. As restantes direções

jogadaH, jogadaV, jogadaD - Para a função play ter um melhor funcionamento criamos três funções auxiliares, **jogadaH**(jogada horizontal),**jogadaV**(jogada vertical) e **jogadaD**(jogada diagonal). O objetivo das mesmas é averiguar se existem condições para serem efetuadas as jogadas nas respetivas direções para a peça com a cor que esta a ser jogada. Todas elas têm os mesmos parâmetros que são um array bidimensional definido pela variável “board”, as variáveis “line” e “col” que representam a linha e a coluna da jogada e a variável “color” que é o inteiro que mostra a cor que está a ser jogada. Em primeiro lugar as funções vão procurar se existe alguma peça da cor que esta a ser jogada numa certa direção. Caso nessa direção existir uma peça da mesma cor irá ser guardada a posição da peça que foi encontrada, esta informação é guardada nas seguintes variáveis “col_right” e ”col_left”,”line_up” e ”line_down”, ”diagonal_down_line” e “diagonal_down_col”,”diagonal_up_line” e “diagonal_up_col, ”diagonal_down_line2” e “diagonal_down_col2”, ”diagonal_up_line2” e “diagonal_up_col2”(dependente da função a ser utilizada,estas variáveis guardam a linha ou a coluna da peça encontrada) .Como foi encontrado uma peça ira ser feito uma análise entre a posição mais próxima da jogada (dependente da direção) e da peça que foi encontrada, se entre elas só constarem peças da equipa adversaria a jogada é dada como valida, logo o “value_f” da função na qual a jogada respeitava as jogadas numa certa direção vai ser igual a 1.

Play_horizontal,Play_vertical,Play_diagonal- Estas funções tem como parâmetros um array bidimensional definido pela variável “board”, as variáveis “line” e “col” que representam a linha e a coluna da jogada , a variável “color” que é o inteiro que representa a cor que esta a ser jogada , sendo que divergem nas structs mencionadas nos parâmetros que recebem, por exemplo a função Play_horizontal possui a struct do tipo “valoresH”. Estas estruturas como possuem os valores das variáveis obtidas pelas funções jogadaH,jogadaV e jogadaD é possível identificar qual ou quais as direçoes que possam permitir virar as peças do adversário. Com isto, caso um dos “value_f” de uma das estruturas ser igual a 1 e caso um dos “value_l”, ”value_r”,”value_d” ,”value_u” ou um dos restantes values da estrutura “valoresD” for igual a 1é efetuado a atualização do tabuleiro introduzindo as peças da cor que esta a ser jogada ao longo das direçoes possíveis que consiga virar as peças do adversário.

- # Conclusão

Após ter sido feito um trabalho de pesquisa acerca deste jogo de tabuleiro, desde as suas regras e alguns truques recolhidos a partir de certas fontes, ter sido planeado como ia ser resolvido cada problema, o grupo confrontou-se com diferentes obstáculos, isto é, problemas em algumas implementações de certas funções, mas que foram ultrapassadas com sucesso, pois foram-nos dadas informações necessárias durante o decorrer do semestre que nos ajudou na implementação deste trabalho. Este trabalho foi importante pois ajudou-nos a perceber a importância do trabalho em equipa e fundamentalmente a prepararmo-nos para um trabalho futuro neste ramo.

Todos os objetivos do trabalho foram alcançados pois foram feitas todas as funções pedidas bem como foram apresentadas outras capacidades adicionais, tal como a jogada inteligente por parte do computador como também o cálculo de todas as jogadas possíveis para uma certa cor num dado num momento do jogo.