

# Relatório Reversi

May 20, 2019

Laboratórios de Informática - PL3  
Universidade do Minho - MIEI

João Gomes  
Marco Pereira  
Carlos Filipe



Figure 1: Tabuleiro Reversi

# 1 Índice

- Introdução - página 4
  - Objetivos - página 4
- O Reversi - página 5
  - Origem - página 5
  - Descrição - página 5
  - Jogadas - página 6
  - Final - página 6
- Análise do jogo - página 7
  - Menu - página 7
  - Interface do jogo - página 8
  - Jogadas válidas - página 9
  - Novo jogo - página 9
  - Histórico - página 10
  - Undo - página 10
  - Ler/gravar jogo - página 11
  - Jogar - página 12
  - Bot - página 13-14
  - Sugestão - página 15
  - Campeonato - página 15

-Conclusão - página 16

## 2 Introdução

Este projeto foi realizado no âmbito da disciplina de Laboratórios de informática 2. O mesmo tem como principal objetivo conceber o jogo Reversi na linguagem C com os conhecimentos adquiridos na cadeira de Programação Imperativa, ou seja, produzir um jogo de tabuleiro com inteligência artificial que permitisse o confronto entre máquina e humanos.

### 2.1 Objetivos

- Implementação de um menu com diferentes linhas de comando;
- Leitura das jogadas via teclado;
- Validação da jogada;
- Visualização atual do estado de jogo: score de cada jogador e tabuleiro;
- Gravação e leitura do estado de jogo em formato ficheiro;
- Visualização das jogadas válidas;
- Implementação de um bot com diferentes níveis de dificuldade;
- Reversão do estado de jogo (Undo).

## 3 O Reversi

### 3.1 Origem

Reversi é um jogo antigo cuja origem é incerta. As referências mais antigas sobre jogos semelhantes remontam ao final do século XIX; esses jogos tinham nomes diferentes e os seus tabuleiros tinham tamanho ou formato diferentes. Em 1870 apareceu um jogo semelhante usando uma placa de forma transversal. Posteriormente apareceu outro jogo jogado em uma placa de forma quadrada 8x8.

### 3.2 Descrição

Dois jogadores participam neste jogo. Ambos usam um tabuleiro com 64 quadrados distribuídos em 8 linhas e 8 colunas, e 64 peças similares de duas cores (normalmente preto e branco).

O objetivo de cada um dos jogadores é terminar o jogo com mais peças no tabuleiro em sua própria cor do que o adversário. Antes de começar, os jogadores decidem qual cor usará cada um deles. De seguida, 4 peças devem ser colocadas nos quadrados centrais do tabuleiro, para que cada par de peças da mesma cor forme uma diagonal entre elas. O jogador com peças pretas se move primeiro. Um único movimento é feito a cada turno.

### 3.3 Jogadas

Uma jogada consiste em colocar de fora uma peça no tabuleiro. As peças colocadas nunca podem ser movidas para outro quadrado no final do jogo.

A incorporação das peças deve ser feita de acordo com as seguintes regras:

-A peça incorporada deve flanquear uma ou mais das peças colocadas pelo adversário

-Para flanquear significa que uma única peça ou uma linha reta (vertical, horizontal ou diagonal) de peças do oponente está em ambos os lados ao lado de suas próprias peças, sem quadrados vazios entre todas essas peças.

-O jogador que faz o movimento vira as peças para fora, tornando-se todas em peças próprias

-Se houver mais de uma linha fora do flanco, todas as partes envolvidas nessas linhas terão que ser viradas

-Se não for possível fazer este tipo de jogada, o turn é perdido e o adversário joga novamente.

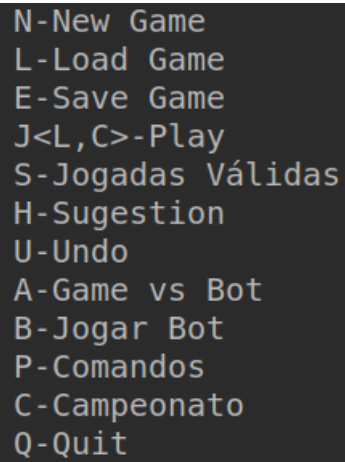
### 3.4 Final

O jogo acaba quando todos os quadrados do tabuleiro são preenchidos ou nenhum dos jogadores se pode mover. Em qualquer caso, o vencedor é o jogador que tem mais peças no tabuleiro. O jogo termina empatado quando ambos os jogadores têm o mesmo número de peças no tabuleiro.

## 4 Análise do jogo

### 4.1 Menu

O menu do nosso jogo apresenta os seguintes comandos:



```
N-New Game  
L-Load Game  
E-Save Game  
J<L,C>-Play  
S-Jogadas Válidas  
H-Sugestion  
U-Undo  
A-Game vs Bot  
B-Jogar Bot  
P-Comandos  
C-Campeonato  
Q-Quit
```

Figure 2: Menu de jogo

Para seleccionar a opção pretendida basta introduzir a letra que se encontra à esquerda da de cada comando.

## 4.2 Interface do jogo

Na interface do jogo está presente, como seria de esperar o tabuleiro com o atual estado de jogo. Decidimos introduzir também informações relevantes ao jogador: a próxima peça a jogar, o modo atual (automático ou manual) e por último, o número de peças de cada jogador.

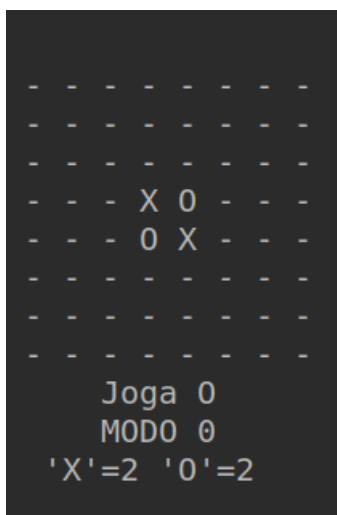


Figure 3: Interface de jogo no comando 'S'



### 4.3 Jogadas válidas

No comando das jogadas válidas, a interface é ligeiramente modificada, adicionando '+' nas posições do tabuleiro onde a jogada é possível e as coordenadas das mesmas na parte inferior do ecrã.

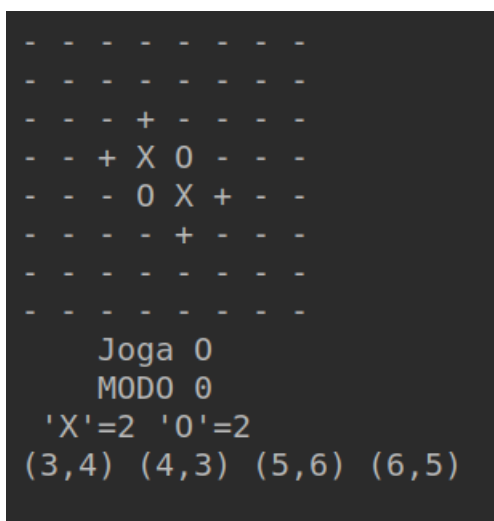


Figure 4: Interface de jogo

### 4.4 Novo jogo

Quando é criado um novo jogo com o comando do mesmo nome, o estado é limpo e passa conter apenas 2 peças de cada jogador dispostas diagonalmente no centro do tabuleiro. O histórico passa a ser nulo. O primeiro jogador pode ser escolhido no comando.

## 4.5 Histórico

Para ser possível reverter uma jogada no comando Undo, foi criada uma struct, ou seja, uma variável que é capaz de armazenar todas as informações relativas aos estados de jogo anteriores. Este tipo de estrutura é também conhecida por lista lista ligada e foi dada o nome de histórico.

```
typedef struct historico {  
    ESTADO past;  
    struct historico* apontador;  
}HIST;
```

Figure 5: Struct Histórico

## 4.6 Undo

Ao fazer Undo, se o histórico for nulo, o jogador será informado que o jogo se encontra no início. Caso contrário, o estado atual passa a ser o primeiro estado presente na lista ligada(o estado da última jogada), movendo o apontador para o próximo estado e libertando a memória ocupada pelo mesmo.

```
e = (*h)->past;  
pas = (*h)->apontador;  
free(*h);  
*h = pas;  
printa(e);
```

Figure 6: Undo

## 4.7 Ler/gravar jogo

Na parte superior do ficheiro, encontram-se gravados o modo de jogo, a próxima peça a jogar e a dificuldade do bot repetidamente. Em baixo decidimos guardar todo o histórico de jogo, ou seja, os estados de todas as jogadas definidas até ao momento.

Ao ler o ficheiro, o estado atual passa a ser o último estado presente no ficheiro. Os restantes estados são guardados na lista ligada do histórico anteriormente mencionado. Para gravar, é usada a mesma lista ligada de modo a imprimir no ficheiro todos os estados presentes no histórico.

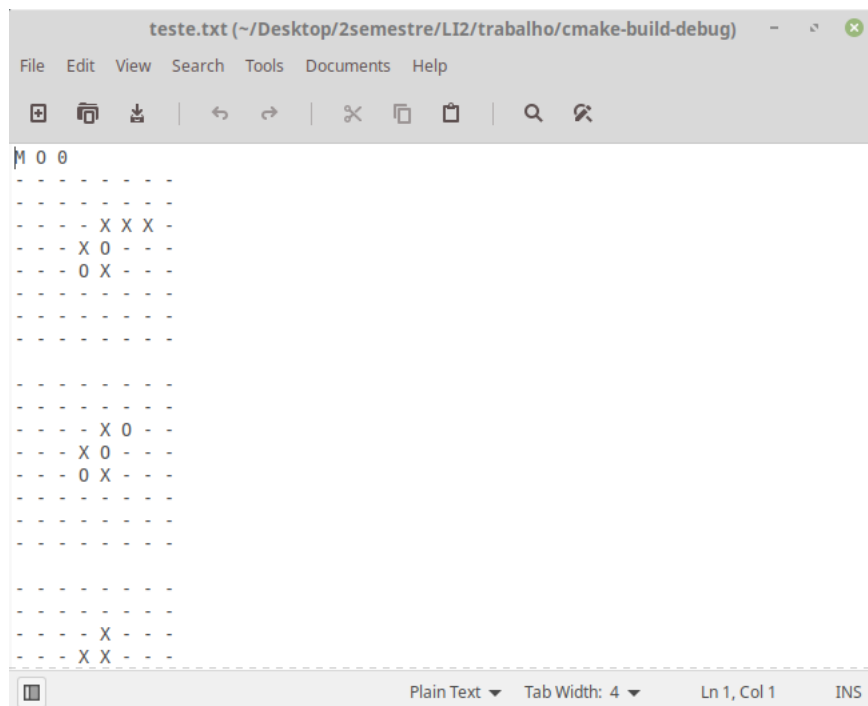


Figure 7: Exemplo de jogo gravado em ficheiro

Nota - Para ler/gravar o jogo é necessário introduzir o nome do ficheiro .

## 4.8 Jogar

Quando este comando é utilizado juntamente com uma posição, o jogo faz a validação da mesma posição usando a seguinte função:

```
int epossivel (int i,int j,ESTADO e)
{
    int u;
    int a [8]= {0,0,0,0,0,0,0,0};
    array (i,j,a,e);
    dispersa (i,j,a,e);
    if (existe (2,a) && e.grelha[i][j]==VAZIA ) return 1;
    return 0;
}
```

Figure 8: Função principal que verifica se jogada é possível

Para além de verificar a jogada e modificar o estado de jogo, o comando Jogar altera o histórico, adicionando o último estado ao mesmo. Para isso, vai alocar memória para guardar o novo estado e andar com o apontador para trás.

```
pas = malloc(sizeof(HIST));
pas->past = e;
pas->apontador = *h;
*h = pas;
```

Figure 9: Alteração do histórico na jogada

## 4.9 Bot

### 4.9.1 MiniMax

No desenvolvimento do nosso bot decidimos adotar a estratégia do Minimax

Em jeito de resumo, Minimax é uma regra de decisão utilizada em inteligência artificial, teoria da decisão, teoria dos jogos, estatística e filosofia para minimizar a possível perda para um cenário de pior caso (perda máxima). Ao lidar com ganhos, é referido como "maximin" - para maximizar o ganho mínimo. Originalmente formulado para a teoria dos jogos de soma zero de dois jogadores, abrangendo os casos em que os jogadores realizam movimentos alternados e aqueles em que fazem movimentos simultâneos, ele também foi estendido a jogos mais complexos e à tomada de decisão geral na presença de incerteza.

### 4.9.2 Estratégia

Para explicar a estratégia, consideremos a seguinte árvore, na qual os degraus max representam as jogadas em que o jogador é o bot e os degraus min representam as jogadas em que o jogador é o adversário:

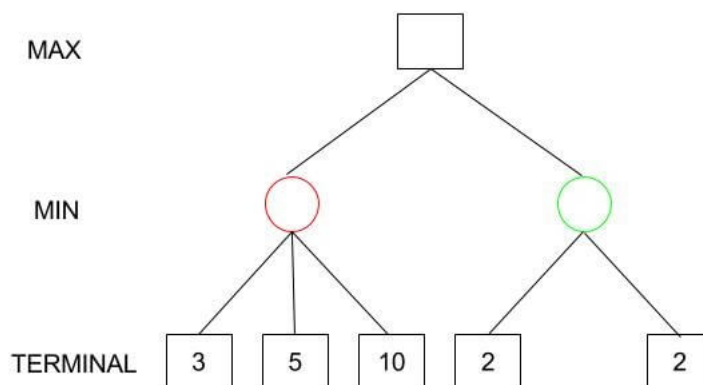


Figure 10: Árvore minimax

O algoritmo começa por calcular a diferença de peças à profundidade desejada, ou seja, a diferença de peças nas jogadas terminais.

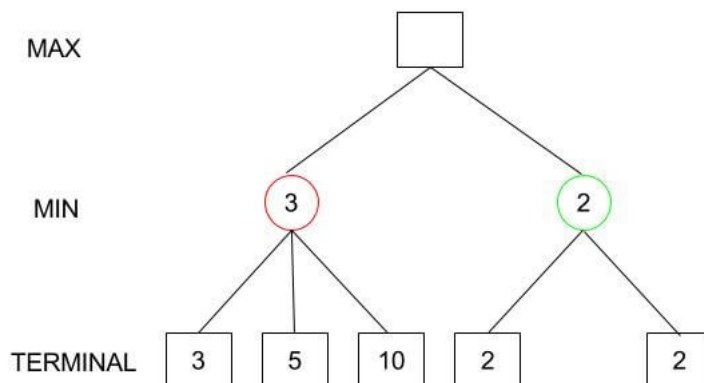


Figure 11: Árvore minimax

Depois é chamada recursivamente a função minimax que escolhe o nodo com a diferença de peças mínima quando é o adversário a jogar e escolhe o nodo com a diferença de peças máxima quando é o bot a jogar. No exemplo acima o algoritmo escolheria a jogada em que a diferença de peças é 3.

Nota: A dificuldade do bot neste trabalho é gerida pela quantidade de jogadas que se deve simular no futuro ou seja a altura da árvore do algoritmo MiniMax, tendo em conta que se dá sempre prioridade aos 4 cantos.

## 4.10 Sugestão

Quando este comando é selecionado, o jogo vai usar o bot com uma profundidade baixa em ordem a dar uma sugestão de jogada que não seja muito forte.

## 4.11 Campeonato

Para avaliar o desempenho dos bots, foi nos solicitado a criação de um comando que permitisse o confronto direto entre grupos. O mesmo é possível devido ao uso de uma pasta partilhada, onde os diferentes grupos podem ler e guardar no mesmo ficheiro. O resultado do jogo é dado por uma extensão no nome do ficheiro: .gO no caso da vitória de O,.gX no caso da vitória de X e .g- em caso de empate.

O nosso grupo decidiu definir a profundidade em 5 ao bot de modo a causar mais problemas ao adversário.

## 5 Conclusão

No final do projeto fomos capazes de realizar todos os objetivos propostos no início do mesmo. Ao mesmo tempo conseguimos demonstrar que fomos capazes de criar uma forma de inteligência artificial com a implementação do bot no jogo.