Universidade Federal Fluminense

Instituto de Ciência e Tecnologia

Departamento de Computação

Trabalho sobre o jogo Reversi (segunda parte)

O jogo Reversi (também conhecido como Othello) é um jogo de tabuleiro para dois jogadores. O tabuleiro utilizado é de tamanho 8x8 e em cada casa são colocadas “pedras” pretas ou brancas (cada pedra tem um lado preto e o outro lado branco). Inicialmente, o tabuleiro é montado como na Figura 1 abaixo, no que é chamado de Posição Inicial.

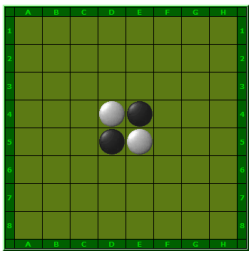


Figura 1: posição inicial do tabuleiro de Reversi

O primeiro jogador a fazer sua jogada é o jogador de pretas. A jogada consiste em colocar uma peça (da cor do jogador) em alguma casa vazia do tabuleiro, de maneira a fazer um “sanduíche” com pelo menos uma peça do adversário. Esse “sanduíche” consiste em uma ou mais peças do adversário, em linha reta (vertical, horizontal ou diagonal), que estejam entre uma pedra já colocada anteriormente e a pedra que acabou de ser colocada. A Figura 2 mostra uma posição do jogo e indica com X vermelho onde o jogador de pretas pode colocar uma pedra para fazer um ou mais sanduíches.

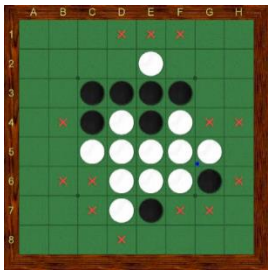
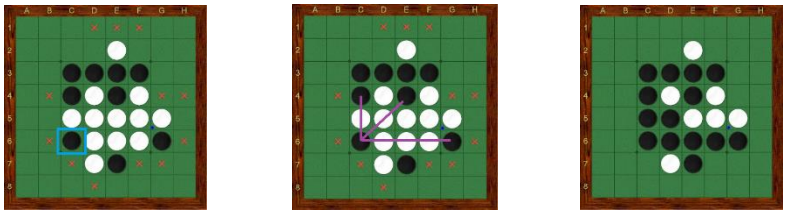


Figura 2: posição intermediária do jogo (o jogador de pretas pode colocar uma pedra em qualquer casa marcada com X vermelho).

Todas as peças que estejam nesses sanduíches são “viradas”, ou seja, passam a pertencer momentaneamente ao adversário, como nas Figuras 3. Todas as pedras do adversário que puderem ser viradas, por estarem em um sanduíche, devem ser viradas imediatamente após a colocação da pedra. Não é possível escolher quais

pedras serão viradas nem deixar pedras sem serem viradas em quaisquer direções possíveis. Uma pedra não pode ser colocada em uma posição onde não vire nenhuma pedra do adversário.



A **B** **C**

Figuras 3: posições intermediárias do jogo. (A) pedra colocada pelo jogador de pretas na casa marcada pelo retângulo azul. (B) todos os sanduíches possíveis com a colocação da pedra são indicados pelos segmentos de reta roxos. (C) a posição intermediária após as pedras brancas (dos sanduíches) serem viradas.

Após colocar uma peça e "virar" as peças do adversário. O jogador passa a vez para o adversário. Se, a qualquer momento, o jogador da vez não puder fazer uma jogada, ele passa a vez para o adversário sem colocar nenhuma pedra. Se os dois jogadores não puderam mais colocar pedras, o jogo termina. O joga também termina quando todas as casas do tabuleiro estiverem com alguma pedra. Ao final, vence o jogador que terminar com mais pedras. Se existirem casas vazias ao final da partida, elas são somadas à pontuação do jogador vencedor.

Mudanças para a Segunda Parte

Na primeira parte do trabalho, o usuário precisava digitar a jogada (linha e coluna) que eram fornecidos à função ExecutaJogada. Essa função tinha dois papéis: descobrir se a jogada era válida e, caso afirmativo, realizar a jogada. Essa não é uma boa prática de programação. O ideal é que exista uma função específica para realizar a jogada e outra para testar se uma jogada é valida ou não.
Na segunda parte do trabalho, nós vamos mudar isso e também a forma de interagir com o usuário na escolha da jogada. Após desenhar o tabuleiro na tela, vamos calcular quais são as possíveis jogadas que podem ser feitas nesse momento. Vamos colocar essas jogadas em uma lista e depois chamar uma função que interage com o usuário. Essa função só encerrará quando o usuário escolher uma jogada válida. Dessa forma, quando a ExecutaJogada for chamada, ela terá certeza que se trata de uma jogada válida e não precisará testar mais nada, apenas realizar a jogada.

O que deve ser implementado?

1) **O struct Posicao (abaixo) representa o tabuleiro do jogo (matriz 8x8 de números inteiros) e indica também qual é o jogador da vez. As casas vazias possuem o valor 0. As pedras brancas são indicadas pelo valor 1 e as pedras pretas pelo valor -1.**
2) **Faça uma função IniciaTabuleiro(), sem parâmetros, que retorne a posição inicial do jogo (o tabuleiro na posição inicial o jogador da vez sendo o jogador de pretas).**

3) **O programa deve ter uma função DesenhaTabuleiro(), que receba um struct Posicao e desenhe na tela as pedras já colocadas no tabuleiro da melhor forma possível (parecendo uma matriz na tela). A função também deve indicar na tela a numeração das linhas e colunas.**
4) **Faça uma função CalculaJogadasValidas(), que deve receber, como parâmetro, a posição atual do jogo e deve retornar uma lista circular duplamente encadeada, contendo todas as possíveis jogadas que podem ser feitas pelo jogador da vez (struct jogada, abaixo). Se não houver nenhuma jogada possível, a função deve retornar NULL. Um ponteiro da main deve receber o resultado da função.**
5) **Faça uma função EscolheJogada(), que receba como parâmetro a lista de jogadas válidas (ponteiro da main). Se essa lista estiver vazia (NULL) a função também deve retornar NULL imediatamente. Se houver pelo menos uma jogada válida, a função deve mostrar todas essas jogadas na tela e deve perguntar ao usuário qual jogada ele quer fazer. O usuário precisa escolher uma das jogadas válidas; caso não escolha, a função ficará em repetição até que a jogada seja escolhida. A jogada escolhida deve ser retornada na forma de um ponteiro para o elemento (escolhido pelo usuário) da lista.**
6) **Faça uma função ExecutaJogada() que receba a posição atual e o valor retornado pela função EscolheJogada(). Se o valor for diferente de NULL, a função deve modificar o tabuleiro de acordo com a jogada indicada e as regras, além de inverter o jogado da vez. Se o valor for NULL, a função deve apenas mudar o jogador da vez, sem alterar o tabuleiro.**
7) **Após a execução da função, se a lista de jogadas válidas não estiver vazia, ela deve ser destruída (memória liberada).**
8) **Após a liberação da memória (se ocorrer), se ainda houver casas vazias no tabuleiro, o programa deve voltar ao passo 3. Caso o tabuleiro esteja completamente cheio, o programa deve indicar quem foi o vencedor (ou se houve empate) e encerrar.**

O que não é necessário implementar (ainda)?

1) **A situação onde não há jogadas válidas disponíveis para os jogadores simultaneamente (ou seja, o programa deve considerar que sempre haverá um dos dois jogadores capaz de realizar alguma jogada).**

```
struct Posicao{

int tabuleiro[8][8];

int jogadorVez;

}

struct jogada{

int linha,coluna;

struct jogada *prox,*ant;

}
```