Desenvolvimento de uma Inteligência Artificial do Jogo de Xadrez, Explorando seus Fundamentos e Limites

UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES - CAMPUS DE ERECHIM - DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO

Matheus Fernando Finatto, Neilor Tonin

052486@aluno.uricer.edu.br, neilortonin@uricer.edu.br

Abstract. *This study develops a chess-playing artificial intelligence (AI) with a focus on Minimax with alpha-beta pruning and evaluation tables for decision-making, detailing the assignment of values to pieces and board positions and illustrating how these affect game-state assessment. Tests indicate that the AI, "FinattoBot," performs well against Stockfish at intermediate depths but could benefit from enhancements in mid-game positional evaluation.*

Resumo. *Este trabalho desenvolve uma inteligência artificial (IA) para jogar xadrez, aprofundando-se nas técnicas do algoritmo Minimax com poda alfa-beta e no uso de tabelas de avaliação para tomada de decisão explorando a importância da atribuição de valores às peças e às posições no tabuleiro e destacando como isso influencia a avaliação do estado do jogo. Testes mostram que a IA, chamada "FinattoBot", teve bom desempenho contra o Stockfish em profundidade intermediária, mas há espaço para aprimoramentos na análise posicional em fases de meio-jogo.*

1. INTRODUÇÃO

O xadrez, ao longo dos séculos, tem sido um campo de batalha intelectual onde estratégia, tática e raciocínio são postos à prova. A busca pela criação de uma inteligência artificial (IA) capaz de desafiar os mestres do xadrez nestes domínios tem cativado mentes e estimulado a imaginação há séculos.

Esse feito, porém, só foi alcançado em 1997, quando Garry Kasparov, o campeão mundial vigente, foi derrotado pelo Deep Blue, um supercomputador desenvolvido pela IBM. Este evento marcou o primeiro triunfo de um computador sobre um campeão mundial em uma partida completa de xadrez, realizada em condições de torneio e com controle de tempo, representando um momento histórico tanto para o xadrez quanto para a inteligência artificial. Esse confronto ficou imortalizado como um dos momentos mais emblemáticos na história do xadrez, acompanhado de perto por entusiastas do jogo, cientistas, especialistas em computação e o público em geral (Chess.com).

Baseado nisso, este trabalho se propõe a desenvolver uma IA voltada para o xadrez, explorando os avanços tecnológicos do século XXI, tanto em potência de hardware quanto técnicas de algoritmos. Assim, não apenas serão examinados os fundamentos teóricos que tornam essa interação entre IA e xadrez possível, mas também o potencial prático da IA em aplicações em partidas reais, buscando entender como essa tecnologia pode influenciar e enriquecer o jogo contemporâneo.

2. REFERENCIAL TEÓRICO

Antes de compreendermos a funcionalidade de uma IA que joga xadrez, precisamos de entender alguns conceitos básicos sobre o jogo e sobre IAs.

2.1. Valor das peças e avaliação de posições

No xadrez, cada peça possui um valor relativo, que serve para avaliar trocas e posições durante o jogo. Esses valores são baseados em sua mobilidade e potencial de influência no tabuleiro. A escala de valores tradicional é a seguinte (CHESS PROGRAMMING WIKI, 2024).

- **Peão (P)**: 10 pontos
- **Cavalo (N)**: 30 pontos
- **Bispo (B)**: 30 pontos
- **Torre (R)**: 50 pontos
- **Dama (Q)**: 90 pontos
- **Rei (K)**: 900

Tais valores são o ponto de partida tanto para jogadores quanto para computadores avaliarem quem está ganhando a partida. No entanto, nem sempre uma dama, que vale nove pontos, tem um valor inferior a duas torres, que juntas somam dez pontos. O contexto da partida é um fator ainda mais importante que o valor das peças para definir o vencedor (Chess.com, 2024).

Desta forma, outro fator amplamente utilizado para obtenção de vantagem posicional é o valor das peças no tabuleiro na posição em que se encontram. Por exemplo, o valor de um peão pode variar de -20 a +50 em seu valor base de 100, dependendo da posição que se encontra. Essas tabelas de posição são conhecidas como "*evaluation tables".*

Abaixo estão anexadas as figuras 1 e 2, exemplos de *evaluation tables* (para o jogador de brancas) encontradas no site Chess Programming (2024):

Figura 1 - Evaluation table relativa a peões

```
// pawn
 0,  0,  0,  0,  0,  0,  0,  0,
50, 50, 50, 50, 50, 50, 50, 50,
10, 10, 20, 30, 30, 20, 10, 10,
 5,  5, 10, 25, 25, 10,  5,  5,
 0,  0,  0, 20, 20,  0,  0,  0,
 5, -5,-10,  0,  0,-10, -5,  5,
 5, 10, 10,-20,-20, 10, 10,  5,
 0,  0,  0,  0,  0,  0,  0,  0
```

Figura 2 - Evaluation table relativa a cavalos

```
// knight
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-30,-30,-30,-30,-40,-50,
```

Figura 3 - Evaluation table relativa a bispos

```
// bishop
-20,-10,-10,-10,-10,-10,-10,-20,
-10,  0,  0,  0,  0,  0,  0,-10,
-10,  0,  5, 10, 10,  5,  0,-10,
-10,  5,  5, 10, 10,  5,  5,-10,
-10,  0, 10, 10, 10, 10,  0,-10,
-10, 10, 10, 10, 10, 10, 10,-10,
-10,  5,  0,  0,  0,  0,  5,-10,
-20,-10,-10,-10,-10,-10,-10,-20,
```

Figura 4 - Evaluation table relativa a torres

```
rook
  0,  0,  0,  0,  0,  0,  0,  0,
  5, 10, 10, 10, 10, 10, 10,  5,
 -5,  0,  0,  0,  0,  0,  0, -5,
 -5,  0,  0,  0,  0,  0,  0, -5,
 -5,  0,  0,  0,  0,  0,  0, -5,
 -5,  0,  0,  0,  0,  0,  0, -5,
 -5,  0,  0,  0,  0,  0,  0, -5,
  0,  0,  0,  5,  5,  0,  0,  0
```

Figura 5 - Evaluation table relativa a damas

```
//queen
-20,-10,-10, -5, -5,-10,-10,-20,
-10,  0,  0,  0,  0,  0,  0,-10,
-10,  0,  5,  5,  5,  5,  0,-10,
 -5,  0,  5,  5,  5,  5,  0, -5,
  0,  0,  5,  5,  5,  5,  0, -5,
-10,  5,  5,  5,  5,  5,  0,-10,
-10,  0,  5,  0,  0,  0,  0,-10,
-20,-10,-10, -5, -5,-10,-10,-20
```

Figura 6 - Evaluation table relativa a reis (meio de jogo)

```
king middle game
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-20,-30,-30,-40,-40,-30,-30,-20,
-10,-20,-20,-20,-20,-20,-20,-10,
 20, 20,  0,  0,  0,  0, 20, 20,
 20, 30, 10,  0,  0, 10, 30, 20
```

Figura 7- Evaluation table relativa a reis (fim de jogo[1])

```
// king end game
-50,-40,-30,-20,-20,-30,-40,-50,
-30,-20,-10,  0,  0,-10,-20,-30,
-30,-10, 20, 30, 30, 20,-10,-30,
-30,-10, 30, 40, 40, 30,-10,-30,
-30,-10, 30, 40, 40, 30,-10,-30,
-30,-10, 20, 30, 30, 20,-10,-30,
-30,-30,  0,  0,  0,  0,-30,-30,
-50,-30,-30,-30,-30,-30,-30,-50
```

Fonte: Chess Programming (2024)

[1] Entende-se “fim de jogo” quando (Chess Programming, 2024):

1. Ambos os lados não possuem mais damas, ou;
2. Ambos os lados possuem apenas a dama e uma peça menor (bispo ou cavalo).

2.2. Algoritmo MiniMax

O algoritmo Minimax é uma técnica fundamental usada para decisão em jogos de soma zero como o xadrez, sendo assim uma escolha lógica para a resolução do problema. Ele explora o espaço de jogo para prever os movimentos futuros dos adversários, maximizando o ganho mínimo (obtendo o melhor resultado, mesmo no pior cenário). A operação básica do Minimax envolve a construção de uma árvore de decisão onde cada nó representa um estado do jogo. O algoritmo alterna entre tentar maximizar a pontuação do jogador e minimizar a pontuação do oponente (DIDERICH e GENGLER, 1995).

- **Maximizing Player**: O jogador que busca maximizar seu benefício.
- **Minimizing Player**: O adversário que busca minimizar o benefício do Maximizing Player.

O Minimax considera todos os movimentos possíveis até uma certa profundidade, avaliando o valor de cada estado de acordo com uma função de avaliação. A decisão final é baseada no melhor movimento encontrado através dessa análise .

2.3. Poda Alfa-Beta

Um dos desafios do algoritmo Minimax é que o número de nós na árvore de decisão cresce exponencialmente com a profundidade da busca. Para mitigar esse custo elevado, podemos notar que nem todos os caminhos gerados pelo Minimax são realmente viáveis. Portanto, algumas subárvores do Minimax, que surgem de decisões que não podem ocorrer segundo a lógica do jogo, podem ser completamente descartadas.

A poda alfa-beta é uma otimização do algoritmo Minimax que reduz o número de nós avaliados na árvore de decisão. Ele faz isso através de poda, descartando ramos que não influenciarão a decisão final. Alpha-Beta utiliza dois valores, segundo Russel e Norvig (2004):

- **Alpha**: O valor mais alto encontrado até o momento ao longo do caminho do maximizador.
- **Beta**: O valor mais baixo encontrado até o momento ao longo do caminho do minimizador.

Durante a pesquisa da árvore de decisão, se Alpha for maior ou igual a Beta, o ramo é podado, pois não pode afetar o resultado final. Essa técnica reduz significativamente o número de cálculos necessários, permitindo que a IA explore maior profundidade na árvore de decisão em menos tempo.

3. METODOLOGIA

3.1. Algoritmo de decisão

O algoritmo de tomada de decisão por jogada possui as seguintes etapas, a fim de encontrar o melhor movimento possível:

- Tabelas de Avaliação

O início do código define as **tabelas de avaliação (*evaluation tables*)** para as peças brancas e pretas. Tais tabelas foram demonstradas previamente (Figura 1 a 7) e simplesmente foram implementadas no código em formato de matrizes. As tabelas das peças pretas são obtidas invertendo as das peças brancas usando a função `reverse_array`, a qual muda a ordem das linhas da matriz.

Essas tabelas ajudam a guiar o motor a avaliar posições mais profundamente, considerando fatores além do valor intrínseco da peça (como mobilidade e controle de áreas estratégicas).

- Função `evaluate_piece`

Esta função avalia o valor de uma peça específica, considerando seu tipo, cor e posição no tabuleiro. Ela usa uma função auxiliar, `get_absolute_value`, que soma o valor base da peça (ex.: 10 para peões, 900 para o rei) ao valor atribuído pela tabela de avaliação para aquela posição.

Para garantir que a avaliação seja consistente, o valor é positivo para peças brancas e negativo para peças pretas. Isso permite calcular a vantagem ou desvantagem relativa no tabuleiro. Por exemplo, um cavalo branco centralizado em `d4` terá uma pontuação positiva, enquanto um cavalo preto na mesma posição terá o mesmo valor, mas negativo.

- Função `evaluate_board`

A função `evaluate_board` calcula a pontuação total do tabuleiro, somando os valores de todas as peças em suas posições atuais. Ela itera sobre cada posição no tabuleiro, acessando as peças e chamando `evaluate_piece` para determinar o valor de cada uma. Desta forma, fornece uma avaliação numérica do tabuleiro em seu estado atual, o que ajuda o algoritmo a determinar qual movimento é mais vantajoso.

- Função `minimax_root`

Essa é a função principal que o motor utiliza para decidir o melhor movimento a ser feito. Ela inicia o processo de busca, considerando todas as jogadas legais no estado atual do tabuleiro. Para cada movimento possível, o tabuleiro é atualizado, e a função recursiva `minimax` é chamada para calcular o valor associado ao movimento. Depois de calcular, o movimento é revertido para restaurar o estado original. O movimento com o maior valor associado é armazenado em `best_move_found`, que é retornado no final como o melhor movimento.

- Função `minimax`

A `minimax` é uma função recursiva que implementa o algoritmo de mesmo nome, com a poda alfa-beta para otimização. Essa função é responsável por explorar as possíveis sequências de jogadas até a profundidade especificada, alternando entre os pontos de vista do jogador maximizador (tentando obter o maior valor possível) e do minimizador (tentando reduzir os ganhos do oponente).

Se a profundidade for 0, ou seja, o limite de exploração for atingido, a avaliação do tabuleiro atual é retornada como o valor da posição. O maximizador e minimizador funcionam da seguinte forma:

- **Maximizador:** Para o jogador que busca a maior pontuação, a função inicializa uma variável `best_move` com um valor muito baixo (-9999) e tenta maximizar seu valor ao iterar sobre os movimentos possíveis. Durante esse processo, a variável `alpha` é atualizada para refletir o melhor valor encontrado até então, e se `alpha` ultrapassar `beta`, a busca é interrompida para evitar explorar caminhos desnecessários.
- **Minimizador:** Similar ao maximizador, mas a função inicializa `best_move` com um valor alto (9999) e tenta minimizar seu valor. A variável `beta` é atualizada, e se `beta` for menor que `alpha`, a busca é interrompida.

Essa alternância entre maximização e minimização simula os dois jogadores do xadrez e busca a melhor estratégia para ambos, resultando em um movimento equilibrado e estratégico.

3.2. Interface gráfica e interação com humanos

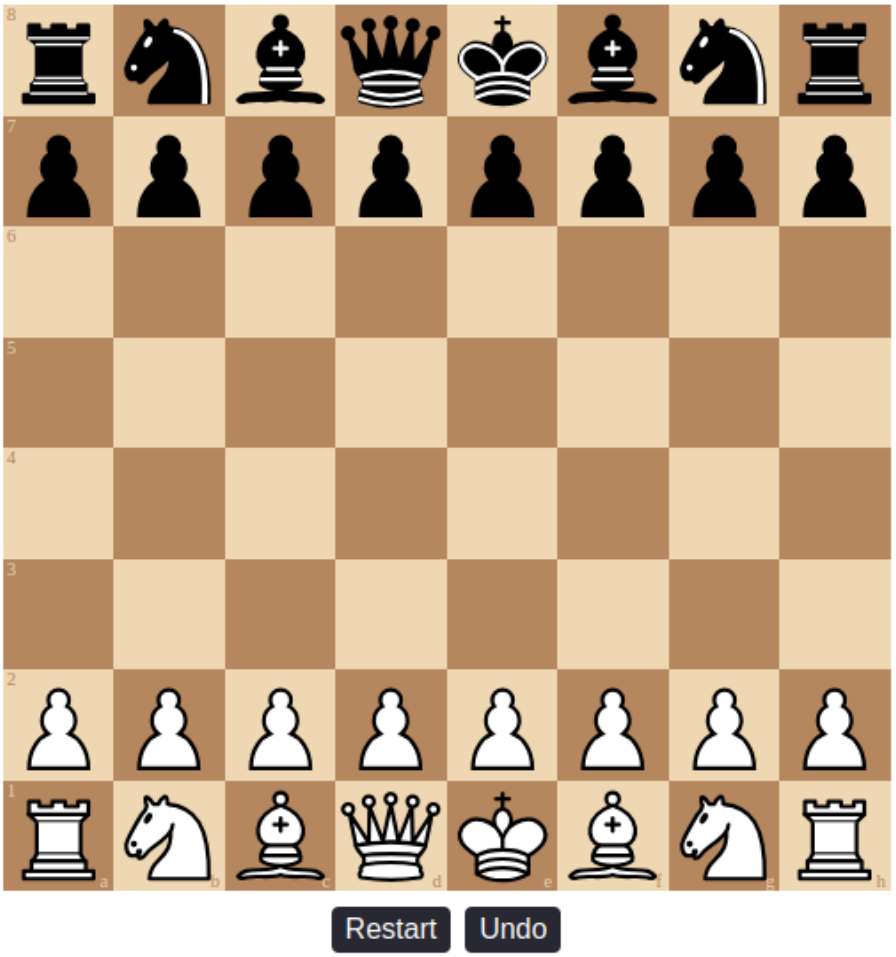
Para que humanos possam interagir com a IA, foi desenvolvida uma interface gráfica utilizando tecnologias web, com React no front-end e Flask no back-end. O back-end, que se comunica diretamente com a IA, receberá as requisições HTTP do front-end com o estado atual do jogo.

Após o processamento dos dados pela IA, a jogada resultante será enviada de volta para o front-end, que a reproduz na interface gráfica, proporcionando um feedback ao usuário. A Figura 3 demonstra o fluxo de informações dentro da aplicação e a Figura 4 apresenta uma imagem da interface gráfica desenvolvida utilizando como apoio bibliotecas para abstração de coisas que não são o foco do projeto, como estilização e definição das regras do xadrez.

Figura 3- Fluxo de informações

Fonte: Desenvolvido pelo autor

Figura 4 - Interface gráfica web para interação com humanos



Fonte: Desenvolvido pelo autor

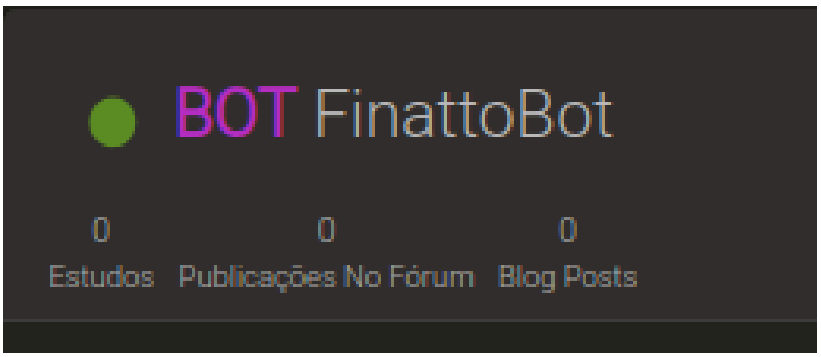
3.3. Lichess BOT

Para a realização dos testes da IA contra outra IA (no caso, o Stockfish 14 disponibilizado pela API do site lichess.com) foi utilizada a Lichess API (Lichess, 2024) com apoio de uma *bridge* desenvolvida para facilitar a comunicação entre Lichess BOTs e chess engines.

Utilizando tais facilitadores, foi realizada a criação de uma *Lichess BOT Account,* uma conta especial dentro do Lichess que permite usar engines para jogar nativamente e de forma ética (já que o uso de engines sem que sejam devidamente identificadas correm risco de banimento por ser considerado um método de trapaça).

Com a *Lichess BOT Account* criada, a engine atribuída ao bot foi o mesmo código minimax utilizado no ambiente Python utilizado no servidor back-end citado no tópico anterior. Com isso configurado, o bot pôde ser executado com êxito e conseguiu jogar partidas contra a engine nativa do Lichess, tendo possibilidade também de jogar contra humanos no próprio site (outra alternativa para interação com humanos). A Figura 5 demonstra uma imagem do perfil do Lichess BOT criada no site do Lichess (a qual foi nomeada como “FinattoBot”), enquanto a figura 14 demonstra uma partida sendo jogada contra uma Engine Stockfish 14 *Level* 3.

Figura 5 - Perfil público do “FinattoBot”



Fonte: Lichess.com

Figura 6 - Partida clássica entre FinattoBot (brancas) e Engine Stockfish 14 *Level* 3

Fonte: Lichess.com

4. TESTES E RESULTADOS

A partir das implementações descritas, foram realizados testes para avaliar o desempenho da inteligência artificial em diferentes cenários. Os testes focaram na taxa de vitórias contra outras engines de xadrez e na precisão das avaliações posicionais.

O “FinattoBot” foi desafiado contra vários níveis de dificuldade do Stockfish para avaliar sua capacidade de manter um desempenho relativamente consistente. Esse processo revelou pontos de melhoria no algoritmo Minimax e na função de avaliação, especialmente em posições complexas.

Durante as partidas, foi possível observar que o “FinattoBot” consegue tomar decisões estratégicas sólidas em posições complexas, indicando que a configuração atual da IA lida bem com uma variedade de situações de jogo, principalmente em fases intermediárias e finais. A análise qualitativa das partidas demonstrou que o bot consegue identificar ameaças e oportunidades com precisão, favorecendo-o em cenários que exigem avaliação posicional e trocas de peças de valor semelhante.

Entretanto, foram identificadas limitações em posições de abertura, onde o Stockfish obtinha vantagem. Isso ocorre pela dificuldade em aberturas inerentes às engines de xadrez, mesmo as mais avançadas.

Para mitigar tal problema, foi utilizado como suporte um livro digital de aberturas chamado *Titans* (GMCHEEMS-ORG, 2024). Antes de cada jogada, a IA consulta o livro para verificar se o movimento de abertura está presente; caso esteja, o movimento é aplicado. Se não estiver, a função minimax é acionada normalmente. Isso garante que a IA siga princípios de abertura nos primeiros lances da partida e, assim que já estiver em posições não-mapeadas, opere livremente. Esse processo é ilustrado na figura 8, que apresenta o trecho correspondente do código.

Figura 8 - Implementação do livro de aberturas “Titans.bin” à decisão

```
def find_best_move(board):
    should_use_book = True

    if not should_use_book:
        best_move = minimax_root(DEPTH, not board.turn, board)
        return best_move

    try:
        with chess.polyglot.open_reader("../Titans.bin") as reader:
            try:
                book_move = reader.find(board)
                if book_move:
                    print(f"Move from book: {book_move.move}")
                    return (book_move.move)
                else:
                    print("No book move found, falling back to Minimax.")
            except IndexError:
                print("No book entry found for this position.")
                should_use_book = False
                best_move = minimax_root(DEPTH, not board.turn, board)
                return best_move

    except FileNotFoundError as e:
        print(f"File not found: {e}")
    except Exception as e:
        print(f"Unexpected error loading opening book: {e}")
```

Fonte: Desenvolvido pelo autor

Esses resultados demonstram o potencial do “FinattoBot” para competir em níveis superiores, indicando que, com ajustes na avaliação de posições iniciais, a IA pode alcançar maior eficácia ao enfrentar adversários com técnicas mais avançadas.

Os testes mostraram que o “FinattoBot” foi capaz de vencer o Stockfish 3 do Lichess em partidas configuradas com uma profundidade de busca de 4 níveis. Esse feito é notável, considerando o nível intermediário do Stockfish 3, e confirma a eficácia da função de avaliação.

A figura 7 expõe o gráfico de uma das partidas contra tal adversário (o algoritmo desenvolvido jogando com as peças pretas e o adversário de brancas), demonstrando pontos altos e baixos da força do algoritmo.

Figura 7 - Gráfico de análise de uma das vitórias contra o stockfish 3

Fonte: Lichess.org (https://lichess.org/ynjYtLPD/black)

5. CONCLUSÃO

Com base nos resultados apresentados, podemos concluir que o algoritmo desenvolvido para o jogo de xadrez possui um desempenho sólido em partidas configuradas com profundidade de busca intermediária. Ele demonstrou capacidade de competir em condições desafiadoras, obtendo vitórias consistentes contra o

Stockfish 3 e mostrando-se eficaz na avaliação posicional e na tomada de decisões estratégicas, especialmente em fases finais do jogo.

A implementação da poda alfa-beta foi essencial para otimizar o processo de decisão, permitindo que o algoritmo explorasse maior profundidade de jogadas sem comprometer a eficiência. A função de avaliação, baseada em uma análise detalhada das peças e suas posições, provou ser adequada para a maioria das situações, embora algumas limitações tenham sido identificadas nas posições de meio jogo. Nessa fase, o algoritmo apresentou dificuldades em acompanhar engines com maior capacidade de refinamento estratégico, especialmente em movimentos de desenvolvimento e controle central.

Esses resultados indicam que o aprimoramento da função de avaliação para contextos específicos dessa etapa poderia elevar ainda mais o nível competitivo da IA, permitindo que o algoritmo enfrente oponentes mais avançados e se adapte melhor a diversas fases do jogo.

Por fim, futuras melhorias podem ser realizadas na mobilidade das peças, estrutura de peões e nas atividades das peças maiores, da seguinte forma: na mobilidade, podem ser verificadas quantas opções de movimento estão disponíveis para cada lado. Isso mede a atividade das peças, um fator importante em posições de meio-jogo. Na estrutura de peões, pode-se avaliar e evitar fraquezas como peões dobrados, e fomentar a criação de peões passados e quanto às atividades das peças maiores, se forem consideradas as torres e damas priorizadas em colunas abertas, tais peças podem ser muito mais eficazes nessas posições.

REFERÊNCIAS

Chess.com, 2018. **Kasparov vs. Deep Blue | The Match That Changed History.** Disponível em: <https://www.chess.com/article/view/deep-blue-kasparov-chess>. Acesso em: 25 mar. 2024.

Chess.com, 2018. **Elo Rating System.** Disponível em: <https://www.chess.com/terms/elo-rating-chess>. Acesso em: 11 abr. 2024.

CHESS PROGRAMMING WIKI. *Simplified Evaluation Function*. Disponível em: <https://www.chessprogramming.org/Simplified_Evaluation_Function>. Acesso em: 04 out. 2024.

Chess.com, 2018. **How to evaluate a position?.** Disponível em: <https://www.chess.com/pt-BR/terms/valor-pecas-xadrez>. Acesso em: 06 jun. 2024.

DIDERICH, Claude G.; GENGLER, Marc. ***A SURVEY ON MINIMAX TREES AND ASSOCIATED ALGORITHM***. Lausanne, Switzerland, 1995. Cap. 2. Disponível em:< https://link.springer.com/chapter/10.1007/978-1-4613-3557-3_2. Acesso em: 06 jun. 2024.

GMCHEEMS-ORG. *Titans.bin*. Disponível em: https://github.com/gmcheems-org/free-opening-books/blob/main/books/bin/Titans.bin. Acesso em: 21 nov. 2024.

LICHESS, 2024. **Lichess.org API reference (2.0.0).** Disponível em: <https://lichess.org/api>. Acesso em: 30 ago. 2024.

LICHESS BOT DEVELOPERS. **lichess-bot**. GitHub, n.d. Disponível em: <https://github.com/lichess-bot-devs/lichess-bot>. Acesso em: 4 out. 2024.

RUSSELL, S.; NORVIG, P **.Inteligência Artificial**. New Jersey, EUA: Pearson, 2004. 161–195 p. (Prentice Hall Series in Artificial Intelligence). ISBN 9780136042594.