# 

Sistemas de Inteligentes

Relatório

Conteúdo

[Análise e Comparação de Algoritmos para o Jogo do Galo 4](#_Toc195534492)

[Descrição do Jogo 4](#_Toc195534493)

[Descrição dos Algoritmos Implementados 5](#_Toc195534494)

[Algoritmo Minimax 5](#_Toc195534495)

[Algoritmo Alpha-Beta 6](#_Toc195534496)

[Discussão das Principais Características dos Algoritmos 7](#_Toc195534497)

[Optimalidade: 7](#_Toc195534498)

[Completude 8](#_Toc195534499)

[Minimax: 8](#_Toc195534500)

[Alpha-Beta: 8](#_Toc195534501)

[Complexidade 9](#_Toc195534502)

[Minimax: 9](#_Toc195534503)

[Alpha-Beta: 9](#_Toc195534504)

[Estudo do Custo de Tempo e Memória 10](#_Toc195534505)

[Discussão dos Resultados 12](#_Toc195534506)

[Testes do Jogo 13](#_Toc195534507)

[14](#_Toc195534508)

[Conclusões Principais 16](#_Toc195534509)

# Análise e Comparação de Algoritmos para o Jogo do Galo

Descrição do Jogo

O Jogo do Galo (conhecido também como Tic Tac Toe) é um jogo de tabuleiro clássico para dois jogadores. O tabuleiro é composto por 9 espaços dispostos em uma matriz 3x3, e os jogadores alternam-se para marcar os espaços com os seus símbolos, geralmente representados por "X" e "O". O objetivo principal do jogo é alinhar três símbolos consecutivos de forma horizontal, vertical ou diagonal. Caso todas as casas do tabuleiro sejam preenchidas sem que um dos jogadores consiga alinhar três símbolos, o jogo termina em empate.

A simplicidade do Jogo do Galo torna-o um excelente ponto de partida para explorar algoritmos de tomada de decisão em jogos. Com uma quantidade limitada de estados possíveis, é possível utilizar algoritmos como Minimax e Alpha-Beta para determinar a jogada ótima em cada momento.

Na implementação apresentada, o jogo oferece diferentes modos de jogo:

Humano vs Computador e Computador vs Computador, permitindo uma análise comparativa dos algoritmos de decisão. Além disso, a opção de Computador vs Computador serve como uma ferramenta útil para observar a interação entre os dois algoritmos, destacando as diferenças no comportamento e no desempenho de cada um.

Descrição dos Algoritmos Implementados

## Algoritmo Minimax

**Implementação**: O algoritmo Minimax é uma técnica recursiva amplamente utilizada em jogos de soma zero com dois jogadores, com o objetivo de determinar a jogada ótima assumindo que ambos os jogadores atuam de forma racional e tentam maximizar ou minimizar a pontuação, respetivamente. A função minimax percorre recursivamente todas as jogadas possíveis a partir do estado atual do tabuleiro, atribuindo uma pontuação a cada resultado com base no desfecho do jogo: vitória, derrota ou empate.

**Casos Base**:

* Se o jogador atual (normalmente o computador) vencer, a função retorna uma pontuação positiva. A pontuação é calculada como 10 - profundidade, de forma a valorizar vitórias mais rápidas, ou seja, quando o número de jogadas necessárias para vencer é menor.
* Se o adversário (normalmente o jogador humano) vencer, a função retorna uma pontuação negativa, calculada como profundidade - 10, penalizando derrotas mais rápidas e valorizando derrotas mais tardias.
* No caso de empate (quando não há vencedor e o tabuleiro está completamente preenchido), a pontuação atribuída é zero, indicando um estado neutro.

**Lógica Recursiva**: A recursão do algoritmo segue a estrutura de uma árvore de decisões. O processo de escolha da jogada ótima ocorre da seguinte maneira:

Jogador que maximiza (tipicamente o computador): A função minimax seleciona a jogada com a maior pontuação entre todas as jogadas possíveis. Este jogador busca maximizar o seu benefício, logo, a jogada mais vantajosa será escolhida.

Jogador que minimiza (tipicamente o adversário): A função seleciona a jogada com a menor pontuação, pois este jogador tenta minimizar as chances de vitória do adversário.

# Algoritmo Alpha-Beta

## Implementação:

O algoritmo Alpha-Beta é uma otimização do algoritmo Minimax, desenvolvido com o objetivo de reduzir o número de nós a serem avaliados durante o processo de busca. Isso é alcançado por meio de uma técnica de poda, que permite descartar subárvores que não precisam ser exploradas, economizando, assim, tempo de execução e recursos computacionais.

Na função alpha\_beta, além dos parâmetros utilizados no Minimax, são introduzidos dois valores adicionais: **alpha** e **beta**.

* **Alpha**: Representa o valor mínimo que o jogador maximizador (tipicamente o computador) está garantido de obter, independentemente das escolhas do adversário.
* **Beta**: Representa o valor máximo que o jogador minimizador (tipicamente o adversário humano) assegura para si.

Poda Alpha-Beta: Durante a recursão, a poda ocorre quando a função identifica que um valor de um nó não pode contribuir para uma melhor decisão para o jogador, considerando os limites impostos por alpha e beta. Quando o valor de um nó ultrapassa ou é inferior a esses limites, a subárvore correspondente é descartada, ou seja, a exploração dessa parte do espaço de decisões é interrompida. Essa abordagem permite reduzir significativamente o número de nós visitados e, consequentemente, acelera o processo de tomada de decisão, sem comprometer a qualidade da jogada ótima escolhida.

# Discussão das Principais Características dos Algoritmos

## Optimalidade:

* **Minimax**:

O algoritmo Minimax garante a escolha da jogada ótima ao explorar completamente a árvore de decisões. Para cada possível jogada, o algoritmo considera todos os cenários subsequentes, buscando maximizar a probabilidade de vitória (ou minimizar a probabilidade de derrota). Como o Minimax avalia todas as possibilidades de forma exaustiva, ele garante que a jogada selecionada seja a melhor possível, considerando que o adversário também está a tomar decisões ótimas.

* **Alpha-Beta**:

O algoritmo Alpha-Beta mantém a mesma garantia de optimalidade que o Minimax, uma vez que a poda não altera a avaliação da melhor jogada, mas sim reduz o número de nós a serem analisados. A poda elimina ramos da árvore de decisões que não podem influenciar a decisão final, preservando assim a escolha ótima. Portanto, o algoritmo Alpha-Beta é igualmente capaz de garantir a melhor jogada possível, com a vantagem de ser mais eficiente em termos de tempo e recursos computacionais, devido à redução do espaço de busca.

# Completude

Ambos os algoritmos, Minimax e Alpha-Beta, são considerados completos quando aplicados a jogos com um espaço de estados finito. A completude de um algoritmo refere-se à sua capacidade de explorar todas as possíveis alternativas de decisão dentro de um espaço de busca finito. No contexto de jogos de soma zero, como o Jogo do Galo (Tic Tac Toe), a completude garante que o algoritmo será capaz de encontrar uma solução (se esta existir), seja uma vitória, um empate ou uma derrota, ao considerar todas as jogadas possíveis.

## Minimax:

O Minimax é um algoritmo de busca exaustiva, o que significa que ele explora completamente a árvore de decisões gerada a partir do estado inicial do tabuleiro, considerando todas as jogadas possíveis até o fim do jogo. A completude do Minimax advém do fato de que ele não poda nenhum ramo da árvore e explora todas as alternativas possíveis até que o jogo atinja um estado terminal (vitória, derrota ou empate). Portanto, se uma solução existe, o Minimax garantirá que ela seja encontrada, independentemente do número de jogadas ou da complexidade do espaço de estados.

## 

## Alpha-Beta:

O Alpha-Beta, embora também seja completo, apresenta uma diferença crucial em relação ao Minimax. Ele realiza a poda de ramos da árvore de decisão que são irrelevantes para a escolha da jogada ótima. Ou seja, ao calcular um nó e detectar que um ramo não pode influenciar a decisão final, o algoritmo descarta esse ramo, evitando assim a exploração de estados que não contribuirão para a escolha ótima. Apesar de eliminar parte da árvore de decisão, a completude do Alpha-Beta é preservada, uma vez que ele ainda garante a exploração completa do espaço de estados relevantes. A poda não afeta a capacidade do algoritmo de encontrar uma solução ótima, mas permite que o processo de busca seja mais eficiente em termos de tempo de execução e recursos computacionais.

# 

# Complexidade

## Minimax:

A complexidade do Minimax é exponencial em relação à profundidade da árvore de decisão, representada de forma geral como O(b^d), onde b é o fator de ramificação (número de jogadas possíveis por movimento) e d é a profundidade da árvore. Isso significa que, à medida que o número de jogadas possíveis aumenta e a profundidade cresce, a quantidade de nós a serem avaliados também cresce exponencialmente. Embora o Minimax seja eficaz para jogos simples com um espaço de estados relativamente pequeno, ele se torna ineficiente e difícil de aplicar em jogos mais complexos ou com grandes árvores de decisão.

## Alpha-Beta:

O Alpha-Beta, ao realizar a poda de ramos irrelevantes da árvore de decisão, reduz drasticamente o número de nós avaliados. Em condições ideais (com poda eficiente), a complexidade pode ser reduzida para O(b^(d/2)), o que significa que, na prática, o algoritmo pode explorar apenas a metade da profundidade da árvore, levando a uma execução muito mais rápida sem comprometer a qualidade da decisão. A poda alpha-beta aumenta significativamente a eficiência do algoritmo Minimax, especialmente em jogos com um grande espaço de busca.

# Estudo do Custo de Tempo e Memória

Na implementação, foram incluídas funções para medir o desempenho de ambos os algoritmos, permitindo a análise do custo de tempo e memória. Os principais parâmetros analisados foram:

* **Tempo de Execução**:

A função obter\_jogada\_computador mede o tempo que cada algoritmo demora a calcular a melhor jogada. Ensaios realizados com tabuleiros de dimensões 3x3 e 4x4 demonstraram que o algoritmo **Alpha-Beta** apresenta tempos de execução significativamente menores em comparação ao **Minimax**, especialmente quando o número de estados possíveis aumenta. A poda realizada pelo Alpha-Beta permite reduzir a exploração de ramos irrelevantes, resultando em uma execução mais rápida.

* **Contagem de Nós Visitados**:

A função contar\_nos\_visitados contabiliza o número de nós explorados durante a execução de cada algoritmo. Os resultados mostram que o **Alpha-Beta** explora consideravelmente menos nós, refletindo uma utilização mais eficiente da memória e dos recursos computacionais. Isso ocorre porque, ao podar ramos desnecessários, o Alpha-Beta reduz o espaço de busca efetivo, mantendo a mesma qualidade na escolha da jogada.

Uma imagem com texto, captura de ecrã, diagrama, Gráfico

Os conteúdos gerados por IA poderão estar incorretos.

# Discussão dos Resultados

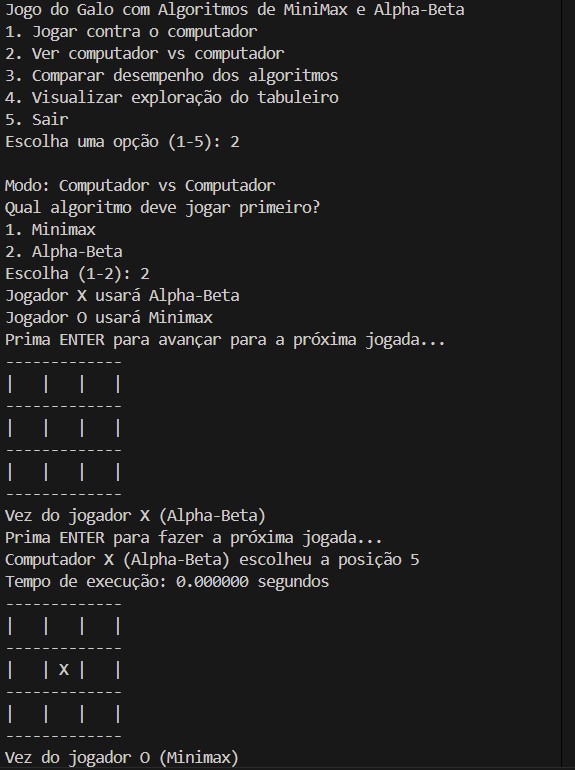
Com base nos testes realizados e nas visualizações gráficas geradas:

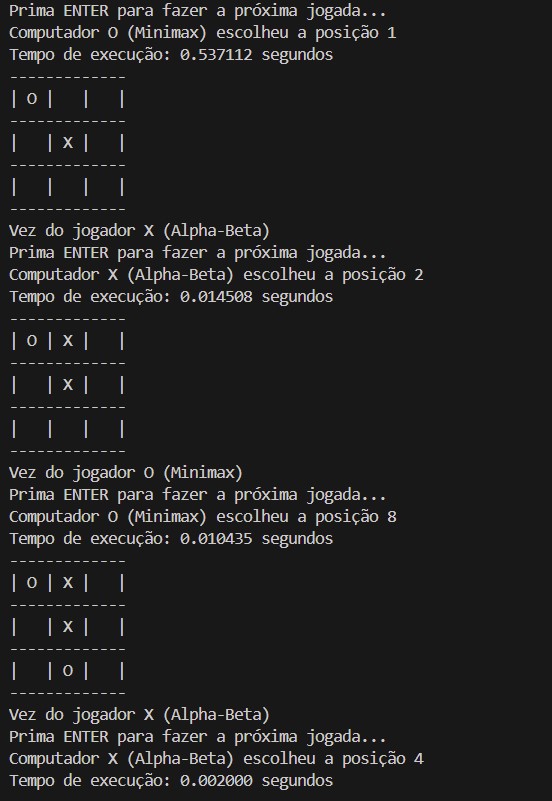
* **Desempenho**:  
  O algoritmo **Alpha-Beta** demonstrou ser mais eficiente em termos de tempo de execução e redução do número de nós visitados, quando comparado ao **Minimax**. Esta vantagem torna o **Alpha-Beta** especialmente adequado para jogos ou cenários com espaços de estados mais extensos, onde a quantidade de possibilidades aumenta significativamente.
* **Eficácia da Poda**:  
  A significativa redução no número de nós explorados valida a eficácia da técnica de poda utilizada pelo algoritmo **Alpha-Beta**. Esta técnica permite ao algoritmo agir de forma mais rápida e com menor uso de memória, sem comprometer a optimalidade da decisão, o que o torna muito mais eficiente do ponto de vista computacional.
* **Aplicabilidade**:  
  Embora ambos os algoritmos sejam viáveis para o **Jogo do Galo**, em contextos de jogos com maior complexidade, a estratégia de poda **Alpha-Beta** torna-se essencial para garantir uma tomada de decisão em tempo útil. Em jogos com um espaço de estados vasto, a eficiência proporcionada pela poda se torna um fator crucial para a viabilidade de uso de tais algoritmos.

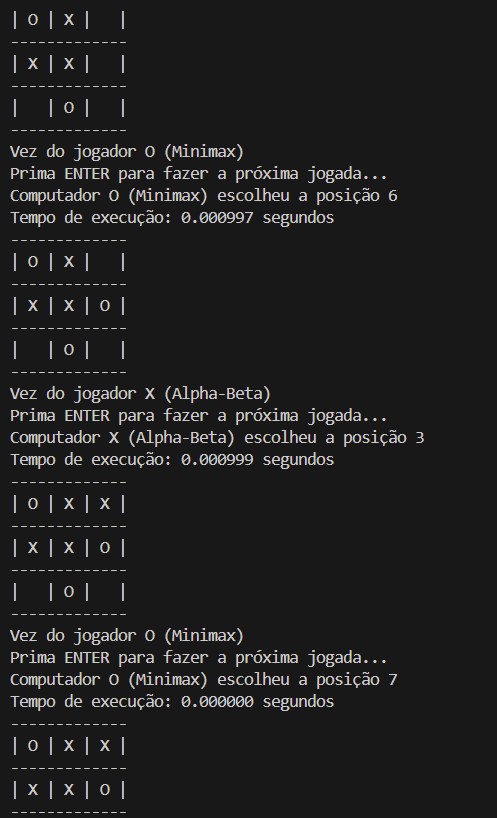
Uma imagem com texto, file, diagrama, Gráfico

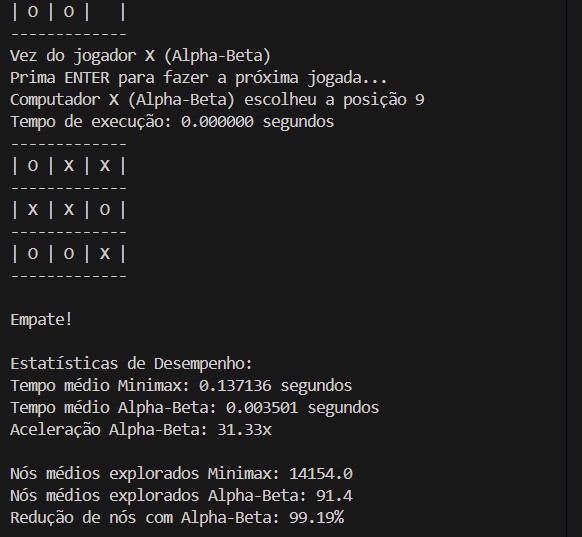
Os conteúdos gerados por IA poderão estar incorretos.

# Testes do Jogo









# Conclusões Principais

* **Optimalidade e Completude**:

Tanto o algoritmo Minimax quanto o Alpha-Beta garantem a escolha da jogada ótima e são completos para jogos com um número finito de estados, como o Jogo do Galo. Isso assegura que, independentemente da estratégia utilizada, o melhor movimento será sempre escolhido, desde que o espaço de estados seja finito.

* **Eficiência do Alpha-Beta:**

A técnica de poda Alpha-Beta apresenta uma melhoria significativa na eficiência computacional ao reduzir o número de nós visitados e o tempo de execução, sem comprometer a optimalidade da decisão. Esta otimização é essencial em jogos ou cenários com espaços de estados grandes, onde a exploração completa da árvore de decisão seria inviável.

* **Custo de Tempo e Memória:**

Embora o Minimax seja simples de implementar, ele apresenta um custo elevado em termos de tempo e memória à medida que a profundidade da árvore de decisões aumenta. Em contrapartida, o Alpha-Beta apresenta uma otimização substancial, tornando-o mais adequado para cenários de maior complexidade, onde a eficiência computacional é crucial.