Atividade 2b: Pesquisa com Adversários/Jogos

João Gama Amaral (up201708805)
FEUP
Porto, Portugal
up201708805@fe.up.pt

I. Introdução

Esta atividade consiste na implementação do algoritmo Minimax no âmbito de resolver o jogo Connect-4.

II. Descrição do Problema

O jogo trata-se de um jogo de Pesquisa com Adversários. O objetivo do jogo é conseguir juntar 4 peças em linha.

III. Formulação do Problema

Representação do jogo: A linguagem utilizada foi Python. O jogo irá ser representado por uma matriz de tamanho 7x6. Nesta matriz as casas vazias irão ser representadas pelo número 0. Enquanto que as peças dos jogadores vão ser representadas pelos números 1 e 2.

Estado inicial: O estado inicial do jogo irá ser sempre o mesmo. Uma matriz de tamanho 6x7 com todas as casas preenchidas pelo valor 0.

Operadores: Existem sete opções possíveis: primeira coluna .. sétima coluna. Cada uma destas ações ocorre numa peça. Para que a ação possa ser realizada a coluna não pode estar preenchida com seis peças.

Teste objetivo: O teste objetivo irá verificar se o estado atual do jogo é um estado final, sendo que este é um estado no qual existem quatro peças de um jogador em linha.

IV. Funções implementadas

Para esta atividade foi-nos pedido para implementarmos três funções essenciais para as conclusões da atividade.

```
def nlinhas4(self, piece):

    linhas = 0

    for c in range(COLUMNS-3):
        for r in range(ROWS):
            if self.board[r][c] == piece and self.board[r][c+1] == piece and self.board[r][c+2] == piece and self.board[r][c+3] == piece:
                linhas += 1
    #vertical
    for c in range(COLUMNS):
        for r in range(ROWS-3):
            if self.board[r][c] == piece and self.board[r+1][c] == piece and self.board[r+2][c] == piece and self.board[r+3][c] == piece:
                linhas += 1
    #diagonal positiva
    for c in range(COLUMNS-3):
        for r in range(ROWS-3):
            if self.board[r][c] == piece and self.board[r+1][c+1] == piece and self.board[r+2][c+2] == piece and self.board[r+3][c+3] == piece
                linhas += 1
    #diagonal negativa
    for c in range(COLUMNS-3):
        for r in range(3, ROWS):
            if self.board[r][c] == piece and self.board[r-1][c+1] == piece and self.board[r-2][c+2] == piece and self.board[r-3][c+3] == piece
                linhas += 1

    return linhas

def nlinhas3(self, piece):...

def central(self, piece):

    points = 0

    for count in range(ROWS):
        if self.board[count][3] == piece:
            points += 2
        if self.board[count][2] == piece:
            points += 1
        if self.board[count][4] == piece:
            points += 1

    return points
```

V. Algoritmo Minimax

```
def minimax(game, depth, alpha, beta, maximizingPlayer):

    if depth == 0 or game.winning_move(1) or game.winning_move(2):
        return None, None, Agente4(game)

    valid_locations = game.get_valid_locations()

    if maximizingPlayer:
        value = -math.inf
        children = game.children(1)

        column = random.choice(valid_locations)
        row = game.is_valid_position(column)

        for child in children:

            new_score = minimax(child[0], depth - 1, alpha, beta, False)

            if new_score[2] > value:
                value = new_score[2]
                column = child[2]
                row = child[1]

            alpha = max(alpha, value)
            if beta <= alpha:
                break
        return column, row, value

    else:
        value = math.inf
        children = game.children(2)

        column = random.choice(valid_locations)
        row = game.is_valid_position(column)

        for child in children:
            new_score = minimax(child[0], depth - 1, alpha, beta, True)
            if new_score[2] < value:
                value = new_score[2]
                column = child[2]
                row = child[1]
            beta = min(beta, value)
            if beta <= alpha:
                break
        return column, row, value
```

VI. Comparação de resultados

Em cada célula é apresentado o agente que conseguiu a vitória.

.Depth: 1

Agentes	1	2	3	4
1		Agente 2	Agente 3	Agente 4
2			Agente 1	Agente 4
3				Agente 3
4				

Depth: 3

Agentes	1	2	3	4
1		Agente 2	Agente 3	Agente 4
2			Agente 3	Agente 4
3				Agente 3
4				

Depth: 5

Agentes	1	2	3	4
1		Agente 1	Agente 3	Agente 4
2			Agente 2	Agente 4
3				Agente 4
4				

Depth: 7

Agentes	1	2	3	4
1		Agente 2	Agente 3	Agente 4
2			Agente 3	Agente 4
3				Agente 3
4				