UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

Título: Análise comparativa do Jogo da Velha com Minimax com Tabela de Transposição (MTT) vs Minimax com Poda Alfa-Beta

Discentes: Gabriel Pereira, Gabriel Neneve dos Santos, Thalita Wiederkehr Pereira e Vinicius Mattos Marcos

Docente: Prof. Dr. Adriana Postal

Data: 20/07/2025

---

1. Introdução

O presente trabalho apresenta o desenvolvimento e a análise comparativa de dois algoritmos que utilizam métodos de busca aplicados ao jogo da velha, com o objetivo de avaliar estratégias de otimização na tomada de decisão em jogos adversários. Este estudo foi realizado no contexto de uma competição entre agentes inteligentes, conforme as diretrizes da disciplina de Inteligência Artificial.

O jogo da velha é um clássico jogo de tabuleiro, caracterizado por ser de soma zero e de informação perfeita, disputado entre dois jogadores que alternam turnos para marcar espaços em um tabuleiro 3x3. O objetivo de cada jogador é alinhar três de suas marcas — ‘X’ ou ‘O’ — em sequência, seja na horizontal, vertical ou diagonal. A partida pode terminar com a vitória de um dos jogadores ou com um empate, caso todas as casas sejam preenchidas sem que se forme uma linha vencedora. Segundo Bem-El-Mechaiekh e Dimand (2005), o jogo da velha é um exemplo clássico de um jogo que pode ser aplicado através do teorema Minimax, que fundamenta a tomada de decisão em cenários de competição.

Para a realização da competição entre algoritmos, foram desenvolvidas duas abordagens baseadas no algoritmo Minimax:

Minimax com Tabela de Transposição (MTT): Esta abordagem utiliza uma estrutura de memorização para evitar a reavaliação de estados repetidos, reduzindo assim o custo computacional.

Minimax com Poda Alfa-Beta: Esta é uma técnica clássica de otimização que elimina, durante a busca, ramos que não afetam o resultado final, mantendo a correção e a profundidade da estratégia.

As partidas foram realizadas automaticamente por meio de uma interface gráfica desenvolvida com Pygame, que permitiu o registro de estatísticas como o número de vitórias, empates e o tempo médio por jogo. O ambiente de jogo foi projetado para alternar os papéis dos jogadores a cada rodada, garantindo imparcialidade na ordem de início. O conjunto de análises busca refletir não apenas a eficácia estratégica dos algoritmos, mas também sua eficiência computacional.

Assim, este trabalho visa não apenas comparar o desempenho das duas variações do Minimax em um jogo resolvido, mas também explorar o impacto de diferentes técnicas de otimização em ambientes com características de decisão determinística. A análise dos resultados permitirá uma compreensão mais profunda das vantagens e desvantagens de cada abordagem, contribuindo para o avanço no campo da inteligência artificial aplicada aos jogos.

2. Metodologia

2.1 Algoritmos Escolhidos e Justificativa

Ambas as equipes escolheram o algoritmo de busca Minimax. O Minimax é um algoritmo baseado em árvore que trabalha de forma recursiva. Ele fornece um movimento ótimo para o jogador, considerando que o oponente também joga de forma ótima. Ele é geralmente utilizado para implementar IA em jogos, como Xadrez, Jogo da Velha e vários outros jogos de dois jogadores.

2.1.1 Minimax com Tabela de Transposição (Thalita e Vinicius)

A Tabela de Transposição funciona como uma heurística que impede o algoritmo Minimax de recalcular estados do jogo que já foram analisados anteriormente, acelerando significativamente o processo de decisão. Ao evitar chamadas recursivas desnecessárias, o algoritmo economiza tempo e recursos, permitindo explorar mais jogadas futuras, tomar decisões de melhor qualidade e adotar estratégias mais eficazes.

Essa otimização é especialmente útil em jogos como o Jogo da Velha, que, apesar de ser mais simples do que jogos como o Xadrez, ainda apresenta muitos

estados possíveis. Com a redução drástica na quantidade de cálculos realizados, o algoritmo se torna muito mais eficiente e responsivo.

O pseudocódigo abaixo mostra como o Minimax com Tabela de Transposição funciona:

```
ALGORITMO MTT:
    Inicializa:
        letra ← símbolo do jogador (X ou O)
        transposicao ← tabela vazia
        nos_explorados ← 0

FUNÇÃO oponente(letra):
    Retorna 'O' se letra for 'X', senão retorna 'X'

FUNÇÃO escolher_jogada(jogo):
    Zera nos_explorados
    Limpa a tabela de transposição

    jogadas ← jogo.jogadas_disponiveis()

    SE número de jogadas for 9:
        Retornar um canto aleatório entre [0, 2, 6, 8]
    SENÃO:
        (pontuacao, jogada) ← MINIMAX(jogo, letra, profundidade = 0)
        Retornar jogada

FUNÇÃO estado_hash(estado):
    Retornar uma string representando o estado atual do tabuleiro

FUNÇÃO MINIMAX(estado, jogador, profundidade):
    Incrementa nos_explorados

    chave ← estado_hash(estado)
    SE chave estiver na transposicao:
        Retornar transposicao[chave]

    SE estado.vencedor_atual == letra:
        Retornar (10 - profundidade, NULO)
    SENÃO SE estado.vencedor_atual == oponente(letra):
        Retornar (-10 + profundidade, NULO)
```

```
SENÃO SE NÃO há espaços vazios:
        Retornar (0, NULO)

SE jogador == letra (MAX):
        melhor_pontuacao ← -infinito
        melhor_movimento ← NULO

        PARA cada movimento em jogadas_disponiveis():
                Salvar vencedor_antes ← estado.vencedor_atual
                estado.fazer_jogada(movimento, jogador)

                (pontuacao, _) ← MINIMAX(estado, oponente(letra), profundidade
        + 1)

                estado.desfazer_jogada(movimento)
                estado.vencedor_atual ← vencedor_antes

                SE pontuacao > melhor_pontuacao:
                        melhor_pontuacao ← pontuacao
                        melhor_movimento ← movimento

SENÃO (MIN - oponente):
        melhor_pontuacao ← +infinito
        melhor_movimento ← NULO

PARA cada movimento em jogadas_disponiveis():
        Salvar vencedor_antes ← estado.vencedor_atual
        estado.fazer_jogada(movimento, jogador)

        (pontuacao, _) ← MINIMAX(estado, letra, profundidade + 1)

        estado.desfazer_jogada(movimento)
        estado.vencedor_atual ← vencedor_antes

                SE pontuacao < melhor_pontuacao:
                melhor_pontuacao ← pontuacao
                melhor_movimento ← movimento

Armazenar (melhor_pontuacao, melhor_movimento) em transposicao[chave]
Retornar (melhor_pontuacao, melhor_movimento)
```

2.1.2 Minimax com Poda Alfa-Beta

O minimax com poda Alfa-Beta funciona da seguinte maneira:

Classe JogadorAlphaBeta:
- letra: representa o símbolo do jogador ('X' ou 'O')
- nos_explorados: contador de nós analisados no processo de decisão

Método oponente:
Retorna o símbolo oposto ao do jogador atual

Método escolher_jogada:
    Inicializa nos_explorados como zero
    Obtém todas as jogadas disponíveis no tabuleiro

    Se o tabuleiro estiver vazio (9 posições livres):
        Retorna o centro do tabuleiro (posição 4)
    Inicializa melhor_pontuacao com o pior valor possível
    Inicializa melhor_jogada como indefinida

    Para cada jogada possível:
        Simula a jogada com o símbolo do jogador
        Calcula a profundidade restante após a jogada
        Chama o método alphabeta com:
            - estado atual
            - profundidade
            - valores iniciais de alpha (-∞) e beta (+∞)
            - maximizando_jogador = False (próximo turno será do oponente)
        Desfaz a jogada
        Limpa o vencedor atual

        Se a pontuação for melhor que melhor_pontuacao:
            Atualiza melhor_pontuacao e melhor_jogada
    Retorna a melhor jogada encontrada

Método alphabeta(estado, profundidade, alpha, beta, maximizando_jogador):

Incrementa o contador de nós explorados

    Se há um vencedor:
        Retorna pontuação positiva ou negativa com base no jogador vencedor

```
    Se não há mais espaços ou profundidade chegou a zero:
        Retorna a avaliação do tabuleiro

    Se é a vez do jogador maximizar:
        Inicializa melhor_valor como -∞
        Para cada jogada disponível:
            Simula jogada do jogador
            Chama alphabeta recursivamente, agora minimizando
            Desfaz jogada e limpa vencedor

            Atualiza melhor_valor com o maior valor entre atual e retorno
            Atualiza alpha com o maior entre alpha e melhor_valor
            Se beta ≤ alpha: interrompe o loop (poda beta)

Retorna melhor_valor
    Senão (minimizando oponente):
        Inicializa melhor_valor como +∞
        Para cada jogada disponível:
            Simula jogada do oponente
            Chama alphabeta recursivamente, agora maximizando
            Desfaz jogada e limpa vencedor

            Atualiza melhor_valor com o menor valor entre atual e retorno
            Atualiza beta com o menor entre beta e melhor_valor
            Se beta ≤ alpha: interrompe o loop (poda alfa)

        Retorna melhor_valor
```

2.2 Detalhes da implementação

2.2.1 Minimax com Tabela de Tranposição (Thalita e Vinicius)

Como mencionado anteriormente, o algoritmo de busca escolhido por essa equipe foi o Minimax. Para otimizar o desempenho do Minimax, foi adotada a heurística da Tabela de Transposição.

A Tabela de Transposição, no contexto do jogo da velha, funciona como uma "memória" que armazena os estados do tabuleiro já avaliados durante a execução do algoritmo. Ao registrar esses estados em uma estrutura de dados, como um dicionário ou tabela hash, o algoritmo pode simplesmente reutilizar o valor já calculado quando se depara com o mesmo estado novamente, evitando o processamento redundante.

Porém, esse tipo de algoritmo também possui limitações. A principal delas é o uso de memória, já que a tabela precisa armazenar todos os estados visitados durante a execução. Em jogos mais complexos, isso pode acarretar um consumo excessivo de recursos.

2.2.2 Minimax com Poda Alfa-Beta (Gabriel Pereira e Gabriel Neneve)

A Poda Alfa-Beta é uma otimização crucial para o algoritmo Minimax que aumenta drasticamente sua eficiência. Basicamente, ela permite que o Minimax "pode" (elimine) ramos da árvore de busca que não influenciarão a decisão final, evitando cálculos desnecessários. Isso acontece porque, durante a análise das jogadas, se o algoritmo perceber que um caminho alternativo já oferece um resultado melhor (ou pior, dependendo da perspectiva do jogador), ele não precisa continuar explorando aquele ramo específico, pois ele já sabe que não será escolhido.

Essa técnica é particularmente útil em jogos como o Jogo da Velha, mesmo que pareça simples, pois a Poda Alfa-Beta consegue reduzir exponencialmente o número de nós que o algoritmo precisa visitar. Ao ignorar partes irrelevantes da árvore de busca, o Minimax com Poda Alfa-Beta economiza tempo e recursos computacionais, permitindo que o algoritmo explore um número maior de jogadas em menos tempo. Isso resulta em decisões mais rápidas e, em muitos casos, na capacidade de o algoritmo alcançar uma profundidade de busca maior, levando a um jogo mais estratégico e eficaz.

3. Critérios de vitória

A eficácia dos métodos será avaliada pela capacidade de conquistar o maior número de vitórias em uma série de partidas. Para garantir a equidade e neutralizar possíveis influências relacionadas à ordem de jogada (se o jogador é o primeiro ou o segundo), a competição será realizada no formato “melhor de N”, com a alternância sistemática dos papéis de primeiro e segundo jogador (jogando como 'X' e 'O') a cada rodada.

4. Métricas de comparação entre os Algoritmos

A análise quantitativa será fundamentada nas seguintes métricas:

- **Vitórias e Derrotas**: Contaremos o total de vitórias e derrotas de cada algoritmo. Essa métrica mostra diretamente a eficácia de cada abordagem em situações competitivas.
- **Empates**: Registraremos o número total de partidas empatadas. Em jogos resolvidos, como o Jogo da Velha, onde jogadas ótimas tendem ao empate, essa métrica é importante para avaliar a capacidade dos algoritmos de evitar derrotas e manter um bom desempenho.
- **Tempo Médio por Jogada**: Embora possamos monitorar o tempo total por partida, o tempo médio por jogada é uma métrica mais significativa da eficiência de cada algoritmo. Assim, refletindo a complexidade computacional na tomada de decisão e permitindo avaliar o impacto do uso de cada uma das técnicas na redução do tempo de processamento.

5. Resultados

A análise dos resultados obtidos após cinco testes, com cinco partidas cada um, na competição entre o algoritmo Minimax com Poda Alfa-Beta e o Minimax com Tabela de Transposição (MTT), revela aspectos relevantes tanto na qualidade das decisões quanto na eficiência computacional de cada abordagem.

Em termos de qualidade da jogada, os dois algoritmos demonstraram um desempenho ótimo, resultando em empate em todas as partidas. Este resultado indica que as implementações de ambos os algoritmos foram capazes de encontrar a solução ideal para o problema, que é esperado para o algoritmo minimax, que por natureza, busca o movimento ideal considerando a melhor jogada do oponente.

Em relação à eficiência, a tabela de médias mostra uma clara vantagem para o algoritmo minimax com Tabela de Transposição (MTT). O Tempo de execução médio do MTT foi de 0,08704, significativamente menor que os 0,24008 do Minimax com Poda Alfa-Beta. Da mesma forma, o número de nós explorados pelo MTT foi de 2.947,4, quase metade dos 5.333,6 nós explorados pelo Minimax com Poda Alfa-Beta. Esta diferença de desempenho é explicada pela forma como cada algoritmo otimiza a busca em árvores.

Médias	Alpha Beta	MTT

Tempo de execução	0,24008	0,08704
Nós explorados	5.333,60	2.947,40
Taxa de vitórias	Empate em todas as partidas	

Tabela 1: Resultado obtidos após cinco testes com cinco partidas cada.

6. Análise crítica

6.1 Pontos fortes e fracos de cada algoritmo no contexto do problema:

6.1.1 Minimax com Poda Alfa-Beta:

- Pontos Fortes:
    - Qualidade da Solução: O algoritmo Minimax é um método de tomada de decisão usado em jogos para dois jogadores que busca um movimento ideal, assumindo que o oponente também joga de maneira ideal. A Poda Alfa-Beta é uma otimização que, embora elimine ramos da árvore de busca, garante que o algoritmo faça o mesmo movimento padrão que o Minimax sem poda. Consequentemente, a taxa de vitórias que resultou em empate em todas partidas demonstra que esta implementação é capaz de jogar de forma ótima para o Jogo da Velha, encontrando a melhor solução que exista.
    - Eficiência da Busca (em relação ao Minimax puro): A Poda Alfa-Beta é um método de busca heurística que acelera o processo de busca ao reduzir o número de nós que precisam ser visitados na árvore. Isso é alcançado ao podar ramos que não podem influenciar a decisão final.
- Pontos Fracos
    - Eficiência Limitada em comparação ao MTT: Apesar de ser uma otimização do Minimax, o algoritmo com Poda Alfa-Beta ainda apresentou um tempo de execução e um número de nós explorados consideravelmente maiores do que o MTT para o mesmo problema. Para o Jogo da Velha, seu tempo médio foi de 0,24008 e explorou 5.333,6 nós. Isso se deve à sua incapacidade de memorizar e reutilizar cálculos para estados de jogo repetidos que são alcançados por diferentes caminhos na árvore de busca.

6.1.2 Minimax com Tabela de Transposição (MTT):

- Pontos Fortes:
- Qualidade da Solução: Assim como o Minimax com Poda Alfa-Beta, o MTT também demonstrou desempenho ótimo, resultando em empate em todas partidas. Isso mostra que o algoritmo encontrou a solução ideal para o Jogo da Velha.
- Eficiência Computacional Superior: O MTT se destacou significativamente em eficiência. Seu tempo de execução médio foi de 0,08704, sendo

aproximadamente 2,75 vezes mais rápido que o Minimax com Poda Alfa-Beta. Além disso, explorou quase metade dos nós (2.947,4) em comparação com o concorrente.

- Reutilização de Cálculos: A principal vantagem do MTT está em sua capacidade de armazenar e reutilizar os resultados de estados de jogo (posições do tabuleiro) que já foram avaliados. Ao evitar o recálculo de subárvores idênticas, ele reduz drasticamente a redundância na busca, levando a uma exploração de nós muito menor e um tempo de execução mais rápido.

◦ Pontos Fracos:

- Requisito de Memória: O uso de uma Tabela de Transposição implica em um consumo de memória para armazenar os estados já visitados e seus respectivos valores. Embora para o Jogo da Velha este não seja um problema crítico, em jogos com espaços de busca substancialmente maiores, o tamanho da tabela pode se tornar proibitivo, afetando a complexidade em termos de espaço do método.

6.2 Situações em que um algoritmo superou o outro e porquê

A competição demonstrou claramente que o Minimax com Tabela de Transposição (MTT) superou o Minimax com Poda Alfa-Beta em termos de eficiência computacional. Essa superioridade foi evidenciada pelas métricas de:

- Tempo de Execução: O MTT executou a busca em 0,08704 segundos em média, enquanto o Minimax com Poda Alfa-Beta levou 0,24008 segundos, tornando o MTT aproximadamente 2,75 vezes mais rápido.
- Nós Explorados: O MTT explorou uma média de 2.947,4 nós, o que é significativamente menor que os 5.333,6 nós explorados pelo Minimax com Poda Alfa-Beta.

Essa superação em eficiência se deve à diferença fundamental na forma como o MTT lida com estados de jogo repetidos. Enquanto a Poda Alfa-Beta otimiza a busca ao eliminar ramos irrelevantes que não afetam a decisão final, ela ainda pode visitar a mesma posição do tabuleiro múltiplas vezes se ela for alcançada por diferentes sequências de movimentos. O MTT, por outro lado, utiliza uma tabela para armazenar os resultados de posições já avaliadas. Ao encontrar uma posição previamente calculada, o algoritmo simplesmente consulta a tabela e reutiliza o valor armazenado, evitando o trabalho de reexplorar toda uma subárvore. Este mecanismo de memorização é o fator chave que conferiu ao MTT uma vantagem de desempenho tão expressiva para o Jogo da Velha, sem comprometer a ótima qualidade de jogo que ambos os algoritmos alcançaram.

7. Conclusão

Este estudo investigou a implementação e a eficácia de duas variantes do algoritmo Minimax no Jogo da Velha: o Minimax com Tabela de Transposição (MTT)

e o Minimax com Poda Alpha-Beta. A comparação dessas abordagens, realizada em um ambiente de competição simulada em Python, revelou importantes insights sobre suas performances e otimizações.

Os resultados confirmaram que, como esperado em jogos resolvidos, todas as partidas entre agentes ótimos terminaram em empates. Isso reitera que, com estratégias ideais, nenhum jogador pode obter uma vantagem decisiva.

A Poda Alpha-Beta demonstrou ser uma otimização altamente eficaz, reduzindo significativamente o número de nós explorados na árvore de busca sem comprometer a qualidade da decisão. Esse impacto gerou uma aceleração notável no processo de tomada de decisão, mesmo em um jogo de complexidade baixa. Por outro lado, a Tabela de Transposição, embora com impacto modesto devido à simplicidade e ao tamanho limitado da árvore de estados do Jogo da Velha, evitou recomputações redundantes. Sua relevância, no entanto, se torna mais evidente em jogos com espaços de estado maiores e árvores de busca mais profundas.

O desenvolvimento deste trabalho proporcionou um aprofundamento significativo na compreensão de algoritmos de tomada de decisão adversarial. A implementação do Minimax solidificou o entendimento dos princípios de maximização e minimização em jogos de soma zero. A Poda Alpha-Beta se destacou como uma técnica indispensável para otimizar o desempenho do Minimax, ao podar ramos da árvore de busca que não influenciam a decisão final.

Embora a Tabela de Transposição tenha mostrado impacto limitado no Jogo da Velha, ela se mostra crucial em jogos mais complexos, onde a repetição de estados é frequente e a economia computacional é vital. A análise também evidenciou que, em jogos resolvidos, o objetivo de um agente ótimo é evitar a derrota, o que naturalmente resulta em empates quando ambos os lados jogam perfeitamente.

A função heurística foi fundamental para guiar o algoritmo em estados não terminais, especialmente em jogos com árvores de decisão extensas. A qualidade da heurística é diretamente proporcional à eficácia do algoritmo em cenários nos quais a busca completa é inviável. Embora a heurística atual seja funcional, futuras iterações poderiam explorar funções de avaliação mais sofisticadas. A integração de técnicas de aprendizado de máquina, que ponderem a importância das diferentes configurações do tabuleiro, poderia aprimorar ainda mais a capacidade de avaliação do agente, possibilitando sua aplicação em jogos mais complexos.

Em síntese, este trabalho não apenas alcançou os objetivos de implementar e comparar algoritmos de busca heurísticos, mas também forneceu valiosas lições

sobre as nuances da inteligência artificial em jogos. Destacou a relevância das otimizações algorítmicas e a complexidade inerente aos jogos de soma zero com informação perfeita. A experiência reforça a importância de selecionar e otimizar algoritmos de IA conforme as características específicas do problema a ser resolvido.

Referências bibliográficas

BEM-EL-MECHAIEKH, Hichem; DIMAND, Robert. The von Neumann minimax theorem revisited. Fixed Point Theory and Its Applications. Warszawa: Polish Academy of Sciences, Institute of Mathematics, 2007. Banach Center Publications, v. 77, p. 23–34.

Softgroup, Sysvale. Entendendo o algoritmo Minimax com o Jogo da Velha. *Medium*, [s.l.], 2019. Disponível em: https://medium.com/sysvale/entendendo-o-algoritmo-minimax-com-o-jogo-da-velha-e7945bb908bd.