Computadores no Xadrez

Um Estudo e Implementação

Gustavo Figueiredo Serra

Orientador: Prof. Kostiantyn Iusenko

Trabalho de conclusão de curso do

Bacharelado em Matemática Aplicada e Computacional

Instituto de Matemática e Estatística

Universidade de São Paulo

2023

"Comparing the capacity of computers to the capacity of the human brain, I've often wondered, where does our success come from? The answer is synthesis, the ability to combine creativity and calculation, art and science, into a whole that is much greater than the sum of its parts."

— *Garry Kasparov*

Agradecimentos

Se formar no Instituto de Matemática e Estatística da Universidade de São Paulo não é tarefa fácil, e ouso dizer que impossível sem apoio algum. Se estou aqui, foi por todos vocês que fizeram parte dessa jornada.

Primeiramente, a meus pais, por me proporcionarem absolutamente todo o necessário para eu sequer conseguir pisar nesse instituto, por sempre me incentivarem a cursar uma graduação e a vivenciar a faculdade por completo e, ao fim, por me lembrarem constantemente de me formar: palavras não são capazes de expressar minha gratidão. A vocês devo tudo.

A meus companheiros de graduação, pelo apoio tanto acadêmico quanto fora: obrigado por ajudarem a fazer desses os melhores anos da minha vida (até agora). Tutu, Dobby, Aurea, Nadia, Feliz, Lorenzo, Curitiba, Isa, Ma, Bia, Caio, Pedro, Fada, Dielle, Gui, Solove e tantos outros que cruzei nesse longo caminho, de aulas a provas, de bandejões a carteados, de reuniões a festas, agradeço por tudo, pois foi parte integral da minha formação.

A meu orientador, professor Iusenko, por topar me orientar nessa aventura sobre um jogo que tanto amamos, e pela paciência e conselhos por todo o caminho.

À mais nova adição, porém não menos importante para a conclusão deste trabalho, obrigado Jaque, pelo incentivo em tempo integral, por acreditar em mim, aguentar os sumiços e surtos e por revisar tudo comigo. Obrigado por tornar essa jornada mais leve.

Resumo

Gustavo Figueiredo Serra. **Computadores no Xadrez - um Estudo e Implementação.** Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

O xadrez é um dos mais icônicos jogos de tabuleiro da cultura ocidental, com milhões de jogadores ao redor do mundo. Sua história remonta de séculos atrás, e o jogo já passou por inúmeras evoluções, tanto em suas regras, como na forma de ser jogado. O xadrez sempre foi visto como um jogo altamente abstrato e até artístico, por exigir uma alta capacidade de visualização e estratégia. Mesmo assim, hoje os computadores estão muito acima da capacidade de qualquer pessoa de jogar xadrez, outrora considerado um jogo tão humano. Esta monografia se propõe a estudar a história desse embate e os métodos que levaram programas de xadrez a superarem humanos. Por fim, ao final é apresentado um programa de implementação autoral, utilizando conceitos explorados nos primeiros capítulos.

Palavras-chave: Xadrez. Jogos. Engine. Inteligência Artificial. Aprendizado de Máquina. Minimax. Alpha-Beta Pruning.

Índice

1 Introdução

O xadrez é um jogo milenar que tem sido objeto de estudo e fascínio de muitas pessoas ao longo dos séculos. Com o advento da computação, o xadrez tornou-se uma área fértil para o desenvolvimento de algoritmos e programas que podem jogá-lo com maestria. Esses programas utilizam uma variedade de técnicas para avaliar as posições no tabuleiro e determinar a melhor jogada a ser realizada.

Este trabalho tem como propósito apresentar uma visão geral das engines de xadrez, incluindo a história e a evolução das heurísticas utilizadas, bem como uma descrição detalhada do funcionamento de diferentes tipos de engines. Além disso, uma engine de xadrez foi desenvolvida não só a partir do estudo de princípios de xadrez, mas também dos pressupostos para a construção de engines - investigando prioridades, pesos e os resultados obtidos. Assim, esta dissertação busca proporcionar uma compreensão mais profunda das engines de xadrez, desde suas técnicas até sua implementação.

Para alcançar esses objetivos, este trabalho está organizado em quatro seções principais além desta, introdutória. Na segunda seção, uma breve introdução à história do xadrez e como a computação se tornou parte integrante do jogo, seguida pela história do Uso de Computadores no Xadrez. Na quarta, serão descritas as técnicas utilizadas pelas engines de xadrez para avaliar as posições no tabuleiro, incluindo análise material, posicional e também uma visão geral dos algoritmos de busca utilizados pelas engines de xadrez. Já na quinta e última seção, será apresentado uma engine de xadrez desenvolvida do zero a fim de explorar a complexidade do jogo e a capacidade computacional de realizar tarefas abstratas e criativas.

2 Uma Breve História do Xadrez

O xadrez é um dos jogos mais populares e antigos do mundo, jogado por milhões de pessoas ao redor do globo. O jogo é conhecido por sua complexidade e estratégia, desafiando jogadores

a pensar de forma criativa e a planejar com antecedência.

As origens do xadrez são incertas, mas acredita-se que o jogo tenha se originado na Índia, por volta do século VI d.C. Originalmente chamado de chaturanga, o jogo tinha peças que representavam diferentes unidades militares indianas, como elefantes, cavalos e guerreiros [1].

Figura 1: *Peças de chaturanga, datadas do século XVIII.*

O chaturanga se espalhou rapidamente para outras partes do mundo, incluindo a Pérsia e a China. Na Pérsia, o jogo foi renomeado para shatranj e passou por várias mudanças, incluindo a adição da peça do bispo e a mudança no movimento do cavalo.

O xadrez moderno, como conhecemos hoje, tomou forma na Europa durante a Idade Média, com a adição de novas peças, como a rainha e o peão, e mudanças nas regras do jogo para algo mais próximo do que conhecemos hoje, com a adição de movimentos como o roque e o *en passant*.

O xadrez tornou-se popular em toda a Europa durante o século XVIII, com o surgimento de torneios e competições. O primeiro campeonato mundial de xadrez foi realizado em 1886, com Wilhelm Steinitz, da Áustria, sendo coroado como o primeiro campeão mundial de xadrez. Desde então, o jogo cresceu muito, e se tornou uma atividade reconhecida internacionalmente, com a Federação Internacional de Xadrez (FIDE) sendo fundada em 1924 para organizar e regulamentá-lo em todo o mundo.

Hoje em dia, o xadrez é amplamente reconhecido como um jogo que estimula a criatividade, estratégia e planejamento, sendo ensinado em diversas escolas para crianças em todo o mundo. Nos últimos anos, o jogo vem ganhando muita popularidade, com a habilidade de ser jogado online durante a pandemia e a aclamada série da Netflix "O Gambito da Rainha" trazendo muitos novos entusiastas para o esporte. O xadrez é um jogo que une pessoas de diferentes origens e culturas, sendo um esporte que não enxerga diferenças físicas ou distâncias.

Figura 2: *Cena da série "The Queens Gambit"(2020)*

3 Uso de Computadores no Xadrez

3.1 Vitória das Máquinas

O uso de computadores no xadrez começou na década de 1950, quando o matemático e pioneiro da computação Claude Shannon criou o primeiro programa de xadrez em um computador. No entanto, a capacidade dos computadores da época era limitada e o programa não era capaz de jogar com grande habilidade. Na época, os programas eram vistos como meras ferramentas para auxílio de cálculo em algumas posições e finais de jogo, mas nunca do mesmo nível de jogadores profissionais de xadrez.

Ao longo das décadas seguintes, o poder de processamento dos computadores aumentou significativamente, permitindo o desenvolvimento de programas cada vez mais avançados. Ao longo da década de 60, programas começaram a participar em torneios de xadrez, vencendo em algumas ocasiões contra jogadores federados. Em 1970, foi criado o primeiro torneio de xadrez somente para computadores e finalmente, em 1981, o programa Cray Blitz venceu Joe Sentef (rating 2262) e se tornou o primeiro a derrotar um mestre em um torneio e a ganhar um rating acima de 2200 (2258). [2]

Nos anos seguintes, os programas de xadrez continuaram a evoluir rapidamente e em 1997, o programa Deep Blue [3], desenvolvido pela IBM, venceu o Campeão Mundial de xadrez Garry Kasparov em uma partida de seis jogos. Essa vitória foi um grande marco na história do xadrez, sendo o momento em que um programa se consagrou como o melhor jogador do mundo, e atestou a capacidade dos computadores superar não apenas Grandes Mestres como o Campeão Mundial de xadrez Garry Kasparov, tido por muitos até hoje como o maior da história. A partir desse momento, as engines só continuaram se distanciando dos jogadores humanos, ao ponto de serem simplesmente imbatíveis. Sequer empatar contra algum dos melhores programas hoje é uma tarefa praticamente impossível.

Figura 3: *Campeão Mundial Garry Kasparov em seu match contra Deep Blue, 1997*

3.2 Para Além da Força Bruta

Outrora tido como a epítome do pensamento criativo e racional humano, o xadrez foi rapidamente vencido pelas máquinas, devido ao desenvolvimento acelerado dos computadores e seu poder de processamento, que tornou possível o desenvolvimento de programas que calculam milhões de posições e a profundidades muito maiores que a capacidade humana. Ainda assim, muitos argumentam que o jogo de xadrez é mais do que apenas a habilidade de calcular jogadas com precisão.

Uma das principais limitações das máquinas é a capacidade dos computadores em avaliar posições muito complexas no tabuleiro. Embora os programas de xadrez possam calcular jogadas com grande precisão, ainda há muitas situações no jogo que são muito difíceis de avaliar e exigem um alto nível de intuição e criatividade. Além disso, os programas de xadrez podem ter um estilo de jogo sólido e robusto, o que torna o jogo menos interessante para as

pessoas, que podem preferir jogar ou até assistir jogos contra humanos, pois têm a capacidade de avaliar a posição no tabuleiro de forma mais abstrata, utilizando a intuição, a criatividade e a capacidade de identificar padrões e ideias posicionais nas jogadas do oponente. Tudo isso contribuía para uma visão das *engines* como tendo um estilo de jogo "robótic", sem margem para criatividade ou ideias fora da caixa, e apenas capazes de vencer humanos por puro poder computacional.

Essa visão vem mudando recentemente, principalmente com os recentes avanços nas áreas de inteligência artificial e aprendizado de máquina (*machine learning*). Em 2016, o programa AlphaGo [4], desenvolvido pela empresa de tecnologia DeepMind, venceu o Campeão Mundial de Go, um jogo que tem um nível de abstração muito acima do xadrez, com um número de jogadas possíveis exponencialmente maior. Por conta disso, Go era considerado um jogo que nunca seria resolvido por máquinas, já que o poder bruto de computação não é de tanta ajuda, e de fato, nunca antes uma *engine* de Go tinha vencido um jogador profissional. O que é ainda mais surpreendente, é que AlphaGo não tinha nenhum conhecimento além das regras do jogo. Todo o seu conhecimento foi adquirido apenas jogando contra si mesmo e aprendendo com as próprias vitórias e derrotas.

Logo em seguida, em 2017, a DeepMind atacou o Xadrez, com sua IA (Inteligência Artificial) AlphaZero [5]. AlphaZero, após 9 horas de treinamento, jogou 100 partidas contra a melhor *engine* de xadrez da época, o StockFish 8, e simplesmente destruiu o StockFish: a IA não perdeu um único jogo, com o impresssionante resultado de 28 vitórias, 0 derrotas e 72 empates. Para o xadrez, que em alto nível tem uma alta taxa de empates, isso é uma vitória avassaladora. E não só AlphaZero demonstrou sua superioridade, como também, assim como AlphaGo, tinha um estilo de jogo muito diferente, com diversos sacrifícios materiais, ideias posicionais e criativas nunca antes vistas de uma máquina. Em nenhum momento AlphaZero foi ensinado sobre valor de peças ou qualquer outro conceito de xadrez que comumente são utilizados em *engines*, tendo apenas jogos contra si mesmo para aprender.

Atualmente, o StockFish 13 [6] também usa técnicas de Inteligência Artificial e Redes Neurais e é amplamente reconhecido como a melhor *engine* do mundo, mas AlphaZero e AlphaGo realmente chacoalharam os nossos conceitos prévios de criatividade e estratégia, e a própria capacidade das máquinas de "pensa" como humanos. A habilidade de Inteligências Artificiais nesses jogos fomenta uma discussão muito mais profunda e nos faz questionar qual o limite da capacidade de abstração de um computador. A que ponto chegaremos? No limite, poderiam ser desenvolvidas IAs semelhantes a humanos, tal qual ficções científicas confabulam há décadas? Poderiam as máquinas nos substituir e superar nas mais complexas e "humana" tarefas?

3.3 O Discípulo se torna o Mestre

Engines de xadrez hoje desempenham um papel fundamental na evolução do jogo. Estamos a um nível de conhecimento como nunca antes visto, com os Grandes Mestres da atualidade jogando em altíssimo nível, muito superior às gerações passadas. Isso foi especialmente potencializado nas últimas décadas, com o avanço de máquinas para patamares cada vez mais altos de jogo, nos ensinando o que há de mais avançado em teorias de abertura, finais de jogo e ideias posicionais como um todo.

Com participação principalmente nas partes iniciais do jogo, muitos vinham considerando que o cenário profissional do xadrez estava sendo arruinado pelas máquinas, e de fato vimos um período de jogos mais travados, um aumento nos empates e memorizações avançadas, o que frequentemente tira toda a beleza do jogo.

Nos últimos anos, no entanto, com o avanço das *engines* para níveis ainda maiores, estamos vivendo uma era pós-moderna do jogo, e as máquinas vêm sendo usadas para ajudar a quebrar conceitos antes fortemente estabelecidos e possibilitar inovações em todos os estágios do jogo. Como consequência, hoje vemos algumas das mais impressionantes performances de xadrez da história e uma competição de altíssimo nível, com Grandes Mestres crescendo e se

desenvolvendo ao lado de máquinas. As *engines* de xadrez vieram a se tornar uma das mais importantes ferramentas para o desenvolvimento do jogo, e ainda temos muito a aprender de suas inteligências artificiais.

4 O Funcionamento de uma Engine de Xadrez

Atualmente, conhecemos duas abordagens diferentes para construção de uma *engine* de xadrez: a tradicional, em que são programados conceitos do jogo para avaliação de posições, e uma mais moderna, utilizada por AlphaZero, onde uma rede neural é treinada para avaliar a posição. Entretanto, a última requer altíssimo poder computacional para treino e sua implementação não aborda as nuances do jogo de xadrez, portanto, esta monografia cobrirá as metodologias mais tradicionais, envolvendo conhecimento de xadrez para avaliação das posições.

Existe uma grande variedade de heurísticas para avaliação de posições e mecanismos de busca para otimizar o tempo de processamento. Nesta seção serão apresentadas algumas das mais comuns, a maioria das quais foram utilizadas na criação do programa do presente trabalho.

4.1 Avaliação de Posição

A avaliação de posição é uma das tarefas mais importantes das *engines* de xadrez, pois é responsável por determinar a melhor jogada em uma determinada posição. Essa avaliação é constituída por uma série de heurísticas para determinar o quão vantajosa é uma posição, possibilitando assim a escolha de qual lance realizar. Ela pode levar em conta não apenas fatores como a diferença de material mas também controle de espaço, a mobilidade das peças, a estrutura de peões e a qualidade das casas ocupadas pelas peças.

4.1.1 Análise Material

A análise material é a heurística mais simples utilizada pelas engines de xadrez. Ela consiste em avaliar a diferença de material entre as duas cores no tabuleiro, levando em conta o valor atribuído a cada peça. Por exemplo, tradicionalmente um peão tem um valor de 1 ponto, um cavalo tem um valor de 3 pontos e assim por diante, contando o número de pontos de diferença entre os dois lados.

A avaliação material normalmente é a parte que mais altera a avaliação geral da posição, pois peças são muito importantes no xadrez e posições com uma ou duas peças importantes de diferença raramente estão a favor do lado com menos material. É essa heurística que ensinamos também a jogadores humanos de xadrez como um bom ponto de partida para se analisar uma posição, e o sistema de contagem de pontos é amplamente reconhecido entre amadores e profissionais.

No entanto, apesar de ser uma das mais importantes, a análise material por si só não é suficiente para avaliar a posição com precisão. É possível que uma posição em que uma cor tenha menos material seja mais vantajosa devido à mobilidade das peças, à estrutura de peões ou a outras características da posição.

Ao longo dos anos, diversas pontuações diferentes foram teoretizadas pelos mais famosos enxadristas. A mais simples e popular classificação hoje data de 1942, quando Reuben Fine publicou em seu livro Chess The Easy Way [7] as seguintes pontuações de peças:

Essa divisão assume forças similares ao Cavalo e Bispo (chamadas comumente de peças menores) e coloca a Torre acima, porém mais fraca que um par de peças menores. Da mesma forma, uma Dama vale mais que uma torre, porém menos que duas, e aproximadamente 3 peças menores.

Valor das Peças	
Peça	Valor
♙	1
♘	3
♗	3
♖	5
♕	9

Tabela 1: Atribuição simples e popular de valores para as peças

Esse tipo de avaliação direta de peças é simplista e releva relações importantes entre as peças, como o poder do par de bispos (já que podem cobrir as duas cores do tabuleiro) ou do par de torres, que são melhores trabalhando juntas. Além disso, hoje em dia reconhecemos o bispo como sendo ligeiramente superior ao cavalo de uma maneira geral. Por fim, os melhores programas hoje também mudam a avaliação das peças à medida que o jogo progride, pois em estágios diferentes as peças desempenham funções diferentes.

4.1.2 Estrutura de Peões

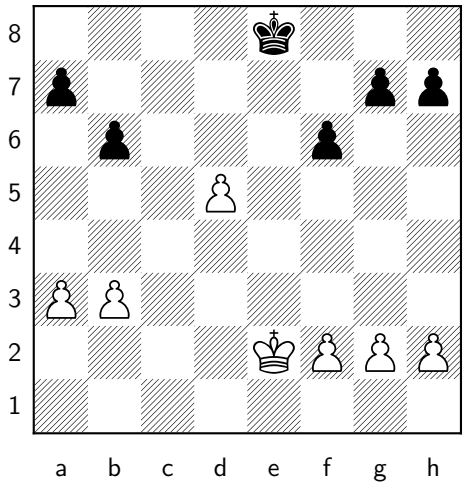
A estrutura de peões é outro fator importante a ser considerado na análise posicional. O peão é uma peça excepcional do xadrez, por ser a única incapaz de se mover para os lados ou para trás - depois de um movimento de peão, é impossível voltar atrás. Além disso, chegando ao final do tabuleiro, o peão se promove para qualquer outra peça, se tornando uma das maiores ameaças no final do jogo. Por causa disso, a forma como os peões são dispostos no tabuleiro pode ter um grande impacto não só na mobilidade das peças, mas também sua ameaça de promoção pode dar enorme vantagem posicional. Ao mesmo tempo que é uma peça fraca por si só, tornando-se alvos fáceis se isolados, uma estrutura de peões sólida pode fornecer uma forte base para as peças.

Na teoria de xadrez, são definidos alguns tipos de peões, de acordo com sua posição relativa ao tabuleiro e a outros peões. São eles:

Peão Isolado

Um peão isolado é um peão que não tem peões aliados em colunas adjacentes. Os peões isolados podem ser uma fraqueza, pois não podem ser protegidos por outros peões e podem ser vulneráveis a ataques inimigos. No entanto, também podem fornecer mobilidade para outras peças e criar oportunidades de avanço.

Na imagem abaixo, o peão das brancas em d5 é um peão isolado:

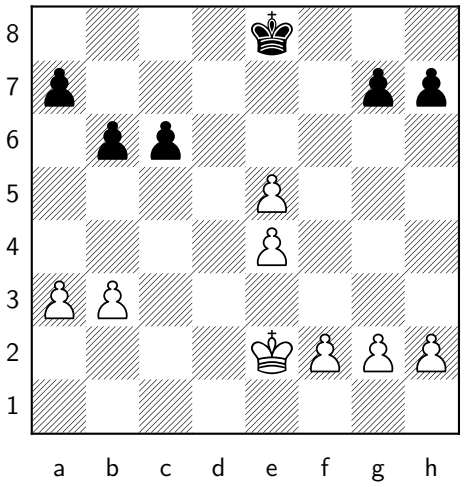


Normalmente, peões isolados recebem uma penalidade.

Peão Dobrado

Um peão dobrado é um peão que tem outro peão aliado na mesma coluna. Os peões dobrados são considerados uma fraqueza, pois têm menos liberdade para atacar e geralmente também estão isolados, ou seja, estão sem defensores e portanto podem ser mais fáceis de atacar.

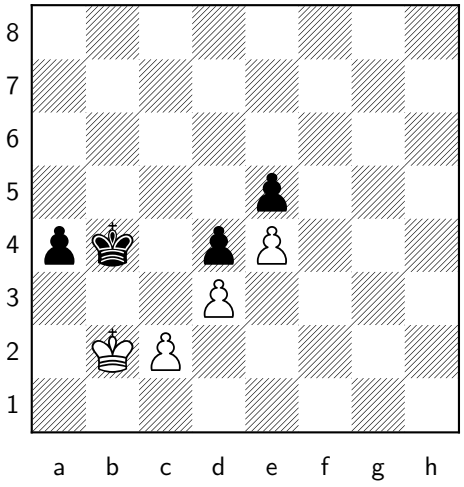
No tabuleiro a seguir, os peões em e4 e e5 são dobrados, pois estão na mesma coluna:



Peões dobrados são comumente atribuídos ao valor de 0,5 peão aproximadamente, podendo variar dependendo do tipo de peão dobrado.

Peão Atrasado

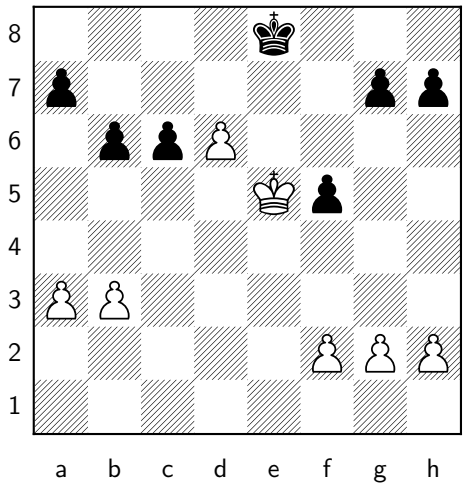
Um peão atrasado é um peão que está atrás dos outros peões em sua própria coluna. Os peões atrasados são penalizados, pois, assim como os peões isolados, não é possível defendê-los com outros peões e geralmente também são impossibilitados de prosseguir por peões inimigos, como o peão das brancas em c2 na figura:



Peão Passado

Um peão passado é um peão que não pode ser bloqueado ou capturado por peões inimigos na mesma coluna ou nas colunas adjacentes. Isso geralmente ocorre quando um peão avança e ultrapassa peões inimigos sem ser capturado, ou captura ele mesmo um peão atrasado inimigo.

Na imagem abaixo, o peão em d6 é um peão passado:



Os peões passados normalmente são considerados uma vantagem, principalmente no final do jogo, pois podem avançar livremente pelo tabuleiro e atacar e peças inimigas, além da constante ameaça de promoção. O valor atribuído costuma aumentar ainda mais dependendo da fileira em que está o peão passado.

Existem ainda outras regras possíveis, diferenciando tipos de peões dobrados, corrente de peões, maioridade e muitas outras heurísticas para se avaliar a estrutura de peões de um jogo. Entretanto, é um cálculo muito custoso, e que exige muita otimização, como por exemplo só atualizar a avaliação de peões quando necessário ou uma matriz de peões e zonas de ataques performática.

4.1.3 Análise de Segurança do Rei

A análise de segurança do rei é uma heurística que leva em conta a posição do rei no tabuleiro e as ameaças que ele pode enfrentar. Uma posição em que o rei de uma cor está exposto a ameaças pode ser considerada desfavorável, mesmo que essa cor possua mais material ou uma posição mais ativa.

Para avaliar a segurança do rei, as engines de xadrez consideram a presença de peças defensoras em torno do rei, a possibilidade de criação de ameaças táticas e o número de atacantes inimigos nas casas adjacentes ao Rei.

4.1.4 Mobilidade das Peças

A avaliação de mobilidade das peças se constitui em um bônus de pontuação cada para lance legal disponível aos jogadores. É consenso geral na comunidade enxadrista que uma maior mobilidade de peças é um ponto positivo no xadrez, pois dessa forma existem mais opções e possibilidades para traçar planos e lidar com as ofensivas inimigas, sem ficar preso pelas outras peças em jogo.

Essa questão foi estudada e analisada estatisticamente no artigo de Elliot Slater "Statistics for the Chess Computer and the Factor of Mobility" [8], onde dados de campeonatos foram utilizados para mostrar uma correlação positiva entre mobilidade e chances de vitória, e existem inclusive engines que são majoritariamente baseadas em métricas de mobilidades. Engines mais avançadas levam também em conta não apenas lances possíveis, mas também número de casas controladas pelo jogador e lances possíveis em casas não controladas pelo inimigo.

4.1.5 Tabelas Piece-Square

As tabelas Piece-Square, também conhecidas como tabelas de avaliação de posição, são uma abordagem heurística para a avaliação de posições no xadrez. Essas tabelas atribuem valores numéricos às posições das peças no tabuleiro, permitindo que um programa de xadrez estime a qualidade de uma posição sem ter que analisar todas as possíveis jogadas futuras. A origem dessa técnica remonta à década de 1970, quando os primeiros programas de xadrez começaram a ser desenvolvidos [9].

Uma das principais utilidades das tabelas Piece-Square é fornecer uma medida simples e eficiente de avaliação de posição para engines de xadrez. Em vez de analisar exaustivamente todas as possibilidades de jogadas futuras, os engines utilizam essas tabelas para orientar seu processo de busca e tomar decisões mais rapidamente. Além disso, as tabelas Piece-Square são fáceis de implementar e requerem pouco espaço de armazenamento.

Normalmente, existem duas Tabelas Piece-Square em uma engine, uma para o começo do jogo e outra para o final. Isso reflete as diferentes estratégias, em que no final do jogo é necessário, por exemplo, ativar o rei e avançar os peões, diferentemente do começo do jogo. Uma estratégia comum para determinar o estágio do jogo é pelo número de peças, e através de uma função as duas tabelas são misturadas, garantindo uma transição contínua e gradual entre elas.

4.2 Algoritmos de busca

4.2.1 O Algoritmo Minimax

O algoritmo Minimax é um dos mais populares algoritmos usados em teoria dos jogos, notavelmente em jogos de tabuleiro clássicos, como o Jogo da Velha, Damas e Xadrez. O algoritmo consiste em encontrar a melhor jogada em um determinado estado do jogo, considerando as melhores respostas possíveis do adversário. Ele foi provado em 1928 por John von Neumann

[10]. e vem sendo estudado e aplicado amplamente na área de teoria dos jogos desde então, sendo parte fundamental dessa área de estudo.

O algoritmo Minimax é baseado na ideia de que o jogador deve tentar minimizar a perda, enquanto o adversário deve tentar maximizar o ganho, do ponto de vista dele. Ao se colocar na posição do adversário e calcular os máximos ganhos dele, o algoritmo busca maximiar os próprios ganhos a partir da jogada ótima adversária.

Tomando como exemplo o jogo da velha, se for possível para o oponente vencer na próxima jogada, o algoritmo considera que essa vai ser a escolha, portanto, evitará se colocar nessa posição na penúltima rodada. Da mesma forma, voltando para o ponto de vista do inimigo, ele tentará forçar uma posicão na antepenúltima rodada que force tal situação de vitória na última, e assim por diante.

Para aplicar esse conceito ao xadrez, como não é possível calcular todas as jogadas até o fim do jogo, utilizamos uma função de avaliação para calcular a melhor jogada, portanto, o algoritmo é limitado pela profundidade de busca e pela eficiência da função de avaliação.

O algoritmo Minimax funciona através da construção de uma árvore de busca, onde cada nó representa um estado do jogo e cada ramificação representa uma jogada possível. Cada nó na árvore é avaliado e um valor numérico é atribuído a esse estado do jogo.

Para entender melhor o funcionamento matemático do algoritmo, primeiro definimos as seguintes notações:

- s é um estado do jogo.
- $P(s)$ é o jogador que tem o turno em um estado s.
- $A(s)$ é o conjunto de ações possíveis em um estado s.

- $S(s, a)$ é o estado resultante de executar a ação a no estado s.
- $E(s)$ é uma função de avaliação que atribui um valor numérico a um estado s.

Com essas notações, o algoritmo Minimax pode ser descrito matematicamente como:

$$\text{maximin}(s) = \begin{cases} E(s) \text{ se terminal}(s) \\ \max_{a \in A(s)} \text{minimax}(S(s, a)) & \text{se } P(s) = \text{maximizador} \\ \min_{a \in A(s)} \text{minimax}(S(s, a)) & \text{se } P(s) = \text{minimizador} \end{cases} \tag{1}$$

onde maximin(s) representa o valor máximo que o jogador maximizador pode obter em um estado s e minimax(s) representa o valor mínimo que o jogador minimizador pode obter em um estado s.

O algoritmo percorre a árvore de busca de forma recursiva, alternando entre os jogadores e fazendo uma avaliação em cada nó da árvore. Quando o algoritmo chega a uma certa profundidade ou ao final do jogo, a função de avaliação é aplicada para determinar o valor numérico do estado.

4.2.2 Poda Alpha-Beta

A poda alpha-beta (Alpha-Beta Pruning em inglês) [12] é uma técnica de busca em árvore utilizada em diversos jogos dentro da teoria dos jogos para reduzir a quantidade de nós a serem avaliados, aumentando assim a eficiência do algoritmo de busca. O algoritmo é uma extensão do Minimax e amplamente utilizado como algoritmo de busca no xadrez, juntamente com uma função de avaliação, como no Minimax.

O algoritmo alpha-beta pruning funciona da seguinte forma:

1. A busca em árvore é realizada usando o algoritmo minimax, que avalia todas as jogadas possíveis até uma profundidade pré-definida.

2. Cada nó da árvore recebe duas valores, α e β, que representam o menor e o maior valor encontrado até o momento em um caminho de busca. Esses valores são inicializados com $-\infty$ e $+\infty$, respectivamente, para o nó raiz da árvore.

3. Quando um nó é avaliado, a sua pontuação é passada para os nós ascendentes.

4. Se a pontuação do nó atual é menor do que o valor mínimo já encontrado em um nó anterior (ou seja, α é maior do que a pontuação), a busca pode ser interrompida, já que o valor atual não pode ser o escolhido. De forma análoga, se a pontuação atual for maior do que o valor máximo já encontrado em um nó anterior (ou seja, β é menor do que a pontuação), a busca também pode ser interrompida.

5. A técnica de poda é aplicada para eliminar ramos inteiros da árvore de busca que não são relevantes para encontrar a melhor jogada, reduzindo assim o tempo de processamento necessário para avaliar todas as possibilidades. Isso é feito porque, se um nó tem um valor fora do intervalo $[\alpha, \beta]$, ele não pode afetar o resultado final da busca, uma vez que já se sabe que existem outras jogadas melhores.

6. O algoritmo continua a busca em outros ramos da árvore até que todos os nós tenham sido avaliados ou até que a busca tenha sido interrompida em um determinado nó (por exemplo, por uma restrição de tempo).

Desde a sua criação, o algoritmo alpha-beta pruning tem sido amplamente utilizado no desenvolvimento de engines de xadrez. Ele é uma das principais técnicas utilizadas para avaliar as jogadas e determinar a melhor estratégia para cada posição no tabuleiro. Para deixar o algoritmo ainda mais rápido, existem algumas outras estratégias que reduzem o tempo de processamento.

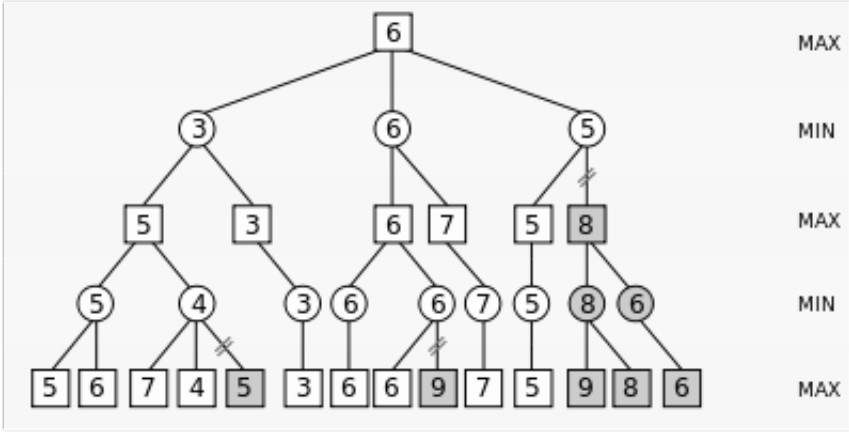


Figura 4: *Funcionamento da poda Alpha-Beta*

4.2.3 Ordenação dos movimentos

Perceba que a eficiência da poda alpha-beta é totalmente dependente do quão cedo podemos identificar posições vantajosas para o jogador maximizador. Portanto, é essencial também implementar uma heurística de ordenação que verifique primeiro lances com uma probabilidade maior de serem os melhores ou lances mais rápidos de serem avaliados.

Algumas possíveis técnicas de ordenação são:

- Verificar posições já guardadas na tabela de transposição.
- Verificar lances da linha principal avaliada em uma busca anterior.
- Verificar xeques e capturas primeiro. Capturas de peças menos valiosas para mais valiosas e xeques de peças de maior valor são avaliados antes. (técnica de Most Valuable Victim - Least Valuable Agressor, MVL-LVA)
- Verificar lances que tiveram um salto de avaliação em alguma análise anterior (chamados *killer moves*.

4.2.4 Tabelas de transposição e Zobrist Hashing

As tabelas de transposição são tabelas utilizadas para guardar posições do tabuleiro que já foram analisadas e suas respectivas avaliações. Essa técnica é uma das mais importantes para reduzir o tempo do algoritmo, e se tornou possível depois de 1970, com a publicação do paper "A New Hashing Method with Application for Game Playin", de Albert L. Zobrist [13]. Um dos grandes problemas da tabela de transposição é a memória ocupada para guardar posições no tabuleiro, pois seria impossível guardar milhões de posições com os métodos até então conhecidos.

Com o "Zobrist Hashin", geramos uma matriz de números pseudoaleatórios:

- Um número para cada tipo de peça em cada casa.
- Um número para indicar de quem é a vez.
- Quatro números para indicar direitos de roque, embora geralmente sejam usados 16 (2^4) para maior velocidade.
- Oito números para indicar o arquivo de uma casa de *En passant* válida, se houver.

Assim, temos uma matriz com 781 (12*64 + 1 + 4 + 8) números aleatórios, e ainda é utilizada uma chave para geração do hash. No entanto, ainda existe dentro do Zobrist Hashing uma probabilidade de existir colisão de duas posiçoes, onde duas posições mapeiam a mesma chave. Isso significa que uma posição guardada pode acabar retornando outra completamente diferente, e esse é um risco que as engines correm. Inclusive, uma das características mais importantes sobre o Zobrist Hashing é que chaves parecidas e pequenas mudanças gerem posições completamente diferentes, o que ajuda com o problema de colisões, pois as posições geradas nesses casos são muito distintas e é fácil de fazer uma verificação por absurdo caso necessário.

Para calcular o tamanho da chave, a chance de colisão é levada em conta. Chaves menores são mais rápidas e eficientes em termos de espaço, enquanto as maiores reduzem o risco de colisão. As chances de ocorrerem foram avaliadas por Robert Hyatt e Anthony Cozzie em seu artigo "The Effect of Hash Signature Collisions in a Chess Progra", de 2005 [14].

Colisões de hash demonstram o "paradoxo do aniversári", ou seja, a chance de colisões se aproxima da certeza em torno da raiz quadrada do número de chaves possíveis. Uma colisão em um hash de 32 bits acontecerá em uma a cada $\sqrt{2^{32}} = 2^{16}$ ou cerca de 65 mil posições. Com um hash de 64 bits, pode-se esperar uma colisão após cerca de 2^{32} ou 4 bilhões de posições. Normalmente, são utilizados hashs de 64 bits nas engines de xadrez, balanceando o espaço ocupado e com uma baixíssima chance de colisão.

4.2.5 Livros de Abertura

Os livros de abertura são uma coleção de sequências de movimentos iniciais no xadrez, normalmente provenientes de partidas jogadas por grandes-mestres e analisadas por computadores. Essas sequências são pré-calculadas e armazenadas em um arquivo, que é consultado por engines de xadrez durante as jogadas iniciais da partida. Os livros de abertura são úteis para economizar tempo e recursos computacionais, além de permitir que as engines utilizem aberturas bem estabelecidas e comprovadas.

O uso de livros de abertura busca reduzir a dificuldade de transmitir conceitos de abertura conhecidos por humanos, e também compensar o fato de que humanos também estudam e memorizam aberturas previamente, pelos mesmos motivos.

4.2.6 Tablebases

As tablebases, também conhecidas como bases de finais de partida, são conjuntos pré-calculados de informações sobre todas as posições possíveis em finais de partida com um número limitado de peças. Essas tablebases fornecem informações precisas sobre a melhor

jogada em cada posição e o resultado esperado (vitória, empate ou derrota). Dessa forma, as engines de xadrez podem consultar as tablebases para tomar decisões ideais durante as fases finais da partida.

O tamanho de uma tablebase cresce exponencialmente com o aumento do número de peças no tabuleiro. A quantidade de posições possíveis pode ser calculada pela seguinte fórmula:

$$N = \binom{64}{k} \times (2k - 2), \tag{2}$$

onde N é o número de posições e k é o número de peças no tabuleiro, incluindo os dois reis.

Atualmente, as tablebases mais extensas disponíveis são as Syzygy Bases [15], que cobrem todos os finais de partida e combinações com até 7 peças no tabuleiro. Essas tablebases ocupam mais de 18 terabytes de espaço em disco e mais de 420 trilhões de posições, e existe hoje um trabalho de desenvolver tablebases para 8 peças e inclusive são feitas análises computacionais muito interessantes como, por exemplo, a vitória forçada mais longa (549 lances é o mais longo encontrado até agora!). No entanto, tablebases de 8 peças são estimadas em 38 quadrilhões de posições e ocupariam um armazenamento calculado em 2 petabytes, além de um tremendo esforço computacional, tornando essa possibiliade praticamente inviável com as tecnologias atuais.

5 Desenvolvimento de Engine autoral

5.1 Metodologia

O programa desenvolvido para o presente trabalho foi desenvolvido em linguagem de programação Python, utilizando a biblioteca de xadrez *python-chess* [16] e com uma GUI (Interface Gráfica) de xadrez utilizando PyGame, em uma versão adaptada do código de William Wu Dennis, disponibilizada no Blog DevGenius [17].

Uma versão web também foi desenvolvida, disponível na página https://gustavo-serra.github.io/chess-engine/, porém em uma versão inferior. O Python não é uma linguagem feita para formato web, portanto a biblioteca PygBag [18], que converte o código do Pygame pra java, foi utilizada. No entanto, essa biblioteca tem limitações de importação de bibliotecas externas, o que impossibilitou o uso de livros de abertura pra versão online. Além disso, a performance em geral é muito prejudicada, portanto nessa versão a profundidade é de apenas 4 e a busca não é feita até posições sem captura, o que prejudica fortemente o poder de cálculo da engine, e mesmo assim existe uma certa demora para processamento dos lances. Assim, a engine sofre muito mais na abertura, movimentando muito a dama e buscando material, mas ainda tem uma boa defesa e lances táticos perigosos, que podem pegar muitos jogadores de surpresa e acredito que valha a experiência.

A seguir estão detalhadas quais das técnicas apresentadas no capítulo 4 para avaliação de posições foram selecionadas para o programa e como foram implementadas, escolhidas com base na complexidade de desenvolvimento e impacto no desempenho e performance.

Análise Material

Para análise material, foi utilizada a avaliação material de AlphaZero, calculada a partir de milhares de jogos que a IA jogou contra si mesma [19]. Os autores do artigo comentam que essa avaliação não deve ser levada como regra de ouro, tendo em vista que os jogos de *engines* tendem a ser mais longos e diferentes de maneira geral do que jogos com humanos, o que pode gerar uma diferença na força de cada peça. Entretanto, como uma homenagem e devido à força do programa, foram escolhidos para a implementação. A tabela a seguir mostra os valores atribuídos às peças:

Nessa avaliação, os bispos são levemente mais valiosos que os cavalos, e as torres e dama são mais fortes, mas as proporções de troca se mantém parecidas à avaliação clássica. É interessante observar que, mesmo sem nenhuma heurística pré-programada, AlphaZero calculou valores tão próximos aos que chegamos através de décadas de estudos.

Tabela 2: Valores atribuídos às peças na avaliação material de AlphaZero

Valor das Peças	
Peça	Valor
♙	1
♘	3.05
♗	3.33
♖	5.63
♕	9.5

Mobilidade das Peças

Para a mobilidade das peças, foi atribuído um bônus de 5 (um vigésimo de peão) para cada lance disponível. Esse valor foi atribuído após testagem com valores diferentes, de forma a não influenciar o resultado mais do que outras heurísticas mais relevantes.

Estrutura de Peões

Para avaliação de estrutura de peões, inicialmente foi implementada uma penalidade para peões atrasados e dobrados e uma pontuação maior para peões passados, baseada parcialmente na proposta de Hans Berliner, um renomado enxadrista e Campeão Mundial de Xadrez por Correspondência, em seu livro The System [20], onde descreve os valores nas tabelas 3, 4 e 5 a seguir:

Tabela 3: Valores do peão (multiplicador do valor base)

Rank	Isolado	Conectado	Passado	Passado e conectado
4	1.05	1.15	1.30	1.55
5	1.30	1.35	1.55	2.3
6	2.1	–	–	3.5

No entanto, após a implementação e testes, foi constatado que o impacto da avaliação de estrutura de peões no desempenho foi muito baixo tendo em vista um grande au-

Tabela 4: Valor do peão não passado na abertura

Rank	Colunas a/h	Colunas b/g	Colunas c/f	Colunas d/e
2	0.90	0.95	1.05	1.10
3	0.90	0.95	1.05	1.15
4	0.90	0.95	1.10	1.20
5	0.97	1.03	1.17	1.27
6	1.06	1.12	1.25	1.40

Tabela 5: Valor do peão não passado no final de jogo

Rank	Colunas a e h	Colunas b e g	Colunas c e f	Coluna d e e
2	1.20	1.05	0.95	0.90
3	1.20	1.05	0.95	0.90
4	1.25	1.10	1.00	0.95
5	1.33	1.17	1.07	1.00
6	1.45	1.29	1.16	1.05

mento no tempo de processamento das posições. Buscar todos os peões um a um e avaliar seu status relativo a outros peões se mostrou um processo muito custoso para a implementação em questão. Uma melhor matriz de peças e um processo de avaliação otimizado de estrutura de peões poderiam melhorar esse processo, no entanto, as tabelas piece-square se mostraram eficazes para o uso de peões e muito mais eficientes. A tabela utilizada para os peões foi baseada nos números de Berliner.

Tabelas piece-square

Sem a avaliação de estrutura de peões, as tabelas Piece-Square são definitivamente a heurística mais importante para avaliação de posições nessa engine. Para construir as tabelas, foram utilizados os números de Hans Berliner mencionados no item de estrutura de peões para tabelas de peões e uma conjunção de conhecimentos pessoais do jogo

com as tabelas do TSCP (Tom's Simple Chess Program), programa open-source com propósito educativo sobre engines de xadrez. As tabelas podem ser consultadas no código-fonte.

Foram criadas duas tabelas para cada peça, uma para começo de jogo e uma para finais, e um processo de sobreposição dessas matrizes é feito com base no número de peças restantes, com os seguintes pesos:

- **PesoPeão** = 0
- **PesoCavalo** = 1
- **PesoBispo** = 1
- **PesoTorre** = 2
- **PesoDama** = 4
- **Total** = $PesoPeao * 16 + PesoCavalo * 4 + PesoBispo * 4 + PesoTorre * 4 + PesoDama * 2$

Então, o número de peças em jogo é multiplicado por cada peso e em seguida é atribuído um valor de 0 a 1 correspondente a $SomaPesos/Total = FaseAtual$, em que 1 é o início do jogo e 0 é o fim. O valor de cada peça em cada posição é calculado multiplicando as matrizes $MatrizInicio$ e $MatrizFinal$ da seguinte forma:

$$ValorPeca = MatrizInicio * FaseAtual + MatrizFinal * (1 - FaseAtual)$$

Livro de Abertura

Para a abertura, foi utilizado o livro de aberturas disponibilizado para uso público da engine The Baron [11], uma engine muito forte que competiu o WCCC 2018 (World Computer Chess Championship). O programa consulta lances do livro até no máximo o décimo lance.

Poda Alpha-Beta

O método utilizado para busca foi a Poda Alfa-Beta [12], com profundidade de 5 lances, para maior poder de busca sem tempos de processamento muito elevados. Também foi utilizada uma técnica de busca até posições sem capturas, ou seja, após a profundidade 5, o programa busca por capturas e analisa posições até que não hajam mais capturas, podendo chegar a altissima profundidade. Embora isso aumente consideravelmente o tempo de processamento (mais de 10 minutos para analisar algumas posições no meio de jogo), isso permite que a engine não interrompa sequências de capturas no meio e avalie corretamente posições onde ainda há trocas para serem resolvidas após a profundidade definida ser atingida.

Ordenação dos movimentos

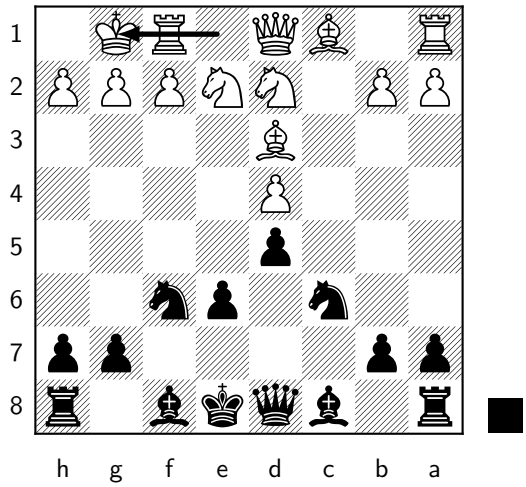
Para otimização da busca, uma ordenação dos movimentos é realizada, priorizando capturas e xeques. Uma pontuação é atribuída a cada lance, com pontuação maior para peças de valor maior sendo capturadas ou dando cheque, e pontuação maior se a peça capturando é de baixo valor (Most Valuable Victim - Least Valuable Aggressor).

5.2 Desempenho

Para avaliar o desempenho da engine e buscar estabelecer uma força, o programa foi colocado contra bots do mais conhecido site de xadrez chess.com [21], que oferece oponentes de diferentes classificações e dificuldades. O bot mais forte derrotado foi o primeiro da categoria "Avançado", com força avaliada em 1500 ELO. A partida em questão, jogada de pretas, será analisado nessa seção.

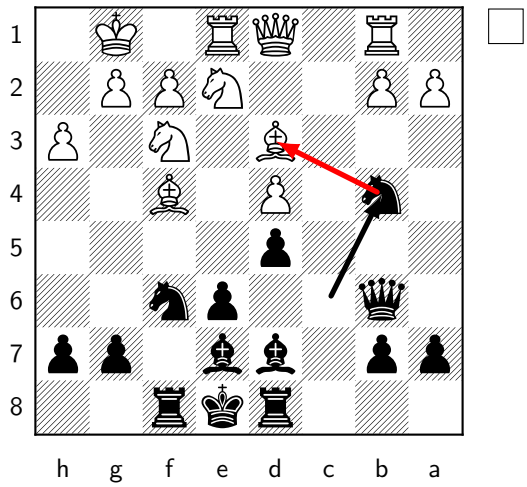
Primeiramente, através do livro de aberturas fornecido, a seguinte posição foi alcançada da abertura após 9 lances:

1 e4 e6 2 d4 d5 3 ♘d2 ♘f6 4 e5 ♘fd7 5 ♗d3 c5 6 c3 ♘c6 7 ♘e2 c×d4 8 c×d4 f6 9 e×f6 ♘×f6 10 O-O Posição final da abertura.



Essa é linha principal da chamada variação *Tarrasch* da defesa Francesa, uma das mais antigas e estudadas aberturas. A engine agora está por conta própria. Ela procede desenvolvendo peças e avança com o cavalo para trocá-lo pelo forte bispo de casas claras das brancas.

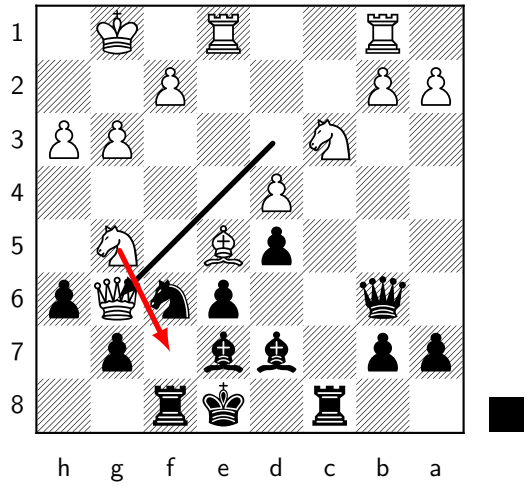
10. . . ♕b6 11 ♘f3 ♗d7 12 ♗f4 ♗e7 13 ♖b1 ♖d8 14 h3 ♖f8 15 ♖e1 ♘b4 Desenvolvendo peças e trocando o cavalo pelo bispo.



Um ponto forte de atenção, que gerou grandes problemas mais pra frente, é a mal colocação das torres. A torre na coluna D estaria melhor colocada na coluna C, que está aberta e livre de peões. Além disso, ao invés do Roque, a torre de A foi movida para f8, talvez se prevenindo de algum ataque imediato, mas com fortes consequências a longo prazo no jogo.

Em seguida, após a troca do bispo pelo cavalo, a engine desenvolvida tentou trocar o outro bispo na ala do rei, porém ele foi defendido (17 ... Cg4, 18. g3). Brancas prosseguiram avançando com o cavalo, ameaçando ganhar o peão em h7, forçando seu avanço para h6, mas possibilitando a infiltração da dama com xeque:

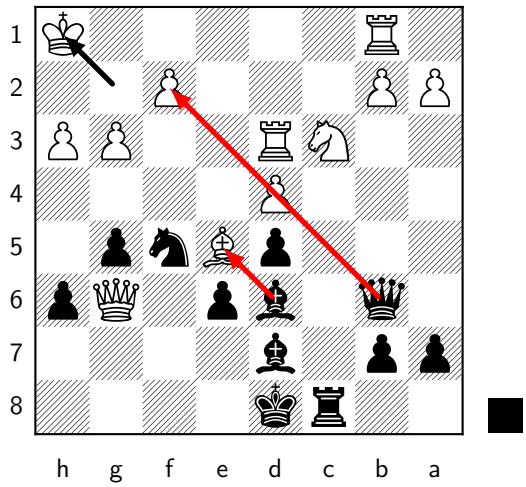
16 ♘c3 ♘×d3 17 ♕×d3 ♘g4 18 g3 ♘f6 19 ♘g5 ♖c8 20 ♗e5 h6 21 ♕g6+ Ataque das brancas e infiltração da dama.



Esse lance força a troca de cavalo por torre, perdendo material, e deixa pretas com a posição muito enfraquecida. O rei no centro do tabuleiro se mostrou um problema para a posição da engine, e foi explorado pelo bot adversário. Aqui, Stockfish dá a posição como ganha para as brancas, porém a linha não é trivial e envolve sacrifício de um cavalo por peão para abrir linhas de ataque da torre e dama, e a o bot oponente não encontra os lances vencedores.

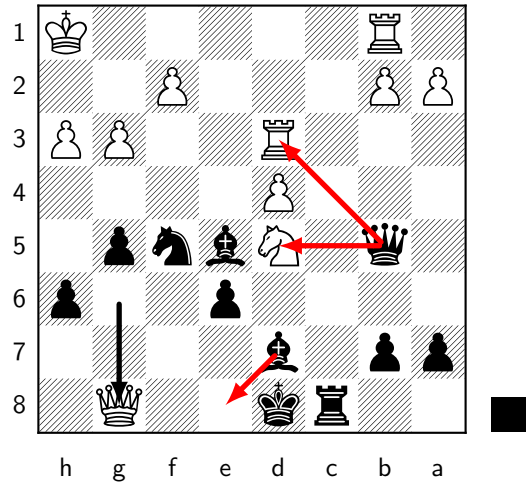
Pretas entram em modo de defesa e coordenam as peças para defender seu material, reposicionando o cavalo e o bispo de casas pretas. Com ótimos lances de recuo, a engine segura a posição por um fio e, no lance 29. Rh2, brancas cometem o primeiro erro. Um lance lento com o rei permite a captura do bispo inimigo, e sua recaptura descobriria a visão da dama para capturar um importante peão em f2:

21... ♔d8 22 ♘f7+ ♖×f7 23 ♕×f7 ♘e8 24 ♖e3 g5 25 ♔g2 ♘d6 26 ♕g6 ♗f8 27 ♖f3 ♘f5 28 ♖d3 ♗d6 29 ♔h1? Erro das brancas.

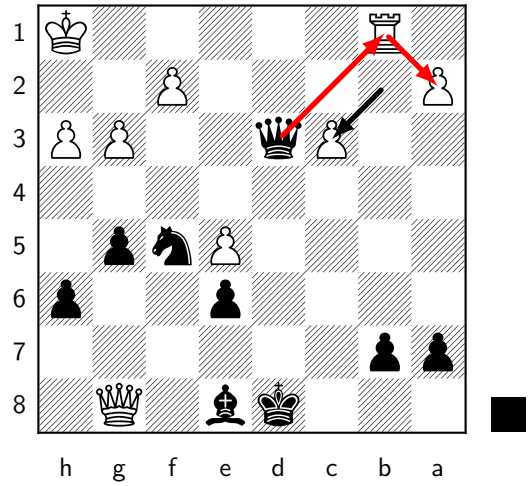


A engine enxerga a ideia, e após a captura do bispo, brancas se recusam a recapturar o Bispo e avançam com o cavalo, e pretas dão ainda a chance de empate para as brancas depois de ... 30. Db5, ao que brancas deveriam deixar a torre e capturar o bispo para possibilitar um xeque perpétuo, mas, com um xeque de desespero, o bot adversário comete mais um erro:

29...♗×e5 30 ♘×d5 ♕b5 31 ♕g8+?? Gafe! Após a defesa do cheque, a perda de material é inevitável

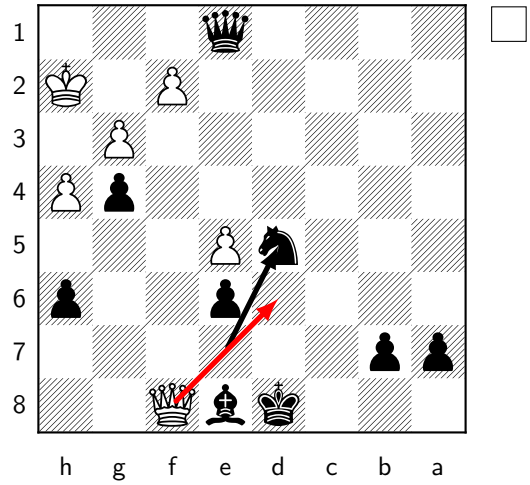


31...♗e8 32 d×e5 ♕×d3 33 ♘c3 ♖×c3 34 b×c3 Pretas capturam as peças restantes.



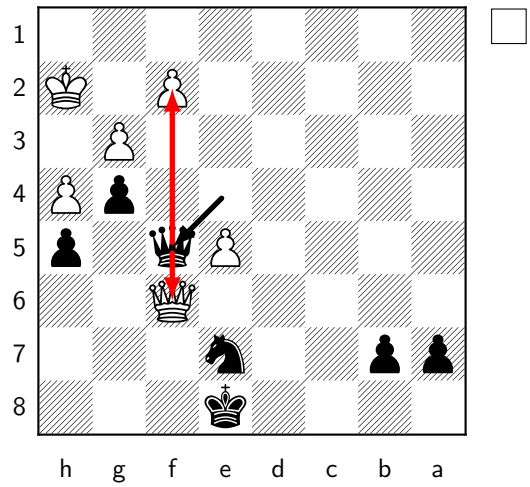
Com duas peças de vantagem, resta converter o jogo. Basta evitar que brancas encontrem um xeque perpétuo, levando ao empate. Após 41... Cd5?, brancas quase conseguem complicar o jogo:

34... ♕×b1+ 35 ♔h2 ♕×a2 36 ♔g1 ♘e7 37 ♕h8 ♕a1+ 38 ♔h2 ♕×c3 39 ♕f6 ♕e1 40 h4 g4 41 ♕f8 ♘d5?



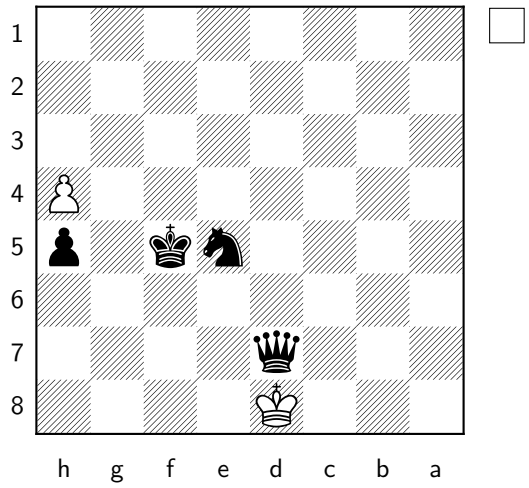
Corretamente avaliando a posição, pretas sacrificam o bispo para interromper os xeques, e após 52... Df5, forçam a troca de damas, simplificando a posição:

42 ♕d6+ ♗d7 43 ♕f8+ ♔c7 44 ♕d6+ ♔c8 45 ♕f8+ ♗e8 46 ♕×e8+ ♔c7 47 ♕f7+ ♔d8 48 ♕f6+ ♔e8 49 ♕×e6+ ♘e7 50 ♔g2 h5 51 ♕f6 ♕e4+ 52 ♔h2 ♕f5 Pretas forçam a troca de damas.



Por fim, basta utilizar a peça adicional e os peões passados para finalizar o jogo.

53 ♔g2 ♕×f6 54 e×f6 ♘d5 55 f3 ♘×f6 56 f×g4 ♘×g4 57 ♔f3 ♔f7 58 ♔f4 ♔f6 59 ♔e4 ♘e5 60 ♔d5 b6 61 ♔d6 ♔f5 62 ♔d5 a5 63 ♔d6 a4 64 ♔c7 a3 65 g4+ ♘×g4 66 ♔×b6 a2 67 ♔c7 a1♕ 68 ♔d8 ♘e5 69 ♔c8 ♕a7 70 ♔d8 ♕d7# Fim de jogo! Vitória das pretas.



Apesar de uma falta clara de conhecimento posicional mais avançado e noção de segurança do rei, a engine demonstrou um ótimo resultado. Com a ajuda da abertura, das tabelas piece-square (para desenvolver peças) e do sistema de busca até a última captura em conjunto com a contagem material, a engine foi capaz de defender bem uma posição quase perdida e encontrar artifícios táticos superiores aos da engine adversária no jogo para virar a situação, superando um bot com rating estimado de 1500, considerado avançado para os padrões do site utilizado.

No quesito de eficiência e tempo de processamento, com as ténicas de otimização utilizadas, o programa demora alguns segundos em posições simples, mas chegou a demorar mais de 10 minutos em posições complexas de meio jogo, o que é longe do ideal. Isso se deve à utilização da linguagem Python e uma biblioteca não totalmente otimizada para performance, e principalmente a falta de técnicas mais complexas e avançadas como tabelas de transposição

e Zobrist Hashing.

Na versão online, com a busca de profundidade limitada em 4 e sem livro de aberturas, além de uma performance bem inferior do Python em uma plataforma web, a engine sofre muito mais na abertura, movimentando muito a dama e buscando material, mas ainda tem uma boa defesa e lances táticos perigosos, que podem pegar muitos jogadores de surpresa.

6 Considerações finais

O presente estudo e construção da engine apresentada nesta tese se mostrou uma grande aventura. Como grande apreciador do Xadrez, da Computação e da Inteligência Artificial em geral, me aprofundar nesse vasto tema proporcionou novos conhecimentos tanto do jogo quanto de programação, e ver a máquina que criei tomando vida foi de uma sensação sem igual. Gostaria ainda de melhorar minha implementação, trazendo uma versão web melhor e implementando as técnicas de transposição e hashing mencionadas, mas finalizo este trabalho com orgulho dessa criação e com esperanças de que eu possa também ter brilhado os olhos de algum leitor amante de xadrez, computação, dos dois, ou nenhum!

Neste trabalho, foram apresentadas as mais diversas e complexas técnicas conhecidas hoje para programas que jogam xadrez, e como podemos transmitir e otimizar um conhecimento abstrato para a linguagem de programação, além de abusar da capacidade de cálculo das máquinas para desenvolver algoritmos mais poderosos do que o poder de processamento de um cérebro humano. É a partir dessa junção de inserção de conhecimento e utilização do poder computacional que reside o assombroso poder da tecnologia e das Inteligências Artificiais. Em tempos de ChatGPT e Dall-E, a discussão sobre os limites de IAs e a potencial substituição de humanos está mais fervorosa do que nunca, e acredito que a humanidade tenha um pouco a aprender com as engines de xadrez. Tal qual hoje vivem os melhores enxadristas da história, aprendendo e crescendo ao lado de máquinas, IAs de processamento de linguagem e geração de imagens podem também impulsionar para limites nunca antes vistos o conhecimento e o potencial humano. É preciso parar de usá-las como meros "trapaceiros" de plataformas virtuais o fazem para ganhar jogos de xadrez online, e nos tornarmos Grandes-Mestres.

7 Referências

[1] H.J.R. Murray, *A History of Chess: The Original 1913 Edition*, Skyhorse Publishing Inc., 2013.

[2] Usuário billwall, Computers and Chess - A History. https://www.chess.com/article/view/computers-and-chess—a-history, acessado em 10/04/2023.

[3] Murray Campbell, A. Joseph Jr Hoane, Feng-hsiung Hsu, *Deep Blue*, Artificial Intelligence, vol. 134, no. 1, pp. 57–83, 1997, Elsevier.

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., *Mastering the game of Go with deep neural networks and tree search*, Nature, vol. 529, no. 7587, pp. 484–489, 2016, Nature Publishing Group.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan and Demis Hassabis, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, arXiv preprint arXiv:1712.01815, 2017.

[6] Romstad, T., Costalba, M., Kiik, J., & Vondele, J. Stockfish chess engine. *https://stockfishchess.org/*

[7] Reuben Fine, *Chess The Easy Way*, David McKay, 1942.

[8] E. Slater, "Statistics for the chess computer and the factor of mobility", in Transactions of the IRE Professional Group on Information Theory, vol. 1, no. 1, pp. 150-152, 1953.

[9] Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 41(314), 256-275.

[10] Kjeldsen, Tinne *John von Neumann's Conception of the Minimax Theorem: A Journey Through Different Mathematical Contexts.* Archive for History of Exact Sciences, 2001.

[11] *http://users.telenet.be/thebaron/html/the_baron.html*. Acessado em 10/02/2023.

[12] Edwards, D.J.; Hart, T.P. *The Alpha–beta Heuristic (Technical report).* Massachusetts Institute of Technology, 1961.

[13] Zobrist, Albert L. (1970). *A New Hashing Method With Application for Game Playing.*

[14] Hyatt, Robert & Cozzie, Anthony. *The Effect of Hash Signature Collisions in a Chess Program.* ICGA Journal, 2005.

[15] Welter, R. (2013). Syzygy Bases. *https://syzygy-tables.info/*

[16] *https://pypi.org/project/chess/*. Acessado em 10/04/2023.

[17] Dennis, William Wu. *Simple Interactive Chess GUI in Python* https://blog.devgenius.io/simple-interactive-chess-gui-in-python-c6d6569f7b6c. Acessado em 10/04/2023.

[18] *https://pypi.org/project/pygbag/0.1.5/*. Acessado em 10/04/2023.

[19] Nenad Tomašev, Ulrich Paquet, Demis Hassabis, Vladimir Kramnik. *Assessing Game Balance with AlphaZero: Exploring Alternative Rule Sets in Chess*, arXiv preprint arXiv:2009.04374, 2020.

[20] Berliner, H. and Burgess, G. (1999). *The System: A World Champion's Approach to Chess*

[21] Chess. www.chess.com. Acessado em 01/04/2023

8 Apêndice

8.1 Código-Fonte

```
import chess
import chess.polyglot
import pygame
import math

# chess.PAWN: chess.PieceType= 1
# chess.KNIGHT: chess.PieceType= 2
# chess.BISHOP: chess.PieceType= 3
# chess.ROOK: chess.PieceType= 4
# chess.QUEEN: chess.PieceType= 5
# chess.KING: chess.PieceType= 6

piece_values = {
    "K": 20000,
    "Q": 950,
    "R": 563,
    "B": 333,
    "N": 305,
    "P": 100,
    "k": -20000,
    "q": -950,
    "r": -563,
    "b": -333,
    "n": -305,
    "p": -100
}

def material_eval(board):
    position_eval = 0

```

```
    for piece in board.piece_map().values():
        position_eval += piece_values[piece.symbol()]

    return position_eval

def get_piece_square_value(board):

    piece_square_value_opening = {
        chess.PAWN: [0, 0, 0, 0, 0, 0, 0, 0,
                     20, 25, 35, 50, 50, 35, 25, 20,
                     6, 12, 25, 40, 40, 25, 12, 6,
                     0, 3, 17, 27, 27, 10, 3, 0,
                     -3, -5, 10, 20, 20, 0, 0, -10,
                     -10, -5, 5, 15, 15, 0, 10, -5,
                     -10, -5, 5, 10, 10, 10, 15, -5,
                     0, 0, 0, 0, 0, 0, 0, 0],

        chess.KNIGHT: [-50, -40, -30, -30, -30, -30, -40, -50,
                       -40, -20, 0, 0, 0, 0, -20, -40,
                       -30, 0, 10, 15, 15, 20, 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],

        chess.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, 0, 0, -10,
                       -10, 10, 5, 5, 5, 5, 10, -10,
                       -10, 20, 5, 10, 10, 5, 20, -10,
                       -20, -10, -10, -10, -10, -10, -10, -20],
```

```
65
66          chess.ROOK: [20, 20, 30, 40, 40, 30, 20, 20,
67                       20, 30, 40, 50, 50, 40, 30, 20,
68                       10, 20, 30, 40, 40, 30, 20, 10,
69                       -5, 10, 20, 30, 30, 20, 10, -5,
70                       -5, 0, 0, 0, 0, 0, 0, -5,
71                       -5, 0, 0, 0, 0, 0, 0, -5,
72                       -15, 0, 0, 0, 0, 0, 0, -15,
73                       -30, -20, 5, 20, 20, 10, -20, -30],
74
75          chess.QUEEN: [-20, 0, 10, 15, 40, 40, 40, 40,
76                      -10, 0, 0, 0, 0, 30, 30, 30,
77                      -10, 0, 5, 5, 5, 30, 30, 30,
78                      -5, 0, 5, 5, 5, 15, 0, -5,
79                      0, 0, 5, 5, 5, 5, 0, -5,
80                      -10, 5, 5, 5, 5, 5, 0, -10,
81                      -10, 0, 15, 0, 10, 0, 0, -10,
82                      -20, -10, -10, 10, -5, -10, -10, -20],
83
84          chess.KING: [-30, -40, -40, -50, -50, -40, -40, -30,
85                      -30, -40, -40, -50, -50, -40, -40, -30,
86                      -30, -40, -40, -50, -50, -40, -40, -30,
87                      -30, -40, -40, -50, -50, -40, -40, -30,
88                      -20, -30, -30, -40, -40, -30, -30, -20,
89                      -10, -20, -20, -20, -20, -20, -20, -10,
90                      5, 10, -5, -50, -50, -20, 20, 20,
91                      -10, 30, 10, -50, -10, -30, 30, 20]}
92
93      piece_square_value_endgame = {
94          chess.PAWN: [0, 0, 0, 0, 0, 0, 0, 0,
95                       55, 35, 25, 15, 15, 25, 35, 55,
96                       45, 29, 16, 5, 5, 16, 29, 45,
97                       33, 17, 7, 0, 0, 7, 17, 33,
98                       25, 10, 0, -5, -5, 0, 10, 25,
```

```
                    20, 5, -5, -10, -10, -5, 5, 20,
                    20, 5, -5, -10, -10, -5, 5, 20,
                    0, 0, 0, 0, 0, 0, 0, 0],

        chess.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],

        chess.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, 10, 15, 15, 10, 5, -10,
                       -10, 0, 10, 15, 15, 10, 0, -10,
                       -10, 10, 5, 15, 15, 5, 10, -10,
                       -10, 0, 5, 10, 10, 5, 0, -10,
                       -20, -10, -10, -10, -10, -10, -10, -20],

        chess.ROOK: [15, 10, 10, 10, 10, 10, 10, 0,
                     15, 15, 15, 15, 10, 10, 10, 5,
                     10, 10, 10, 5, 5, 5, 5, 5,
                     5, 5, 10, 5, 0, 0, 5, 5,
                     5, 5, 5, 5, 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],

        chess.QUEEN: [-10, 20, 20, 30, 30, 20, 10, 20,
                      -20, 20, 30, 40, 50, 30, 30, 0,
                      -20, 0, 5, 40, 40, 40, 0, 10,
```

```
                      5, 20, 25, 40, 40, 40, 50, 30,
                      0, 30, 25, 50,30, 30, 40, 20,
                      -10, 5, 15, 5, 5, 15, 10, 0,
                      -10, 0, 5, 0, 0, 0, 0, -10,
                      -20, -10, -10, -5, -5, -10, -10, -20],

        chess.KING: [-50, -40, -20, -20, -10, 10, 5, -20,
                     -10, 20, 15, 15, 15, 40, 20, 10,
                     10, 15, 25, 15, 20, 40, 40, 10,
                     -5, 0, 25, 25, 25, 30, 25, 10,
                     -20, 0, 20, 25, 25, 25, 10, 5,
                     -10, 0, 10, 20, 25, 15, 5, -10,
                     -30, -10, 5, 15, 15, 5, -5, -20,
                     -50, -30, -20, -10, -30, -15, -25, -45]}

    phase_dict = {chess.PAWN : 0,
    chess.KNIGHT : 1,
    chess.BISHOP : 1,
    chess.ROOK : 2,
    chess.QUEEN : 4}
    TotalPhase = phase_dict[chess.PAWN]*16 + phase_dict[chess.KNIGHT]*4 +
    phase_dict[chess.BISHOP]*4 + phase_dict[chess.ROOK]*4 + phase_dict[chess.
    QUEEN]*2

    phase = TotalPhase
    opening = 0
    endgame = 0

    for piece_type in range(1,6):
        phase -= phase_dict[piece_type]*len(board.pieces(piece_type,chess.
    BLACK))
        for square in board.pieces(piece_type,chess.BLACK):
            opening -= piece_square_value_opening[piece_type][square]
            endgame -= piece_square_value_endgame[piece_type][square]
```

```
164         phase -= phase_dict[piece_type]*len(board.pieces(piece_type,chess.
    WHITE))
165         for square in board.pieces(piece_type,chess.WHITE):
166             opening += piece_square_value_opening[piece_type][(7-square//8)*8
    + square%8]
167             endgame += piece_square_value_endgame[piece_type][(7-square//8)*8
    + square%8]
168 
169     phase = (phase * 256 + (TotalPhase / 2)) / TotalPhase
170 
171     piece_eval = ((opening * (256 - phase)) + (endgame * phase)) / 256
172 
173     return piece_eval
174 
175 
176 def mobility_eval(board):
177 
178     if board.turn == chess.WHITE:
179         temp_board = board.copy()
180         temp_board.push(chess.Move.null())
181         return 5*(board.legal_moves.count()-temp_board.legal_moves.count())
182 
183     if board.turn == chess.BLACK:
184         temp_board = board.copy()
185         temp_board.push(chess.Move.null())
186         return -5*(board.legal_moves.count()-temp_board.legal_moves.count())
187 
188 
189 def evaluate(board):
190 
191     return get_piece_square_value(board) + material_eval(board) +
    mobility_eval(board)
192 
193 
```

```
194 def order_moves(board, only_captures = False):
195 
196     dict_move_scores = dict()
197 
198     for move in board.legal_moves:
199 
200         move_score = 0
201         capturing_piece = board.piece_at(move.from_square)
202         captured_piece = board.piece_at(move.to_square)
203         last_moved_piece = board.piece_at(board.peek().to_square)
204 
205         if captured_piece != None:
206             move_score += piece_values[captured_piece.symbol()]
207 
208         if board.gives_check(move):
209             move_score += piece_values[capturing_piece.symbol()]
210 
211         if move.promotion != None:
212             move_score += piece_values[chess.Piece(move.promotion,board.turn).
    symbol()]
213 
214         if not (only_captures and not board.is_capture(move)):
215             dict_move_scores[move] = move_score
216 
217     return dict(sorted(dict_move_scores.items(), key=lambda item: item[1],
    reverse = True)).keys()
218 
219 
220 def search_alphabeta_pruning(board, depth, ply_from_root=0, alpha=-math.inf,
    beta=math.inf):
221 
222     global best_move
223 
224     if board.turn == chess.WHITE:
```

```
225         color_bool = 1
226 
227     if board.turn == chess.BLACK:
228         color_bool = -1
229 
230     if depth == 0:
231         return search_all_captures(board, ply_from_root, alpha, beta)
232 
233     if board.is_checkmate():
234         return (-1000000 + ply_from_root)
235 
236     if board.is_stalemate() or board.can_claim_threefold_repetition():
237         return 0
238 
239     if ply_from_root == 0:
240         print(order_moves(board))
241 
242     for move in order_moves(board):
243         board.push(move)
244         evaluation = -search_alphabeta_pruning(board, depth - 1, ply_from_root
    + 1, -beta, -alpha)
245         board.pop()
246 
247         if evaluation >= beta:
248             return beta
249         if evaluation > alpha:
250             alpha = evaluation
251             if ply_from_root == 0:
252                 best_move = move
253 
254 
255     return alpha
256 
257 
```

```
258 def search_all_captures(board, ply_from_root=0, alpha=-math.inf, beta=math.inf
    ):
259 
260     global best_move
261 
262     if board.turn == chess.WHITE:
263         color_bool = 1
264 
265     if board.turn == chess.BLACK:
266         color_bool = -1
267 
268     evaluation = color_bool*evaluate(board)
269 
270     if evaluation >= beta:
271         return beta
272 
273     alpha = max(alpha,evaluation)
274 
275     if board.is_checkmate():
276         return (-1000000 + ply_from_root)
277 
278     if board.is_stalemate() or board.can_claim_threefold_repetition():
279         return 0
280 
281     for move in order_moves(board, only_captures = True):
282         board.push(move)
283         evaluation = -search_all_captures(board, ply_from_root + 1, -beta, -
    alpha)
284         board.pop()
285         if evaluation >= beta:
286             return beta
287         if evaluation > alpha:
288             alpha = evaluation
289             if ply_from_root == 0:
```

```
                best_move = move

    return alpha


def engine(board):

    if board.fullmove_number <= 10:
        with chess.polyglot.open_reader("openings/baron30.bin") as reader:
            try:
                return reader.choice(board).move
            except:
                pass

    global best_move

    best_move = chess.Move.null()

    depth = 5

    evaluation = search_alphabeta_pruning(board, depth)

    print(best_move)

    return best_move
```

Listing 1: Código-Fonte da Engine