Pesquisa de Publicação – Min-Max

Marco Bockoski – 7° Período de Engenharia de Computação

INTRODUÇÃO

Essa pesquisa é uma síntese da publicação “Entendendo o algoritmo Minimax com o Jogo da Velha” no site Medium, o autor da publicação é Lucas Eliaquim, um Analista de Data Science da Empresa Sysvale Softgroup. Essa empresa é uma empresa de Software que tem um perfil nesse site com várias publicações relacionadas à programação, desenvolvimento web, startups, dentre outros tópicos.

Como o nome sugere, essa pesquisa sobre o Min-Max é um exemplo de aplicação do algoritmo sobre o jogo da velha, e a proposta do autor é ensinar como implementar, tal como ressaltar aspectos de melhoria do algoritmo e ilustrar aspectos do algoritmo funcionando na prática.

ATIVIDADES DESENVOLVIDAS

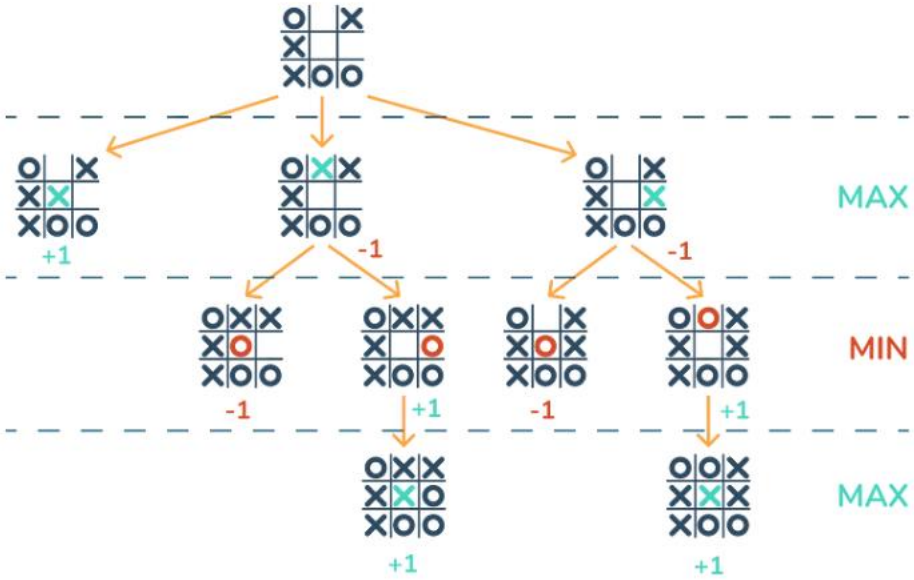
Apresentando o tópico de sua publicação, o autor ressalta a importância de conhecer pré-requisitos para entendimento pleno da explicação, sendo eles: conhecimento da linguagem Python, recursividade; e estrutura de dados com enfoque em árvores.

Na seção “preparando o jogo da velha”, o autor expõe sua implementação de critérios de empate, vitória, verificação de fim de jogo e exibição do tabuleiro. De maneira bem simplificada e breve, esses critérios são vários condicionais e a exibição do tabuleiro é exibição pelo console.

No tópico “O minimax”, o autor começa explicando em qual área é o escopo desse algoritmo, sendo na parte de tomada de decisão e jogos de tabuleiro um contra um, ele considera a melhor jogada para seu oponente e reage com a melhor jogada para seu jogador.

Esse algoritmo corresponde a uma árvore recursiva, que traça todas as possibilidades de jogada e toma a decisão baseada na mais próxima à vitória. O autor explica toda a lógica de comportamento adotando MAX como jogador e MIN como oponente, na qual, todo movimento de MAX deve ser destino à vitória do jogador, enquanto MIN faz as jogadas para minimizar as chances do jogador de ganhar, convencionam-se valores para a tomada de decisão da Inteligência Artificial (IA) dentro da árvore, como +1 para vitória de MAX, -1 para vitória de MIN e 0 para empate. Assim a árvore presente na Figura 1 ilustra perfeitamente como a árvore é construída e como a decisão é tomada (sempre escolhendo escolhas +1).

Figura 1 - Árvore Recursiva Min-Max



A parte subsequente do artigo chama-se “Implementando a abordagem ingênua” referente ao algoritmo Min-Max, o autor denomina desse jeito por ser uma implementação simplista do Min-Max sem considerar fatores de otimização.

O autor implementa uma função de candidatos (possíveis jogadas tanto do jogador quanto do adversário) que, na prática, gera uma lista de estados, cada um com jogadas possíveis nos espaços em brancos. Cada lista recebe uma avaliação se ele aproxima o jogador da vitória (+1), afasta (-1) ou indefere (0).

Ele explica em etapas a função minimax como função recursiva, tendo argumentos como o estado e uma variável lógica para verificar se é para maximizar (vez do jogador) ou minimizar (vez do oponente). Condição de parada utiliza-se a função de verificação de fim de jogo, caso contrário verifica o estado lógico da variável min-max para ver se deve maximizar ou minimizar, gera os candidatos e pega o melhor dentre eles. Ao chamar recursivamente outra função minimax, ele sempre inverte o valor lógico, para provocar o efeito causado pela Figura 1, de turnos que cada jogador tem para jogar. Finalizando com a implementação do autor no método principal para evocar o método.

A seção “Otimizando com a poda Alfa-Beta” do artigo é a justificativa do título da seção anterior, o autor explica que esse método serve para reduzir a quantidade de nós da árvore descartando possibilidades inviáveis ou impossíveis, por meio da não expansão desse nó. Objetivamente falando, a poda serve para reduzir o número de estados candidatos.

Nesse método, considera-se alfa como o melhor MAX e beta como melhor MIN, deste modo, valores menores que alfa e maiores que beta não tem expansão (implementados por condicionais). Uma mudança é notável no gasto de chamadas recursivas durante a implementação da poda, o autor apura a redução de 94%.

O último tópico “Aprimorando ainda mais com heurísticas” apresenta a introdução das heurísticas para evitar o número total de expansões até o tabuleiro do jogo se encher por completo. O autor explica sobre as questões de utilização de heurística, do quão ajudam na otimização do algoritmo em relação à redução do problema em 90%. A implementação da heurística segue a lógica da Figura 2 e é implementada conforme a Figura 3.

Figura 2 - Critérios da Heurística

- Se o computador venceu, retorne Infinito positivo.
- Se o computador perdeu, retorne Infinito negativo.
- Para cada 2 símbolos na linha, coluna ou diagonal em favor do computador (uma posição vazia) soma +10
- Para 1 símbolo na linha, coluna ou diagonal em favor do computador (duas posições vazia) soma +1
- Pontuações negativas, -10 e -1 considerando as mesmas situações, só que a favor do oponente
- Caso contrário soma 0 (sequências vazias ou com simbolos do computador e do oponente ao mesmo tempo)

Figura 3 - Implementação da Heurística (Python)

```
def evaluate_heuristic_sequence(sequence):
    number_of_blank = sequence.count(' ')
    has_two_simbols = (CPU_PLAYER in sequence and HUMAN_PLAYER in
sequence)

    if number_of_blank == 3 or has_two_simbols:
        return 0

    if number_of_blank == 2:
        return 1 if CPU_PLAYER in sequence else -1

    return 10 if CPU_PLAYER in sequence else -10


def heuristic(board):
    if is_win(board, CPU_PLAYER):
        return float('+inf')

    if is_win(board, HUMAN_PLAYER):
        return float('-inf')

    score = 0

    # Avaliando as linhas do tabuleiro
    for line in board:
        score += evaluate_heuristic_sequence(line)

    # Avaliando as colunas do tabuleiro
    for j in range(len(board)):
        column = [board[i][j] for i in range(len(board))]
        score += evaluate_heuristic_sequence(column)

    # Avaliando as diagonais do tabuleiro
    main_diag = [board[i][i] for i in range(len(board))]
    secondary_diag = [board[i][j] for i, j in zip(range(3), range(2,
-1, -1))]

    score += evaluate_heuristic_sequence(main_diag)
    score += evaluate_heuristic_sequence(secondary_diag)

    return score
```

Apesar dessa publicação apresentar 3 variações do método de busca competitiva Minimax (Básico, Poda Alfa-beta e Heurísitca), cada uma é

entendida como aprimoramento da outra, portanto, a Figura 4 apresenta a implementação de minimax com poda alfa-beta juntamente de heurística.

Figura 4 - Implementação do Min-Max (Poda Alfa-beta e Heurística)

```
def minimax_heuristic(board, depth=2, alpha=float('-inf'),
                      beta=float('+inf'), maximizing=False):
    if depth==0 or is_terminal(board):
        return heuristic(board)

    if maximizing:
        value = float('-inf')

        for child in candidates(board, CPU_PLAYER):
            value = max(
                value,
                minimax_heuristic(child, depth - 1, alpha, beta,
False)
            )

            if value >= beta:
                break

            alpha = max(alpha, value)
    else:
        value = float('+inf')

        for child in candidates(board, HUMAN_PLAYER):
            value = min(
                value,
                minimax_heuristic(child, depth - 1, alpha, beta,
True)
            )

            if value <= alpha:
                break

            beta = min(beta, value)

    return value
```

O autor conclui com uma síntese de sua publicação: implementar jogo da velha, explicar minimax, implementar no jogo suas 3 versões.

TABELA DE FORMULAÇÃO DO PROBLEMA

No intuito de responder aspectos importantes da metodologia Min-Max dado o problema da publicação, a Tabela 1 é construída para respondê-las de modo objetivo.

Tabela 1 – Formulação do Problema

Aspecto do Minimax	Descrição	Implementação
Estados	Disposição de símbolos no tabuleiro indicando diferentes combinações possíveis	Matriz 3x3 de char com certa disposição de símbolo
Ações	Escolhe possibilidades de um símbolo ocupar certa posição do tabuleiro	Função Candidatos, conjunto de estados (matrizes) com diferentes disposições si, em relação ao novo símbolo
Função de Avaliação	Dado critérios, percebe se o jogador ou o adversário está na vantagem ou se há empate	Função Heurística presente na Figura 3
Corte Alfa-Beta	Elimina a expansão de nós impossíveis na árvore de possibilidade de jogadas	Condicionais do Min-Max, presente como argumento de função e dentro do “loop for” da Figura 4

CONCLUSÃO

A publicação é uma boa fonte de aprendizado teórico e prático do algoritmo Min-Max, explicações bem detalhadas e fundamentadas, sem exigir muito conhecimento técnico e aprofundado, é um material demasiadamente didático.

BIBLIOGRAFIA

ELIAQUIM, L. Entendendo o algoritmo Minimax com o Jogo da Velha. Medium, 2022. Disponivel em: <https://medium.com/sysvale/entendendo-o-algoritmo-minimax-com-o-jogo-da-velha-e7945bb908bd>. Acesso em: 30 maio 2024.