Universidade Federal de Minas Gerais **Inteligência Artificial**
Instituto de Ciências Exatas **2º Semestre de 2014**
Departamento de Ciência da Computação **Prof. Luiz Chaimowicz**

Trabalho Prático 1
Othello/Reversi
Entrega: 23/10

Instruções preliminares

As implementações podem ser feitas na sua linguagem de programação favorita. Em troca, seu agente deverá respeitar o protocolo de jogadas do torneio, definido neste enunciado. Além disso, certifique-se que seu código pode ser compilado (caso use uma linguagem compilada), e executado em uma máquina GNU/Linux.

Para este trabalho, um kit com o servidor do torneio e um agente 'random' está disponível no moodle (arquivo `kit_tp1.zip`).

Introdução

O objetivo do trabalho é explorar a implementação das técnicas de busca com adversários. Você deve implementar um agente capaz de jogar Othello (também conhecido como Reversi).

Basicamente, o jogo consiste em um tabuleiro onde dois jogadores (preto e branco) tentam capturar o maior número de posições com suas peças. O tabuleiro é formado por um arranjo 8x8 de posições cujas células centrais estão preenchidas por duas peças brancas e pretas, conforme a Figura 1.

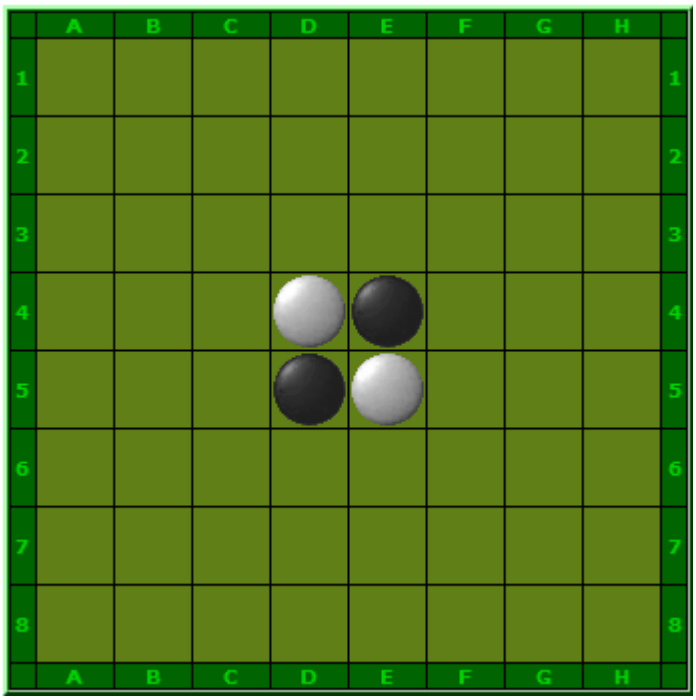

Figura 1: Estado inicial do jogo

As pretas começam jogando e, em cada turno, um jogador deve colocar a próxima peça somente em posições onde uma peça adversária seja capturada. Uma ou mais peças do adversário são capturadas quando existe uma linha reta – horizontal, vertical ou diagonal – entre a peça colocada pelo jogador

e uma das suas outras peças. Todas as peças capturadas mudam de cor e o jogo continua alternadamente. Caso um jogador não possua uma jogada válida, ele deve deve passar a vez. O jogo termina quando não houver jogadas válidas para ambos os jogadores. O vencedor é aquele com o maior número de peças no tabuleiro.

Acesse os seguintes links para entender mais sobre o jogo:

- http://en.wikipedia.org/wiki/Reversi (regras, história, etc)
- http://www.mah-jongg.ch/reversi (jogo online para praticar)

Tarefa

- Implemente uma função `play(tabuleiro, cor)` que recebe o tabuleiro na representação de sua preferência e uma variável que indica a cor com a qual a jogada deve ser feita. A função deve calcular e retornar uma jogada. Você pode definir a representação que preferir para uma jogada.

- Sua função `play` deverá utilizar o algoritmo Minimax com poda alfa-beta para decidir qual será a jogada. No entanto, devido ao vasto número de estados no jogo, não será possível explorar completamente a árvore de busca. Portanto, você deve determinar uma estratégia que defina a profundidade máxima da busca, assim como a função de avaliação dos estados. Diversas características do tabuleiro podem ser utilizadas como função de avaliação (número de peças, quinas, posições estáveis, etc.) e várias estratégias de parada podem ser implementadas (profundidade máxima fixa, variada, quiescence search, singular extension, etc). Cabe ao aluno decidir qual é o melhor método a ser implementado.

- Implemente um programa para um humano jogar Othello contra a sua função `play`. Seu programa deve permitir a escolha da cor com a qual os jogadores vão jogar e processar as jogadas do jogador humano e da sua função `play`. Escreva um script `othello.sh` que execute este programa.

- Haverá um torneio entre os programas dos alunos. Para isso, você deve implementar um outro programa que siga o protocolo de jogadas definido na próxima seção.

Torneio

Os diferentes programas (agentes) irão competir entre si através de troca de arquivos mediada por um servidor. Quando for sua vez de jogar, seu programa será chamado, recebendo o caminho de um arquivo com uma representação padrão do tabuleiro e a cor com a qual a jogada deve ser feita (`black` ou `white`). Seu programa deve converter a representação contida no arquivo para aquela aceita por sua função `play`, que será chamada e retornará a jogada. A jogada deverá ser convertida para a representação padrão do servidor e escrita em um arquivo.

Você deverá escrever um script `launch.sh` que recebe os dois parâmetros e chama o seu programa. Observe a chamada de exemplo, que dará como entrada o `arquivo_estado_tabuleiro` e pedirá que seu agente jogue com as pretas:

```
./launch.sh arquivo_estado_tabuleiro black
```

No arquivo com o estado do tabuleiro, `W` representa uma peça branca (white), `B` uma peça preta (black) e `.` (ponto) representa um espaço livre. No exemplo abaixo, temos a representação do estado inicial da Figura 1. Observe que cada linha do arquivo possui exatamente 8 caracteres

seguidos por uma quebra de linha.

```
........
........
........
...WB...
...BW...
........
........
........
```

Seu agente terá 5 segundos para ler este arquivo, chamar a sua função `play` e escrever a jogada em um arquivo chamado `move.txt`, que deverá ser criado no mesmo diretório do seu `launch.sh`. O arquivo `move.txt` conterá simplesmente a coordenada `x,y` correspondendo ao lugar onde sua peça será colocada. As coordenadas vão de 0 a 7. O eixo x cresce da esquerda para a direita e o eixo y cresce de cima para baixo. O exemplo abaixo mostra o sistema de coordenadas.

```
  01234567 --> eixo x
0 ........
1 ........
2 ........
3 ...WB...
4 ...BW...
5 ........
6 ........
7 ........
|
|
v
eixo y
```

Considerando o estado inicial, um dos movimentos válidos para as pretas é em `x = 5` e `y = 4`. O arquivo `move.txt` de um agente que decidiu por este movimento terá o seguinte conteúdo:

```
5,4
```

O arquivo deverá conter somente os dois números separados por vírgula, sem espaço entre eles. Caso não haja jogada válida para seu jogador, escreva `-1,-1` no arquivo.

Fluxo de jogo

Durante uma partida, seu `launch.sh` será chamado diversas vezes, uma para cada jogada a ser feita pelo seu jogador. Dessa forma, após a jogada, seu agente deverá encerrar sua execução, reiniciando quando o `launch.sh` for chamado novamente. De fato, o servidor encerrará seu script `launch.sh` após os 5 segundos que seu agente tem para fazer a jogada.

Basicamente, o servidor escreverá o estado do tabuleiro no seu diretório, chamará o seu `launch.sh`, aguardará sua jogada e então lerá a jogada feita através do `move.txt` escrito pelo seu programa. Em seguida o servidor escreverá o novo estado no diretório do adversário, chamará o `launch.sh` do adversário e lerá a jogada feita por ele. Isso se repetirá até o fim do jogo.

Entrega

Você deve entregar um arquivo `.zip` contendo:

- O código fonte completo.
- O script `othello.sh` que execute o programa que permite um humano jogar contra seu agente.
- O script `launch.sh` que executará o programa que faz uma jogada, para o torneio.
- Um script `compila.sh` que compile todo o seu código, caso programe em linguagem compilada.
- Documentação detalhada sobre a implementação (`.pdf`) contendo:
  - Descrição dos algoritmos, da função de avaliação, da estratégia de parada; eventuais melhorias (quiescence search, singular extensions, etc); eventuais decisões de projeto e dificuldades encontradas; testes do algoritmo (usando o servidor fornecido no `kit_tp1.zip` e eventuais outros testes); bibliografia completa (sites inclusos).

Observações gerais

- O trabalho pode ser feito em duplas.
- O tempo de 5 segundos é estipulado tendo como referência uma máquina linux do laboratório de graduação, com a seguinte configuração: processador Core2Duo 2.93 Ghz e 4Gb de memória DDR2 800MHz (http://crc.dcc.ufmg.br/infraestrutura/laboratorios/labs_unix).
- Penalização por atraso: 2^dias – 1.
- Trabalhos copiados de colegas ou da internet serão penalizados com a nota zero.
- Haverá um torneio entre todos os agentes recebidos. Por isso, agentes que não conseguirem aderir ao protocolo serão desclassificados do torneio.
- A nota depende da correta implementação e de uma boa documentação. Isto é, um mau desempenho no torneio não resultará em penalidade na nota (desde que seu agente consiga aderir ao protocolo). No entanto, como incentivo, os melhores colocados no torneio receberão pontuação extra.

Dicas

- Leia o `README` do `kit_tp1.zip`, ele contém instruções para a execução do servidor e do jogador 'random'.
- No torneio, após realizar a jogada, seu agente deverá encerrar sua execução. Ele deverá se inicializar novamente suas estruturas e preparar outra busca quando seu `launch.sh` for chamado novamente com o novo estado do tabuleiro, obtido após a jogada do oponente.
- Para facilitar a detecção de erros, é permitido imprimir o que for necessário no terminal durante o torneio, uma vez que as partidas são jogadas via arquivos. Você pode aproveitar o protocolo do torneio para iniciar o seu agente a partir de um estado que esteja causando erros (você escreve o arquivo com o estado problemático e executa seu `launch.sh` manualmente).
- Cuidado com a representação interna do tabuleiro adotada pelo seu agente. No protocolo do torneio, a jogada é `x,y` (coluna, linha) enquanto a representação via matriz endereça primeiramente a linha e depois a coluna.
- Use o jogador 'random' disponível no kit deste trabalho para testar a operação básica do seu agente, com relação ao torneio. O 'random' não é competitivo, portanto é encorajado que os alunos joguem partidas entre si antes do torneio (usem o servidor fornecido no kit), mas a troca de código é proibida.