Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Aplicada

Inteligência Artificial
Prof. Anderson Rocha Tavares

Trabalho 2

Poda alfa-beta ou MCTS em Othello/Reversi

Instruções preliminares

Para este trabalho, um kit com os arquivos necessários para executar o trabalho está disponível no moodle (arquivo `kit_othello.zip`).

As implementações devem ser feitas em Python 3. Assuma que os códigos serão executados em uma máquina com interpretador Python 3.9, com Miniconda, Pip, numba, numpy, pandas). Caso precise de instalar bibliotecas adicionais, descreva-as no seu `Readme.md`

IMPORTANTE: é proibido o uso de bibliotecas que resolvam os problemas, apenas que implementem estruturas de dados (e.g. fila, pilha, etc.) e rotinas auxiliares, (e.g. leitura de arquivos). Recomenda-se a desativação do Github Copilot e similares, vide política de plágio.

Introdução

O objetivo do trabalho é explorar a implementação de poda alfa-beta. 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.

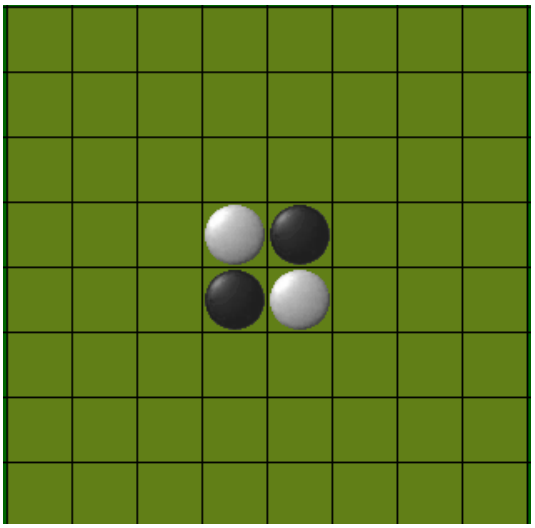

Figura 1 - Estado inicial do Othello

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 passa 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 no navegador)
- Aplicativo Reversi Free (disponível para Android na Play Store)

Tarefa

- Implemente o algoritmo Minimax com poda alfa-beta OU o algoritmo de busca de Monte Carlo em Árvores (MCTS). Sua implementação receberá o estado do jogo (configuração do tabuleiro e a cor que deve fazer a jogada). Para a poda alfa-beta, 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, *iterative deepening*, *quiescence search*, *singular extensions*, etc). Cabe ao grupo decidir qual é o melhor método a ser implementado.

- Você deve preencher a função no arquivo `agent.py` do kit. Note que o arquivo `othello/gamestate.py` contém a implementação do estado e o `othello/board.py` contém a implementação do tabuleiro na classe Board.

Partidas

As partidas são mediadas por um servidor. Quando for sua vez de jogar, a função `make_move(state)` do seu `agent.py` será chamada. A função recebe `state`, um objeto da classe `GameState` que contém um tabuleiro (objeto da classe `Board` e o jogador a fazer a jogada (um caractere) (`B` para as pretas ou `W` para as brancas). Para os detalhes, veja `othello/gamestate.py` e `othello/board.py`.
Em um objeto da classe `Board`, o atributo `tiles` contém a representação do tabuleiro como uma matriz de caracteres (ou lista de strings ;). `W` representa uma peça branca (white), `B` uma peça preta (black) e `.` (ponto) representa um espaço livre. No exemplo a seguir, temos a representação do estado inicial da Figura 1.

```
[
“........”,
“........”,
“........”,
“...WB...”,
“...BW...”,
“........”,
“........”,
“........”,
]
```

O eixo x cresce da esquerda para a direita e o eixo y cresce de cima para baixo. O exemplo a seguir mostra o sistema de coordenadas para o estado inicial.

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

Sua função `make_move` **terá 5 segundos para executar** e deve retornar uma tupla com as coordenadas `x, y` (coluna, linha) onde a jogada será feita, ou `-1, -1` caso não haja jogadas válidas. As coordenadas vão de 0 a 7, pois são os índices da matriz de caracteres. Considerando o estado inicial, um dos movimentos válidos para as pretas, o qual pode ser retornado pela função `make_move` é `(5, 4)`.

IMPORTANTE: cuidado com o sistema de coordenadas vs a indexação de matrizes. Sua função make_move deve retornar as coordenadas `x, y` (coluna, linha) enquanto a representação de matriz endereça primeiramente a linha e depois a coluna.

Torneio

Haverá um torneio entre os programas dos estudantes, em formato a ser definido. Durante cada partida, sua função `make_move` será chamada diversas vezes, uma para cada jogada a ser feita pelo seu jogador. Por isso, a função não deve gerar novos processos ou threads que rodem em background após seu término, sob pena de desclassificação no torneio.

Basicamente, em cada partida o servidor chama o `make_move` de um jogador com o estado do jogo, aguardará sua jogada (com tempo limite de 5 segundos), e chamará o `make_move` do próximo jogador com o estado resultante. Isso se repetirá até o fim do jogo.

Entrega

O kit do trabalho contém um diretorio "`your_agent`", no qual você deve fazer sua implementação. Renomeie o `your_agent` com o nome do seu agente. Use as convenções e nomes de pacotes de python, já que seu agente é visto pelo servidor de partidas como um pacote.

Você pode criar arquivos e pacotes auxiliares dentro do seu diretório, mas você deve preencher as funções do arquivo `agent.py`, pois ele é o ponto de contato do servidor de partidas com a sua implementação.

Ao final, gere um arquivo `.zip` contendo sua implementação e um arquivo `Readme.md` com um pequeno relatório da sua implementação. O relatório deve conter:

- Nomes, cartões de matrícula e turma dos integrantes do grupo;
- Bibliotecas que precisem ser instaladas para executar sua implementação.
- Descrição da função de avaliação, da estratégia de parada; eventuais melhorias (quiescence search, singular extensions, etc para a poda alfa-beta ou qualquer melhoria para o MCTS); decisões de projeto e dificuldades encontradas; e bibliografia completa (incluindo sites).

Supondo que o trio renomeou o diretorio `your_agent` do kit como `alan_turing`, a estrutura do arquivo `.zip` a ser entregue deve ser a seguinte:

```
alan_turing <-- diretorio na raiz do .zip
|-- agent.py  <-- com as funcoes preenchidas
|-- Readme.md <-- com seu relatorio
\-- [outros arquivos e subdiretorios de sua implementacao]
```

Ou seja, você pode supor que sua implementação será executada na mesma estrutura de diretórios do kit do trabalho.

Critérios de correção

Item	Percentual da nota
Aderência à especificação (e.g. estrutura de diretorios, protocolo de jogadas)	10
Relatorio e codigo (correta implementação da poda alfa-beta ou do MCTS)	25
Teste basico (contem as funcoes do test_agent.py do kit e outras mais)	15
Desempenho contra random	20
Desempenho contra minimax 1	20
Desempenho contra minimax 2	10
Torneio (pontos extras)	10
Total	**110**

Importante:

- Nos criterios “Desempenho contra X” serão feitas 2 partidas contra X. Na primeira seu agente joga com as pretas (primeiro a jogar) e, na segunda, X joga com as pretas.
- Nos criterios “Desempenho contra minimax Y”, seu agente irá jogar contra uma implementação básica de minimax com profundidade máxima Y e uma função simples de avaliação de estados.

Observações gerais

- O trabalho deve ser feito em trios.
- O tempo de 5 segundos é estipulado tendo como referência uma máquina linux com a seguinte configuração: processador Core i7 2.93 Ghz e 4Gb de memória RAM.
- Fiquem atentos à política de plágio!
- A nota depende do correto funcionamento da implementação e de um bom relatório. Isto é, um mau desempenho no torneio não resultará em penalidade na nota (desde que seu agente respeite o protocolo e não seja desclassificado). No entanto, como incentivo, os melhores colocados no torneio receberão pontuação extra.
- Não é garantido, mas vamos tentar realizar uma grande final contra o bot campeão dos semestres anteriores e uma partida "humano vs máquina", onde uma pessoa e o bot vencedores se enfrentarão.

Dicas

- Leia o `README.md` do `kit_othello.zip`, ele contém instruções para a execução do servidor e do jogador 'random'.
- Use o comando `tree` no linux para verificar sua estrutura de diretórios.

- Você pode usar as funções do `gamestate.py` para gerar a lista de jogadas válidas e os estados resultantes das mesmas.
- Você pode aproveitar o servidor de partidas para iniciar o seu agente a partir de um estado que esteja causando erros (você escreve a representação com o estado problemático e executa seu agente).

Política de Plágio

Trios poderão apenas discutir questões de alto nível relativas a resolução do problema em questão. Poderão discutir, por exemplo, questões sobre as estruturas de dados utilizadas, as heurísticas de avaliação de estados, vantagens e desvantagens, etc. Não é permitido que os trios utilizem quaisquer códigos fonte provido por outros trios, ou encontrados na internet. **Recomenda-se a desativação do Github Copilot e similares**, pois eles aprendem através de implementações existentes na internet.

Pode-se fazer consultas na internet ou em livros apenas para estudar o modo de funcionamento das técnicas de IA, e para analisar o pseudo-código que as implementa. Não é permitida a análise ou cópia de implementações concretas (em quaisquer linguagens de programação) da técnica escolhida. O objetivo deste trabalho é justamente implementar as técnicas do zero e descobrir as dificuldades envolvidas na sua utilização para resolução do problema em questão. Toda e qualquer fonte consultada pelo trio (tanto para estudar os métodos a serem utilizadas, quanto para verificar a estruturação da técnica em termos de pseudo-código) precisa obrigatoriamente ser citada no relatório.

Usamos rotineiramente um sistema anti-plágio que compara o código-fonte desenvolvido pelos trios com soluções enviadas em edições passadas da disciplina, e também com implementações disponíveis online.

Qualquer nível de plágio (ou seja, utilização de implementações que não tenham sido 100% desenvolvidas pelo trio) poderá resultar em nota zero no trabalho. Caso a cópia tenha sido feita de outro trio da disciplina, todos os envolvidos (não apenas os que copiaram) serão penalizados. Esta política de avaliação não é aberta a debate. Se você tiver quaisquer dúvidas se uma determinada prática pode ou não, ser considerada plágio, não assuma nada: pergunte ao professor e aos monitores.

Note que, considerando-se os pesos das avaliações desta disciplina (especificados e descritos no Plano de Ensino) nota zero em qualquer um dos trabalhos de implementação abaixa muito a média final dos projetos práticos, e se ela for menor que 6, não é permitida prova de recuperação. Ou seja: caso seja detectado plágio, há o risco direto de reprovação. Os trios deverão desenvolver o trabalho sozinhos.