Instruções

- Instalar Docker Desktop, disponível em: https://www.docker.com/products/docker-desktop/
- Utilizar base de código em Python, disponível em: https://bitbucket.org/luisteofilo/ai-solve-games
- Seguir as instruções disponíveis no ficheiro README.md

Jogo do Galo com Tabuleiro de Dimensões Variáveis

Objetivo: Implementar o jogo do galo (também conhecido como Tic Tac Toe) com um tabuleiro de dimensões variáveis (mínimo 3x3) usando o algoritmo minimax e uma base de código em Python.

Regras do jogo:

1. Os jogadores alternam-se para colocar as suas peças (X ou O) no tabuleiro.
2. O vencedor é o jogador que conseguir formar maior número de sequências de 3 peças seguidas na horizontal, vertical ou diagonal.
3. O jogo termina quando o tabuleiro está completamente preenchido ou quando não for possível adicionar novas sequências.

Algoritmo Minimax:

O algoritmo minimax é um método utilizado em jogos com dois jogadores para determinar a melhor jogada, levando em consideração as possíveis respostas do adversário. Este explora a árvore de estados do jogo, alternando entre os níveis "min" e "max", onde "min" representa o pior resultado para o jogador que está a jogar, e "max" representa o melhor resultado.

Tarefas:

1. Utilizar a estrutura base do 4 em linha já implementado para servir como base ao jogo do galo. Para isso os alunos devem copiar o diretório “connect4” e criar um novo chamado “tictactoe”. Todas as classes devem ser renomeadas. Exemplo: Connect4State -> TicTacToeState ou Connect4Simulator -> TicTacToeSimulator.
2. Adaptar a classe **TicTacToeAction** para permitir jogar numa determinada coordenada do tabuleiro X e Y.
3. Adaptar a classe **TicTacToeState** implementando o construtor **__init__**. Este deverá inicializar uma grelha de TicTacToe vazia. A grelha será sempre quadrada pelo que apenas é preciso passar um único argumento para as dimensões do tabuleiro. O tamanho mínimo deve ser 3.
4. Adaptar a classe **TicTacToeState** implementando o método **validate_action**. Neste deveremos verificar se a jogada é valida, garantindo que a ação está dentro das dimensões do tabuleiro e a casa onde pretendermos jogar estar vazia.
5. Adaptar a classe **TicTacToeState** implementando o método **update**. Nesta deveremos atualizar o estado do tabuleiro mudando o valor da célula onde estamos a jogar na grelha. Após mudar o

valor, deveremos verificar se já existe um vencedor no jogo, criando uma variável na classe para o efeito. Finalmente, caso não haja vencedor, teremos de trocar o turno. O vencedor é determinado pela quantidade de 3 peças seguidas na horizontal, diagonal ou vertical, independentemente do tamanho do tabuleiro. Assim, o jogo só termina quando o tabuleiro estiver cheio ou se não for possível fazer mais nenhuma sequência de 3 peças.

6. Adaptar a classe **TicTacToeState** implementando o método **display** que deverá desenhar a grelha de jogo na consola tendo em conta o estado atual.
7. Adaptar a classe **TicTacToeState** implementando o método **is_finished** que deverá retornar verdadeiro caso o jogo já tenha terminado.
8. Adaptar a classe **TicTacToeState** implementando o método **get_acting_player** que deverá retornar o índice do jogador que está a jogar no turno atual (pode ser 0 ou 1).
9. Adaptar a classe **TicTacToeState** implementando o método **clone** que deverá criar um novo estado duplicando o atual.
10. Adaptar a classe TicTacToeState implementando o método **get_result** que deverá retornar o resultado do jogo, se disponível, para o índice do jogador (0 ou 1). Adaptar a classe **TicTacToeResult** se necessário.
11. Adaptar quaisquer outros métodos da classe **TicTacToeState** que ache necessário.
12. Adatar a classe **TicTacToeSimulator**. Em vez de se inicializar o tabuleiro no construtor com uma largura e altura, deverá ser apenas uma dimensão que indica o tamanho da grelha. Adaptar o método **init_game** para retornar um **TicTacToeState**.
13. No diretório **players** existem várias implementações de jogadores. A ideia é implementarmos várias estratégias:
    a. Implementar o jogador humano que imprime o estado do tabuleiro e pede um input ao utilizador para jogar. Esta implementação vai permitir jogarmos contra o computador.
    b. Implementar o jogador aleatório que selecionada uma posição aleatória do tabuleiro que esteja vazia e joga na mesma.
    c. Implementar o jogador **OffensiveGreedy** que deverá dar prioridade a jogar na casa que lhe dê mais peças seguidas. Se existirem várias hipóteses, o jogador deverá selecionar uma aleatória dentro das jogadas empatadas, dando prioridade às casas do meio do tabuleiro.
    d. Implementar o jogador **DefensiveGreedy** que deverá dar prioridade a jogar em casas adjacentes ao adversário. Quanto mais peças adjacentes do adversário, maior deverá ser a probabilidade de escolher essa casa. No caso de várias opções, deverá dar prioridade às casas mais próximas do meio do tabuleiro.
    e. Implementar o jogador **Minimax** com a heurística baseada na alínea c).
    f. Implementar o jogador **Minimax** com a heurística baseada na alínea d).
14. Adaptar o ficheiro main.py para correr simulações para o jogo do galo.