Universidade Federal do Rio Grande do Sul **Inteligência Artificial**
Instituto de Informática **Prof. Joel Luís Carbonera**
Departamento de Informática Aplicada

Trabalho 4
Busca com Adversário

Instruções preliminares

Este trabalho envolve a implementação de algoritmos de busca com adversário em dois jogos. Um kit com os arquivos necessários para executar o trabalho está disponível no moodle (arquivo `kit_games.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: ao contrário dos trabalhos anteriores, o uso de assistentes de IA para programação não é proibido. Pelo contrário! É encorajado. Você leu certo! Pode usar o ChatGPT, Copilot, etc. Como é a primeira vez que isso está sendo testado (talvez em todo o INF), seu feedback é muito importante.

1. Os Jogos

Você implementará algoritmos para dois jogos: jogo da velha invertido (tic-tac-toe misere) e Othello (também conhecido como Reversi). Cada jogo é implementado em dois módulos (`board.py` e `gamestate.py`) em um pacote no kit. Jogo da velha invertido está no pacote `tttm` (abreviatura para tic-tac-toe misere) e Othello está no pacote `othello`.

O `gamestate.py` contém uma classe `GameState`, cujos objetos armazenam o tabuleiro no atributo `board` e o jogador a fazer a jogada no atributo `player` (um caractere, `B` para as pretas ou `W` para as brancas. Isso vale para ambos os jogos (jogo da velha invertido também é jogado com 'pretas' ou 'brancas' ao invés dos clássicos 'X' e 'O'). As funções de gamestate.py permitem consultar a lista de ações válidas, obter a lista de sucessores de um estado, consultar se o estado é terminal e quem venceu o jogo.

O `board.py` contém a classe `Board`, cujos objetos contém a representação interna do jogo. Ela somente será útil para calcular avaliações heurísticas na qual você precisará examinar o estado internamente.

1.1. Jogo da velha invertido

O tamanho do tabuleiro e as jogadas válidas são da mesma forma que na versão original, porém, no

invertido você PERDE ao alinhar 3 peças em um grid 3x3. Este é um jogo pequeno: o fator de ramificação começa em 9 e diminui em 1 a cada turno. A profundidade máxima da árvore do jogo é 9.

A seguir temos uma descrição mais detalhada do `tttm/board.py`. O tabuleiro é representado 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 de um estado de vitória das pretas (brancas alinharam 3 peças na diagonal).

```
[
“..W”,
“BWB”,
“WB.”
]
```

Em nosso sistema de coordenadas, 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 aquele mesmo estado acima.

```
  012 --> eixo x
0 ..W
1 BWB
2 WB.

|
v
eixo y
```

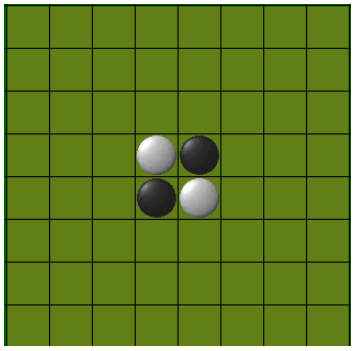
Uma jogada é dada por uma tupla com as coordenadas x, y (coluna, linha) da peça a ser colocada, de acordo com o sistema de coordenadas acima.

Para praticar o jogo e o sistema de coordenadas, use o servidor do kit com o comando (iniciará uma partida em que ambas as cores são controladas via terminal):

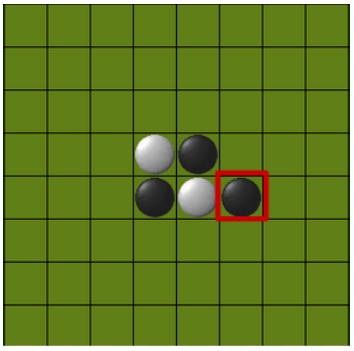
```
python server.py tttm advsearch/humanplayer/agent.py
advsearch/humanplayer/agent.py
```

1.2. Othello

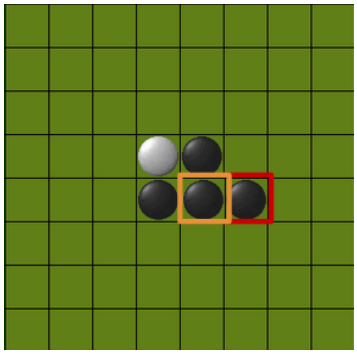
O tabuleiro é um grid de 8x8 posições cujas células centrais começam preenchidas por duas peças brancas e pretas, conforme a Figura 1 (a).



(a) Estado inicial



(b) Jogada válida (em



(c) Peça capturada (em laranja)

	vermelho)	

Figura 1 - Jogo de 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.

O link http://en.wikipedia.org/wiki/Reversi possui mais detalhes sobre regras, história, etc.
Para praticar online, você pode utilizar sites como este.

A seguir temos uma descrição mais detalhada do `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 (a).

```
[
“........”,
“........”,
“........”,
“...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
```

Uma jogada é dada por uma tupla com as coordenadas x, y (coluna, linha) da peça a ser colocada, de acordo com o sistema de coordenadas acima.

Para praticar o jogo e o sistema de coordenadas, use o servidor do kit com o comando (iniciará uma partida em que ambas as cores são controladas via terminal):

```
python server.py othello advsearch/humanplayer/agent.py
advsearch/humanplayer/agent.py
```

2. Os algoritmos

Você deverá implementar o algoritmo minimax com poda alfa-beta. O algoritmo deve ser implementado para ambos os jogos. Todas as implementações consistem em preencher as funções especificadas em determinados arquivos do kit.

2.1. Minimax com poda alfa-beta

Você deve implementar a poda alfa-beta na função `minimax_move` do arquivo `minimax.py`. A função recebe o estado do jogo, a profundidade máxima (-1 pra profundidade ilimitada) e uma função de avaliação (que você implementará em um momento posterior. Ela deve executar a poda alfa-beta até a profundidade máxima e chamar a função de avaliação quando encontrar um estado terminal ou nos estados na profundidade máxima.

Sua implementação da poda alfa-beta deve ser independente do jogo (ambos os jogos usam a mesma interface para consultar as jogadas e gerar os sucessores). A única parte que depende do jogo específico é a função de avaliação de estados, a qual é implementada em outro arquivo, conforme a seguir. Todas as funções de avaliação recebem o estado a ser avaliado e o jogador cujo ponto de vista deve ser considerado. Esse jogador deve ser o mesmo do estado raiz da árvore de busca, já que a decisão será feita para ele.

2.1.1. Jogo da velha invertido

Para o jogo da velha invertido, preencha a função `utility` do arquivo `tttm_minimax.py` com a função de avaliação. Assuma que ela avaliará estados terminais do jogo. Por fim, a função que retornará uma jogada para o servidor de jogos é a `make_move`. Ela receberá um estado e executará a poda alfa-beta sem limite de profundidade, passando sua função `utility` para avaliação dos estados. O jogo é pequeno e é esperado que seja possível atingir a profundidade máxima. Quando testada, sua função deve retornar uma jogada com o tempo limite de 1 minuto.

2.1.2. Othello

Othello é um jogo mais complexo, com fator de ramificação variável, mas com profundidade máxima de 64, o que torna inviável resolvê-lo completamente. Você implementará 3 heurísticas em arquivos diferentes.

(obrigatório) Em cada arquivo, você deverá preencher o `make_move` para executar sua poda alfa-beta com profundidade limitada e a respectiva heurística de avaliação implementada naquele mesmo arquivo. Você deve definir a profundidade máxima para retornar jogadas no tempo limite de 5 segundos. As 3 heurísticas são descritas a seguir.

(opcional) Você pode pensar em melhorias para a poda alfa-beta: por exemplo, busca com aprofundamento iterativo, poda antecipada, etc.

2.1.2.1. Heurística da contagem de peças

Essa heurística simplesmente retorna a diferença entre a quantidade de peças do jogador e do seu oponente. A poda alfa beta com essa heurística deve ser implementada na função `evaluate_count` do arquivo `othello_minimax_count.py`. O `make_move` desse arquivo deve chamar sua poda alfa-beta usando `evaluate_count` como função de avaliação.

2.1.2.2. Heurística do valor posicional

Essa heurística atribui valores para posições do tabuleiro (e.g. quinas valem muito, pré-quinas valem pouco) e retorna a diferença de valores das posições ocupadas pelo jogador e o oponente. A poda alfa beta com essa heurística deve ser implementada na função `evaluate_mask` do arquivo `othello_minimax_mask.py`. O valor das posições é uma matriz pré definida que serve como máscara para o tabuleiro. Não mude esses valores. O `make_move` desse arquivo deve chamar sua poda alfa-beta usando `evaluate_mask` como função de avaliação.

2.1.2.3. Heurística customizada

Use a heurística que quiser. A poda alfa beta com essa heurística deve ser implementada na função `evaluate_custom` do arquivo `othello_minimax_custom.py`. O `make_move` desse arquivo deve chamar sua poda alfa-beta usando `evaluate_custom` como função de avaliação.

2.3. Avaliação

Para o Tic-Tac-Toe misere, relate se o desempenho da sua implementação do minimax com poda alfa-beta: embora não seja simples provar que o jogo está sendo jogado com perfeição, avalie as evidências:

(i) O minimax sempre ganha do randomplayer?
(ii) O minimax sempre empata consigo mesmo?
(iii) O minimax sempre empata contra as jogadas perfeitas recomendadas pelo https://nyc.cs.berkeley.edu/uni/games/ttt/variants/misere ? Para verificar isso, use o humanplayer. No link, faça as jogadas do minimax, e no servidor do kit, faça as jogadas recomendadas (amarelo ou verde) do link.

Para o Othello, faça um mini-torneio (intercalando quem começa a partida) utilizando as diferentes heurísticas de todos contra todos.

(i) Represente em uma matriz de 3 X 3 onde as linhas representam o jogador que inicia (player 1) e as colunas representam o player 2 e em cada célula, indique se a partida resultou em vitória (1), derrota (-1) ou empate (0) entre os agentes com cada uma das heurísticas.
(ii) Observe e relate qual implementação foi a mais bem-sucedida.
(iii) Reflita sobre o que pode ter tornado cada heurística melhor ou pior, em termos de performance.

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.

3. Entrega

O kit do trabalho contém um diretorio "`your_agent`", no qual você deve fazer sua implementação.

Você pode criar arquivos e pacotes auxiliares dentro do seu diretório, mas você deve preencher as funções dos arquivos especificados, já que elas serão testadas. Não mude a assinatura (parâmetros e

retorno) das funções. Isso poderá inviabilizar a execução e/ou a correção das mesmas.

Ao final, gere um arquivo `.zip` contendo sua implementação e um arquivo `Readme.md` com um 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;
- Resultado da sua avaliação dos algoritmos (ver item 2.3);
- Feedback: quão fácil ou difícil foi realizar o trabalho? Utilizaram o auxílio da IA para o desenvolvimento do trabalho? Como foi trabalhar com o auxílio da IA? Quais sugestões teria para melhorar o trabalho?

```
your_agent<-- diretorio na raiz do .zip
|-- __init__.py
|-- minimax.py           <-- implementação da poda alfa-beta
|-- othello_minimax_count.py   <-- heuristica de contagem
|-- othello_minimax_custom.py  <-- heuristica customizada
|-- othello_minimax_mask.py    <-- heuristica posicional
|-- tttm_minimax.py    <-- minimax que joga o tic-tac-toe misere
|-- Readme.md <-- com seu relatorio
\-- [outros arquivos e subdiretorios de sua implementacao]
```

Sua implementação será executada na mesma estrutura de diretórios do kit do trabalho.

4. Critérios de correção

Item	**Percentual da nota**
Aderência à especificação (e.g. estrutura de diretórios, protocolo de jogadas)	10
Relatório	30
Implementação do minimax para o Tic-Tac-Toe misere	15
Implementação do minimax para Othello	15
Implementação da heurística contagem de peças	10
Implementação da heurística valor posicional	10
Implementação da heurística customizada	10
Extras (melhorias no minimax)	10
Total	**110**

A nota satura em 100, mas atividades extras podem compensar eventuais problemas nos demais itens.

Observações gerais

- O trabalho deve ser feito em grupos.
- O tempo de 5 segundos é estipulado tendo como referência uma máquina linux com a seguinte configuração: processador Xeon E5-2650 Sandy Bridge (Q1'12), 2,0 GHz e 4Gb de memória RAM (possivelmente essa será a configuração do computador do torneio).
- Fiquem atentos à política de plágio!
- A nota depende do correto funcionamento da implementação e de um bom relatório.

Dicas

- Em algumas ocasiões, o mesmo jogador pode jogar duas (ou mais) vezes seguidas, pois o oponente fica sem jogadas. Ajuste sua implementação da poda alfa-beta ou MCTS para verificar e tratar isso apropriadamente.
- Leia o `README.md` do `kit_games.zip`, ele contém instruções para a execução do servidor e do jogador 'random'.
- 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

O uso de assistentes de IA para programar é encorajado, mas os grupos não podem copiar uns aos outros. Os grupos 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.

Usamos rotineiramente um sistema anti-plágio que compara o código-fonte desenvolvido pelos grupos uns com os outros e com soluções enviadas em edições passadas da disciplina.

Qualquer nível de plágio entre grupos poderá resultar em nota zero no trabalho. 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.