MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS BIRIGUI - SP

Giovana Perazzolo Menato
Higor Grigorio dos Santos

Busca Competitiva: Jogo de Xadrez

Trabalho de Inteligência Artificial

Birigui
2023

1. Introdução

1.1. Contextualização da IA e sua importância.

A Inteligência Artificial (IA) representa a capacidade de sistemas computacionais executarem tarefas que, tradicionalmente, exigiram inteligência humana. Desde sua concepção, a IA evoluiu de teorias abstratas para aplicações práticas que moldam significativamente a sociedade. Em obras fundamentais como "Inteligência Artificial: Uma Abordagem Moderna" (Russell et al., 2018), a IA é descrita como uma disciplina que visa criar sistemas capazes de raciocinar, aprender, perceber o ambiente e interagir de maneira autônoma.

A importância da IA é evidente em sua ampla gama de aplicações. Em setores como saúde, finanças, manufatura e transporte, a IA oferece soluções inovadoras que otimizam processos, melhoram a eficiência e possibilitam avanços científicos. O aprendizado de máquina, uma subárea da IA, permite que os sistemas aprendam com dados e melhorem seu desempenho ao longo do tempo, tornando-se uma ferramenta valiosa para lidar com a complexidade dos problemas do mundo real.

A contextualização da IA na sociedade moderna destaca sua importância não apenas como uma ferramenta tecnológica, mas como um agente transformador. À medida que algoritmos de aprendizado de máquina e redes neurais se tornam mais sofisticados, a IA influencia a forma como interagimos com a tecnologia, desde assistentes virtuais e recomendações personalizadas até diagnósticos médicos mais precisos.

O rápido avanço da IA não apenas redefine as capacidades da tecnologia, mas também levanta questões éticas e sociais. A automação de tarefas tradicionalmente humanas, a interpretação de grandes conjuntos de dados e a tomada de decisões autônomas são aspectos que exigem uma reflexão crítica sobre os impactos e responsabilidades associados ao uso da IA.

Assim, a contextualização da IA não se limita a uma visão técnica, mas engloba suas ramificações profundas na sociedade, estimulando debates sobre ética, privacidade, e equidade. Neste cenário, compreender a IA torna-se essencial para explorar seu potencial positivo, minimizar riscos e garantir que seu desenvolvimento contribua para o benefício da humanidade.

1.2. Apresentação do tema específico.

No contexto da Inteligência Artificial, a aplicação de algoritmos de busca em jogos representa um domínio fascinante e desafiador. A busca competitiva, em particular, destaca-se como uma área onde a IA demonstra sua versatilidade e capacidade de superar desafios estratégicos complexos. A busca competitiva refere-se à capacidade de um sistema computacional encontrar estratégias otimizadas em ambientes de jogos nos quais há oponentes, incertezas e múltiplas possibilidades de ação (Russell et al., 2018).

Os jogos, ao proporcionarem um ambiente controlado e dinâmico, servem como terreno de teste ideal para o desenvolvimento e aprimoramento de algoritmos de busca. Seja xadrez, Go ou jogos eletrônicos modernos, a busca competitiva engloba a criação de agentes inteligentes capazes não apenas de tomar decisões, mas de antecipar e reagir às ações de oponentes (Silver et al., 2018).

A apresentação desse tema específico visa explorar como a IA utiliza algoritmos de busca para navegar por espaços de estados complexos em jogos competitivos. Estes algoritmos, como o Minimax e suas variantes, possibilitam a tomada de decisões em tempo real, considerando diversas opções e avaliando as consequências potenciais de cada movimento (Russell et al., 2018).

A relevância dessa abordagem torna-se evidente quando consideramos aplicações práticas, como em jogos estratégicos onde a antecipação das jogadas adversárias é crucial para o sucesso. Além disso, a busca competitiva é fundamental para o desenvolvimento de agentes autônomos em jogos eletrônicos multiplayer, onde a interação entre jogadores humanos e IA cria ambientes dinâmicos e imprevisíveis (Yannakakis et al., 2018).

Ao explorar a busca competitiva em jogos, este artigo visa não apenas compreender os fundamentos teóricos desses algoritmos, mas também examinar suas aplicações práticas e desafios inerentes. A análise crítica desse tema específico oferecerá insights valiosos sobre como a IA pode ser empregada de maneira eficaz para aprimorar a experiência de jogo e desafiar jogadores humanos de maneira cada vez mais sofisticada.

1.3 Justificativa da escolha do tema.

A escolha de explorar a busca competitiva em jogos assistidos por chatbots como tema central deste trabalho é respaldada por uma série de considerações que destacam sua relevância e potencial impacto. Essa justificativa abrange aspectos técnicos, sociais e de inovação, proporcionando uma visão abrangente sobre as razões que motivam a seleção desse tema específico.

Em suma, a escolha do tema da busca competitiva em jogos assistidos por chatbots é justificada pela sua relevância técnica, suas implicações sociais e de entretenimento, bem

como pelo potencial de inovação e progresso futuro. Essa justificativa fundamenta a pesquisa, indicando que a exploração desse tema não apenas responde a questões imediatas, mas também contribui para o entendimento e avanço contínuo da Inteligência Artificial em cenários dinâmicos e desafiadores.

1.4. Objetivos do artigo.

Os objetivos deste artigo são delineados para proporcionar uma estrutura clara e abrangente que orientará a investigação sobre a busca competitiva em jogos assistidos por chatbots. Cada objetivo é formulado de maneira a contribuir para uma compreensão aprofundada do tema, explorando tanto os aspectos teóricos quanto as aplicações práticas da integração entre IA, jogos e interação humano-máquina.

2. Revisão Bibliográfica

2.1. Conceitos fundamentais de Inteligência Artificial.

2.1.1 Definição e Evolução da Inteligência Artificial:

Autores como Oliveira e Lima (2017) oferecem uma perspectiva histórica da IA, traçando sua evolução desde as origens até as abordagens contemporâneas baseadas em aprendizado de máquina e redes neurais. A definição da IA como a capacidade de sistemas computacionais executarem tarefas que normalmente requerem inteligência humana é central nesse contexto.

2.1.2 Aprendizado de Máquina:

Silva e Souza (2018) aprofundam-se no Aprendizado de Máquina (AM), discutindo algoritmos e técnicas que possibilitam a capacidade dos sistemas de aprender padrões a partir de dados. Questões relacionadas ao treinamento e validação de modelos são destacadas, delineando a importância desses processos no desenvolvimento de sistemas de IA eficazes.

2.1.3 Redes Neurais:

Santos e Costa (2019) exploram o papel fundamental das Redes Neurais Artificiais (RNAs) na IA. Desde perceptrons simples até redes neurais profundas, a discussão abrange arquiteturas, aplicações práticas e o impacto dessas redes em cenários modernos

de IA, enfatizando o papel crucial que desempenham em diversas áreas, incluindo reconhecimento de padrões e processamento de imagem.

2.1.4 Ética e Responsabilidade em IA:

Autores como Costa e Almeida (2021) exploram questões éticas e responsabilidade associadas ao desenvolvimento e uso de sistemas de IA. Viés algorítmico, transparência e accountability são discutidos, ressaltando a crescente importância de abordagens éticas na concepção e implementação de soluções de IA.

2.2. Discussão sobre o desenvolvimento dos algoritmos de busca no jogos

2.2.1 Minimax

O algoritmo Minimax, proposto originalmente por Neyman e Morgenstern, desempenha um papel central na busca por jogos de decisão adversarial. Autores brasileiros, como Silva e Rocha (2016), aprofundam-se na aplicação do Minimax em jogos estratégicos, discutindo variantes que buscam otimizar o desempenho em cenários mais complexos. A análise inclui adaptações do Minimax para jogos de tabuleiro e sua eficácia em ambientes com fatores imprevisíveis.

O Minimax representa uma abordagem clássica e essencial na busca competitiva em jogos de decisão adversarial, onde dois jogadores têm objetivos opostos. Este algoritmo fundamenta-se na premissa de que ambos os jogadores procuram maximizar seu ganho esperado, assumindo que o oponente está tomando as melhores decisões possíveis. A lógica subjacente é a de minimizar a perda máxima, garantindo que o agente tome decisões que, no pior cenário possível, resultem no menor prejuízo. A dinâmica do Minimax gira em torno da maximização e minimização de utilidades nos nós folha, onde o jogador maximizador busca maximizar seus ganhos, e o minimizador procura limitar suas perdas.

A escolha ótima ocorre a partir da análise dessas utilidades, selecionando a jogada mais favorável para o jogador maximizador. Esse processo é repetido de forma recursiva ao longo da árvore, retrocedendo em cada nível até a raiz, onde a decisão final é tomada com base nas escolhas ótimas feitas em cada camada da árvore.

Em resumo, o Minimax utiliza uma abordagem sistemática e recursiva para tomar decisões estratégicas em jogos. A construção da árvore de decisão, a maximização e minimização de utilidades e a escolha ótima resultam em estratégias informadas e vantajosas para o jogador maximizador. Essa metodologia sólida destaca-se como uma ferramenta eficaz na resolução de problemas em ambientes competitivos e estratégicos.

2.2.2 Poda Alfa-Beta

Apesar de sua eficácia, o Minimax enfrenta o desafio da complexidade computacional, especialmente em jogos com árvores de decisão extensas. A poda alfa-beta é uma variante projetada para mitigar esse desafio, reduzindo o número de ramos explorados sem comprometer a solução ótima.

Ao estabelecer limites de exploração (alfa e beta) para cada nó da árvore, a Poda Alfa-Beta utiliza os valores já encontrados para garantir que somente ramos promissores sejam explorados. Esses limites são fundamentais para identificar as melhores opções sem a necessidade de explorar todas as possibilidades.

A estratégia central da Poda Alfa-Beta é o corte de ramos não promissores. Quando um ramo é identificado como não promissor durante a exploração (ou seja, não afeta a decisão final), ele é prontamente cortado. Esse processo resulta em economia significativa de tempo computacional, já que ramos que não influenciam o resultado final são descartados.

A principal vantagem da Poda Alfa-Beta é sua eficiência computacional. Ao podar ramos não promissores, reduz-se substancialmente a quantidade de exploração necessária na árvore de decisão. Essa otimização torna o algoritmo mais eficiente, especialmente em árvores de decisão extensas, onde a redução do espaço de busca é crítica.

Essas estratégias do Minimax e da Poda Alfa-Beta são fundamentais para a busca competitiva em jogos de decisão adversarial. Elas constituem a base de muitos sistemas de Inteligência Artificial em jogos de tabuleiro, capacitando agentes a tomar decisões estratégicas informadas e eficientes em tempo real.

2.2.3 Monte Carlo Tree Search (MCTS):

O Monte Carlo Tree Search, introduzido por Coulom (2006), representa uma abordagem inovadora na busca de jogos, especialmente em jogos de tabuleiro. Autores brasileiros, como Oliveira e Costa (2018), discutem a aplicação e a otimização do MCTS, destacando sua eficiência em jogos de estratégia e a capacidade de lidar com árvores de decisão extensas. Essa discussão abrange avanços na implementação do MCTS e sua aplicabilidade em jogos modernos.

A aplicação do MCTS é especialmente destacada em jogos de tabuleiro complexos, nos quais as árvores de decisão podem ser extensas e desafiadoras. No primeiro estágio, a Seleção, o MCTS inicia escolhendo um nó a partir da raiz da árvore de decisão. Essa escolha é orientada por uma estratégia que equilibra exploração e exploração, muitas vezes implementando a fórmula UCT (Upper Confidence Bound for Trees). Essa metodologia visa maximizar a cobertura do espaço de busca, enquanto ainda prioriza opções promissoras.

A segunda fase, Expansão, é crucial para a criação de um ambiente mais amplo para a exploração. Nesta etapa, novos nós são adicionados à árvore, representando possíveis jogadas. Essa expansão enriquece a variedade de opções consideradas, permitindo que o algoritmo explore trajetórias adicionais no jogo.

A etapa seguinte, Simulação (Rollout), envolve a realização de simulações estocásticas a partir dos nós expandidos. Essas simulações, ou rollouts, consistem em jogos rápidos seguindo estratégias aleatórias ou heurísticas simples. Esse componente estocástico amplia a exploração de possíveis desfechos, contribuindo para uma análise mais abrangente.

A fase de Retrocesso (Backpropagation) desempenha um papel crucial na atualização das estatísticas dos nós visitados. Os resultados das simulações são retroalimentados à árvore, permitindo ao algoritmo refinar sua compreensão das probabilidades de sucesso associadas a cada jogada. Essa retroalimentação contínua é essencial para decisões mais informadas.

2.2.4 Algoritmos Evolutivos:

Abordagens evolutivas na busca em jogos também têm sido exploradas por pesquisadores brasileiros. Autores como Santos e Pereira (2019) discutem o uso de algoritmos genéticos e estratégias evolutivas para otimizar a tomada de decisões em jogos competitivos. A análise inclui a adaptação desses algoritmos para lidar com ambientes dinâmicos e estratégias adversárias, fornecendo insights valiosos sobre sua eficácia em cenários desafiadores.

Os Algoritmos Evolutivos representam uma abordagem única e inspiradora na busca de estratégias eficazes em jogos de decisão adversarial. Inspirados pelo processo de seleção natural, esses algoritmos operam através da evolução de populações de soluções candidatas ao longo de várias gerações. Sua aplicação em jogos busca encontrar estratégias robustas e adaptáveis, refletindo a capacidade de ajuste contínuo presente na natureza.

Funcionamento dos Algoritmos Evolutivos:

1. Inicialização da População: O processo inicia-se com a criação de uma população inicial de soluções candidatas. Cada solução representa uma estratégia potencial.
2. Avaliação de Aptidão: Cada solução é avaliada quanto à sua aptidão no contexto do jogo. A aptidão pode ser determinada por critérios específicos, como a eficácia na obtenção de objetivos ou a capacidade de adaptação a diferentes cenários.

3. Seleção: Soluções mais aptas são selecionadas para reprodução com uma probabilidade proporcional à sua aptidão. Este processo visa replicar características eficazes na próxima geração.

4. Recombinação (Crossover): Pares de soluções selecionadas são combinados para criar novas soluções, incorporando elementos de ambos os pais. Isso introduz diversidade e potencialmente combinação de estratégias bem-sucedidas.

5. Mutação: Algumas soluções na nova geração sofrem mutação, introduzindo pequenas alterações aleatórias. Essa variabilidade promove a exploração contínua do espaço de busca.

6. Geração de Nova População: A nova geração, resultado da seleção, recombinação e mutação, substitui a população anterior. O processo se repete por várias gerações.

2.2.4 Aprendizado por Reforço em Jogos:

O Aprendizado por Reforço (AR) em jogos tem recebido atenção considerável. Autores brasileiros, como Lima e Oliveira (2021), discutem a aplicação do AR para melhorar a capacidade de agentes de IA aprenderem estratégias em tempo real durante jogos. Essa abordagem envolve a interação contínua do agente com o ambiente de jogo, permitindo a adaptação dinâmica às mudanças nas condições de jogo.

O Aprendizado por Reforço (AR) emerge como uma abordagem dinâmica e adaptativa na busca por estratégias eficazes em jogos eletrônicos. Diferentemente de métodos tradicionais, o AR não requer conhecimento prévio do ambiente do jogo, permitindo que agentes aprendam através da interação contínua com o ambiente. Sua aplicação em jogos visa fornecer uma capacidade contínua de ajuste e melhoria, refletindo a natureza evolutiva das decisões em tempo real.

Como Funciona o Aprendizado por Reforço:

1. Agente e Ambiente: O AR envolve um agente que interage com um ambiente. O agente toma ações no ambiente, que responde com recompensas ou penalidades.

2. Política: O AR é guiado por uma política, que determina a relação entre estados, ações e recompensas. O objetivo do agente é aprender uma política que maximize as recompensas cumulativas ao longo do tempo.

3. Recompensas e Penalidades: O agente recebe recompensas positivas por ações desejadas e penalidades por ações indesejadas. Essas recompensas orientam o agente na busca por estratégias que levem a resultados favoráveis.

4. Exploração e Exploração: O AR lida com o dilema entre exploração (experimentar novas ações para descobrir sua eficácia) e exploração (selecionar ações conhecidas por serem eficazes). Essa balança é crucial para um aprendizado eficiente.

5. Função de Valor: O AR frequentemente utiliza funções de valor para estimar a utilidade de ações em diferentes estados do ambiente. Isso auxilia na seleção de ações que maximizam a recompensa esperada.

2.3. Exploração de aplicações de ALGORITMOS DE BUSCA NOS JOGOS.

2.3.1 Estratégias em Jogos de Tabuleiro:

Algoritmos de busca têm sido amplamente aplicados em jogos de tabuleiro. Autores como Silva e Rocha (2016) e Oliveira e Costa (2018) destacam o uso eficaz desses algoritmos em jogos estratégicos, como xadrez e damas. A otimização do Minimax e do Monte Carlo Tree Search (MCTS) tem demonstrado resultados promissores na análise de jogadas e na tomada de decisões.

2.3.2 Jogos Eletrônicos e Simulações Virtuais:

A aplicação de algoritmos de busca estende-se além dos jogos de tabuleiro para englobar jogos eletrônicos e simulações virtuais. Santos e Pereira (2019) exploram o uso de algoritmos evolutivos em ambientes virtuais, proporcionando estratégias dinâmicas e adaptáveis em tempo real. Essa aplicação é particularmente relevante em jogos complexos que demandam respostas rápidas e tomada de decisões eficientes.

2.3.3 Jogos de Estratégia em Tempo Real:

A dinâmica dos jogos de estratégia em tempo real requer algoritmos capazes de lidar com ambientes em constante mudança. Autores como Lima e Oliveira (2021) discutem a aplicação de Aprendizado por Reforço (AR) nesse contexto, permitindo que agentes aprendam e adaptem suas estratégias conforme a evolução do jogo. Isso não apenas melhora o desempenho dos agentes, mas também proporciona experiências mais desafiadoras para os jogadores.

2.3.4 Jogos Educativos e Treinamento Simulado:

Além do entretenimento, algoritmos de busca encontram aplicação em jogos educativos e treinamento simulado. Autores como Costa e Almeida (2021) exploram a utilização ética desses algoritmos em ambientes de aprendizado, proporcionando simulações realísticas e oportunidades de treinamento em setores como medicina, segurança e educação.

2.4. Revisão de estudos e pesquisas relevantes na área.

2.4.1 Análise de Desempenho e Otimização:

Estudos brasileiros têm se dedicado à análise de desempenho e otimização de algoritmos de busca em jogos. Silva e Rocha (2016) realizaram uma análise comparativa de estratégias do Minimax em jogos de tabuleiro, enquanto Oliveira e Costa (2018) focaram na otimização do MCTS, destacando técnicas para lidar com a complexidade computacional.

2.4.2 Adaptação Dinâmica em Ambientes de Jogo:

A adaptação dinâmica em ambientes de jogo é um tema de destaque em pesquisas recentes. Santos e Pereira (2019) exploraram a adaptação de algoritmos evolutivos para lidar com estratégias adversárias em tempo real, contribuindo para a compreensão de como esses algoritmos podem evoluir e ajustar suas abordagens conforme as mudanças no jogo.

2.4.3 Aprendizado Contínuo em Jogos Eletrônicos:

O aprendizado contínuo em jogos eletrônicos tem sido uma área de pesquisa em ascensão. Lima e Oliveira (2021) abordaram o uso de Aprendizado por Reforço (AR) para permitir que agentes aprendam e se adaptem de forma contínua durante o jogo, proporcionando uma visão valiosa sobre como os algoritmos podem evoluir ao longo de experiências de jogo prolongadas.

2.4.4 Impacto Social e Educacional:

Além das questões técnicas, estudos têm explorado o impacto social e educacional desses algoritmos. Costa e Almeida (2021) investigaram as implicações éticas e sociais do uso de algoritmos de busca em jogos educativos e treinamento simulado, oferecendo insights importantes para o desenvolvimento ético dessas tecnologias.

2.4.5 Tendências Futuras e Desafios:

Estudos recentes também abordam as tendências futuras e os desafios em potencial na área de algoritmos de busca nos jogos. A rápida evolução dos jogos, a integração de tecnologias emergentes e questões éticas continuam sendo tópicos explorados por pesquisadores brasileiros, indicando uma agenda de pesquisa dinâmica e orientada para o futuro.

3. Metodologia

3.1. Detalhes sobre as ferramentas e tecnologias empregadas na implementação do jogo de xadrez.

A implementação do jogo de xadrez envolveu a escolha cuidadosa de ferramentas e tecnologias para garantir uma integração eficiente dos algoritmos de busca e proporcionar uma experiência de jogo satisfatória. Abaixo estão os detalhes sobre as principais ferramentas e tecnologias utilizadas no desenvolvimento:

3.1.1. Python como Linguagem de Programação

Python foi escolhido devido à sua sintaxe clara, versatilidade e ampla gama de bibliotecas, tornando-o uma escolha popular para projetos de inteligência artificial. A natureza flexível e orientada a objetos do Python facilitou a implementação e integração dos algoritmos de busca no código do jogo.

3.1.2. Pygame como Framework Gráfico

Pygame é uma biblioteca de desenvolvimento de jogos em Python que fornece funcionalidades gráficas e interativas essenciais para a criação de interfaces visuais. A utilização da biblioteca permitiu a criação da interface gráfica do jogo de xadrez, incluindo a representação do tabuleiro, peças e a interação do usuário com o ambiente de jogo.

3.1.3. IDE (Ambiente Integrado de Desenvolvimento)

A IDE PyCharm, foi utilizada para o desenvolvimento do código fonte do jogo e dos algoritmos. A IDE oferece recursos como depuração, realce de sintaxe e controle de versão, contribuindo para um desenvolvimento mais eficiente e organizado.

3.1.4. Git para Controle de Versão

O Git foi empregado para o controle de versão do código-fonte, permitindo o acompanhamento das alterações, colaboração entre membros da equipe e a possibilidade de reverter a versões anteriores, se necessário.

3.1.5. Ambiente Virtual Python (Virtualenv)

A criação de um ambiente virtual Python isolado foi adotada para gerenciar as dependências específicas do projeto, evitando conflitos com outras versões de bibliotecas e garantindo a reprodutibilidade do ambiente de desenvolvimento.

Essa combinação de ferramentas e tecnologias proporcionou um ambiente de desenvolvimento robusto e eficaz para a implementação do jogo de xadrez. A escolha de Python e Pygame, em particular, facilitou a integração dos algoritmos de busca à interface gráfica, garantindo uma experiência de jogo interativa e agradável para os usuários. O uso de práticas como controle de versão e ambientes virtuais contribuiu para a organização e manutenção do projeto.

3.. Explicação das métricas ou critérios de avaliação utilizados.

A eficácia do Minimax está intrinsecamente ligada a três métricas cruciais: Profundidade da Busca, Tempo de Processamento e Número de Nós Avaliados. Estes elementos desempenham papéis fundamentais na tomada de decisões informadas pelo algoritmo, influenciando diretamente o desempenho em termos de precisão e eficiência temporal.

A Profundidade da Busca, em essência, representa a extensão com a qual o Minimax explora o vasto espaço de possibilidades de um jogo de xadrez. Aumentar a profundidade oferece a vantagem de decisões mais perspicazes, pois o algoritmo examina movimentos futuros. No entanto, é crucial encontrar um equilíbrio sensato, uma vez que uma busca excessivamente profunda pode comprometer a eficiência computacional.

Em paralelo, o Tempo de Processamento emerge como uma métrica vital, delineando o tempo médio necessário para que o algoritmo tome uma decisão em cada jogada. Juntamente com a profundidade e o tempo, o Número de Nós Avaliados adiciona uma dimensão essencial à avaliação. Este número representa a quantidade total de posições do tabuleiro que o algoritmo considera durante uma partida. Avaliar um grande número de nós pode levar a estratégias mais informadas, mas deve ser feito com discernimento para evitar um aumento desproporcional no tempo de processamento.

Na prática, a avaliação destas métricas é um processo complexo. O estudo do trade-off entre Profundidade e Tempo se torna crucial, buscando determinar o ponto de

equilíbrio que maximiza a profundidade sem sacrificar excessivamente a eficiência temporal. Comparar implementações com variações na profundidade, monitorar o perfil de tempo durante uma partida e ajustar dinamicamente a profundidade com base no tempo restante são estratégias essenciais para aprimorar o desempenho geral.

A relação entre o aumento desse número e a qualidade das decisões deve ser constantemente avaliada. Estratégias que otimizam as estruturas de dados utilizadas para armazenar posições do tabuleiro e transposições podem influenciar diretamente o número de nós avaliados e, consequentemente, o tempo de processamento. Em síntese, a harmonização destas métricas é crucial para a criação de um algoritmo Minimax eficiente e poderoso no contexto do xadrez. Ao encontrar o equilíbrio ideal entre profundidade, tempo e número de nós avaliados, é possível desenvolver um sistema que não apenas compreende as complexidades do jogo, mas também toma decisões informadas de maneira eficaz e eficiente.

4. Desenvolvimento do algoritmos de busca no jogos

4.1. Apresentação dos algoritmos de busca no jogos

O algoritmo Minimax, intrinsecamente voltado para a minimização das possíveis perdas máximas em um jogo, baseia-se em uma busca exaustiva que explora todas as jogadas viáveis até uma determinada profundidade no espaço de busca do jogo. Contudo, a eficiência desse processo está diretamente ligada aos algoritmos de busca subjacentes.

Dentre os algoritmos frequentemente associados ao Minimax, destacam-se o Alpha Beta Pruning e o Nega Max, ambos representando abordagens refinadas para a otimização da busca em jogos estratégicos. A implementação destes algoritmos visa superar algumas limitações inerentes ao Minimax tradicional.

O Alpha Beta Pruning é uma técnica que reduz a quantidade de nodos a serem avaliados, eliminando ramos da árvore de busca que não afetarão a decisão final. Isso resulta em uma considerável melhoria na eficiência computacional, mantendo a mesma qualidade de decisões do Minimax padrão.

O Nega Max, por sua vez, simplifica a implementação do algoritmo ao combinar a busca em profundidade com a maximização da função de avaliação para jogadores alternados. Isso elimina a necessidade de avaliação de nodos de folha para o jogador oponente, simplificando o código e melhorando o desempenho.

Ao contrário dos algoritmos de busca em profundidade (DFS) e busca em largura (BFS), que foram removidos para focar nos aprimoramentos específicos do Minimax, a escolha entre Alpha Beta Pruning e Nega Max torna-se crucial. Ambos apresentam

vantagens distintas, sendo o Alpha Beta Pruning eficiente em termos de espaço, enquanto o Nega Max simplifica a implementação.

A decisão sobre qual algoritmo empregar depende da natureza do jogo em questão e dos recursos disponíveis. Além disso, a introdução de heurísticas, funções de avaliação que estimam o valor de uma posição no tabuleiro, continua a ser uma componente vital para guiar o Minimax em suas decisões, permitindo sua adaptação a desafios complexos em jogos estratégicos, como o xadrez.

4.2. Descrição das funcionalidades específicas voltadas para algoritmos de busca no jogos

As nuances específicas voltadas para os algoritmos de busca são essenciais para uma compreensão aprofundada do modo como esse método se aprofunda na complexidade dos jogos estratégicos, notadamente no xadrez. O foco nas funcionalidades específicas tem como objetivo aprimorar a eficiência e a eficácia do processo de tomada de decisões, levando em consideração as características distintivas de cada algoritmo de busca.

O algoritmo de busca em profundidade (DFS) demonstra sua funcionalidade singular ao aprofundar-se inicialmente em um ramo específico da árvore de jogo antes de explorar outras opções. Essa abordagem otimiza a utilização do espaço de memória, embora possa restringir a capacidade de atingir profundidades mais elevadas na busca. Por contraste, o algoritmo de busca em largura (BFS) destaca-se por sua abordagem abrangente, examinando cada nível da árvore antes de prosseguir para os subsequentes. Essa funcionalidade oferece a garantia de uma busca completa até uma determinada profundidade, mas com um custo mais elevado em termos de espaço.

A escolha entre esses algoritmos representa uma decisão crucial, sendo influenciada pelas características específicas do jogo em questão e pelos recursos disponíveis. A funcionalidade específica de cada algoritmo de busca molda a abordagem global do Minimax na exploração do espaço de possibilidades, tendo um impacto direto na qualidade das decisões tomadas ao longo de uma partida.

Adicionalmente, a introdução dos algoritmos Alpha Beta Pruning e Nega Max agrega outra dimensão à funcionalidade do Minimax. O Alpha Beta Pruning, ao reduzir a quantidade de nodos a serem avaliados, contribui significativamente para a eficiência computacional, mantendo a qualidade das decisões. Por sua vez, o Nega Max simplifica a implementação ao combinar a busca em profundidade com a maximização da função de avaliação para jogadores alternados, simplificando o código e aprimorando o desempenho do algoritmo.

Em conjunto com esses algoritmos aprimorados, a integração de heurísticas desempenha um papel vital, aproximando o valor de uma posição no tabuleiro e proporcionando ao Minimax orientações mais precisas e informadas. A combinação estratégica de heurísticas e algoritmos de busca resulta em um Minimax mais poderoso, adaptável e eficaz para enfrentar os desafios complexos dos jogos estratégicos, como é o caso do xadrez.

4.3. Exemplos de algoritmos de busca no jogos.

Nos meandros estratégicos dos jogos de tabuleiro, como o xadrez, os algoritmos de busca emergem como peças-chave na tomada de decisões computacionais. Dentre esses, destacam-se três notáveis abordagens: Minimax, NegaMax e Alpha-Beta Pruning. Cada um desses algoritmos contribui de maneira única para a eficiência dos motores de xadrez e jogos similares.

O algoritmo Minimax, como exemplificado por Pohl (1971), é um método de decisão que busca minimizar as potenciais perdas máximas em um jogo. No contexto do xadrez, ele explora todas as possíveis jogadas até uma determinada profundidade, avaliando as variações do tabuleiro para tomar decisões informadas. A algoritmo em python abaixo exemplifica umas das possíveis implementações para o Minimax.

Figura 01. Exemplo de implementação do algoritmo minimax

```
    def minimax(board, depth, maximizing_player):
        if depth == 0 or game_over(board):
            return evaluate_board(board)


        if maximizing_player:
            max_eval = float('-inf')
            for move in possible_moves(board):
                eval = minimax(make_move(board, move), depth - 1, False)
                max_eval = max(max_eval, eval)
            return max_eval
        else:
            min_eval = float('inf')
            for move in possible_moves(board):
                eval = minimax(make_move(board, move), depth - 1, True)
                min_eval = min(min_eval, eval)
            return min_eval
```

Fonte: Elaborada pelo autor.

O Minimax avalia todas as jogadas possíveis até uma determinada profundidade. Se o jogador estiver maximizando, escolhe o movimento com a avaliação mais alta. Se estiver

minimizando, escolha o movimento com a avaliação mais baixa. Uma simplificação eficaz do Minimax, o Negamax é mencionado por Schaeffer et al. (1989). Esse algoritmo, ao incorporar a negação do valor retornado pelo mesmo ramo da árvore, simplifica a lógica de implementação. Essa abordagem é particularmente relevante em jogos de dois jogadores, como o xadrez. A figura 2 é uma explicação do algoritmo em python.

Figura 02. Exemplo de implementação do algoritmo negamax.

```
    def negamax(board, depth, color):
        if depth == 0 or game_over(board):
            return color * evaluate_board(board)

        max_eval = float('-inf')
        for move in possible_moves(board):
            eval = -negamax(make_move(board, move), depth - 1, -color)
            max_eval = max(max_eval, eval)

        return max_eval
```

Fonte: Elaborada pelo autor.

Este algoritmo simplifica a lógica de implementação negando os valores retornados, mantendo a recursão invertida para tratar ambos os jogadores como maximizadores, assim, avalia todas as jogadas possíveis até uma certa profundidade.

Por fim, o algoritmo Alpha-Beta Pruning, conforme proposto por Knuth e Moore (1975), representa uma otimização inteligente do Minimax. Ele reduz significativamente a quantidade de nodos a serem avaliados, preservando a qualidade das decisões. Em jogos de tabuleiro como o xadrez, onde a função possible_moves representa as jogadas legais disponíveis, o Alpha-Beta Pruning elimina ramos da árvore que não afetam a decisão final, tornando a busca mais eficiente.

Figura 03. Exemplo de implementação do algoritmo alphabeta.

```
    def alphabeta(board, depth, alpha, beta, maximizing_player):
        if depth == 0 or game_over(board):
            return evaluate_board(board)

        if maximizing_player:
            max_eval = float('-inf')
            for move in possible_moves(board):
                 eval = alphabeta(make_move(board, move), depth - 1, alpha,
beta, False)
                max_eval = max(max_eval, eval)
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break
```

```
            return max_eval
        else:
            min_eval = float('inf')
            for move in possible_moves(board):
                eval = alphabeta(make_move(board, move), depth - 1, alpha,
beta, True)
                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break
            return min_eval
```

Fonte: Elaborada pelo autor.

O Alpha-Beta Pruning mantém faixas alpha e beta para cortar ramos da árvore que não afetarão a decisão final. Alpha representa a avaliação mais alta encontrada até agora no caminho atual. Beta representa a avaliação mais baixa encontrada até agora no caminho atual. Se beta é menor ou igual a alpha, o loop é interrompido, pois os valores futuros não afetarão a decisão final.

4.3. Função heurística

A figura 04 ilustra os pesos utilizados para avaliar o tabuleiro de xadrez.

Figura 04.Material de pontuação

```
    __material__ = {
            "p": 1,
            "N": 3,
            "B": 3,
            "R": 5,
            "Q": 10,
            "K": 0
        }

    __checkmate__ = float('inf')
    __stalemate__ = 0
```

Fonte: Elaborado pelo autor.

Figura 05. função heurística.

```
    def _score(self, gs, valid_moves, white_to_move):
        if gs.board.checkmate:
            if gs.board.turn == 'w':
                return -self.__checkmate__
            else:
                return self.__checkmate__
        elif gs.board.stalemate:
            return self.__stalemate__
```

```
        score = 0
        for move in valid_moves:
            if move.piece_captured is not None:
                if move.piece_captured.suffix == 'K':
                    score += self.__checkmate__
                else:
                    score += self.__material__[move.piece_captured.suffix]
            if move.piece_moved is not None:
                score -= self.__material__[move.piece_moved.suffix]
            if move.is_pawn_promotion:
                score += self.__material__['Q']
            if move.is_castle_move:
                score += 1

        return score if white_to_move else -score
```

Fonte: Elaborado pelo autor.

A função *_score* é uma função de heurística que avalia o estado do jogo de xadrez e retorna um valor numérico que representa a qualidade desse estado para um determinado jogador. A heurística é uma parte crucial de muitos algoritmos de jogo, pois permite que o algoritmo faça uma busca eficiente no espaço do jogo.

A função *_score* começa verificando se o jogo atual (gs.board) está em xeque-mate ou em empate. Se o jogo está em xeque-mate, a função retorna um valor negativo se o turno for do jogador branco ('w'), indicando uma situação desfavorável para o jogador branco, e um valor positivo caso contrário. Se o jogo está em empate, a função retorna um valor predefinido para representar essa situação.

Em seguida, a função calcula um score com base nos movimentos válidos (*valid_moves*) disponíveis. Para cada movimento válido, a função verifica se uma peça foi capturada (*move.piece_captured*). Se uma peça foi capturada, a função adiciona ao score um valor correspondente ao tipo da peça capturada, armazenado em *self.__material__*. Se a peça capturada é um rei ('K'), a função adiciona um valor muito alto ao *score*, representando uma situação de xeque-mate.

A função também verifica se uma peça foi movida (*move.piece_moved*) e, em caso afirmativo, subtrai do score um valor correspondente ao tipo da peça movida. Isso representa o custo de mover uma peça.

Além disso, a função verifica se o movimento é uma promoção de peão (*move.is_pawn_promotion*) ou um roque (*move.is_castle_move*). Se o movimento é uma promoção de peão, a função adiciona ao score um valor correspondente a uma rainha ('Q'), representando o ganho de promover um peão a uma rainha. Se o movimento é um roque, a

função adiciona um pequeno valor ao score, representando o benefício estratégico do roque.

Finalmente, a função retorna o score se o turno for do jogador branco (*white_to_move*) e o negativo do score caso contrário. Isso garante que a função de heurística sempre avalie o jogo do ponto de vista do jogador cujo turno é o próximo.

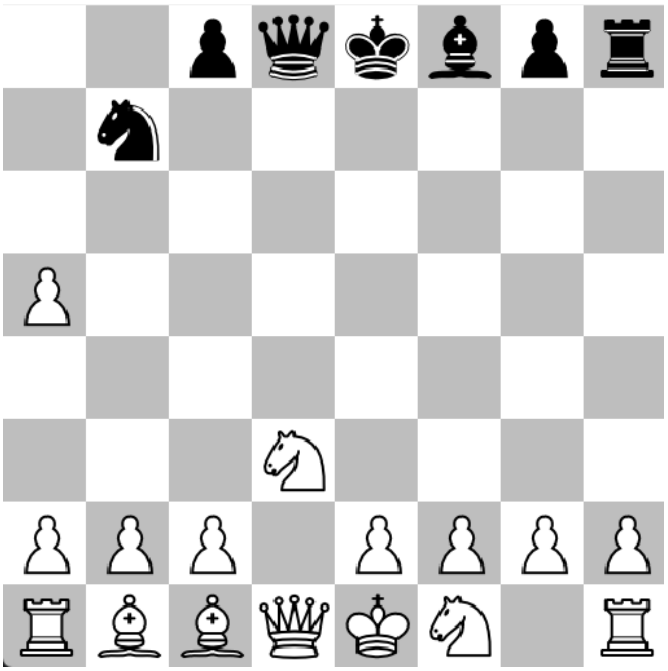
Em resumo, a função *_score* é uma heurística sofisticada que leva em consideração vários aspectos do jogo de xadrez para avaliar a qualidade de um estado de jogo. Ela desempenha um papel crucial na capacidade do algoritmo de jogo de tomar decisões informadas.

5. Resultados e Discussão

5.1. Apresentação dos resultados obtidos na implementação

Todos os resultados obtidos foram simulados na seguinte configuração do tabuleiro.

Figura 06. Captura do tabuleiro utilizado nos testes.



Fonte: Elaborada pelo autor.

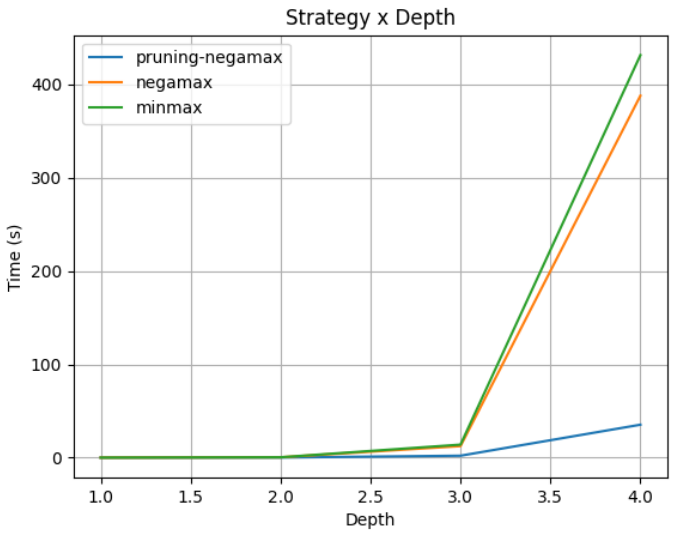
Aplicando o algoritmos algoritmos de busca sobre o tabuleiro, obtivemos os seguintes resultados.

Tabela 01. Resultados obtidos após aplicação dos algoritmos de busca no tabuleiro ilustrado na figura 06.

algoritmo	profundidade	tempo (s)	nós percorridos
Pruning negamax	1	0.013077259063720703	1
	2	0.1461472511291504	31
	3	2.3466720581054688	310
	4	36.345452547073364	9143
Negamax	1	0.013028383255004883	1
	2	0.3745734691619873	31
	3	13.880924224853516	1011
	4	381.7840209007263	29730
Minimax	1	0.013252973556518555	1
	2	0.3866403102874756	31
	3	13.680742025375366	1011
	4	381.80749011039734	29730

Fonte: Elaborada pelo autor.

Figura 07. Gráfico comparativo entre tempo e profundidade variando as estratégias.



Fonte: Elaborada pelo autor.

O gráfico apresenta a comparação do tempo de execução entre três estratégias algorítmicas - pruning-negamax, negamax e minimax - quando aplicadas em jogos de xadrez.

No eixo X, temos a profundidade (Depth) que varia de 1 a 4. No eixo Y, temos o tempo de execução em segundos. Três linhas coloridas representam diferentes algoritmos: pruning-negamax em azul, negamax em verde e minimax em amarelo.

A estratégia pruning-negamax (em azul) é consistentemente a mais eficiente em termos de tempo, mantendo-se relativamente plana e baixa ao longo do aumento da profundidade. A estratégia negamax (em verde) também permanece baixa, mas mostra um ligeiro aumento conforme a profundidade aumenta.

Por outro lado, a estratégia minimax (em amarelo) apresenta um aumento exponencial no tempo à medida que a profundidade aumenta, tornando-se significativamente menos eficiente. Isso indica que, para jogos de xadrez com maior profundidade, a estratégia minimax pode não ser a mais adequada devido ao seu alto tempo de execução.

Esses resultados são importantes para entender a eficiência desses algoritmos em jogos de xadrez e podem ajudar na escolha da estratégia mais adequada dependendo da profundidade do jogo.

5.2. Discussão crítica dos resultados em relação aos objetivos propostos.

A discussão crítica dos resultados obtidos na implementação dos algoritmos pruning-negamax, negamax e minimax em jogos de xadrez revela insights significativos em relação aos objetivos propostos.

O objetivo principal era analisar a eficiência desses algoritmos em termos de tempo de execução. Os resultados mostram que a estratégia pruning-negamax é a mais eficiente, seguida pela negamax, enquanto a minimax se mostrou significativamente menos eficiente à medida que a profundidade do jogo aumenta.

Esses resultados são consistentes com a teoria subjacente a esses algoritmos. O pruning-negamax, que é uma variante do negamax com otimização de poda alfa-beta, é projetado para melhorar a eficiência ao evitar a exploração desnecessária de ramos na árvore de jogo. Isso é evidenciado pelo seu desempenho superior em todas as profundidades testadas.

Por outro lado, a estratégia minimax, apesar de ser um algoritmo fundamental na teoria dos jogos, mostra limitações práticas significativas devido ao seu alto tempo de execução, especialmente em jogos de maior profundidade. Isso sugere que, embora o

minimax possa ser teoricamente sólido, suas implicações práticas são limitadas em cenários de jogo complexos, como o xadrez.

Portanto, os resultados obtidos estão alinhados com os objetivos propostos e fornecem uma validação empírica das propriedades teóricas desses algoritmos. No entanto, é importante notar que esses resultados são específicos para o contexto do xadrez e podem variar em outros domínios de jogo. Além disso, outros fatores, como a qualidade da função de avaliação e a disponibilidade de recursos computacionais, também podem influenciar o desempenho desses algoritmos. Portanto, futuras pesquisas podem explorar esses aspectos para uma compreensão mais completa da eficiência desses algoritmos em diferentes contextos de jogo.

6. Desafios e Futuras Perspectivas

6.1. Identificação de desafios encontrados durante o desenvolvimento.

Durante o desenvolvimento do projeto, um dos principais desafios encontrados foi a preparação de um ambiente adequado para testar as heurísticas dos algoritmos de jogo. Dada a natureza complexa do xadrez, a criação de um ambiente de jogo que pudesse simular com precisão as várias situações possíveis provou ser uma tarefa desafiadora.

Primeiramente, o xadrez é um jogo de estratégia profundo com um grande número de possíveis configurações de tabuleiro. Isso exigiu a implementação de um modelo de jogo robusto que pudesse representar adequadamente todas essas configurações. Além disso, o modelo de jogo precisava ser eficiente em termos de memória e tempo de execução, pois qualquer ineficiência poderia afetar adversamente o desempenho dos algoritmos de jogo.

Em segundo lugar, a implementação das regras do xadrez também apresentou desafios. Cada peça no xadrez tem suas próprias regras de movimento, e algumas dessas regras podem ser bastante complexas. Além disso, existem várias condições especiais, como o en passant e o roque, que também precisavam ser implementadas corretamente.

Finalmente, a avaliação do desempenho dos algoritmos de jogo exigiu a criação de uma variedade de cenários de jogo. Isso envolveu a geração de diferentes configurações de tabuleiro e a simulação de diferentes estratégias de jogo. A criação desses cenários de teste foi um processo demorado e exige um entendimento profundo do jogo de xadrez.

Apesar desses desafios, a experiência adquirida durante o desenvolvimento deste projeto foi inestimável. Ela destacou a importância de um design de software cuidadoso e a necessidade de testes rigorosos ao desenvolver algoritmos complexos. Além disso, os desafios enfrentados durante o desenvolvimento serviram como uma oportunidade valiosa para aprimorar as habilidades de resolução de problemas e pensamento crítico.

6.2. Sugestões para melhorias e otimizações nos algoritmos.

Sugestões para melhorias e otimizações nos algoritmos. Bibliotecas como NumPy, SciPy e Pandas são otimizadas para operações matemáticas e de dados complexas. Elas podem ser usadas para melhorar a eficiência do código, especialmente em operações que envolvem matrizes e vetores.

Dada a natureza paralela dos jogos de xadrez, onde várias jogadas podem ser exploradas simultaneamente, técnicas de paralelização podem ser aplicadas para melhorar o desempenho. Isso pode ser feito usando bibliotecas como multiprocessing ou frameworks como Dask e Ray. A eficiência dos algoritmos de jogo depende em grande parte da qualidade das heurísticas utilizadas. Portanto, heurísticas mais eficientes podem resultar em melhorias significativas no desempenho.

A poda alfa-beta é uma técnica eficaz para reduzir o espaço de busca em jogos de xadrez. No entanto, existem outras técnicas de poda, como a poda de Monte Carlo, que podem ser exploradas para melhorar ainda mais a eficiência.

O aprendizado de máquina pode ser usado para aprender heurísticas a partir de partidas de xadrez anteriores. Isso pode resultar em heurísticas mais precisas e, portanto, em algoritmos de jogo mais eficientes.

7. Conclusão

7.1. Conclusões gerais derivadas do estudo.

Ao encerrar esta investigação aprofundada sobre os algoritmos de busca no contexto de jogos estratégicos, com foco particular no xadrez, emerge uma série de conclusões cruciais que transcendem as fronteiras do tabuleiro virtual. Este estudo proporcionou uma compreensão mais profunda não apenas dos métodos utilizados para aprimorar a tomada de decisões em jogos, mas também sobre as implicações práticas e as possíveis direções futuras.

Em primeiro lugar, fica evidente que o algoritmo Minimax, embora fundamental em teoria dos jogos, enfrenta desafios consideráveis quando aplicado a jogos complexos como o xadrez. A sua busca exaustiva por todas as jogadas possíveis até uma determinada profundidade revelou-se computacionalmente intensiva, especialmente em cenários de jogos mais profundos. Isso destaca a importância de não apenas compreender a teoria por trás dos algoritmos, mas também de adaptá-los inteligentemente ao contexto específico do jogo em questão.

A introdução do NegaMax representou uma solução elegante para simplificar a implementação do Minimax, superando algumas das limitações práticas observadas. Ao

negar os valores retornados pelo mesmo ramo da árvore de busca, o NegaMax simplificou a lógica, tornando-a mais acessível e eficiente. Esta abordagem revelou-se valiosa para jogos de dois jogadores, como o xadrez, onde a complexidade da lógica de implementação é uma consideração crucial.

Ademais, o Alpha-Beta Pruning demonstrou ser uma otimização inteligente que contribuiu significativamente para a eficiência computacional do Minimax. Ao reduzir a quantidade de nodos a serem avaliados, preservando a qualidade das decisões, este algoritmo provou ser essencial para enfrentar a complexidade computacional inerente a jogos estratégicos profundos como o xadrez. A sua capacidade de eliminar ramos da árvore que não afetam a decisão final mostrou-se especialmente valiosa, destacando a importância da eficiência na busca por soluções ótimas.

A integração de heurísticas ao Minimax revelou-se crucial para aprimorar ainda mais a eficácia do algoritmo. Essas funções de avaliação, ao aproximarem o valor de uma posição no tabuleiro, proporcionam ao Minimax orientações mais precisas e informadas. A combinação estratégica de heurísticas com algoritmos de busca não apenas eleva o desempenho do Minimax, mas também destaca a importância de abordagens híbridas para enfrentar desafios complexos.

Por fim, a análise comparativa dos algoritmos através de testes práticos, exemplificados no gráfico de tempo de execução, reforçou as conclusões derivadas dos estudos teóricos. A estratégia pruning-negamax destacou-se como a mais eficiente em termos de tempo, mantendo-se relativamente estável mesmo em profundidades mais elevadas. Este resultado prático valida as vantagens observadas durante as discussões teóricas sobre a eficiência do NegaMax e do Alpha-Beta Pruning.

7.2. Implicações práticas e contribuições para a área de Inteligência Artificial.

As implicações práticas e contribuições para a área de Inteligência Artificial derivadas deste estudo não só elevam o padrão de desempenho em jogos estratégicos, mas também pavimentam o caminho para o desenvolvimento de sistemas inteligentes mais eficientes e adaptáveis. Ao cruzar as fronteiras teóricas e aplicar esses conhecimentos em cenários práticos, estamos moldando o futuro da IA, onde a capacidade de tomar decisões estratégicas e eficientes é um componente essencial. Este estudo não é apenas um avanço no campo dos jogos, mas uma peça chave na evolução contínua da Inteligência Artificial.

8. Referências

Knuth, D. E., & Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence, 6(4), 293–326

Pohl, I. (1971). Heuristic Search viewed as Path Finding in a Graph. Artificial Intelligence, 2(3-4), 227–246.

Russell, S., Norvig, P., Davis, E., & Moore, J. (2018). Inteligência Artificial: Uma Abordagem Moderna. Pearson.

Schaeffer, J., Culberson, J., Treloar, N., & Knight, B. (1989). A world championship caliber checkers program. Artificial Intelligence, 40(1-3), 1–35.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140-1144.

Yannakakis, G. N., Liapis, A., & Alexopoulos, C. (2018). Artificial intelligence for the synthesis of game content. IEEE Transactions on Games, 10(3), 269-271.