

Universidade do Minho

Escola de Engenharia

Departamento Informática

**Mestrado Integrado em Engenharia
Informática**

Laboratórios de Informática II

Projeto LI2

Reversi

	1	2	3	4	5	6	7	8
1	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-
4	-	-	-	X	O	-	-	-
5	-	-	-	O	X	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-

Grupo 9 (PL2)

Luís Araújo (a86772)

Gonçalo Rodrigues (a90439)

Índice

1 - Resumo	2
2 - Regras Do Jogo O Reversi.....	3
3 – Interpretador (Interpretador.c)	3
3.1 – Novo Jogo Manual (NovoJogo.c)	3
3.2 – Novo Jogo Automático (NovoJogo.c)	4
3.2.1 – Bot Fácil (JogadasBotFacil.c)	4
3.2.2 – Bot Médio (JogadasBotMedio.c)	5
3.2.3 – Bot Difícil (JogadasBotDificil.c)	6
3.3 – Modo Competitivo (FuncoesAuxiliares.c)	7
3.4 – Salvar Jogada (Salvar.c)	9
3.5 – Descarregar Ficheiro (Descarregar).....	10
3.6 – Remover Ficheiro (ControloDeFicheiros.c).....	10
3.7 – Jogar (ControloJogadas.c).....	10
3.8 – Trocar Peças (TrocCalcons.c)	10
3.9 – Jogadas Validas (MatrizAjuda.c)	11
3.10 – Sugestão de Jogada	12
3.11 – Undo (Stack.c)	12
3.12 – Ficheiros Guardados (ControloDeFicheiros.c)	13
3.13 – Ranking (FuncoesAuxiliares.c)	13
3.14 – Sair.....	13
4 – Conclusão	13

1 - Resumo

Este documento contém a explicação do raciocínio que seguimos para elaborar este projeto “O Jogo do Reversi” e ainda em detalhe mais aprofundado as diferentes dificuldades dos bots existentes no jogo.

Começamos por enunciarmos brevemente as regras e em que consiste o próprio jogo para um melhor esclarecimento do mesmo. De seguida referimos o processo de desenvolvimento do Jogo, as etapas de realização, as dificuldades com que nos deparamos e satisfação de quando as superamos.

Para concluirmos este projeto teremos também uma breve conclusão/analise sobre o projeto e as diferentes estratégias de jogo.

2 - Regras Do Jogo O Reversi

O Reversi é um jogo de tabuleiro de estratégia para dois jogadores no formato de um contra um, em que um jogador joga com a peça X e o outro com a peça O. Este é jogado num tabuleiro de dimensão 8x8 (64 posições) em que o jogo é iniciado sempre da mesma forma, com a peça X nas posições (4,5) e (5,4) e a peça O nas posições (4,4) e (5,5).

Um jogador tem como objetivo obter ou todas as peças do jogador adversário ou conseguir o maior número de peças no tabuleiro até final do jogo. Para se realizar uma jogada é obrigatório virar, pelo menos, uma peça do jogador oponente, a qual se sucede quando temos uma ou mais peças do jogador adversário entre peças do mesmo, em quaisquer direções (vertical, horizontal, diagonais). O jogo é terminado automaticamente caso um dos jogadores fica sem peças no tabuleiro, ou caso nenhum dos jogadores tenham jogadas onde sejam possível virar peças do adversário (jogadas válidas/possíveis).

3 – Interpretador (Interpretador.c)

Com o interpretador que nos foi sugerido pelos docentes da cadeira e adicionando também algumas novas opções da nossa autoria, optamos por criar uma junção de menu com interpretador, o qual revela as opções possíveis a ser efetuadas pelo utilizador (Figura 1 - Interpretador).

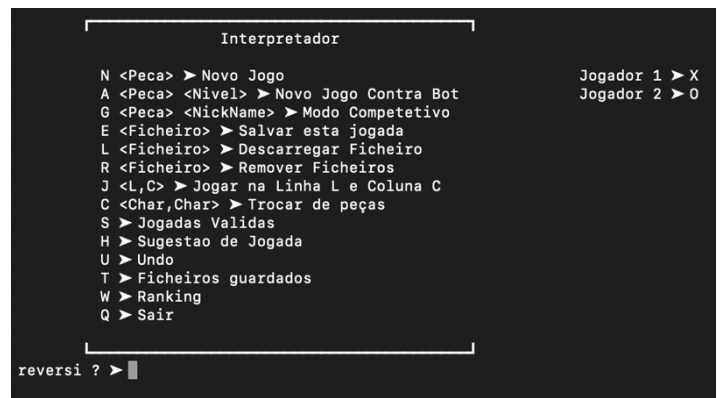
A screenshot of a terminal window titled "Interpretador". It displays a menu of options for the Reversi game, each preceded by a letter and a right-pointing arrow. The options are: N <Peca> > Novo Jogo, A <Peca> <Nivel> > Novo Jogo Contra Bot, G <Peca> <NickName> > Modo Competetivo, E <Ficheiro> > Salvar esta jogada, L <Ficheiro> > Descarregar Ficheiro, R <Ficheiro> > Remover Ficheiros, J <L,C> > Jogar na Linha L e Coluna C, C <Char,Char> > Trocar de peças, S > Jogadas Validas, H > Sugestao de Jogada, U > Undo, T > Ficheiros guardados, W > Ranking, and Q > Sair. On the right side of the terminal, it says "Jogador 1 > X" and "Jogador 2 > O". At the bottom left, the prompt "reversi ? >" is visible with a cursor.

Figura 1 - Interpretador

3.1 – Novo Jogo Manual (NovoJogo.c)

Nesta hipótese de escolha necessitamos de 2 parâmetros, um primeiro para destingir a escolha de comando, neste caso o caracter ‘N’, e um segundo para escolher o jogador que começará a jogar. Como vemos à direita da Figura 1, temos 2 jogadores aos quais podemos escolher (Jogador 1 – X; Jogador 2 – O).

Este comando irá alterar o estado do jogo para um estado inicial, modificando várias opções do estado, tais como:

- A matriz grelha passará uma matriz inicial;
- O modo do jogo será agora ‘M’;
- Armazenar o valor do primeiro jogador;

- Iremos limpar a stack, existente ou não;
- Colocar a dificuldade a 0, a qual corresponde o modo manual;
- Guardar a primeira matriz (inicial) na stack;

3.2 – Novo Jogo Automático (NovoJogo.c)

Em comparação ao Novo Jogo Manual este comando não defere em muito, alterando apenas os parâmetros necessários para a validação do mesmo, e a dificuldade dos diferentes bots.

3.2.1 – Bot Fácil (JogadasBotFacil.c)

Como nunca tínhamos feito um bot com linguagem c, optamos por começar pelo de dificuldade mais fácil.

Para esta dificuldade tivemos então a ideia de ir a “MatrizAjuda” (!) e procurar todos os ‘.’ presentes na mesma, para tal chamamos a função “NumeroP” (!) com a “MatrizAjuda” encontrando assim o número de ‘.’ existentes nessa matriz.

Em seguida, geramos um número aleatório (“numeroRandom”), este terá, no entanto, algumas restrições, pois o mesmo só poderá ser um número gerado de 1 até ao número de jogadas possíveis.

Sabendo que os números aleatórios são calculados através de uma equação e que esses valores são influenciados por valores anteriores, decidimos usar as funções “srand (time (0))” e “rand” para tal não ocorrer.

Para realizar a jogada optamos por percorrer a “MatrizAjuda” e sempre que encontrássemos um caracter ‘.’ subtraíamos um (-1) ao número aleatório gerado, quando esse número chegar ao valor um (1) e fosse encontrado um caracter ‘.’ na “MatrizAjuda” poderíamos então concluir que esta seria posição de jogada do bot.

Após encontrada a posição de jogada, o bot vai efetuar a sua jogada na grelha e vai fazer as devidas alterações no tabuleiro.

```
void JogaBot1 (ESTADO *e) {
    int i,j,encontrado = 0;
    int numeroP;
    int numeroRandom;

    numeroP = NumeroP(e->MatrizAjuda);

    srand(time(0));

    numeroRandom = (rand() % numeroP) +1;

    for (i=0 ; i<8 && encontrado == 0; i++) {
        for (j=0 ; j<8 && encontrado == 0; j++) {
            if (numeroRandom == 1 && e->MatrizAjuda[i][j] == '.') {
                e->grelha[i][j] = e->peca;
                ConvertePecas(e->grelha,e->peca,i,j);
                encontrado = 1;
            }
            else if (e->MatrizAjuda[i][j] == '.') numeroRandom--;
        }
    }
}
```

Figura 2 - JogaBot1

Notas:

- **MatrizAjuda** - Uma matriz (de caracteres) auxiliar presente no “estado” a qual guardamos como pontos as jogadas validas/possíveis do jogador
- **NumeroP** - Funão que retorna o número de jogadas validas de um jogador

3.2.2 – Bot Médio (JogadasBotMedio.c)

Como no Bot medio tínhamos de dificultar ligeiramente a vida do jogador para o obrigar a usar o cérebro, decidimos adotar a estratégia de executar a jogada onde o bot obtém mais peças.

Para inicializar a criação deste bot, começamos por executar uma nova função “CopiaMatrizBot” que copia uma nova matriz a parte da principal, esta nova matriz tem como objetivo gerar jogadas e verificar em qual delas o bot ganha mais peças. Após a criação desta nova matriz vamos procurar na “MatrizAjuda” o primeiro ‘.’, quando encontrado vamos realizar a jogada na matriz “MatrizAjudaBot” e guardamos nas variáveis “maior” e “coordenadas” quantas peças foram ganhas e as coordenadas dessa jogada, respetivamente. De seguida voltamos a copia a matriz “MatrizAjudaBot” a partir da matriz principal com a função “CopiaMatrizBot” para podermos neste realizar novas jogadas. Seguindo para os ‘.’ subsequentes da “MatrizAjuda” realizamos o mesmo processo utilizado anteriormente, porém comparamos agora a nossa variável “maior” a uma nova variável “competidor” que tem o mesmo propósito que a “maior” (guarda quantas peças são ganhas quando se joga naquele ponto especifico) caso este seja superior, substituímo-lo e passamos a ter um novo “maior” e uma nova “coordenadas”.

No fim de percorrermos todos os ‘.’ da “MatrizAjuda” e obtermos o “maior” número de peças que podemos obter, vamos a posição guardada pela variável “coordenadas” e efetuamos a jogada na grelha principal.

```
void JogaBot2 (ESTADO *e) {
    int i, flag=0, j, maior, competidor;
    Posicao coordenadas;

    CopiaMatrizBot(e->MatrizAjudaBot, e->grelha);

    for (i=0 ; i<8 ; i++) {
        for (j=0 ; j<8 ; j++) {

            if (flag==0 && e->MatrizAjuda[i][j] == '.') {

                maior = ConvertePecas(e->MatrizAjudaBot, e->peca, i, j);
                coordenadas.x = i;
                coordenadas.y = j;

                CopiaMatrizBot(e->MatrizAjudaBot, e->grelha);
                flag++;

            }
            else if (e->MatrizAjuda[i][j] == '.') {

                competidor = ConvertePecas(e->MatrizAjudaBot, e->peca, i, j);

                if (maior < competidor) {

                    maior = competidor;
                    coordenadas.x = i;
                    coordenadas.y = j;

                    CopiaMatrizBot(e->MatrizAjudaBot, e->grelha);

                }

            }

        }
    }

    e->grelha[coordenadas.x][coordenadas.y] = e->peca;
    ConvertePecas(e->grelha, e->peca, coordenadas.x, coordenadas.y);
}
```

Figura 3 - JogaBot2

3.2.3 – Bot Difícil (JogadasBotDifícil.c)

Chegamos por fim a última dificuldade do modo automático, o bot difícil, que implica um maior ponderamento e inteligência pelo parte do jogador.

Este bot vem com a particularidade de casos, pois apercebemo-nos que os cantos do tabuleiro são as melhores jogadas possível, devido a que estas são as posições de maior impacto. No entanto e como é logico nem sempre é possível jogar nos cantos, logo temos de ter outro processo para realizar jogadas. Seguindo a mesma ideia do bot medio, começamos por percorrer a “MatrizAjuda” até encontrar os ‘.', após encontrado um ponto efetua-se a função “CalculaMobilidade” que realiza uma jogada no ponto encontrado e verifica a “mobilidade” do jogador adversário, ou seja, calcula o número de jogadas validas/possíveis do oponente, após a jogada do bot. Guarda-se então esse valor na variável “menor” sendo este o valor de referência.

Para aumentar ainda mais a dificuldade do bot, decidimos que quando houvesse uma igual mobilidade do jogador oponente em dois pontos de jogadas possíveis, neste caso optamos pela estratégia do bot medio, comparando entre estes dois pontos o que ganha mais peças no tabuleiro.

No entanto, podemos nos deparar ainda com o excecional caso de nestes dois pontos, o ganho de peças ser o mesmo, para desempatar esta igualdade vamos a “matrizValores” (!) verificar os valores nesses pontos, escolhendo assim o maior valor entre os dois e efetuando então a jogada nessa posição.

É possível ainda que os valores sejam iguais. Neste caso especialíssimo o bot vai apenas escolher o valor de referência, ou seja, o ‘.’ que apareceu em primeiro na “MatrizAjuda” e por fim irá concretizar a jogada final na grelha.

```
void JogaBot4 (ESTADO *e) {
    int i,j,menor,challenger,flag = 0;
    int maiorPecas,challengerPecas;
    Posicao coordenadas;

    for(i=0 ; i<8 ; i++) {
        for (j=0 ; j<8 ; j++) {

            if (e->MatrizAjuda[0][0] == '.') {
                coordenadas.x = 0;
                coordenadas.y = 0;
                flag++;
            }

            else if (e->MatrizAjuda[7][0] == '.') {
                coordenadas.x = 7;
                coordenadas.y = 0;
                flag++;
            }

            else if (e->MatrizAjuda[0][7] == '.') {
                coordenadas.x = 0;
                coordenadas.y = 7;
                flag++;
            }

            else if (e->MatrizAjuda[7][7] == '.') {
                coordenadas.x = 7;
                coordenadas.y = 7;
                flag++;
            }

            else if (e->MatrizAjuda[i][j] == '.' && flag == 0) {
                menor = CalculaMobilidade(e,e->grelha,i,j);
                coordenadas.x = i;
                coordenadas.y = j;
                flag++;
            }

        }
    }
}
```

Figura 4 - JogaBot4 (Parte I)

```
else if (e->MatrizAjuda[i][j] == '.') {
    challenger = CalculaMobilidade(e,e->grelha,i,j);

    if (menor >= challenger) {
        if (menor > challenger) {
            menor = challenger;
            coordenadas.x = i;
            coordenadas.y = j;
        }

        else if (menor == challenger) {
            maiorPecas = ConvertePecas(e->MatrizAjudaBot,e->peca,coordenadas.x,coordenadas.y);
            copiaMatrizBot(e->MatrizAjudaBot,e->grelha);
            challengerPecas = ConvertePecas(e->MatrizAjudaBot,e->peca,i,j);
            copiaMatrizBot(e->MatrizAjudaBot,e->grelha);

            if (maiorPecas >= challengerPecas) {
                if (maiorPecas > challengerPecas) {
                    menor = challenger;
                    coordenadas.x = i;
                    coordenadas.y = j;
                }

                else if (maiorPecas == challengerPecas) {
                    if (e->matrizValores[coordenadas.x][coordenadas.y] < e->matrizValores[i][j]) {
                        menor = challenger;
                        coordenadas.x = i;
                        coordenadas.y = j;
                    }
                }
            }
        }
    }
}
```

Figura 5 - JogaBot4 (Parte II)

```
}

if (flag > 0) {
    e->grelha[coordenadas.x][coordenadas.y] = e->peca;
    ConvertePecas(e->grelha, e->peca, coordenadas.x, coordenadas.y);
}
}
```

Figura 6 - JogaBot4 (Pate III)

Notas:

- **matrizValores** – Uma matriz armazenada no estado sempre que é inicializado o programa, esta atribui um valor para casa posição do tabuleiro (Figura 7).

```
{120, -20, 20, 5, 5, 20, -20, 120,
-20, -40, -5, -5, -5, -5, -40, -20,
20, -5, 15, 3, 3, 15, -5, 20,
5, -5, 3, 3, 3, 3, -5, 5,
5, -5, 3, 3, 3, 3, -5, 5,
20, -5, 15, 3, 3, 15, -5, 20,
-20, -40, -5, -5, -5, -5, -40, -20,
120, -20, 20, 5, 5, 20, -20, 120,};
```

Figura 7 - MatrizValores

3.3 – Modo Competitivo (FuncoesAuxiliares.c)

Uma boa competição faz bem a qualquer um e como tal tivemos a ideia de criar um novo modo de jogo da nossa autoria, o “Modo Competitivo” que consistia em jogar contra um bot mais difícil, não tendo a possibilidade de pedir ajuda (comando ‘H’), de ter as jogadas validas (comando ‘S’), de poder salvar o jogo (comando ‘E’) e de descarregar um jogo (comando ‘L’) já guardado. Neste modo de jogo também temos um sistema “score” que nos permite a classificação de cada jogador num ranking exclusivo para este modo.

Para começar vamos explicar como é que funciona o sistema que permite atribuir um “score” a um jogador. Ao principio, nos não tínhamos a certeza qual seria a forma mais difícil de ganhar um jogo, se seria acabar com o maior numero de peças porem com muitas jogadas ou se acabar com menos peças mas com poucas jogadas, por isso optamos por fazer um balanceamento entre as duas, que consiste em acabar com menos jogadas possíveis e com o maior numero de peças. No fim do jogo é então calculado o score do jogador mesmo tendo este ganho ou perco.

```
float CalculaScore (ESTADO *e) {
    float Jscore = 0;
    int n,j;

    if (e->peca == VALOR_X) n = NumeroJ1(e);
    else n = NumeroJ2(e);

    j = e->jogadas-30;

    if (n <= 20) Jscore = n;
    else if (n <= 40) Jscore = 2 * n;
    else if (n <= 55) Jscore = 3 * n;
    else Jscore = 4 * n;

    if (j <= 20) Jscore += (j * 25);
    else if (j <= 23) Jscore += (j * 15);
    else if (j <= 27) Jscore += (j * 10);
    else if (j < 30) Jscore += (j * 5);
    else Jscore += (j * 2);

    return Jscore;
}
```

Figura 8 - CalculaScore

Entender o funcionamento do score agora não deve ser um problema, e a forma de guardar os diferentes jogadores e os seus “scores” também não o deve ser, pois estes estão todos armazenados numa lista ligada ordenada pelos scores em que cada jogador, tendo em cada nodo da lista o nickname (max. 8 caracteres) e “score”.

Mas e se um jogador quiser melhorar o seu score? Basta introduzir o mesmo nickname e caso este tenha feito um melhor “score” o seu score e classificação no ranking ira ser atualizado, caso jogue outra vez e tenha uma pior prestação não há problema pois o programa ira guardar apenas a melhor tentativa do jogador em questão.

Agora que o funcionamento do “Modo Competitivo” esta esclarecido vamos passar ao funcionamento deste novo e melhorado bot.

Este bot usa o algoritmo do “Alpha-Beta” para realizar as suas jogadas que consiste em procurar o maior ganho com o menor número de perdas.

Continuando o nosso bot começa por copiar a matriz que estamos a receber como argumento para uma matriz a parte “MatrizJogadas” e convertemos essa matriz em uma matriz de caracteres “MatrizPontos”, sendo assim uteis as funções “CopiaMatrizBot” e “ConverteMatriz”. Para alem disso também é necessário saber em que posições é que o bot pode efetuar a sua jogada, para tal coloca-se os ‘.’ na matriz a parte de caracteres “MatrizPontos”, com isto podemos efetuar jogadas numa matriz a parte da principal.

De seguida, vai se então percorre a matriz “MatrizPontos” e procura os ‘.’ disponíveis. Após encontrado um ‘.’ o bot realizará uma jogada nesse ‘.’, como é obvio, numa matriz a parte “MatrizJogadas”. Este processo ira ocorrer recursivamente, subtraindo sempre a variável “Alcance” por 1, e a alterar no jogador introduzido nos argumentos de cada iteração. Quando a variável “Alcance” chegar ao valor 0, ou seja, se inicialmente esta variável estava a 10, este bot ira percorrer 10 iterações/jogadas á frente do jogo atual.

```
void AcrescentaScore (ESTADO *e, float Jscore) {
    LRanking acres;
    LRanking ant = 0;
    LRanking aux = e->tabela;

    if ( ProcuraJogador(e, Jscore) == 0) {

        acres = malloc(sizeof(struct Ranking));
        acres->jogador.score = Jscore;
        strcpy(acres->jogador.nome, e->nomeTemporario);

        while (aux && aux->jogador.score > Jscore) {
            ant = aux;
            aux = aux->prox;
        }

        acres->prox = aux;

        if (ant) ant->prox = acres;
        else e->tabela = acres;
    }
}
```

Figura 9 - AcrescentaScore

```
int MinMax (ESTADO *e, VALOR Matriz[][8], int Alcance, VALOR Jogador, int alpha, int beta) {
    VALOR MatrizJogadas[8][8];
    char MatrizPontos[8][8];
    VALOR JogadorAdversario;
    int max = -500, min = 500;
    int challenger;
    int calcula = -500;
    int flag = 0;
    int i, j;
    Posicao coordenadas;

    CopiaMatrizBot(MatrizJogadas, Matriz);
    ConverteMatriz(e, MatrizJogadas, MatrizPontos);
    JogadasValidas(MatrizJogadas, Jogador, MatrizPontos);

    if (Jogador == VALOR_X) JogadorAdversario = VALOR_O;
    else JogadorAdversario = VALOR_X;

    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            if (MatrizPontos[i][j] == '.') {

                if (Alcance == 0) {
                    if (flag == 0) {
                        calcula = e->matrizValores[i][j];
                        flag++;
                    }
                    else if (calcula < e->matrizValores[i][j]) {
                        calcula = e->matrizValores[i][j];
                    }
                }
            }
        }
    }
}
```

Figura 10 - MinMax (Parte I)

```
else {
    if (e->peca == Jogador) {
        MatrizJogadas[i][j] = Jogador;
        ConvertePecas(MatrizJogadas, Jogador, i, j);

        challenger = MinMax(e, MatrizJogadas, Alcance - 1, JogadorAdversario, alpha, beta);

        if (challenger > max) {
            if (challenger > max) max = challenger;

            if (Alcance == 10) {
                coordenadas.x = i;
                coordenadas.y = j;
            }
        }

        if (max > alpha) {
            alpha = max;
        }

        if (alpha >= beta) {
            return max;
        }
    }
}
```

Figura 11 - MinMax (Parte II)

Como neste exemplo o bot esta feito para se ver 10 jogadas á frente, na decima iterao do jogo o bot ira ver qual o maior resultado consoante as jogadas possíveis e os valores dessas posies na “matrizValores”. Após isto, a funo ira retornar esse mesmo valor.

Voltamos agora atras para quando a variável “Alcance” tem valor 1, pois neste ponto da funo ainda não retornamos nenhum valor. Neste ponto da funo a variável “challenger” passara a ter o valor de retorno desta funo quando o “Alcance” tinha valor 0. É necessário também reparar que como efetuamos uma jogada na “MatrizJogadas” esta necessita de voltar atras na jogada feita, para tal executamos a funo “CopiaMatrizBot”. Iremos por fim comparar variáveis e tomara decisies a partir aqui.

Por último, depois desta funo recursiva ser efetuada e encontrado o melhor valor consoante as diversas comparaes de variáveis, o bot efetua uma jogada.

```

} else {
    MatrizJogadas[i][j] = Jogador;
    ConvertePecas(MatrizJogadas, Jogador, i, j);

    challenger = MinMax(e, MatrizJogadas, Alcance - 1, JogadorAdversario, alpha, beta);
    CopiaMatrizBot(MatrizJogadas, Matriz);

    if (challenger < min) {
        min = challenger;
    }

    if (min < beta) {
        beta = min;
    }

    if (alpha >= beta) {
        return min;
    }
}

```

Figura 12 - MinMax (Parte III)

```

if (Alcance == 0) return calcula;
else {
    if (NumeroP(MatrizPontos)) {
        if (Alcance == 10) {
            e->greilha[coordenadas.x][coordenadas.y] = e->peca;
            ConvertePecas(e->greilha, e->peca, coordenadas.x, coordenadas.y);
        }

        if (e->peca == Jogador) return max;
        else return min;
    } else {
        if (Jogador != e->peca) Alcance--;
        return MinMax(e, MatrizJogadas, Alcance, JogadorAdversario, alpha, beta);
    }
}
}

```

Figura 13 - MinMax (Parte IV)

3.4 – Salvar Jogada (Salvar.c)

Nesta opo do interpretador, foi-nos pedido que fosse implementado a gravao da jogada que o utilizador estava a executar. Porem nós fomos para além do que nos foi pedido e gravamos o jogo inteiro, isto é, gravamos num ficheiro não só a jogada que o utilizador estava a executar como também a stack existente naquele momento. Isto permitia-nos assim que quando descarregássemos esse ficheiro, conseguíssemos realizar o comando “undo” e voltar atras no jogo.

É também aqui, na funo de salvar o jogo, que sempre que o realizarmos, adicionamos o nome do ficheiro no ficheiro “Ficheiro.txt”, o qual armazena todos os nomes de ficheiros guardados.

3.5 – Descarregar Ficheiro (Descarregar)

Nem sempre é possível finalizar um jogo no momento, por vezes temos tempo do acabar, para tal temos a opção de guardar. Quando finalmente temos a possibilidade de o jogar, temos então a opção de o descarregar.

Esta parte do interpretador descarrega ficheiros previamente guardados. Para tal, o programa limpa a stack atual e começa a descarregar o jogo para a stack. Finalmente após o jogo ter sido todo descarregado para a stack, é nos apresentado o estado atual do jogo que foi gravado, posto o mesmo no tabuleiro principal.

3.6 – Remover Ficheiro (ControloDeFicheiros.c)

Fazer o chamado “reset” e apagar todo o nosso progresso realizado faz parte, a “RemoveFicheiro” é a escolha a fazer na hora de tirar o “lixo” presente na diretoria.

Com isto, este comando permite-nos remover ficheiros existentes na diretoria, tendo em conta que este só pode remover os ficheiros criados por algum utilizador (com o comando E do interpretador).

3.7 – Jogar (ControloJogadas.c)

Existem diferentes formas de jogar “O jogo do Reversi” (manual, automáticos e as suas diferentes dificuldades), como tal a opção “jogar” comporta se de diferentes maneiras consoante com o modo de jogo que esta a ser jogado.

Para efetuar uma jogada o jogador tem de usar o caracter ‘J’ e indicar o número da linha e coluna, respetivamente, onde quer realizar a jogada. Após sabermos onde o jogador quer efetuar a jogada, e chamada então a função “ControloDeJogadas” que não só verifica o modo de jogo atual, como consoante o modo realiza a jogada. Por outro lado, um jogador por ficar sem peças, ou o número de jogadas exceder o limite e nesta exceção a função que controla as jogadas irá termina automaticamente o jogo, mostrando qual do jogador ganhou o jogo e o número de peças de cada jogador no tabuleiro.

3.8 – Trocar Peças (TrocIcons.c)

Jogar sempre com o ‘X’ e o ‘O’ pode ser cansativo pensamos nos, como tal decidimos criar a possibilidade dos jogadores escolherem os seus próprios “icons”.

Começamos por criar um array de “icons” com todos os caracteres usados pelo jogo (X, O, -, ., ?), deste modo podemos constatar que o ‘X’ e o ‘O’ estão nas duas

primeiras posiões do array. Seguindo este raciocínio nos perguntamos ao utilizador quais os “icons” que deseja utilizar e substituímos esses dois novos “icons” nas duas primeiras posiões do array, respetivamente. Deste modo o jogador pode variar os “icons” com que joga e criar uma nova dinâmica ao jogo.

3.9 – Jogadas Validas (MatrizAjuda.c)

Nem sempre vemos tudo o que esta ao nosso alcance, “O jogo do Reversi” não é exceão logo para nossa felicidade e ajuda a expandir horizontes temos as “Jogadas Validas”.

Ora, a funcionalidade desta é bastante direta, começa por converter copiando a matriz principal para uma matriz de caracteres a parte, chamada “MatrizAjuda” que esta presente no estado. De seguida acrescenta os pontos na matriz com o auxílio da funão “JogadasValidas” que procura todas as possibilidades de jogada/todas as posiões onde viramos peças do oponente.

Esta o faz da seguinte forma, começamos por atribuir um valor ao uma variável local “JogadorAdversario”, conforme o valor do jogador que esta a efetuar a jogada. De seguida, começamos por percorrer a matriz principal até encontrarmos a uma das peças do jogador. Quando a tal for possível, esta ira entrar num conjunto de comandos de controlo, para verificar se existem ou não possibilidades dessa peça inverter peças do adversário. Para que isto acontea, é necessário que se verifique a seguinte condião, caso esta peça esteja entre, pelo menos uma das peças do jogador adversário e uma peça de valor “Vazia”, sendo que nenhuma peça do jogador possa estar também presentes no meio destas, então se isto for verdade é porque existe na peça de valor “Vazia” uma jogada possível. Iremos então replicar esta ideia para os 8 lados existentes para a peça.

```
void JogadasValidas (VALOR Matriz[][8], VALOR Jogador, char MatrizAjuda[][8]) {
    int l, c, i, flag, encontrado, ok;
    char JogadorAdversario;

    if (Jogador == VALOR_X) JogadorAdversario = VALOR_O;
    else JogadorAdversario = VALOR_X;

    for (l=0; l < 8; l++) {
        for (c=0; c < 8; c++) {
            if (Matriz[l][c] == Jogador) {
                flag = 0;
                encontrado = 0;
                ok = 0;
                for (i = 1; l + i <= 7 && encontrado == 0 && ok == 0; i++) {
                    if (Matriz[l + i][c] == JogadorAdversario) flag = 1;
                    else if (Matriz[l + i][c] == VAZIA) encontrado = 1;
                    else if (Matriz[l + i][c] == Jogador) ok = 1;
                }
                i--;
                if (flag == 1 && encontrado == 1 && ok == 0) MatrizAjuda[l + i][c] = '.';
            }
        }
    }
}
```

Figura 14 - JogadasValidas

Após encontradas as jogadas válidas, concluímos por apresentar essa matriz ao utilizador.

3.10 – Sugesto de Jogada

At os grandes jogadores precisam de uma mozinha, e  nestas situaes que a “sugesto de jogada” entra em vigor.

Decidimos que a melhor forma de mostrar uma boa jogada seria escolher a jogada possvel desse jogador que tinha mais valor na “matrizValores”. Temos de ter ateno tambm que esta sugesto depende bastante do tipo de estratgia que optamos por seguir.

Para encontrarmos ento um boa jogada comeamos por procurar na “MatrizAjuda” todos os ‘.’ (possibilidades de jogada) e nas posies dos respetivos pontos iremos comparar os valores da “matrizValores”. Ficamos no fim com o a posio do “maior” dos valores e substituímos na “MatrizAjuda” o ‘.’ de “maior” valor por um ‘?’ para o auxilio do jogador. Finalmente apresentamos a matriz “MatrizAjuda” ao utilizador.

3.11 – Undo (Stack.c)

Errar  humano, como tal existir uma hiptese que nos permita refazer/jogar novamente  algo essencial no jogo.

Para tal temos o Undo, a capacidade de regredir no “estado”. Para esse fim, comeamos por criar uma lista ligada no estado. Sempre que iniciamos um jogo a lista ligada do estado  inicializada e quando o terminamos a mesma lista ligada  limpa. Apos a criao desta lista ligada a cada jogada efetuada acrescentamos um novo nodo no fim da lista passando este a ser o novo topo da stack.

Nesta lista esto guardadas no s as matrizes anteriores, como tambm os jogadores que estavam a efetuar a jogada.

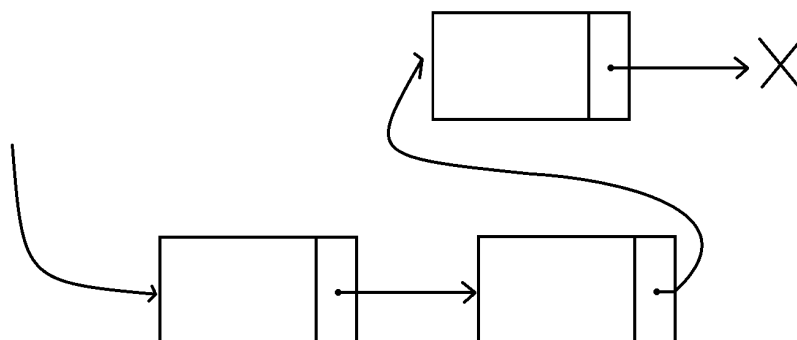


Figura 15 - Stack (Exemplo)

3.12 – Ficheiros Guardados (ControloDeFicheiros.c)

Este comando foi criado para que não fosse preciso ir até a diretoria todas as vezes que queríamos saber os o nome dos ficheiros que já foram guardados. Para tal, este comando mostra a lista de ficheiros guardados pelo utilizador.

É importante também destacar que este comando apenas mostra ficheiros que foram guardados através do comando ‘E’ do interpretador.

3.13 – Ranking (FuncoesAuxiliares.c)

Como ver a classificação do modo competitivo faz parte, esta opção permite nos fazer isso mesmo. Abre ficheiro “Ranking.txt” e mostra a grande tabela classificativa dos jogadores que praticaram este modo até ao fim, aqui podemos observar a classificação, o nickname e o score dos jogadores.

3.14 – Sair

Na última hipótese do nosso interpretador temos uma saída do programa que simplesmente fecha-o.

4 – Conclusão

Concluimos o nosso relatório sobre “O jogo do Reversi” com uma breve análise as diferentes estratégias presentes no nosso jogo, como podemos constatar ao longo do relatório (no uso dos mesmos em bots), existem várias estratégias possíveis a adotar neste jogo.

As diferentes estratégias que nós optamos por usar nos nosso bots foram:

- **Maximum Disks** – esta estratégia consiste em jogar nas posições onde um dos jogadores inverte o maior número de peças do adversário. Utilizamos esta estratégia no bot medio pois esta embora não seja muito eficiente, visto que ao jogar com o objetivo de obter os cantos do tabuleiro aniquila esta estratégia, porem continua a ser uma estratégia bastante solida para quem ainda não tem muita familiaridade com o jogo.

- **Positional** - tem como objetivo efetuar jogadas nas posiões de maior valor no tabuleiro correspondentes a cada posião do tabuleiro, este é provavelmente a estratégia mais poderosa (na nossa opinião). Desta forma decidimos que o nosso bot de do modo competitivo iria jogar consoante este processo.
- **Mobility** - limita-se a jogar nas posiões onde o adversário terá o menor numero de jogadas possíveis retirando assim algo poder de decisão e a dificultar o jogo ao oponente, como esta estratégia pode causar pânico no oponente graças ao bloqueio da mobilidade do mesmo, esta é também uma boa estratégia a ser utilizada, o único senão desta estratégia é caso seja utilizada conta um bot, pois este não iria falhar enquanto um humano devido a limitação tenda a cometer mais erros, por isso decidimos utilizar esta estratégia no nosso bot difícil.

O nosso Bot de dificuldade mais elevada no modo automático usa uma junão das três estratégias aqui apresentadas (Mobility, Maximum Disks e Positional).

No entanto não utilizamos todas as estratégias possíveis, devido a grande variedade destas, um outro exemplo de uma estratégia é:

- **Stable disks** - resume se a jogar em posiões onde as peças não consigam ser viradas de modo a obter peças “fixas” no tabuleiro.

Para mais estratégias utilizadas deixamos aqui a nossa sugestão onde as procurar:

http://samsoft.org.uk/reversi/strategy.htm?fbclid=IwAR1M-PesnVPQWBe_h4SPRjr_ZdYUtG25SVE_IWhyJM1nCwsTzSYi4CRK0hs