Relatório Final
IA de xadrez usando grafos

Henrique Alves de Fernando, 236538
Professor: Ruben Interian

***Resumo*—Este projeto apresenta o desenvolvimento de uma Inteligência Artificial (IA) de xadrez que utiliza a teoria dos grafos para modelar e analisar posições no tabuleiro. Essa abordagem possibilitou a identificação e o aprimoramento de padrões estratégicos e táticos com base em dados de partidas reais, promovendo uma tomada de decisão informada e eficiente. A IA foi construída com uma função de avaliação personalizada e implementada através do algoritmo minimax, otimizado com alfa-beta pruning para aumentar a velocidade e a precisão da análise de jogadas. O sistema demonstrou desempenho competitivo em partidas, validando a eficácia da abordagem teórica adotada.**

***Index Terms*—Xadrez, Inteligência Artificial, Grafos**

I. Introdução

O XADREZ é um dos jogos mais antigos e complexos, servindo há décadas como campo de estudo e desafio para o desenvolvimento de Inteligências Artificiais (IA). Este projeto busca utilizar as capacidades da IA no xadrez por via da teoria dos grafos para modelar e analisar posições no tabuleiro, explorando assim a complexidade do jogo de forma estruturada. A implementação se baseia em uma função de avaliação detalhada, construída a partir de dados históricos de partidas, e em um algoritmo minimax otimizado com alfa-beta pruning, permitindo a identificação e exploração de padrões estratégicos e táticos. Essa abordagem visa desenvolver uma IA capaz de jogar xadrez com desempenho competitivo, contribuindo para a pesquisa em IA aplicada a jogos de alta complexidade.

II. Objetivos

O objetivo do projeto é a implementação de um algoritmo capaz de analisar uma posição de xadrez, calcular uma série de lances e tomar uma boa decisão de melhor lance a partir disso. Com esse intuito, foi feita a análise de um grande volume de dados de partidas reais e foram extraídas informações mostrando a relação entre a avaliação de uma posição com uma série de fatores relacionados às peças e ao jogo, que serão discutidos na seção III. Com esses dados, foi possível implementar uma função de avaliação que determina quão boa uma posição é para o branco ou preto.

Feito isso, o algoritmo minimax [1] com otimização alpha-beta pruning [2] será empregado, de forma que o algoritmo possa ao mesmo tempo calcular de forma mais rápida e mais precisa.

Finalmente, esse algoritmo foi publicado como um BOT de xadrez no site lichess [3], onde ele pode jogar contra seres humanos online.

III. Coleta, análise e aplicação de dados

A. Coleta e pré-processamento de dados

O conjunto de dados utilizado neste projeto se baseou em 200.000 posições de xadrez, cada uma representada por uma string no formato FEN (Forsyth-Edwards Notation) junto com uma avaliação pré-calculada. As strings FEN descrevem a disposição de todas as peças no tabuleiro, enquanto as avaliações são valores numéricos que indicam a vantagem relativa de um dos lados. Esses dados foram extraídos de partidas reais disponibilizadas pelo repositório público Chess-DataContributor [4], totalizando aproximadamente 40 MB de informações.

No banco de dados criado, armazenam-se os valores heurísticos descritos e discutidos na seção III, os quais auxiliam na avaliação das posições e o banco como um todo possui 365MB e pode ser acessado em [5].

Para processar as posições, foi utilizada a biblioteca python-chess, que facilitou o carregamento e a manipulação dessas posições, permitindo a extração de informações fundamentais, como as localizações das peças e os movimentos legais.

Os resultados foram armazenados em um banco de dados SQLite, utilizando a notação FEN como chave primária para referência. Todo o processo de inserção de dados foi automatizado através de scripts que processaram as posições e inseriram as avaliações correspondentes no banco de dados, os quais podem ser vistos no diretório *data_analysis* de [6].

Os grafos desempenham um papel essencial neste processo, sendo a base para a análise de cada característica. Por exemplo, os grafos possibilitam a construção de relações entre mobilidade e controle central em milhares de posições, revelando tendências que podem não ser imediatamente perceptíveis a partir dos dados brutos. Além disso, a distribuição da conectividade entre peças, fortemente baseada em grafos, permite detectar padrões e correlações de forma mais eficaz. Esses grafos foram gerados a partir dos dados processados e armazenados no banco de dados, tornando a análise mais profunda e intuitiva.

B. Análise dos grafos

Neste projeto, o tabuleiro e as peças foram representados como um grafo, onde cada casa é um vértice e as conexões entre casas (movimentos possíveis das peças) formam as arestas. Nesta estrutura, cada casa possui uma lista de adjacência associada, que representa as casas "atacadas"ou alcançáveis a partir dela, bem como um registro das casas que apontam para ela. Com essas informações, torna-se possível analisar

detalhadamente uma série de aspectos estratégicos e táticos da posição, que foram definidos da seguinte forma:

- **Contagem de Material:** A diferença material entre os lados, ponderada pelo valor de cada peça (por exemplo, peões, cavalos, bispos, torres e dama). Valores positivos indicam vantagem para as brancas, enquanto valores negativos favorecem as pretas.
- **Material Total:** O valor agregado de todas as peças no tabuleiro, independentemente da cor.
- **Mobilidade:** Número de movimentos legais disponíveis para cada lado, refletindo a liberdade de ação das peças.
- **Controle Central:** A influência sobre as casas centrais (e4, d4, e5, d5), que são importantes para o domínio posicional e para o desenvolvimento de peças.
- **Segurança do Rei:** A vulnerabilidade de cada rei, analisando se as casas ao redor dele estão sob ataque.
- **Conectividade:** Como as peças de uma mesma cor se apoiam mutuamente, formando uma rede de defesa e controle.
- **Posição das Peças:** As casas ocupadas por cada peça e a média das avaliações nessas posições, criando um "mapa de calor"da eficácia posicional das peças.

Esses fatores fornecem uma visão detalhada e integrada das forças e fraquezas da posição, permitindo uma análise e avaliação mais profunda e contextualizada das jogadas possíveis e das configurações táticas.

Com o banco de dados já populado com as propriedades das posições, fez-se uma regressão linear usando o valor de cada uma delas como variável dependente e a avaliação da posição como variável independente (a implementação está no arquivo 'heat_map.py' de [6]). Encontrou-se os seguintes coeficientes:

Tabela I
COEFICIENTES DE AVALIAÇÃO DAS CARACTERÍSTICAS

Característica	Coeficiente
Mobilidade	2.89
Controle Central	-1.12
Segurança do Rei	17.07
Conectividade	0.05

Isso sugere que a segurança do rei influencia muito na avaliação da posição e deve ser levada em conta em uma boa função de avaliação, enquanto a conectividade é uma característica dispensável, já que não tem tanta influência. Ao contrário da regra empírica de que o controle central tem um impacto positivo na avaliação, os dados mostraram que o controle das casas centrais é levemente prejudicial e, devido a essa discordância, optou-se por não levar em conta esse aspecto na função de avaliação.

Um fator crucial no cálculo desses coeficientes foi a consideração exclusiva de posições em que havia um equilíbrio material. Isso se deve ao fato de que a quantidade de material é o elemento mais significativo na avaliação de uma posição. Em situações onde um dos lados possuía consideravelmente mais material do que o outro, mesmo que todas as outras características favorecessem um dos lados, o desequilíbrio material comprometeria a avaliação, pois ele é determinante para o resultado final.

Em relação à avaliação da posição das peças, foram criados "mapas de calor"com a avaliação média da posição de acordo com a posição de cada peça no tabuleiro, que podem ser visualizados no Anexo A. Para algumas peças, a avaliação média calculada é coerente com o conhecimento empírico, por exemplo, peões brancos na 7ª fileira (próximos à promoção) resultam em boas avaliações médias. Entretanto, muitos dos resultados se distanciam bastante do conhecimento já bem estabelecido e não parecem ajudar tanto na avaliação. Isso provavelmente se dá pela quantidade insuficiente de dados, pois, apesar de termos 200 000 posições, há 64 casas em que as peças podem estar e, por simplicidade, considerando que as peças podem estar em qualquer casa com uma mesma probabilidade, há aproximadamente 6000 dados para cada casa (considerando duas peças de cada tipo, $\frac{200000}{64} \times 2 \approx 6000$), o que de fato não é uma quantidade que gera dados muito precisos. Portanto, optou-se por usar os mapas de calor que seguem o conhecimento empírico e podem ser vistos em [7], o que de fato fez com que o computador jogasse melhor.

A função de avaliação final pode ser vista como pseudocódigo abaixo e também no arquivo *AI.ts*, de [6].

Algorithm 1 Função de avaliação

```
function EVALUATE
    if board.isCheckmate() then
        return −∞
    else if board.isDraw() then
        return 0
    end if
    eval ← 0
    for piece in board.allPieces do
        eval ← eval + piece.value
        eval ← eval + piece.positionEvaluation
    end for
    eval ← eval + 3 * mobilityBalance
    eval ← eval + 10 * kingSafetyBalance
    return eval
end function
```

Aqui, os valores 3 e 10 são próximos aos coeficientes encontrados na tabela I e demonstraram maior sobriedade na decisão de melhores lances

IV. ESTRUTURAS DE DADOS

A. *Bitboards*

Apesar de não terem sido a estrutura de dados escolhida para o desenvolvimento do projeto, os bitboards são mais adequados no desenvolvimento de bots de xadrez e vale a pena entender o básico por trás deles.

Bitboards são uma estrutura de representação compacta e eficiente de tabuleiros em jogos como o xadrez, onde cada posição é mapeada para um conjunto de bits de um número inteiro (geralmente 64 bits, correspondendo a um tabuleiro de 8x8). Cada peça ou tipo de peça é representado em um bitboard separado, facilitando operações rápidas de manipulação e consulta de posições com operadores de bits, como AND, OR, XOR, e shifts. Essas operações, além de serem executadas em

tempo constante, são processadas diretamente pelo hardware, o que torna a manipulação de dados extremamente rápida e eficiente.

Em um bitboard, cada bit representa uma casa específica do tabuleiro. Por exemplo, um bitboard para as torres pode ter um bit "ligado"(1) nas posições ocupadas pelas torres e "desligado"(0) nas demais posições. Para realizar uma jogada, basta operar diretamente no bitboard correspondente — movimentando uma peça ou verificando se uma casa está ocupada, tudo isso com operações bitwise. Por exemplo, um deslocamento de bits pode simular o movimento de uma peça na direção desejada (uma torre pode "andar"para a direita ao realizar um shift no bitboard).

Conforme discutido, o xadrez também pode ser modelado como grafos, e, embora eles sejam úteis para algumas análises de jogo e para representar a conexão entre movimentos, sua estrutura é mais complexa e consome mais memória e tempo para os cálculos, pois precisam armazenar explicitamente as conexões entre casas.

Dessa forma, o uso de bitboards possui diversas vantagens em relação a uma implementação baseada em grafos. Dentre elas, estão:

- **Eficiência em Memória e Processamento:** Os bitboards usam uma representação extremamente compacta (64 bits por tabuleiro ou peça), enquanto grafos exigem listas de adjacência ou matrizes de adjacência, que consomem significativamente mais memória e são menos diretas de manipular em um nível binário.
- **Operações em Tempo Constante:** Com bitboards, ações comuns, como verificar ameaças a uma peça, mover peças e validar ocupações de casas, podem ser realizadas em tempo constante (O(1)) por meio de operações de bits. Em uma estrutura de grafo, essas operações envolveriam buscas ou consultas em listas ou matrizes, resultando em um desempenho menos eficiente. No caso da implementação feita, essas buscas e consultas aconteceram em O(n), com n sendo o tamanho do tabuleiro.
- **Implementação Simplificada de Movimentos e Máscaras:** No xadrez, há várias regras que restringem o movimento de peças, como os movimentos diagonais dos bispos e a movimentação em “L” dos cavalos. Com bitboards, essas direções de movimento podem ser facilmente representadas com máscaras predefinidas (bitmasks) e operações de deslocamento de bits. Isso simplifica muito a implementação de regras em comparação com a necessidade de verificar todas as arestas possíveis em um grafo.
- **Facilidade na Implementação de Avaliações e Máscaras de Ataque:** Em engines de xadrez, é importante calcular rapidamente áreas controladas por peças e ameaças potenciais. Com bitboards, é possível pré-calcular máscaras que representam todos os possíveis movimentos de uma peça a partir de uma posição específica e aplicar essas máscaras para simular ataques em uma única operação. Em grafos, seria necessário realizar buscas a partir de cada posição para encontrar todas as casas possíveis de ataque.

B. Grafos

Embora menos eficientes, os grafos também são uma abordagem interessante para modelar o tabuleiro de xadrez e as interações entre as peças. No xadrez, o tabuleiro pode ser representado como um grafo em que cada casa é um nó e cada movimento possível entre duas casas é uma aresta direcionada. Assim, cada peça possui um conjunto específico de restrições de movimento que define as arestas que ela pode percorrer. Por exemplo, a torre estaria limitada a mover-se ao longo das arestas horizontais e verticais, enquanto o bispo se moveria apenas ao longo das diagonais. Essa estrutura permite que o algoritmo de IA simule o jogo a partir de uma perspectiva gráfica.

Ao tratar o tabuleiro como um grafo, pode-se empregar algoritmos de busca clássicos, como busca em largura (BFS) e busca em profundidade (DFS), para explorar movimentos e possíveis sequências de jogadas. No caso do minimax, a implementação da busca de melhor lance com DFS é mais adequada e foi implementada, como será discutido mais adiante.

Além de auxiliar na busca por movimentos, o uso de grafos permite identificar padrões no tabuleiro que podem indicar vantagens ou fraquezas estratégicas. Por exemplo, a conexão entre peões e o suporte mútuo entre peças defensivas podem ser vistos como subgrafos específicos, onde certos padrões representam configurações táticas valiosas. Essa análise é essencial para que a IA compreenda formações de peças que oferecem vantagens posicionais, como o controle do centro do tabuleiro ou a criação de redes defensivas, que foi analisado como a "conectividade"entre as peças neste projeto.

Além disso, em momentos defensivos, a IA pode utilizar grafos para mapear não apenas as suas próprias jogadas, mas também as ameaças do oponente. Um "grafo de ameaças"pode representar as casas vulneráveis e os movimentos potenciais do adversário, permitindo que a IA compreenda as áreas mais ameaçadas e antecipe movimentos do oponente, como feito com a "segurança do rei". Esse tipo de estrutura gráfica pode ser especialmente útil em partidas complexas, onde o algoritmo precisa reconhecer padrões de ataque para implementar contra-ataques ou defesas eficazes.

C. tabelas de transposição

As tabelas de transposição [8] são estruturas essenciais na implementação de algoritmos de busca em jogos como xadrez, onde a árvore de busca é complexa e o mesmo estado de jogo pode ser alcançado por diferentes sequências de movimentos. Essa técnica se baseia no conceito de “memorizar” estados de jogo já analisados para evitar avaliações redundantes, economizando tempo de processamento e memória.

No xadrez, uma mesma posição (ou transposição) pode surgir de várias sequências de movimentos. A tabela de transposição evita que o algoritmo analise a mesma posição repetidas vezes, aproveitando os resultados das avaliações anteriores. Isso reduz o número de nós explorados na árvore de busca, aumentando a eficiência.

Uma tabela de transposição é geralmente implementada como uma hashtable onde cada posição é usada como chave,

e os dados sobre a posição (incluindo sua avaliação e outros metadados) são armazenados como valores.

Embora as tabelas de transposição sejam fundamentais para a eficiência dos algoritmos de busca em IA de xadrez, elas apresentam algumas limitações que podem impactar a precisão das avaliações. Colisões de hash, substituição de posições importantes devido a restrições de memória e dependência de limites de avaliação podem levar a decisões imprecisas em posições críticas. Esses fatores, combinados com o consumo significativo de memória, tornam necessário um gerenciamento cuidadoso das tabelas de transposição para mitigar erros e garantir que a IA aproveite ao máximo as vantagens dessa técnica sem comprometer a qualidade da busca. Assim, optou-se por não aplicar essa técnica no projeto, apesar de ser uma poderosa forma de economizar tempo de processamento e melhorar o desempenho.

V. Algoritmos

A. *Minimax*

Um dos algoritmos a ser utilizado é o minimax, que é uma técnica de decisão utilizada em jogos de dois jogadores, como o xadrez, onde um jogador busca maximizar seu ganho (Max) enquanto o outro busca minimizar a perda (Min). Ele explora todas as possíveis jogadas, criando uma árvore de decisão, e avalia cada posição, assumindo que ambos os jogadores jogam de forma ótima.

Além disso, foi utilizado a técnica de alfa-beta pruning, uma otimização do algoritmo minimax que reduz a quantidade de nós na árvore de decisão que precisam ser avaliados. Nesse método, dois parâmetros são mantidos: alfa, que representa a melhor avaliação possível para o jogador branco até o momento, e beta, que representa a melhor avaliação possível para o jogador preto. Um valor alto de alfa indica uma posição favorável ao branco, enquanto um valor baixo de beta indica uma posição favorável ao preto (pois uma avaliação negativa é boa para ele). Durante a busca em profundidade na árvore de posições, quando uma posição é avaliada e se verifica que a melhor opção para o branco (alfa) é maior que o melhor valor que o preto pode garantir (beta), a busca nessa ramificação é interrompida (poda-se a árvore). Isso ocorre porque o jogador preto, sendo racional, nunca escolheria um lance que levaria a uma posição pior do que outra já disponível. Assim, ao podar a árvore, evitam-se avaliações desnecessárias, tornando o algoritmo mais eficiente. A implementação do algoritmo em pseudocódigo pode ser vista abaixo e na função *alfabeta* de *IA.ts*, de [6].

Note também que calcular o minimax começando pelos melhores lances possíveis é uma estratégia que potencializa a eficiência da busca, quando combinado com o alfa-beta pruning. Ao explorar primeiro as jogadas mais promissoras, é mais provável que a poda ocorra mais cedo na árvore de decisão, reduzindo drasticamente o número de posições que precisam ser avaliadas. Isso acontece porque, ao avaliar rapidamente bons movimentos, as janelas alfa e beta são ajustadas para valores mais restritos, possibilitando a eliminação de outras ramificações que levariam a posições menos favoráveis ou redundantes. Essa abordagem não apenas acelera o processo de decisão, mas também permite à IA analisar mais profundamente as opções de jogo dentro do mesmo limite de tempo, resultando em decisões mais informadas e estratégicas. Com isso em mente, os lances possíveis foram ordenados começando pelos xeques e capturas, que têm uma probabilidade maior em resultar em boas posições.

Algorithm 2 Minimax

```
1:  function MINIMAX(node, depth, maximizingPlayer)
2:      if depth = 0 or node is a terminal node then
3:          return Evaluate(node)
4:      end if
5:      if maximizingPlayer then
6:          bestValue ← −∞
7:          for each child of node do
8:              val ← MINIMAX(child, depth - 1, false)
9:              bestValue ← max(bestValue, val)
10:         end for
11:         return bestValue
12:     else
13:         bestValue ← +∞
14:         for each child of node do
15:             val ← MINIMAX(child, depth - 1, true)
16:             bestValue ← min(bestValue, val)
17:         end for
18:         return bestValue
19:     end if
20: end function
```

B. *Quiescence Search*

Esse algoritmo é uma extensão do minimax, que ajuda a evitar a chamada "instabilidade de horizonte", onde o computador continua analisando apenas posições obtidas por capturas ou xeques, por exemplo. A Quiescence Search explora essas posições instáveis além do limite normal da profundidade de busca, até alcançar uma posição "quieta", onde não há capturas imediatas ou xeques. Pode-se definir uma profundidade máxima em que o computador continua a busca até não encontrar esses tipos de lance e o principal benefício é uma menor probabilidade da IA perder uma peça ou cair em uma sequência tática devido à profundidade da busca ser muito pequena.

Essa técnica chegou a ser implementada no projeto, mas levava a um consumo de tempo muito grande e a IA passou a perder muitas partidas por falta de tempo. Por isso, essa técnica não foi aplicada na versão final.

C. *Geração de lances*

Ao mover uma peça, nem todas as peças no tabuleiro precisam ter seus lances recalculados. O cálculo seletivo dos movimentos ajuda a evitar um grande número de operações desnecessárias, economizando tempo e recursos computacionais, especialmente em posições complexas onde o tabuleiro pode estar quase totalmente ocupado.

Quando uma peça se move, o impacto de seu movimento geralmente é limitado a uma área específica. Por exemplo, ao

movimentar um peão, é improvável que lances de peças no lado oposto do tabuleiro sejam afetados, a menos que o peão crie ou interrompa uma linha de defesa ou ataque. Portanto, é possível focar no recalculo de movimentos de peças que estão próximas à casa de origem e de destino do peão, além de considerar se o movimento cria alguma nova rota para as peças de longo alcance, como torres, bispos ou damas.

Peças de longo alcance, como torres, bispos e damas, exigem uma atualização mais cuidadosa. Ao mover uma dessas peças, as linhas de ataque e defesa são afetadas ao longo de toda a extensão de seu movimento, possivelmente alterando as casas que outras peças controlam ou podem acessar. Por exemplo, uma torre movendo-se para uma coluna diferente pode abrir uma linha para outra torre ou bispo aliado, ou bloquear a visão de uma peça adversária. Por isso, o cálculo de movimentos após o deslocamento de uma peça de longo alcance frequentemente exige atualizar o controle de casas ao longo dessas linhas. Esse cálculo seletivo foi implementado de forma personalizada para cada peça, de acordo com seus movimentos, de maneira que o recálculo dos lances possíveis fosse minimizado.

VI. Avaliação

Feito o algoritmo, criou-se uma conta no lichess e foram executados os passos necessários para transformar a conta em uma conta de bot. Para isso, foi gerado um token de API usado na autenticação entre o bot e a plataforma e também o código lichess-bot [9] para conectar-se ao servidor. Assim, foi possível configurar o ambiente com o algoritmo construído jogando tanto contra jogadores reais quanto contra outros bots da plataforma.

Ao longo do tempo em que ele ficou online, obteve-se os seguintes resultados:

Tabela II
Estatísticas do bot em partidas reais

Partidas	99
Vitórias	33
Derrotas	58
Empates	8
Contra humano	49
Vitórias contra humano	29
Vitórias contra bots	4

Com esses resultados, ele ficou com 1394 pontos de rating na modalidade blitz (partidas de aproximadamente 5 minutos), o que significa que ele é melhor que 41,8% dos jogadores do site, como pode-se ver na figura 1.

A análise dos dados da tabela permite uma visão clara do desempenho do bot em diferentes contextos de partida. Com um total de 99 jogos, ele apresenta uma taxa de vitória geral de aproximadamente 33%, que é relativamente baixa em comparação com as derrotas, que somam 58 jogos (cerca de 59% das partidas). Os empates representam 8% do total, indicando que o bot é capaz de alcançar posições equilibradas em algumas situações, mas ainda tem dificuldades para transformar esses jogos em vitórias.

A análise segmentada entre partidas contra humanos e contra outros bots revela informações interessantes. Dos 49

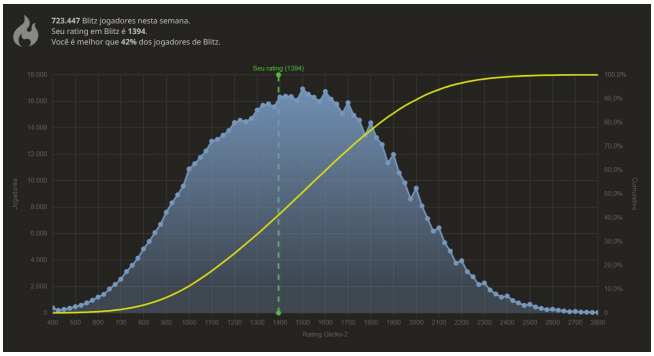

Figura 1. Desempenho do algoritmo jogando online

jogos contra jogadores humanos, o bot obteve 29 vitórias, o que representa uma taxa de vitória de cerca de 59% nesse tipo de confronto. Esse desempenho sugere que o bot consegue competir bem contra jogadores humanos com habilidades variadas, possivelmente se beneficiando de erros humanos ou jogadas menos calculadas em partidas rápidas de blitz. Em especial, isso se deve ao fato de que humanos cometem muito mais erros táticos (que perdem peças ou levam xeque-mate) que computadores. Como esses erros grosseiros têm um impacto muito grande na partida, surge-se uma maior taxa de vitórias.

Por outro lado, o desempenho do bot contra outros bots é significativamente mais fraco, com apenas 4 vitórias, sugerindo uma taxa de vitória de 8% nesse contexto. Isso pode indicar que o bot enfrenta dificuldades em confrontos com outros algoritmos, onde a precisão e a otimização de movimentos são muito mais rigorosas. Bots geralmente são menos propensos a erros e possuem estratégias consistentes e bem calculadas, o que explica o baixo desempenho do algoritmo aqui desenvolvido em comparação com esses oponentes.

O rating de 1394 na modalidade blitz, que coloca o bot acima de 41,8% dos jogadores, sugere que ele possui um desempenho intermediário. Esse resultado implica que o bot está acima de muitos jogadores casuais, mas ainda não possui uma força suficiente para competir em alto nível. Esse nível de rating reflete o potencial do bot, mas também destaca áreas para aprimoramento, especialmente no ajuste da estratégia contra outros algoritmos, onde ele tem um desempenho significativamente inferior.

Com isso, pode-se listar uma série de pontos de melhoria e otimização, tais como:

- **Uso de bitboards:** implica em uma geração mais rápida de lances, e por consequência, melhor desempenho;
- **Implementação usando C++:** pra fins de testes em uma interface gráfica user-friendly, o projeto foi desenvolvido em typescript, mas uma alternativa muito mais perfomática seria o desenvolvimento em C++, que é uma linguagem muito mais rápida de ser executada, o que implicaria em um considerável aumento de performance;
- **Definição da profundidade de cálculo de forma dinâmica:** Ao invés de usar uma profundidade fixa, como feito aqui, ela poderia ser variável, de acordo com a quantidade de material, tempo disponível e número bruto

de posições analisadas, por exemplo. Como no meio-jogo há muitas peças e, por consequência muitas possíveis ramificações, é mais custoso chegar a uma profundidade de 5 ou 6 lances no cálculo do minimax. Entretanto, em finais em que não há tantos lances possíveis, o cálculo se estender por mais lances é bem mais viável e é uma melhoria que poderia fazer muita diferença na força de jogo, especialmente em finais, que o algoritmo se mostrou bastante fraco;

- **Uso da quiescence search e transposition tables:** Essas técnicas já discutidas são amplamente utilizadas em bots de xadrez e certamente têm potencial de tornar este algoritmo melhor. Optou-se por não implementá-las no código final porque o consumo de tempo seria muito grande e isso levaria a muitas derrotas por falta de tempo. Entretanto, com melhorias na performance e viabilidade do uso dessas técnicas, o bot seria levado a outro patamar de força de jogo.

REFERÊNCIAS

[1] "Minimax" *Wikipedia*, 2023. [Online]. Disponível: https://pt.wikipedia.org/wiki/Minimax. [Acessado: Ago. 22, 2024].

[2] "Alpha-beta pruning" *Wikipedia*, 2024. [Online]. Disponível: https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning. [Acessado: Ago. 22, 2024].

[3] Lichess 2024. [Online]. Disponível: https://lichess.org/@/BOT_MC859. [Accessed: Aug. 22, 2024].

[4] "ChessDataContributor" *ChessDataContributor*, 2020. [Online]. Disponível: https://github.com/r2dev2/ChessDataContributor. [Acessado: Out. 31, 2024].

[5] "Pre-evaluated chess position and its features " *Pre-evaluated chess position and its features* , 2020. [Online]. Disponível: https://data.mendeley.com/datasets/7gpr9xtdpc/1. [Acessado: Out. 31, 2024].

[6] "Repositório do projeto" *Repositório do projeto*, 2024. [Online]. Disponível: https://github.com/HenriqueAlves16/MC859. [Acessado: Nov. 02, 2024].

[7] Piece-square tables [Online]. Disponível: https://www.chessprogramming.org/Simplified_Evaluation_Function. [Acessado: Nov. 03, 2024].

[8] Tabelas de transposição [Online]. Disponível: https://www.chessprogramming.org/Transposition_Table. [Acessado: Nov. 03, 2024].

[9] lichess-bot [Online]. Disponível: https://github.com/lichess-bot-devs/lichess-bot. [Acessado: Nov. 03, 2024].

APÊNDICE A
ANEXO I: MAPAS DE CALOR

Abaixo estão representados os mapas de calor gerados para cada peça de cada cor. Quando a avaliação é boa para o branco, a casa fica branca e tem um valor positivo, ao passo que a avaliação boa para o preto deixa a casa escura e tem um valor negativo. Para os casos em que não houve uma quantidade suficiente de dados (menos de 40 dados), nenhum valor foi atribuído à casa e ele foi definido como 0 no mapa.

Para peões, percebe-se que quanto mais próximos das casas de promoção eles estão, melhor tende a ser a avaliação. Para cavalos, bispos, torres e damas, percebe-se que a avaliação tende a ser melhor quando a peça está avançada, ou seja, próxima do território inimigo. Para os reis, percebe-se que quanto mais no canto do tabuleiro e em seu território, melhor tende a ser a avaliação, já que ele fica mais protegido.

Apesar dessas tendências que fazem sentido, vários valores estão muito discrepantes do que se espera. Por exemplo, não faz sentido a avaliação de um cavalo branco ser tão boa quando ele está em d8, que é uma casa perto da borda do tabuleiro (ele perde muita mobilidade). Da mesma forma, não faz sentido a avaliação de um rei preto em c6 ser melhor que ele em b8. Por vários exemplos como esses, optou-se por não usar esses dados na função de avaliação.

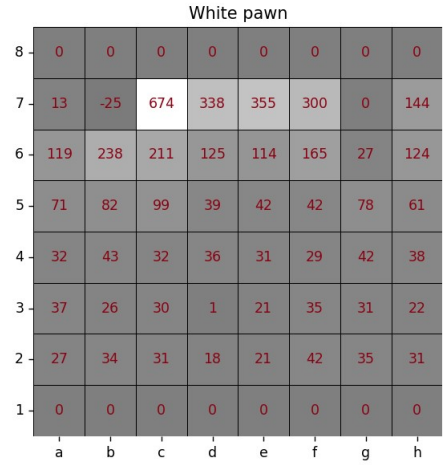

Figura 2. Mapa de calor para peões brancos.

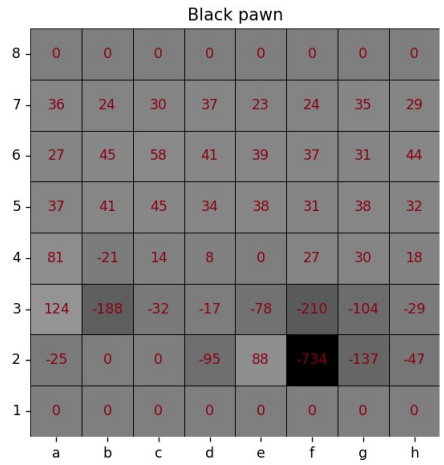

Figura 3. Mapa de calor para peões brancos.

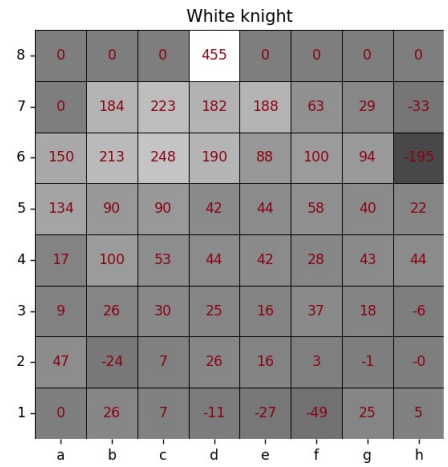

Figura 4. Mapa de calor para cavalos brancos.

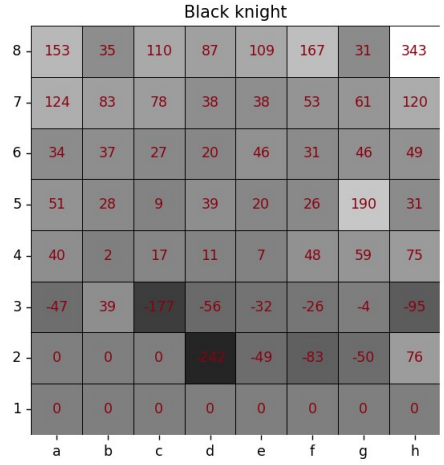

Figura 5. Mapa de calor para cavalos pretos.

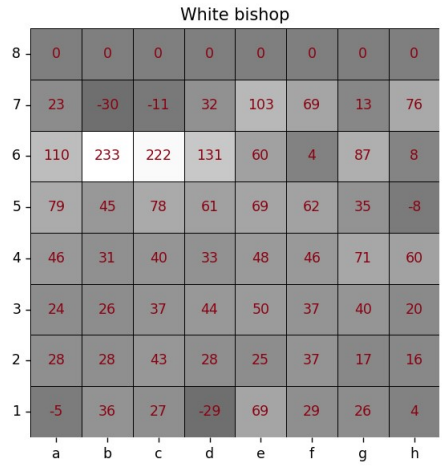

Figura 6. Mapa de calor para bispos brancos

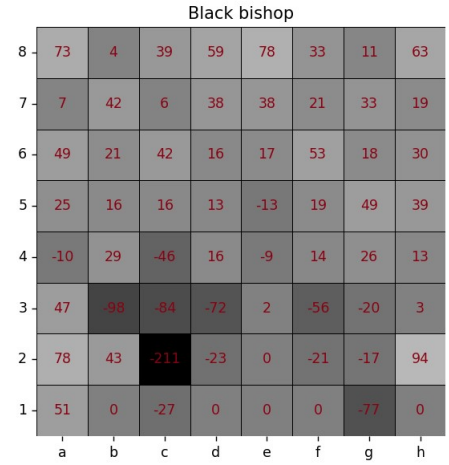

Figura 7. Mapa de calor para bispos pretos

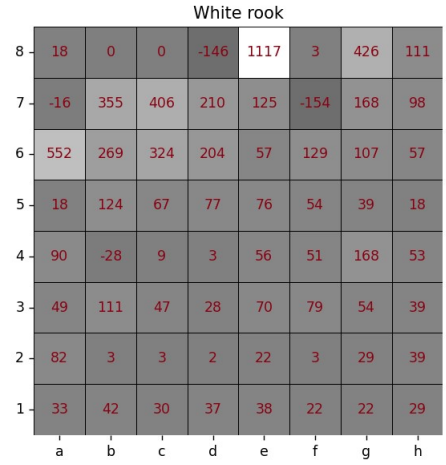

Figura 8. Mapa de calor para torres brancas

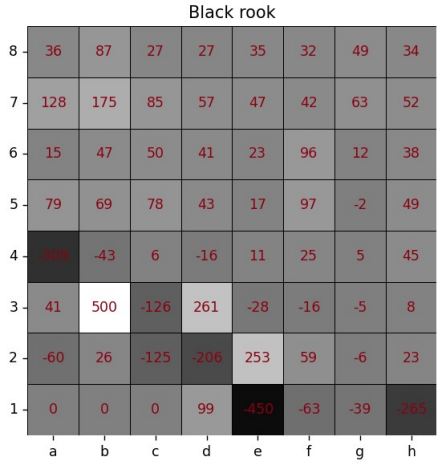

Figura 9. Mapa de calor para torres pretas

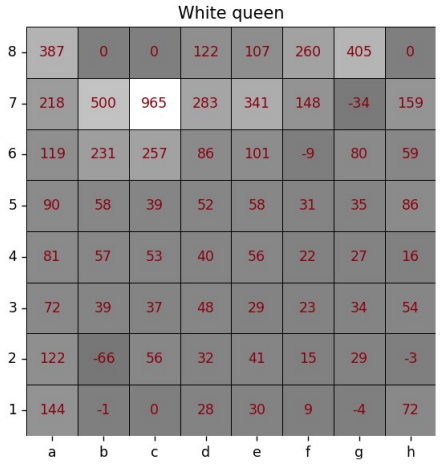

Figura 10. Mapa de calor para damas brancas

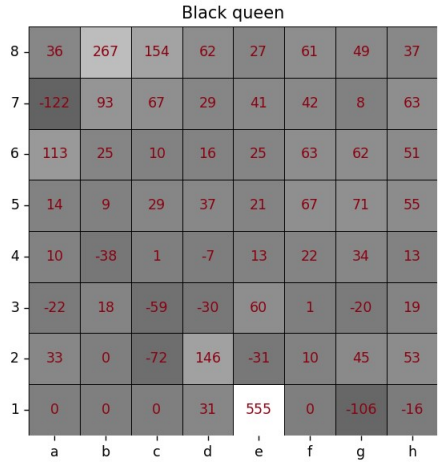

Figura 11. Mapa de calor para damas pretas

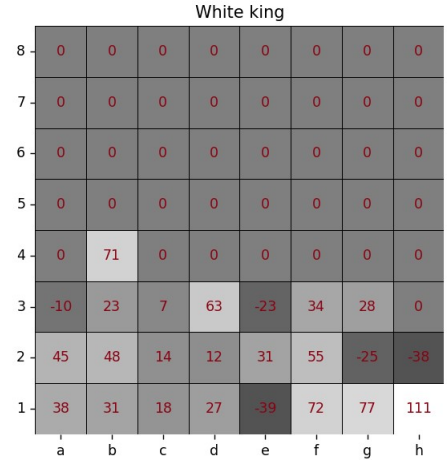

Figura 12. Mapa de calor para reis brancos

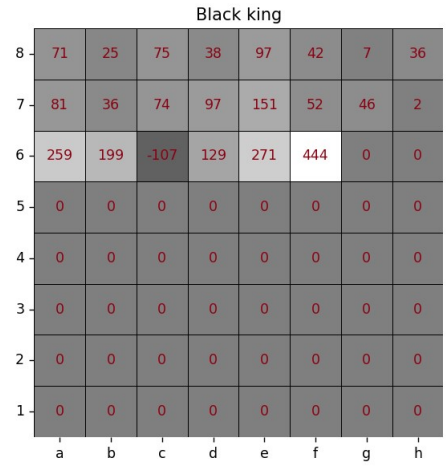

Figura 13. Mapa de calor para reis pretos