Nome: Fabio Grassiotto RA 890441

IA006 – 2S2023

Relatório – Uma Implementação do Algoritmo MinMax para o Jogo da Velha

1. Concepção teórica

O algoritmo MiniMax busca a minimização da perda em cenários de pior caso em jogos de zero-soma (ou seja, jogos onde a vitória de um dos lados implica na derrota do outro lado).

A implementação do algoritmo para a escolha do próximo movimento em um jogo é recursiva: um valor de avaliação é computado para cada jogada possível, e o algoritmo busca maximizar o valor mínimo obtido ao se explorar em uma árvore de jogadas todas as possibilidades de jogo.

Para o jogo da velha, existem sempre no máximo 9 possibilidades de jogo, e, portanto, o custo computacional necessário é baixo.

2. Concepção do projeto

Para a implementação deste projeto, foram seguidas as etapas abaixo:

2.1 Definição da linguagem de programação e busca de interfaces

Inicialmente procurei por implementações do jogo da velha em sites da internet de programação para definir qual seria a linguagem de programação mais apropriada assim como quais bibliotecas/toolkits são os mais utilizados nesse tipo de solução.

Notei, por exemplo, no site [2] que a linguagem de programação Python e a biblioteca Pygame [1] é comumente utilizada para a implementação de jogos simples, facilitando o gerenciamento de janelas, detecção de eventos como o acionamento de teclas e/ou clicks dos botões do mouse entre outras atividades.

Tendo essas informações como referência, busquei implementações de interface do jogo da velha no Github como por exemplo [3], onde uma interface simples é implementada sem qualquer integração com engines de IA ou do algoritmo MiniMax.

Verifiquei que a licença utilizada pela implementação em [3] é do tipo MIT. Este tipo de licença permite a utilização do código publicado para a implementação final deste exercício, desde que preservada a publicação da licença. Portanto, para evitar duplicação de trabalho, utilizei como base o código disponibilizado no repositório de [3] para a implementação da interface gráfica.

2.2 Implementação do algoritmo MiniMax

Após a obtenção da base de código da interface do jogo da velha, foram efetuadas algumas alterações mínimas nas imagens utilizadas e procedi à implementação do algoritmo MiniMax. Para tanto, uma nova classe denominada AI() foi implementada, que será detalhada na seção 3.

Para simplificar a implementação a ser realizada, o jogador humano sempre inicia a partida (utilizando o )e executa o algoritmo de maximização do ganho, enquanto que a AI, utilizando o  , busca a minimização do ganho do jogador humano.

2.3 Testes e Resultados

Testamos o comportamento da AI jogando algumas partidas, que estão documentadas na seção 4.

3. Código final comentado

Conforme descrito acima, foi criada uma nova classe para a implementação da tomada de decisão do algoritmo.

```
class AI:
    def __init__(self) -> None:
        pass
```

Esta classe possui uma função principal que inicia o processo de avaliação da IA, play(). Esta função é iniciada buscando qual é o melhor movimento disponível para o segundo jogador, ou seja, a IA.

```
    # Main Play Function
    def play(self, board):
        # Min-Max algorithm.
        bestEval = -1000 # evaluation
        bestMove = (-1, -1) # movement
```

Para avaliar qual é melhor movimento disponível, a função miniMax() é implementada, onde a avaliação do campo de jogo é realizada, retornado um valor.

```
        # Board search for the best move for the second player.
        for line in range(3):
            for col in range(3):
                if (board[line][col] == 0):
                    # position available
                    board[line][col] = 2
                    eval = self.miniMax(board, 0, False)
                    # reset move
                    board[line][col] = 0
```

```
                    # Was this the best move?
                    if (eval > bestEval):
                        bestMove = (line, col)
                        bestEval = eval
        return bestMove
```

A função principal do algoritmo miniMax, miniMax() é uma função recursiva que retorna os valores de avaliação para o conjunto de jogadas no campo de jogo. Esta função é iniciada procurando a jogada do segundo jogador (neste caso a AI).

```
    # Minimax algorithm
    def miniMax(self, board, depth, maxMove):
        # First evaluate board score
        myScore = self.evaluateBoard(board)
        # Maximizer won the game
        if (myScore == 10):
            return myScore
        # Minimizer won the game
        if (myScore == -10):
            return myScore
        # Draw
        if (self.anyMovesLeft(board) == False):
            return 0
```

Após as verificações de score acima (que permitem a saída do loop da função recursiva), é iniciado o processo de se encontrar o movimento que maximiza ou minimiza o score, de acordo com a jogada sendo efetuada.

Primeiro é feita a busca do movimento de maximização.

```
        # Maximizer move
        if (maxMove):
            bestMove = -1000
            for line in range(3):
                for col in range(3):
                    if (board[line][col] == 0):
```

```
                        # make the move
                        board[line][col] = 2
                        # Recursively find the best move possible
                        bestMove = max(bestMove, self.miniMax(board,
depth+1, not maxMove))
                        board[line][col] = 0
            return bestMove
```

Após o processo de maximização, é efetuada a busca do movimento de mimimização, para a criação da lista de jogadas no processo de recursão.

```
        # Minimizer move
        else:
            bestMove = 1000
            for line in range(3):
                for col in range(3):
                    if (board[line][col] == 0):
                        # make the move
                        board[line][col] = 1
                        # Recursively find the worst move possible (for the
opponent)
                        bestMove = min(bestMove, self.miniMax(board,
depth+1, not maxMove))
                        board[line][col] = 0
            return bestMove
```

A função de avaliação do jogo evaluateBoard() verifica qual é o score relativo ao campo do jogo. Assim, se o jogador humano obtém 3 jogadas em uma linha, coluna ou diagonal o score é +10; caso a IA obtenha as 3 jogadas, o score é -10; e caso contrário, a avaliação retorna zero.

```
    # Evaluation function
    def evaluateBoard(self, board):
        eval = 0
        # Check rows
        for row in range(3):
            if (board[row][0] == board[row][1] and board[row][1] ==
board[row][2]):
```

```
                if (board[row][0] == 1):
                    eval = -10
                    return eval
                elif (board[row][0] == 2):
                    eval = 10
                    return eval
        # Check cols
        for col in range(3):
            if (board[0][col] == board[1][col] and board[1][col] ==
board[2][col]):
                if (board[0][col] == 1):
                    eval = -10
                    return eval
                elif (board[0][col] == 2):
                    eval = 10
                    return eval
        # Check diags
        if (board[0][0] == board[1][1] and board[1][1] == board[2][2]):
            if (board[0][0] == 1):
                eval = -10
                return eval
            elif (board[0][0] == 2):
                eval = 10
                return eval
        if (board[0][2] == board[1][1] and board[1][1] == board[2][0]):
            if (board[0][2] == 1):
                eval = -10
                return eval
            elif (board[0][2] == 2):
                eval = 10
                return eval

        # in all other cases return default value = 0
        return eval
```

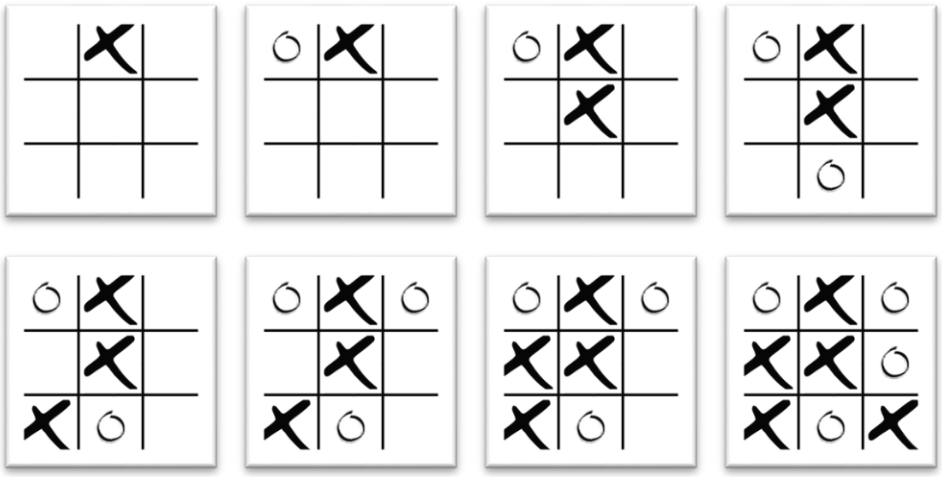
Esta última função anyMovesLeft() é uma função utilitária para verificar se existem movimentos a serem realizados ainda no campo do jogo.

```
    # Any Moves left
    def anyMovesLeft(self, board) :
        for line in range(3) :
            for col in range(3) :
                if (board[line][col] == 0) :
                    return True
        return False
```

4. Exemplos de partidas jogadas

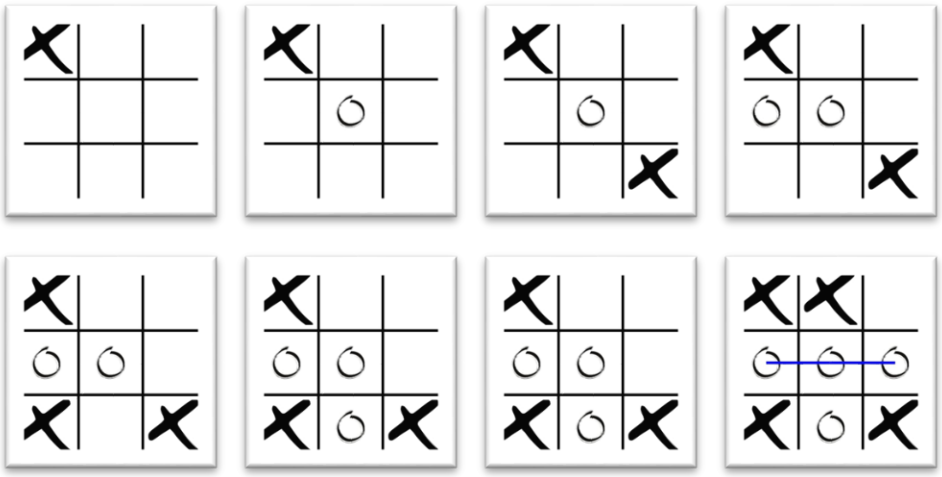
4.1 IA empata a partida

Neste primeiro exemplo o jogador humano tenta ganhar a partida. Podemos notar que a IA tende a jogar com o objetivo de bloquear a partida, ao invés de tentar ganhar, devido à maneira que é calculado o score do jogo.



4.2 IA vence a partida

Neste segundo exemplo a IA vence a partida, quando o jogador humano força um erro para que IA vença. Notamos, no entanto, que uma vez que no algoritmo MiniMax a IA pretende minimizar o score do jogador humano, ela perde uma oportunidade de vitória na jogada 6, para bloquear o jogador humano. Apenas quando não existe essa necessidade, a IA consegue vencer.



5. Referências

[1] Biblioteca para programação de jogos Pygame: https://www.pygame.org/news

[2] Exemplo de implementação do jogo da velha: Tic Tac Toe GUI In Python using PyGame - GeeksforGeeks

[3] Implementação de interface para o jogo da velha: timotheeMM/tic-tac-toe: Tic Tac Toe game made with Python and Pygame (github.com)