UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

LEONARDO DALLA ROSA DE MORAES

JOGADOR DE XADREZ ROBÓTICO COM VISÃO COMPUTACIONAL

CAXIAS DO SUL

2021

LEONARDO DALLA ROSA DE MORAES

JOGADOR DE XADREZ ROBÓTICO COM VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Ricardo Vargas Dorneles
Coorientador: Prof. Dr. Guilherme Holsbach Costa

CAXIAS DO SUL

2021

LEONARDO DALLA ROSA DE MORAES

JOGADOR DE XADREZ ROBÓTICO COM VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 22/06/2021

BANCA EXAMINADORA

Prof. Dr. Ricardo Vargas Dorneles
Universidade de Caxias do Sul - UCS

Prof. Dr. Guilherme Holsbach Costa
Universidade de Caxias do Sul - UCS

Prof. Dra. Scheila de Avila e Silva
Universidade de Caxias do Sul - UCS

“A vida é como jogar uma bola na parede:
Se for jogada uma bola azul, ela voltará azul;
Se for jogada uma bola verde, ela voltará verde;
Se a bola for jogada fraca, ela voltará fraca;
Se a bola for jogada com força, ela voltará com força;
Por isso ’nunca jogue uma bola na vida’ de forma que você não esteja pronto a recebê-la.
A vida não dá nem empresta;
não se comove nem se apieda.
Tudo quanto ela faz é retribuir e transferir aquilo que nós lhe oferecemos."
Albert Einstein

RESUMO

Com este projeto foi criado um jogador de xadrez automatizado que, através de visão computacional e automação, consegue detectar as jogadas do adversário e responder com suas próprias jogadas de forma autônoma em um tabuleiro físico. Para tal, foi construído um tabuleiro próprio, contendo dois eixos perpendiculares, movidos com motores de passo, que juntos posicionam um eletroímã responsável por movimentar as peças de xadrez. Sobre esta estrutura está montada uma câmera para a captação das imagens a partir do topo do tabuleiro e, fazendo uso da biblioteca OpenCV, realizar o reconhecimento de cada objeto na cena. Para o processamento do lance de xadrez está sendo utilizada a *engine* StockFish como o algoritmo de IA principal, enquanto para as análises uma IA própria baseada na literatura sobre o DEEP BLUE e no próprio StockFish. O computador responsável por executar toda a aplicação é um Raspberry Pi.

Palavras-chave: Xadrez. Inteligência Artificial. Visão computacional. Rastreio de objetos. Robótica.

ABSTRACT

This project creates an automated chess player that, through computer vision and automation, can detect the opponent's moves and respond with their own moves autonomously on a physical board. To this end, a board was built, containing two perpendicular axes moved with stepper motors, which together position an electromagnet responsible for moving the chess pieces. A camera is mounted on this structure to capture images from the top of the board and, using the OpenCV library, recognize each object in the scene. For the processing of the chess move, the engine StockFish is being used as the main AI algorithm, while for the analysis a proprietary AI based on the literature on DEEP BLUE and on StockFish itself was created. The computer responsible for executing the entire application is a Raspberry Pi.

Keywords: Chess. Artificial Intelligence. Computer vision. Object tracking. Robotics.

LISTA DE FIGURAS

Figura 1 – Tabuleiro do Chaturanga 14
Figura 2 – Representação do espectro eletromagnético 18
Figura 3 – Exemplos de processamento de imagem 19
Figura 4 – Representação visual de um canal de cor 20
Figura 5 – Varredura do gradiente da imagem 23
Figura 6 – Limiar de histerese 24
Figura 7 – Resultados da detecção de bordas Canny 24
Figura 8 – Imagem de referência para as próximas operações morfológicas 25
Figura 9 – Resultado da operação de erosão 25
Figura 10 – Resultado da operação de dilatação 26
Figura 11 – Resultado da operação de abertura 26
Figura 12 – Resultado da operação de fechamento 26
Figura 13 – Árvore de contornos 27
Figura 14 – Retângulo delimitador 28
Figura 15 – Aproximações de polígono 29
Figura 16 – Notação Algébrica do xadrez 31
Figura 17 – Árvore do xadrez 33
Figura 18 – Algoritmo MinMax 36
Figura 19 – Algoritmo Alpha-Beta 37
Figura 20 – Equivalência de posições 38
Figura 21 – Raspberry PI 3 b+ 44
Figura 22 – Diagrama de um motor de passo 45
Figura 23 – Motor de passo mais próximo dos encontrados hoje 45
Figura 24 – Caixa de redução 46
Figura 25 – Etapas do software 47
Figura 26 – Foto do projeto final 48
Figura 27 – Etapas do mapeamento 49
Figura 28 – Mapeamento das casas do tabuleiro 49
Figura 29 – Detecção da jogada 50
Figura 30 – Estrutura dentro do tabuleiro 53
Figura 31 – Tabuleiro com apenas o vidro 53
Figura 32 – Diagrama da ligação elétrica do motor de passo 54
Figura 33 – Tabuleiro 64
Figura 34 – Fila e Coluna 65
Figura 35 – Representação das Peças 65
Figura 36 – Posição inicial das peças 66

Figura 37 – Flancos e nomenclatura das peças 66
Figura 38 – Movimentação do rei 67
Figura 39 – Movimentação da torre 68
Figura 40 – Movimentação do bispo 68
Figura 41 – Movimentação da dama 69
Figura 42 – Movimentação do cavalo 69
Figura 43 – Movimentação do peão 70
Figura 44 – Passos do roque 71
Figura 45 – *En passant* 72
Figura 46 – Xeque mate 73
Figura 47 – Xeque mate abafado e de banda 73
Figura 48 – Rei Afogado 74
Figura 49 – Xeque perpétuo 74

LISTA DE TABELAS

Tabela 1 – Resultados dos jogos contra o tabuleiro físico 56
Tabela 2 – Estatísticas dos jogos contra o tabuleiro físico 58
Tabela 3 – Perfilação da sessão de jogos 58

LISTA DE QUADROS

Quadro 1 – Notação das peças de xadrez 30
Quadro 2 – Sinais convencionais e abreviaturas comuns 31
Quadro 3 – Revisões do Raspberry Pi. 43

LISTA DE ABREVIATURAS E SIGLAS

CNC	Controle Numérico Computadorizado
COI	Comitê Olímpico Internacional
FIDE	*International Chess Federation*
HSV	*Hue Saturation Value*
IA	Inteligência Artificial
LRU	*Least recently used*
NNUE	*Efficiently Updatable Neural Networks*
RGB	*Red Green Blue*
RGBA	*Red Green Blue Alpha*
TCEC	*Top Chess Engines Competition*
UCI	*Universal Chess Interface*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 15
1.2 ESTRUTURA DO TRABALHO 16

2 VISÃO COMPUTACIONAL 17
2.1 PROCESSAMENTO DIGITAL DE IMAGEM 18
2.1.1 Imagem Digital 19
2.2 OPENCV 20
2.2.1 opencv4nodejs 21
2.3 DETECÇÃO DE OBJETOS 21
2.3.1 Detector de Borda - Canny 22
2.3.1.1 Encontrando o gradiente de intensidade da imagem 22
2.3.1.2 Supressão de não máximos 23
2.3.1.3 Limiar de histerese 23
2.3.2 Operações Morfológicas 24
2.3.2.1 Erosão 25
2.3.2.2 Dilatação 25
2.3.2.3 Abertura 26
2.3.2.4 Fechamento 26
2.3.3 Detecção de Contornos 27
2.3.3.1 O que são contornos? 27
2.3.3.2 Retângulo delimitador 28
2.3.3.3 Aproximação de polígono 29

3 XADREZ 30
3.1 NOTAÇÕES 30
3.1.1 Representação das Peças e Sinais Convencionais 30
3.1.2 Notação Algébrica 30
3.1.3 Notação das Partidas 32
3.2 SOLUÇÕES ALGORÍTMICAS 32
3.2.1 Fundação 32
3.2.2 Primeiros Algoritmos - Pesquisa MinMax (1950) 33
3.2.3 Algoritmo Alpha-Beta (1966) 36
3.2.4 Tabelas de Transposição (1967) 37
3.2.5 Aprofundamento Iterativo (*Iteratively-deepening*) (1975) 39
3.2.6 StockFish 39

3.2.6.1 Avaliação do posicionamento 40
3.2.6.2 Técnicas de avaliação 41
3.2.6.3 StockFish.js 41

4 HARDWARE 43
4.1 RASPBERRY PI 43
4.2 MOTORES DE PASSO 44

5 APLICAÇÃO 47
5.1 DETECÇÃO DO TABULEIRO 48
5.2 JOGADA DO USUÁRIO 48
5.3 PROCESSAMENTO DA JOGADA DO COMPUTADOR 51
5.3.1 Implementando com o StockFish 51
5.3.2 Implementando o MinMax 51
5.4 TABULEIRO E MOVIMENTAÇÃO DAS PEÇAS 52

6 RESULTADOS 55
6.1 LIMITAÇÕES DO PROJETO 55
6.1.1 Suporte à configuração da câmera pelo OpenCV 55
6.1.2 Detecção de jogadas inválidas 55
6.2 VALIDAÇÃO DO TABULEIRO 56
6.2.1 Detecção das jogadas 56
6.2.2 Movimentação das peças 57
6.3 VALIDAÇÃO DA IA 57
6.3.1 Resultados da IA 59

7 CONSIDERAÇÕES FINAIS 60
7.1 SUGESTÕES DE TRABALHOS FUTUROS 60

REFERÊNCIAS 62

ANEXO A – DESCRIÇÃO DO JOGO DE XADREZ 64

1 INTRODUÇÃO

O xadrez é um jogo muito conhecido por todo o mundo. É jogado entre 2 jogadores, cada um com 16 peças para si (brancas para um lado e pretas para o outro) em um tabuleiro de 64 casas, cujo objetivo é eliminar o rei adversário.

Atualmente o xadrez é regulamentado pela organização *International Chess Federation* (FIDE), fundada em 20 de julho de 1924, em Paris. A FIDE hoje é reconhecida pelo Comitê Olímpico Internacional (COI) como a organização responsável por organizar e conduzir campeonatos internacionais em níveis continentais, e conta com cerca de 190 federações nacionais filiadas para campeonatos de nível nacional.

Enquanto hoje considerado um esporte, sua origem, no entanto, remonta a mais de mil anos (GIUSTI, 2002).

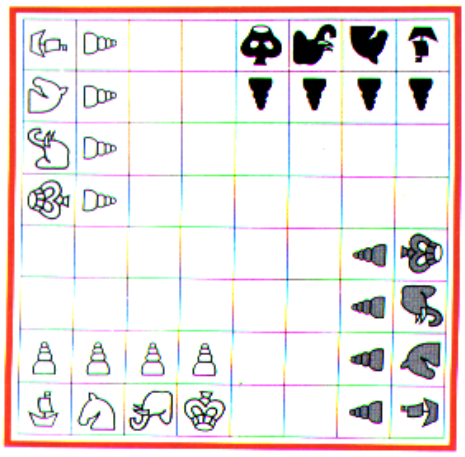
> E, ainda, uma arte: pode criar beleza - em partidas e problemas que produzem, no enxadrista, a emoção estética. E como responde a regras, leis e situações, cuja pesquisa e estudo norteiam os jogadores e lhes dão maior domínio no jogo - O xadrez é, também, uma ciência. (BECKER, 2004)

Embora civilizações antigas como o Antigo Egito e a China Dinástica tenham sido apontadas como a origem do xadrez, a conclusão de novos estudos, no entanto, é que o xadrez tenha origem em um jogo originado da Índia, chamado Chaturanga. O Chaturanga (jogo dos 4 elementos) era um jogo para quatro pessoas, com cada jogador com 8 peças para si. Um ministro, cavalo, elefante, navio/carruagem (equivalente a dama, bispo e torre respectivamente) e com quatro Peões. Segundo arqueólogos, o Chaturanga é praticado na Índia há pelo menos mil e quinhentos anos, porém estudando as peças do jogo pode-se notar semelhança com elementos do antigo exército indiano, até este ser derrotado por Alexandre, o Grande, no ano de 326 A.C. Portanto, pode-se supor que mesmo neste período ele já era praticado (GIUSTI, 2002).

O Chaturanga (Figura 1) foi exportado da Índia para diversas partes do mundo por volta do século VI, através de rotas comerciais com outros países. Para a China, tornou-se o "Jogo do Elefante", para Japão e Coreia tornou-se o "Jogo do General"(Shogi) e para a Pérsia, passa a ser chamado de Chatrang (Jogo do xadrez). É na Pérsia, porém, que o número de jogadores é reduzido a dois e cria-se um novo elemento no jogo, a peça Xá (Rei) (GIUSTI, 2002).

O Chatrang dos árabes, o precursor do xadrez, foi introduzido na Europa através da Espanha, Itália e França, por volta do ano 1000 e no século XI, já era conhecido por toda a Europa. Porém, o xadrez moderno, o qual possui as regras como o conhecemos hoje, levou muito tempo para tomar forma. O primeiro livro a conter todas as regras aceitas hoje foi escrito apenas em 1749, pelo escritor Philidor (GIUSTI, 2002).

Figura 1 – Tabuleiro do Chaturanga



Fonte: COMMONS (2005)

O interesse pelo xadrez, como é possível ver, é muito antigo. Logo, é apenas natural que durante o curso da história, tenha havido tentativas de criar uma máquina capaz de jogar o xadrez. Uma das tentativas mais famosas certamente é "O Turco". O Turco, como ficou conhecido, foi inventado em 1769 pelo Barão Kempelen, um nobre de Pressburg na Hungria, que posteriormente o entregou, junto com o segredo de seu funcionamento, a Johann Nepomuk Mälzel. Logo após sua conclusão, foi exibido em Pressburg, Paris, Viena e outras cidades continentais como Londres e mesmo as principais cidades dos Estados Unidos (POE, 1836).

O Jogador de xadrez Autômato, como era descrito na época, era uma máquina que consistia em um manequim em tamanho real de uma cabeça e torso, com uma barba negra e olhos cinza, vestido com uma túnica turca e um turbante. A sua frente, existia um grande gabinete, que possuía 3 portas e onde estava o mecanismo por trás de sua operação. Durante a apresentação, cada porta era aberta e mostrado seu interior ao público como prova de que a máquina era legítima (POE, 1836) para, após isso, serem fechadas novamente e o apresentador, dando corda na máquina, começar a desafiar espectadores curiosos.

Mas como era de se esperar, após muitos anos e muitas tentativas de desvendar o mistério, o Turco acabou sendo desmistificado como um engenhoso "trote". O que acontecia na verdade era que, um verdadeiro jogador de xadrez se esgueirava dentro do gabinete da máquina e, durante a partida, conseguia observar os movimentos do tabuleiro por debaixo da mesa através de uma ligação magnética com um ímã dentro de cada peça, e controlar o braço esquerdo do autômato através de uma série de alavancas pantográficas (LEVITT, 2000). Infelizmente, o Turco foi destruído em um incêndio no Teatro Nacional da Filadélfia, em 1854.

Foi apenas com a criação dos primeiros computadores que surgiu uma máquina que possui a capacidade de realmente jogar xadrez. Independentemente de projetos protótipos, o primeiro software do gênero a competir com sucesso em um campeonato foi o "MAC HACK"

de Richard Greenblatt, em 1967. Em 1976, o time de programadores David Slate, Larry Atkin, e Keith Gorlen foram os primeiros a ganharem em um campeonato de mestres, o "Minnesota State Championship", com um software (PANDOLFINI, 1997).

Porém, foi apenas em 1997 que uma equipe da IBM conseguiu desenvolver o primeiro software a conseguir ganhar do campeão mundial da época, Garry Kasparov, em um *match* de seis partidas. O IBM's DEEP BLUE sintetizou e modificou as muitas ideias dos trabalhos anteriores durante os anos, adicionando algumas ideias suas também, e criou uma poderosa máquina que criou um marco no mundo do xadrez (PANDOLFINI, 1997).

Infelizmente, após essa conquista, não houve mais pesquisas que chamaram grande atenção na área. E com os algoritmos já desenvolvidos, hoje em dia, jogos do gênero estão por toda a parte na internet. Porém, isso não quer dizer que o assunto está estagnado. Com o avanço do poder computacional dos computadores nas últimas décadas e com as pesquisas em novas formas de inteligência artificial, novos projetos foram criados e são mantidos pela própria comunidade de programadores, como o StockFish. O StockFish é um motor de inteligência artificial, de código livre, que utiliza redes neurais para implementar o algoritmo do jogador e está ativo até hoje.

Tendo a premissa do xadrez como ponto de partida, este trabalho irá utilizá-lo como tema central de um projeto de automação, visão computacional e inteligência artificial. Será criado um tabuleiro físico capaz de analisar a posição de cada peça, calcular sua próxima jogada e movimentar as peças por conta própria, tudo isso sem mais intervenções externas.

1.1 OBJETIVOS

O objetivo final deste trabalho é a criação de um jogo de xadrez, capaz de desafiar o adversário em um tabuleiro físico de forma autônoma, ou seja, sem interação extra por parte de outra pessoa. O computador irá aguardar a jogada do adversário, para após isso realizar a sua, até o término da partida.

Para realizar o processamento e controle dos equipamentos físicos, será utilizada a plataforma Raspberry Pi 3B. Conhecida por seu baixo custo, ela irá proporcionar a mobilidade do equipamento, sem renunciar a muito desempenho.

As metas deste trabalho podem então ser classificados como:

- Realizar a leitura do tabuleiro físico, assim como detectar as jogadas do adversário, através de visão computacional.
- Utilizar-se de métodos de inteligência artificial para processar a jogada do computador.
- Utilizar um equipamento físico para movimentar as peças do tabuleiro físico e completar a jogada.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado da seguinte forma:

- No Capítulo 2 são apresentados os conceitos e métodos por trás do algoritmo de identificação de objetos e rastreio de movimento. Estes algoritmos serão utilizados para o sistema analisar o movimento das peças que o adversário realizou. Como referência, serão utilizados os livros (KAEHLER, 2013) e (GONZALEZ, 2009), além da documentação do OpenCV e alguns artigos sobre detecção de objetos.
- O Capítulo 3 introduz as notações de xadrez, importantes para descrever formalmente o andamento do jogo e realizar a comunicação com os motores de xadrez, assim como os métodos de inteligência artificial utilizados para processar as jogadas. Como referência, serão utilizados o livro Manual de xadrez (BECKER, 2004) para as regras do xadrez, o projeto do DEEP BLUE (HSU, 2002) e seus predecessores para os métodos de inteligência artificial e artigos referentes ao StockFish, o motor de xadrez utilizado no trabalho.
- No Capítulo 4 será apresentada a estrutura que será utilizada para montar o tabuleiro físico, assim como os equipamentos utilizados para a movimentação.
- No Capítulo 5 são apresentadas as principais etapas do desenvolvimento da aplicação.
- No Capítulo 6 estão descritos os testes realizados no projeto, assim como sua assertividade, *benchmarks* para análise de desempenho e problemas encontrados que podem ser resolvidos com um investimento adicional.
- No Capítulo 7 são apresentadas as considerações finais e sugestões de trabalhos futuros.

2 VISÃO COMPUTACIONAL

Dentre os 5 sentidos que nós seres humanos possuímos, a visão é o mais avançado de todos eles. Ela não requer contato com o objeto de interesse, pode em alguns instantes reconhecer o ambiente, identificar distâncias, auxiliar no equilíbrio etc. Dessa forma, não é surpresa que a visão exerça, dentre os nossos sentidos, o papel mais importante na nossa percepção de mundo (GONZALEZ, 2009).

E justamente porque para nós humanos é tão fácil realizar estas tarefas, se torna relativamente fácil ter a errônea impressão que tarefas relacionadas à visão computacional (processamento de imagens) são triviais (KAEHLER, 2013).

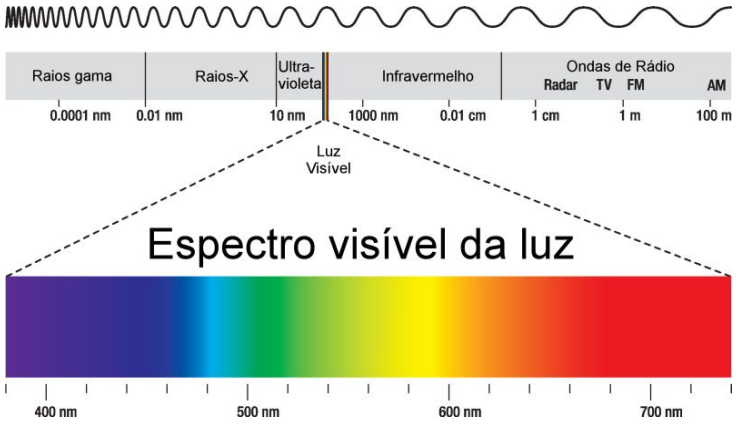
> Î...]O cérebro humano divide o sinal da visão em muitos canais que transmitem diferentes pedaços de informação em seu cérebro. Ele tem um sistema de atenção que identifica, em forma de dependência de tarefas, partes importantes de uma imagem para examinar, ao mesmo tempo que suprime o exame de outras. [...] Existem associações generalizadas de entradas de sensores de controle muscular e todos os outros sentidos que permitem ao cérebro recorrer a associações cruzadas criadas durante anos de vida do indivíduo. [...] Os ciclos de feedback no cérebro remontam a todos os estágios de processamento, incluindo os próprios sensores de hardware (os olhos), que controlam mecanicamente a iluminação via a íris e ajustam a recepção na superfície da retina. (KAEHLER, 2013)

No entanto, o que acontece em um software é bem diferente. O computador recebe apenas uma matriz de números, que pode vir de uma câmera ou de um arquivo local, e é isso. Não há reconhecimento de padrões integrado, sequer controle automático de foco e abertura (KAEHLER, 2013). O software deve implementar cada um desses passos de forma manual, sistemática e programática.

Ainda assim, o que um computador pode conseguir de informações vai muito além da capacidade do "*hardware*" humano. É possível encontrar equipamentos que juntos conseguem cobrir quase todo o espectro eletromagnético, das ondas de rádio até raios gama, enquanto nossos olhos são restritos ao espectro visível (Figura 2). É possível extrair imagens de fontes como ultrassom e mesmo algo como microscopia eletrônica, coisas que raramente eram associadas com imagens (GONZALEZ, 2009).

Dessa forma, por conta do amplo campo de aplicações que a visão computacional proporciona, se torna necessário categorizar essas imagens de alguma forma. Uma forma fácil de realizar isso é categorizá-las de acordo justamente com a sua fonte (espectro visível, raios x, infravermelho, ultrassom etc.) e suas aplicações (GONZALEZ, 2009). Hoje, fotos e vídeos sobre a luz visível reinam soberanos como principal fonte/aplicação. Porém, também utilizamos ondas de rádio para transmitir informação, ondas de raio-x para medicina e inspeção industrial, acústica e ultrassônica para localização e mapeamento, entre muitos outros.

Figura 2 – Representação do espectro eletromagnético



Fonte: CARVALHO (2020)

2.1 PROCESSAMENTO DIGITAL DE IMAGEM

O que é chamado de processamento digital de imagens, portanto, é o trabalho em cima dos dados brutos gerados pelos equipamentos físicos, para modificar ou extrair alguma informação do mesmo. Esse processo pode ser algo mais simples como gerar uma nova imagem a partir da original, a até a extração de atributos e inclusive o reconhecimento de objetos individuais dentro dela (GONZALEZ, 2009).

Dentro deste contexto, GONZALEZ categorizou o processamento de imagens em três níveis distintos, sendo estes:

* **O nível baixo**, envolvendo operações primitivas nas imagens, como o pré-processamento para reduzir o ruído, realce de contraste e o aguçamento de imagens, transformações de cores etc. Este nível é caracterizado pelo fato de tanto a entrada quanto a saída serem imagens.

* **O nível médio**, que envolve tarefas como segmentação (separação de uma imagem em regiões ou objetos), a descrição e a classificação (reconhecimento) de objetos individuais. É caracterizado pelo fato de geralmente receberem imagens como entradas, mas as saídas são atributos extraídos dessas imagens, isto é, bordas, contornos e a identidade de objetos individuais.

* **O nível alto** envolve "dar sentido" aos objetos reconhecidos anteriormente, realizando funções cognitivas normalmente associadas à visão, como análise de imagens, reconhecimento de padrões etc.

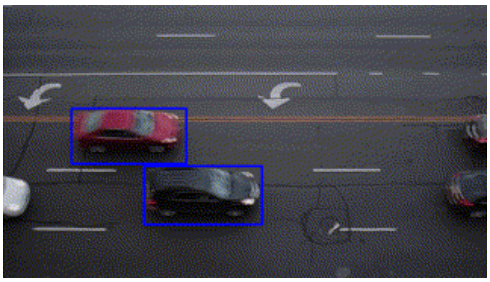
A seguir estão alguns exemplos de processamentos de imagens, cada um em seu respectivo nível. Na primeira Figura 3a está sendo realizada uma operação simples de rotação de

imagem em 90º. Já na segunda, Figura 3b, estão sendo detectados carros atravessando uma rodovia, segmentando e recortando pedaços da imagem original e observando as diferenças entre cada *frame*. Por último, a Figura 3c representa não só a segmentação do rosto de cada personagem, mas também faz uso de Inteligência Artificial (IA) "*Deep Learning*" para reconhecer quem é a pessoa.

Figura 3 – Exemplos de processamento de imagem

(a) Operação de Rotação

Fonte: OPENCV (2020e)



(b) Operação de detecção

Fonte: JUSTADUDEWHOHACKS (2020)

(c) Operação de reconhecimento

Fonte: JUSTADUDEWHOHACKS (2020)

As ações ou decisões que o sistema de visão computacional precisa fazer dependem do contexto no qual este sistema está inserido e a tarefa que deve realizar. Softwares para robôs autônomos, por exemplo, irão empregar estratégias diferentes de um software para câmeras de segurança fixas, seja da necessidade de remover ruído ou dano da imagem, ou para reconhecer e evitar obstáculos. Como regra geral: quanto mais restrito for o contexto de visão computacional, mais simplificada e confiável será a solução final (KAEHLER, 2013).

2.1.1 Imagem Digital

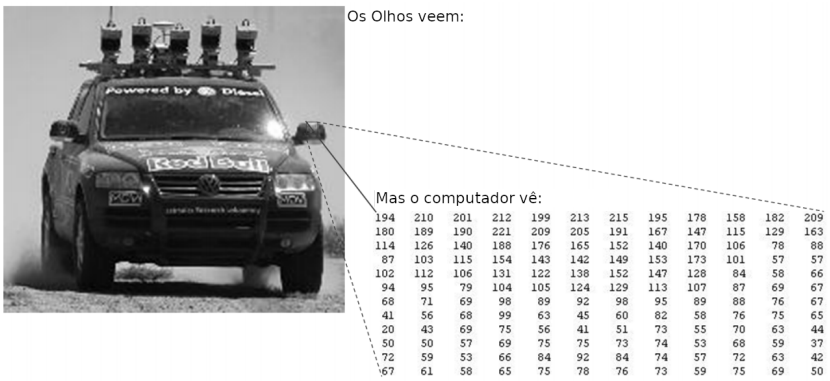
Quando falamos de processamento digital de imagens, por processamento estamos nos referindo a um processamento por um computador e, por ser digital, estamos nos referindo a uma gama discreta de valores. Diferentemente do olho humano, o qual consegue trabalhar em uma escala luminosa na ordem de 10^{10}, o que o computador utiliza é uma escala geralmente de 8 bits, que vai de 0 a 255 (GONZALEZ, 2009). Para cada valor dentro desta escala, então, é dada uma coordenada cartesiana (x e y), e o conjunto destes valores nós chamamos de imagem digital.

> Uma imagem pode ser definida como uma função bidimensional, f(x, y), em que x e y são coordenadas *espaciais* (plano), e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de *intensidade* ou *nível de cinza* da imagem nesse ponto. Quando x, y e os valores de intensidade de f são quantidades finitas e discretas, chamamos de *imagem digital*. (GONZALEZ, 2009)

Cada coordenada deste plano representa um *pixel* na imagem. Para cada *pixel*, está atrelado um ou mais valores de intensidade, cada um deles chamados de *canal*. Estes canais geralmente estão mapeados para um determinado *espaço de cores*. Esse espaço, junto às combinações dos valores de cada canal, torna possível gerar qualquer cor que a percepção humana possa reconhecer. Um exemplo conhecido de um espaço de cores é o *Red Green Blue* (RGB), o qual utiliza 3 canais para mapear cada cor, mas também existem o *Red Green Blue Alpha* (RGBA) no qual o *alpha* representa a transparência e o *Hue Saturation Value* (HSV), que possui os componentes matiz, saturação e brilho.

A Figura 4 mostra a fotografia de um carro em que, no lado do motorista, é possível ver o espelho retrovisor. Enquanto nós como humanos rapidamente extraímos essa informação, o que o computador, na falta de uma palavra melhor, "vê", é apenas uma matriz de números. Números esses que representam a intensidade do pixel (neste exemplo, possui apenas a escala de cinza, ou um canal). Porém, cada um destes números, devido à forma como cada imagem é captada, possui algum grau de ruído associado a ele e por causa disso, sozinho, ele nos dá muito pouca informação. A tarefa da visão computacional, então, se torna transformar esta matriz ruidosa de números na percepção de espelho lateral que nós humanos possuímos (KAEHLER, 2013).

Figura 4 – Representação visual de um canal de cor



Fonte: KAEHLER (2013), adaptado

2.2 OPENCV

O OpenCV (*Open Source Computer Vision Library*) é uma biblioteca de código livre, voltada à resolução de problemas de visão computacional e *machine learning*. Ele foi construído para prover infraestrutura para aplicações de visão computacional em tempo real, hoje

disponibilizando mais de 2500 algoritmos otimizados para o desenvolvedor de forma gratuita (OPENCV, 2020d).

Dentre as funcionalidades destes algoritmos estão a detecção de rostos, identificação de objetos, reconhecer ações humanas em vídeos, rastrear movimentos de câmera, rastrear objetos em movimento, extrair modelos 3D de objetos e reconhecer marcadores de realidade aumentada, entre muitos outros (OPENCV, 2020d). Em alguns casos, é possível inclusive que as próprias funções de alto nível sejam suficientes para a resolução do problema, porém, mesmo quando não é o caso, suas funções básicas são genéricas o suficiente para permitir uma implementação própria do problema.

O OpenCV tem mais de 47 mil usuários na comunidade e um número de downloads excedendo 18 milhões. É extensivamente utilizado em empresas, grupos de pesquisa e inclusive órgãos governamentais (OPENCV, 2020d).

Escrito em C++, ele também possui interfaces para Python, Java e MATLAB, suportando os sistemas operacionais Windows, Linux, Android e Mac OS (OPENCV, 2020d). Neste trabalho, porém, será utilizada uma interface não oficial conhecida como *opencv4nodejs* para Node JS, criada e mantida pelo programador conhecido por *JUSTADUDEWHOHACKS*.

2.2.1 opencv4nodejs

O *opencv4nodejs* é uma interface para API (Application Programming Interface) do OpenCV e OpenCV-contrib modules. Por ser uma interface, ele apenas realiza a ligação entre o código JavaScript do Node JS com a implementação do OpenCV em C++, permitindo realizar as operações da biblioteca com todas as funcionalidades do JavaScript (JUSTADUDEWHOHACKS, 2020).

Este trabalho não entrará em detalhes sobre o funcionamento do *opencv4nodejs*, como essa ligação é realizada, ou mesmo as peculiaridades desta biblioteca. Ao invés disso, serão abordadas as assinaturas das funções do OpenCV na sua forma "crua", em C++, utilizando a sua documentação oficial.

Mais detalhes sobre este projeto, juntamente com um manual de instruções, podem ser encontrados em <https://github.com/justadudewhohacks/opencv4nodejs>

2.3 DETECÇÃO DE OBJETOS

A detecção de objetos utilizada será a sugestão do HANOğLU, o qual fez uso dos contornos de objetos em imagens para reconhecer retângulos, triângulos e outras formas em um vídeo. Esta detecção faz uso de várias técnicas do OpenCV seguindo os seguintes passos:

1. Transformação do frame de uma imagem colorida para cinza

2. Borramento leve da imagem para amenizar o ruído
3. Detecção de bordas com *Canny Edge Detector*
4. Operação morfológica de fechamento na imagem (opcional)
5. Extração de contornos na imagem.

O objetivo do primeiro e segundo passo é reduzir o número de informações na imagem, na tentativa de deixar a detecção de bordas mais direta e suave. Com menos detalhes, facilita inclusive o processo de detecção e extração de contornos. Por último, será abordado como a função *Canny* e a extração de contornos funcionam, visto que são de fundamental importância e relativa complexidade.

Como complemento, às operações morfológicas serão explicadas de forma superficial, pois se trata de operações simples. Já a transformação para escala de cinza e o borramento na imagem, por outro lado, serão omitidos deste capítulo.

2.3.1 Detector de Borda - Canny

O detector de borda Canny é uma evolução do antigo filtro Laplace (não será abordado neste trabalho), refinado por John F. Canny em 1986. Este método calcula o gradiente de intensidade nas direções horizontais e verticais da imagem, para detectar a presença ou não de uma borda. Os pontos onde o valor do gradiente são máximos locais são considerados candidatos para montagem das arestas (KAEHLER, 2013).

2.3.1.1 Encontrando o gradiente de intensidade da imagem

A partir da imagem suavizada, é varrido cada um de seus *pixels* na direção horizontal e vertical para obter a primeira derivada (G_x e G_y respectivamente), gerando duas imagens. A partir dessas duas imagens, é possível encontrar o gradiente de borda e a direção de cada pixel da seguinte maneira de acordo com, respectivamente, a Equação 2.1 e a Equação 2.2 (OPENCV, 2020a):

$$f(G) = \sqrt{G_x^2 + G_y^2}. \tag{2.1}$$

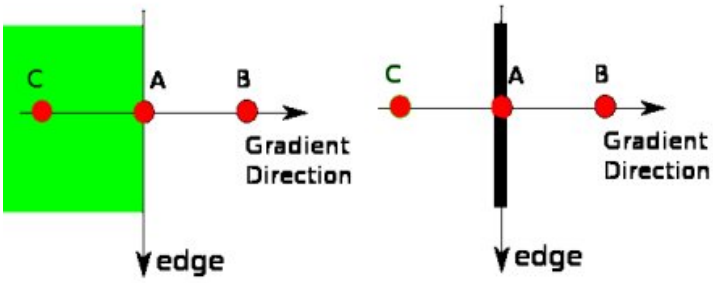
$$f(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right). \tag{2.2}$$

A direção do gradiente é sempre perpendicular às arestas. É arredondado em um dos quatro ângulos que representam as direções vertical, horizontal e as duas diagonais.

2.3.1.2 Supressão de não máximos

Depois de obter a magnitude e a direção do gradiente, uma varredura completa da imagem é feita para remover quaisquer *pixels* indesejados que possam não constituir uma borda (OPENCV, 2020a). Para isso, a cada *pixel*, é verificado se ele é um máximo local em sua vizinhança na direção do gradiente (Figura 5).

Figura 5 – Varredura do gradiente da imagem



Fonte: OPENCV (2020a)

O ponto *A* está na borda (na direção vertical). A direção do gradiente é normal para a borda. Os pontos *B* e *C* estão na direção do gradiente. Portanto, o ponto *A* é verificado com os pontos *B* e *C* para ver se ele forma um máximo local. Em caso afirmativo, é considerado para a próxima etapa, caso contrário é suprimido (colocado a zero). Como resultado, é obtida uma imagem binária (branca e preta) com “bordas finas” (OPENCV, 2020a).

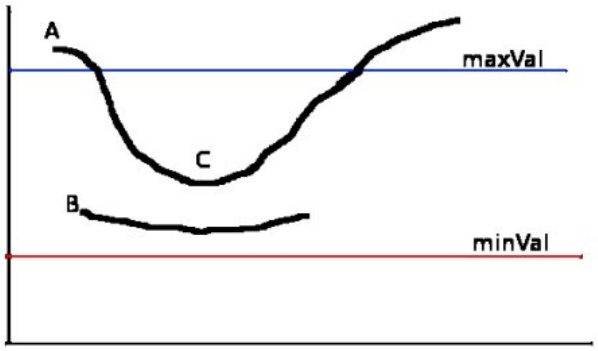
2.3.1.3 Limiar de histerese

Este estágio determina quais arestas são realmente bordas e para isso é preciso de dois valores de limite, *minVal* e *maxVal*. Quaisquer arestas com gradiente de intensidade maior que *maxVal* são certamente arestas e aquelas abaixo de *minVal* certamente não são, sendo, portanto, descartadas. Para as que estão entre esses dois limites, no entanto, são classificadas como bordas de acordo com sua conectividade. Se elas estiverem conectadas a *pixels* que correspondem a "arestas seguras", serão considerados como parte das arestas, caso contrário também são descartados (OPENCV, 2020a).

Na Figura 6, a aresta *A* está acima de *maxVal*, portanto considerada “aresta segura”. *C*, no entanto, está abaixo de *maxVal*, porém está conectada à aresta *A*, de modo que também é considerada uma aresta válida e é obtida a curva completa. Já a aresta *B*, embora esteja acima de *minVal* e na mesma região da aresta *C*, não está conectada a nenhuma “aresta segura”, de modo que é descartada (OPENCV, 2020a). Canny recomendou uma proporção dos valores de *maxVal* e *minVal* entre 2:1 e 3:1 (KAEHLER, 2013).

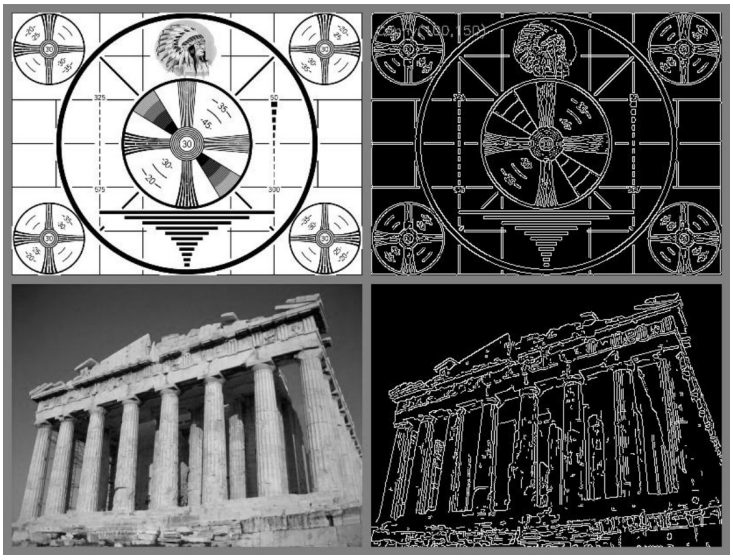
No final deste processo, é possível obter todas as bordas "fortes" presentes na imagem (Figura 7).

Figura 6 – Limiar de histerese



Fonte: OPENCV (2020a)

Figura 7 – Resultados da detecção de bordas Canny



Fonte: KAEHLER (2013)

2.3.2 Operações Morfológicas

As transformações morfológicas são algumas operações simples baseadas na forma da imagem, executadas em imagens binárias. Essa transformação tem duas entradas, a imagem original e o elemento estruturante (também chamado de *kernel*) que decide a natureza da operação.

Dois operadores morfológicos básicos são a erosão e dilatação. Em seguida, combinando-os, é chegado em suas formas variantes como por exemplo abertura e fechamento. Já para o *kernel*, este pode ser interpretado como uma máscara binária, aplicada a cada pixel da imagem e seus vizinhos (OPENCV, 2020c). A Figura 8 traz a imagem original que será utilizada nos próximos exemplos, para comparação.

Fonte: OPENCV (2020c)

Figura 8 – Imagem de referência para as próximas operações morfológicas

2.3.2.1 Erosão

A ideia básica de erosão é como a erosão do solo, ela desgasta os limites do objeto em primeiro plano. O *kernel* desliza pela imagem posicionando-se *pixel* a *pixel* e para cada *pixel*, será considerado branco se e apenas se todos os *pixels* sob o *kernel* forem brancos. Caso contrário, ele é corroído (reduzido a preto) (OPENCV, 2020c).

O resultado é que todos os *pixels* próximos às bordas serão descartados, de acordo com o tamanho do *kernel*, reduzindo a espessura e tamanho do objeto em primeiro plano ou mesmo sumindo com ele de vez. A Figura 9 representa a operação de erosão com um *kernel* 5x5:

Fonte: OPENCV (2020c)

Figura 9 – Resultado da operação de erosão

2.3.2.2 Dilatação

É exatamente o oposto da erosão. Aqui, um *pixel* é branco se pelo menos um *pixel* sob o *kernel* for branco, aumentando o tamanho do respectivo objeto (ou a região branca da imagem).

Normalmente, em casos como remoção de ruído, a erosão é seguida de dilatação. Isso porque, enquanto a erosão remove os ruídos brancos, ela também encolhe o objeto. Como o ruído foi reduzido a zero, ele não voltará, mas o objeto permanece e logo será apenas dilatado de volta para seu tamanho original (Figura 10), porém com a forma um pouco alterada.

Fonte: OPENCV (2020c)

Figura 10 – Resultado da operação de dilatação

2.3.2.3 Abertura

Abertura é apenas o nome utilizado para a operação de erosão seguida de dilatação. Esta operação é útil para remover pequenos ruídos brancos, desconectar dois objetos conectados etc. (OPENCV, 2020c). Essa operação é mostrada na Figura 11.

Figura 11 – Resultado da operação de abertura

Fonte: OPENCV (2020c)

2.3.2.4 Fechamento

O fechamento, por consequência, é a operação inversa da abertura: dilatação seguida por erosão. É útil para fechar pequenos orifícios dentro dos objetos em primeiro plano ou pequenos pontos pretos no objeto, como também para unir partes quebradas de um objeto (OPENCV, 2020c). Essa operação é mostrada na Figura 12.

Figura 12 – Resultado da operação de fechamento

Fonte: OPENCV (2020c)

2.3.3 Detecção de Contornos

Embora algoritmos como o Canny possam ser usados para encontrar os *pixels* das bordas que separam os diferentes segmentos da imagem, eles não informam nada sobre o contexto destas bordas como entidades em si. A próxima etapa é ser capaz de montar esses *pixels* de aresta em contornos (KAEHLER, 2013).

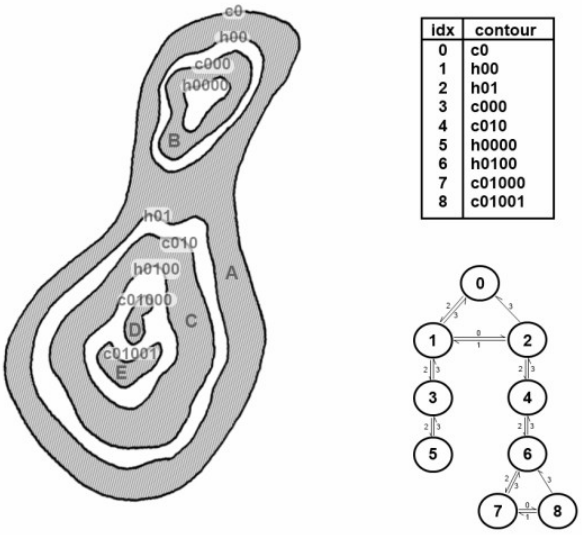
2.3.3.1 O que são contornos?

Os contornos podem ser explicados simplesmente como uma curva que une todos os pontos contínuos ao longo das fronteiras de objetos, tendo a mesma cor ou intensidade (OPENCV, 2020b)

Dito isso, vale a pena dedicar um momento para entender exatamente o que é um contorno e como grupos de contornos podem estar relacionados entre si antes de prosseguir para sua utilização. No OpenCV, os contornos são encontrados a partir de objetos brancos em fundos pretos (OPENCV, 2020b). Utilizando métodos como o Canny, é possível extrair apenas a borda de tais objetos e passando essa nova imagem para a função de contornos, ela vai agrupar todas as arestas conectadas. Cada agrupamento de arestas é o chamado "contorno".

Um conceito particularmente interessante é o conceito de árvore de contornos, que é um recurso importante para o reconhecimento de objetos aninhados em uma estrutura mais complexa (KAEHLER, 2013).

Figura 13 – Árvore de contornos



Fonte: KAEHLER (2013)

A Figura 13 mostra uma imagem de teste (lado esquerdo). Nesta imagem, existem cinco regiões coloridas (marcadas como A, B, C, D e E), mas seus contornos são formados pelas

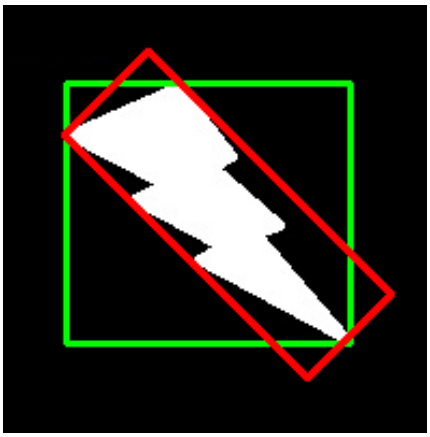
bordas externas e internas de cada região colorida, o que resulta em nove contornos no total. Cada contorno é identificado e aparece em uma lista de saída (canto superior direito).

O conceito de contenção mostrado aqui é importante em muitas aplicações. Por esta razão, geralmente os contornos são retornados como uma árvore de contornos que codifica as relações de contenção em sua estrutura. Uma árvore de contorno correspondente à Figura 13 teria o contorno denominado *c0* como nó raiz, com os buracos *h00* e *h01* como seus filhos. Estes, por sua vez, teriam como filhos os contornos *c000* e *c010*, e assim por diante. (KAEHLER, 2013)

2.3.3.2 Retângulo delimitador

Com a informação do contorno, o OpenCV também consegue trabalhá-los de forma a extrair mais informações. Um exemplo é a função *boundingRect*. O objetivo desta função é retornar os 4 pontos mais externos do contorno, a fim de destacá-lo como um retângulo na imagem ou extrair sua posição. Ele existe em duas formas, o retângulo direto e o retângulo mínimo. O retângulo direto não considera a rotação do objeto e retorna diretamente os pontos onde o contorno se encontra na imagem. Já o retângulo mínimo considera a rotação e retorna o retângulo com a menor área possível para acomodar o objeto (Figura 14).

Figura 14 – Retângulo delimitador



Fonte: OPENCV (2020b)

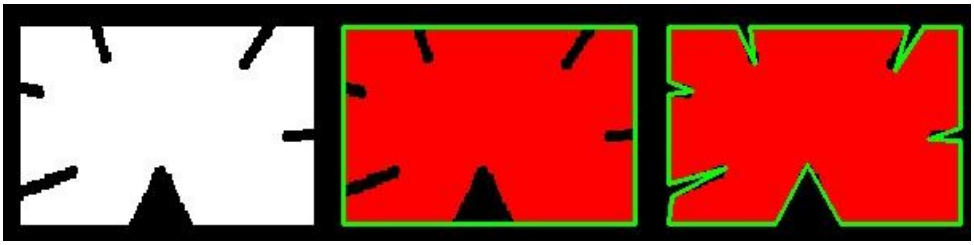
Algumas outras variações dessa função são: *minEnclosingCircle*, *minEnclosingTriangle*, *fitEllipse* etc. Essas funções irão buscar, respectivamente, o menor círculo, triangulo ou elipse que caiba no grupo de pontos do contorno (OPENCV, 2020e). A função *minEnclosingCircle* em particular é interessante pois será utilizada para encontrar as peças de xadrez e seu centro na imagem da câmera.

2.3.3.3 Aproximação de polígono

Aproxima uma nova forma geométrica com menor número de vértices, a partir do conjunto original de contornos. É implementado a partir do algoritmo Douglas-Peucker (OPENCV, 2020b) que não será abordado neste trabalho.

Na Figura 15, por exemplo, foi tirada uma foto de um retângulo, mas devido a alguns problemas na captura da imagem ele veio em um "formato ruim". Com esta função, é possível chegar a uma forma muito aproximada do formato original, apenas configurando a tolerância máxima da distância da deformidade com o contorno aproximado (OPENCV, 2020b).

Figura 15 – Aproximações de polígono



Fonte: OPENCV (2020b)

3 XADREZ

O jogo do xadrez é complexo, possuindo muitas regras e detalhes. Para este trabalho, uma breve descrição sobre o jogo foi reunida no Apêndice A a partir do trabalho de BECKER, que realizou uma compilação de todas as regras, termos e formalização do jogo oficial e as traduziu para o português.

Este capítulo irá assumir que o leitor já conhece o básico sobre o jogo e seguirá diretamente das notações de xadrez, ainda fazendo uso da formalização de BECKER, pois elas serão importantes para ilustrar o funcionamento dos algoritmos que virão em seguida. Os algoritmos apresentados são os mais relevantes (realizaram os maiores avanços) e serão citados em ordem cronológica, juntamente de suas melhorias em relação aos anteriores.

3.1 NOTAÇÕES

Para descrever o andamento da partida, isto é, indicar cada lance que ocorreu durante o jogo, foram criados dois sistemas principais de notação: o *descritivo* e o *algébrico*.

Pelo fato de a biblioteca de xadrez utilizada neste projeto usar apenas a notação algébrica, apenas ela será explicada neste trabalho. Mais detalhes sobre a notação *descritiva* podem ser conferidos na literatura de BECKER.

3.1.1 Representação das Peças e Sinais Convencionais

As peças designam-se pela sua inicial, em letra maiúscula conforme (Quadro 1).

Quadro 1 – Notação das peças de xadrez

Peça	**Inicial**
rei	R
dama	D
torre	T
bispo	B
cavalo	C
peão	P

Já para os sinais convencionais, utiliza-se conforme (Quadro 2).

3.1.2 Notação Algébrica

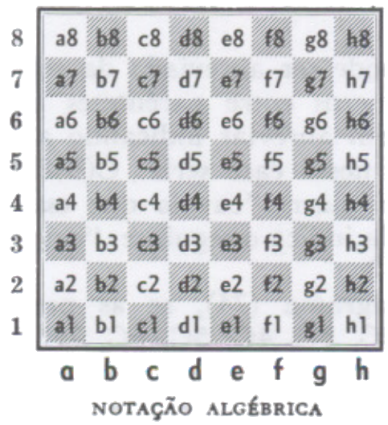
A notação algébrica aponta as casas e os movimentos apenas do ponto de vista das *brancas* (é mais conveniente para notação dos problemas e é menos confusa). Nela, com exceção dos peões, todas as peças são designadas pela sua inicial (Quadro 1).

Quadro 2 – Sinais convencionais e abreviaturas comuns

Sinal	Abreviaturas
o-o	pequeno roque
o-o-o	grande roque
X	toma ou captura
+ ou xq	xeque
++ ou mate	xeque-mate
!	lance bom
!!	lance ótimo
?	lance fraco ou mau
??	lance péssimo
!?	lance duvidoso
+ desc. ou xq desc.	xeque-descoberto
+ dup. ou xq dub.	xeque-duplo
a.p. ou e.p.	ao passar, passando, à passagem

As oito colunas, da esquerda para a direita, indicam-se com as letras a, b, c, d, e, f, g e h. As oito filas numeram-se de 1 a 8, partindo da base das brancas (Figura 16).

Figura 16 – Notação Algébrica do xadrez



Fonte: BECKER (2004), adaptado

Nesta notação, escreve-se a inicial da peça seguida da casa de saída e da casa de chegada. Para os peões indica-se apenas as casas. Em caso de notação abreviada, só se apontam as casas de chegada. Exemplos: Cg1-f3 indica que o cavalo situado em g1 vai a f3. Abreviadamente, escreve-se Cf3; d7-d5 indica que o peão de d7 avança até d5. Abreviadamente, escreve-se d5.

Em caso de duas peças iguais poderem ir à mesma casa, a notação não pode ser inteiramente abreviada. É necessário adicionar uma informação que identifique a peça, como a indicação da fila ou coluna.

Quando um peão é promovido, costuma-se indicar a peça da promoção entre parênteses, ou logo após um sinal de igualdade. Exemplo: para um peão que chega à casa D8 e faz dama, pode ser escrito como PD8(D) ou PD8=D. Alguns simplificam e escrevem PD8.

3.1.3 Notação das Partidas

Os lances das partidas enumeram-se consecutivamente a partir do primeiro lance (1) em diante. Cada número representa um *lance completo*, abrangendo uma jogada das brancas e a resposta das pretas. Há três sistemas para agrupar as notações: *colunar*, *fracionário* e *linear*.

Colunar:

```
1    1. e4    c6
2    2. d4    d5
3    3. Cc3   dXe
```

Fracionária: 1. $\frac{e4}{c6}$ 2. $\frac{d4}{d5}$ 3. $\frac{Cc3}{dXe}$

Linear: 1. e4, c6; 2. d4, d5; 3. Cc3, dXe;

3.2 SOLUÇÕES ALGORÍTMICAS

Após esta explicação do funcionamento do jogo, a partir de agora serão apresentados os principais algoritmos desenvolvidos para o jogo do xadrez, seguindo cronologicamente seus avanços até a partida de Kasparov contra o DEEP BLUE, o primeiro programa de computador a vencer um campeão mundial de xadrez (PANDOLFINI, 1997).

Após isso, será abordado sobre o motor de xadrez chamado StockFish. Este motor é um software relativamente novo, atualmente na sua 12ª versão, construído em C++ e utiliza não só implementações clássicas de IA como MinMax, mas também redes neurais (disponíveis apenas em sua versão mais recente) para a implementação do jogador. Este será o motor utilizado neste trabalho.

> Era uma vez, imagino, que todos os pássaros exibiam sorrisos em seus rostos emplumados, observando os esforços da humanidade para voar. Aviadores humanos aventureiros prenderam asas nas costas, pularam de penhascos, bateram os braços como loucos e caíram no chão. Certamente eles nunca aprenderão a voar como nós, pensaram os pássaros. Por muitos anos, os pássaros riram e riram, mas aos poucos ficaram fascinados com o progresso que estava sendo feito e começaram a sentar-se em silêncio nos galhos, observando o mundo mudar. Enquanto observavam, eles permaneceram em um estado de negação, argumentando que embora a humanidade estivesse fazendo algum progresso, o que eles estavam assistindo não poderia ser chamado de voo. Hoje, construímos aeronaves que foram à lua e além, e os pássaros estão resignados a compartilhar seu espaço com essas maravilhas metálicas elegantes, mas ainda não há nenhum pássaro que concordaria que essas engenhocas inflexíveis realmente podem voar. (PANDOLFINI, 1997)

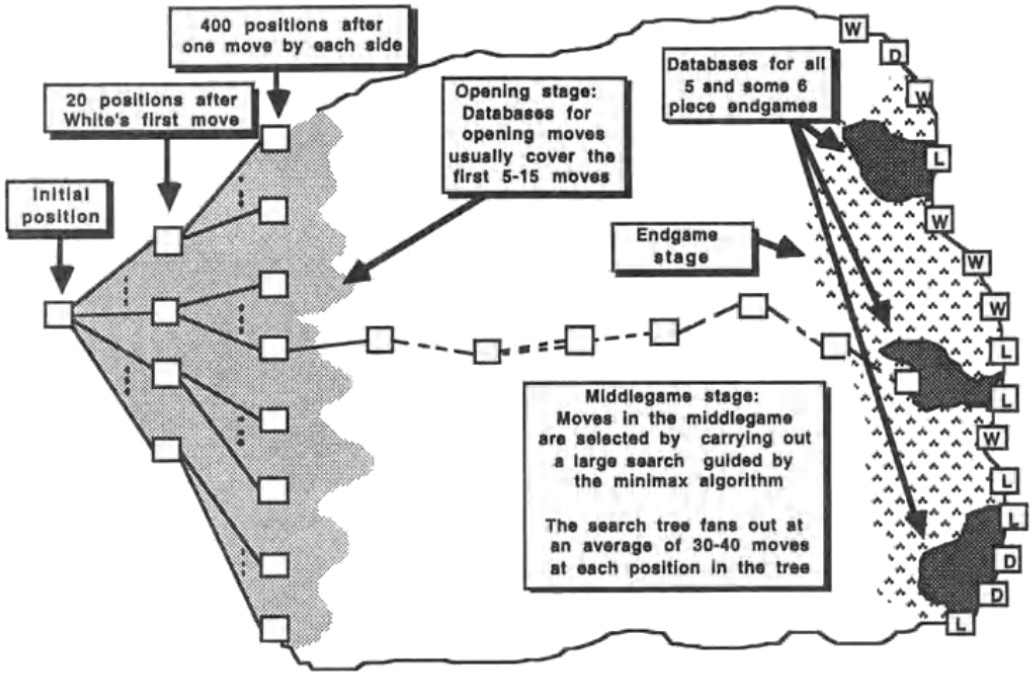
3.2.1 Fundação

O jogo do xadrez consiste em três etapas distintas: a fase de abertura, meio de jogo e o final do jogo.

A fase de abertura dura em torno de 5 a 15 lances, onde os jogadores montam a sua estratégia inicial. Para auxiliar nesta etapa de preparação, os softwares geralmente fazem uso de um grande banco de dados contendo as estratégias iniciais. Contra jogadores profissionais, no entanto, esta estratégia serve mais como um dispositivo de controle de danos do que qualquer outra coisa (PANDOLFINI, 1997). Esta mesma estratégia também é utilizada no final do jogo, onde restam poucas peças no tabuleiro e viabilizam jogadas já processadas.

É na fase de meio de jogo, quando nenhum banco de jogadas é relevante, onde o principal esforço em programação foi investido. Nesta etapa, há em média cerca de trinta a quarenta movimentos possíveis para cada lance, cada um levando a novos trinta a quarenta novos movimentos, e assim por diante. Cerca de mil posições surgem após explorar um lance completo, que aumentam para cerca de um milhão de posições no segundo lance, e assim por diante. Mesmo no final do jogo, que por possuir menos peças esta taxa cresce consideravelmente mais lenta, ainda é inviável pesquisar a árvore inteira devido ao seu custo exponencial (PANDOLFINI, 1997). Esta árvore pode ser visualizada na Figura 17.

Figura 17 – Árvore do xadrez



Fonte: PANDOLFINI (1997)

3.2.2 Primeiros Algoritmos - Pesquisa MinMax (1950)

Os primeiros algoritmos para o jogo do xadrez começaram a aparecer antes mesmo da criação do primeiro computador. Os esforços mais notáveis nesta área podem ser atribuídos a Claude Shannon e Alan Turing, onde embora não tenham trabalhado juntos desenvolveram

métodos semelhantes para resolver o problema. Este método envolvia verificar cada posição decorrente de cada movimento, com o auxílio de uma função de avaliação, e dar-lhe uma "nota" para a nova disposição das peças no tabuleiro (PANDOLFINI, 1997).

Visto que é impossível verificar a árvore inteira, a abordagem de Claude Shannon tornou-se a principal alternativa dos primeiros algoritmos de xadrez. Shannon propôs duas abordagens: A primeira, ou pesquisa tipo A, pode ser descrita como uma pesquisa de "profundidade fixa". Já a segunda, ou tipo B, de pesquisa de "profundidade variável". Embora nenhum programa real tenha sido construído precisamente desta forma, quase todos seguem suas ideias gerais (PANDOLFINI, 1997).

A estratégia Tipo A de Shannon é exatamente o método comentado acima, podendo ser considerada como "força bruta". Um movimento é escolhido a partir de uma determinada posição, explorando-se então todos os lances de jogo até alguma profundidade fixa e, em seguida, atribuindo uma pontuação à posição no final de cada galho da derivação (PANDOLFINI, 1997). Para decidir a pontuação a ser atribuída, Shannon sugeriu os três seguintes parâmetros:

1. **Peças**: Atribui-se valores para cada tipo de peça: Dama = 9, Torre = 5, Bispo = 3, Cavalo = 3, Peão = 1 e Rei = 200. O valor do rei deve ser maior que todos os outros juntos, para garantir que nunca será sacrificado em uma jogada.

2. **Formação de peões**: Uma penalidade de 0,5 para cada peão dobrado (na mesma fila), isolado ou recuado.

3. **Mobilidade**: É desejável ter quantos movimentos forem possíveis para si e poucos para o oponente. Será creditado 0.1 pontos para cada movimento disponível.

Baseado nestes três fatores, e assumindo que o lado branco irá se mover, a pontuação SCORE(P) da posição P pode ser dada pela fórmula:

$$SCORE(P) = 200(R - R') + 9(D - D') + 5(T - T') + 3(B - B' + C - C') + (P - P') - 0.5(DP - DP' + IP - IP' + RP - RP') + 0.1(Mob - Mob')$$

Onde R, D, T, B, C e P representam as peças; DP, IP e RP representam os peões dobrados, isolados e recuados respectivamente e Mob o número legal de movimentos para as brancas. As peças pretas são seguidas pelo traço "'". O resultado desta conta é o quanto a posição é favorável para um dos lados, sendo geralmente o número positivo destinado para as brancas e negativo para as pretas.

Embora essa função de pontuação simples funcione surpreendentemente bem, existem inúmeras situações em que ela falha. Além disso, como já comentado anteriormente, em média um jogo em fase de meio de jogo possui de 30 a 40 movimentos legítimos que precisam ser verificados, o que para apenas dois lances completos resulta em mais de um milhão de posições. Isso significa que um computador, avaliando uma posição terminal a cada microssegundo

(0.000001 segundos), precisa de 16 minutos para processar um único lance completo (PANDOLFINI, 1997).

Assim, Shannon introduziu dois novos conceitos na sua segunda estratégia (o tipo B): estabilidade e poda. A primeira busca todos os ramos até uma certa profundidade e, caso o ramo seja "instável", ele vai algumas posições além (PANDOLFINI, 1997). Esta é uma pesquisa de quiescência, pois visa resolver uma posição dinâmica em uma estável, onde a função de avaliação será mais precisa, além de seu fator de ramificação ser consideravelmente menor que a busca principal. Esse ramo "instável" nada mais é do que um movimento em que resulta em uma captura de uma peça. (WALKER, 2020)

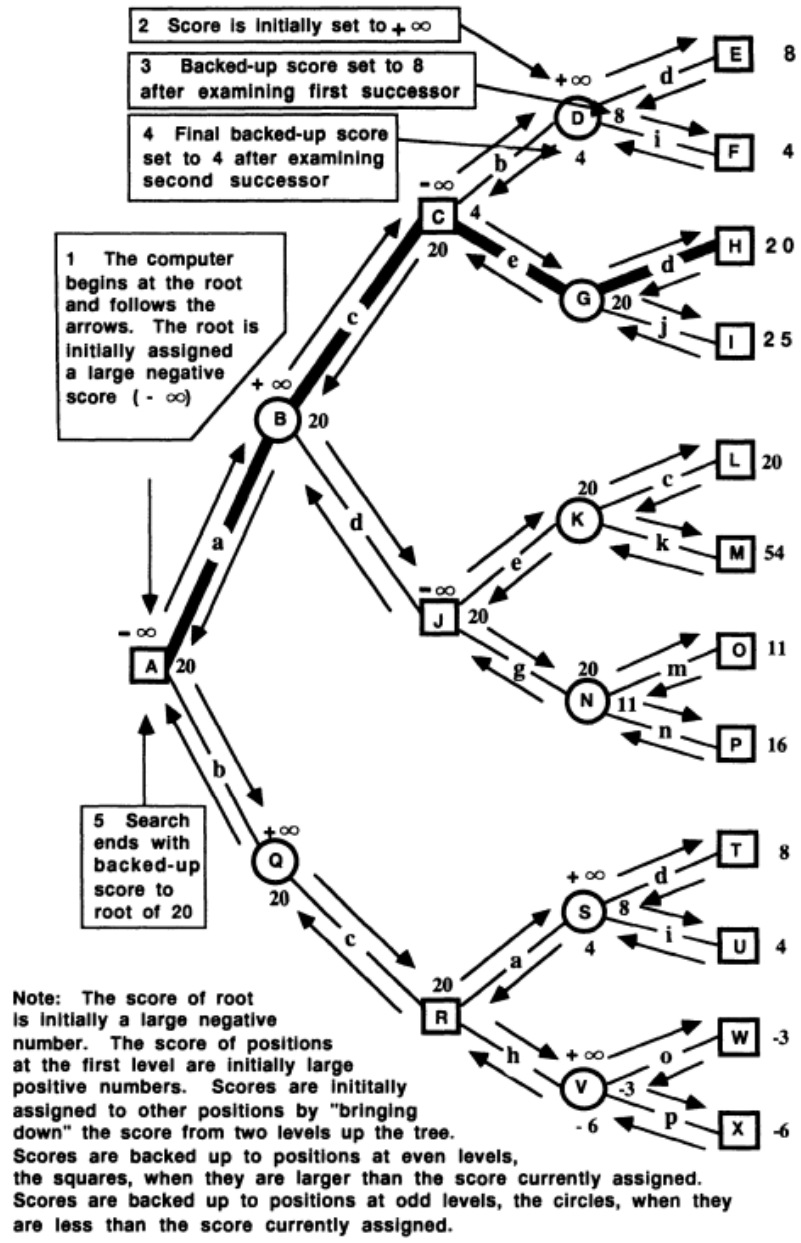
Já a poda é uma seleção preliminar de movimentos, onde os "obviamente ruins" são eliminados sem sequer serem avaliados (PANDOLFINI, 1997). Esta diferenciação do Tipo A é um pouco diferente do "alpha-beta pruning" que será tratado mais para frente, pois aqui ele apenas ignora os movimentos. A decisão de quais movimentos serão ignorados é feita pela própria função de avaliação em conjunto com heurísticas como “verificações-capturas-ameaças”. Esses cortes mantêm o fator de ramificação baixo na árvore de pesquisa, para que possam pesquisar com uma profundidade maior, porém uma vez que eles não consideram muitos movimentos, eles estão propensos a perder muitas táticas importantes (WALKER, 2020).

Todavia, independente da estratégia escolhida, o objetivo do algoritmo é a partir do nó raiz da árvore encontrar o nó terminal (ou folha) que mais favorece o jogador. Tomando como exemplo o computador jogando com as brancas, para cada jogada sua o algoritmo irá buscar o nó com a maior pontuação, maximizando a pontuação de sua jogada. Já as pretas (inimigas), por sua vez, tentarão buscar a jogada com a menor pontuação (que é sua melhor jogada), e o computador pode tomar vantagem disso para dentro deste ramo escolher sua próxima melhor jogada. É a partir deste jogo de maximizar e minimizar a pontuação que se deu a este algoritmo o nome de MinMax.

Na Figura 18 é possível acompanhar este raciocínio. O algoritmo começa na raiz da árvore, na posição A, atribuindo o menor valor possível ($-\infty$). É utilizado o menor valor possível pois qualquer jogada que venha a ocorrer terá uma pontuação no mínimo maior que $-\infty$, permitindo ao algoritmo escolher ao menos uma jogada, por pior que ela seja (PANDOLFINI, 1997). Posteriormente, o algoritmo fará a cópia de uma pontuação até a raiz sempre que um de seus filhos tiver um valor mais positivo que o atual.

O próximo nó da árvore, a posição B, será processado de maneira semelhante, porém com valor inicial de $+\infty$, copiando o valor para si de qualquer filho com valor menor que o atual. Essa sequência irá se repetir, trocando o valor do sinal, até chegar nas folhas da árvore, ou as posições terminais, as quais serão avaliadas com a função de pontuação e seu valor propagado de volta para os nós pais (PANDOLFINI, 1997).

Figura 18 – Algoritmo MinMax



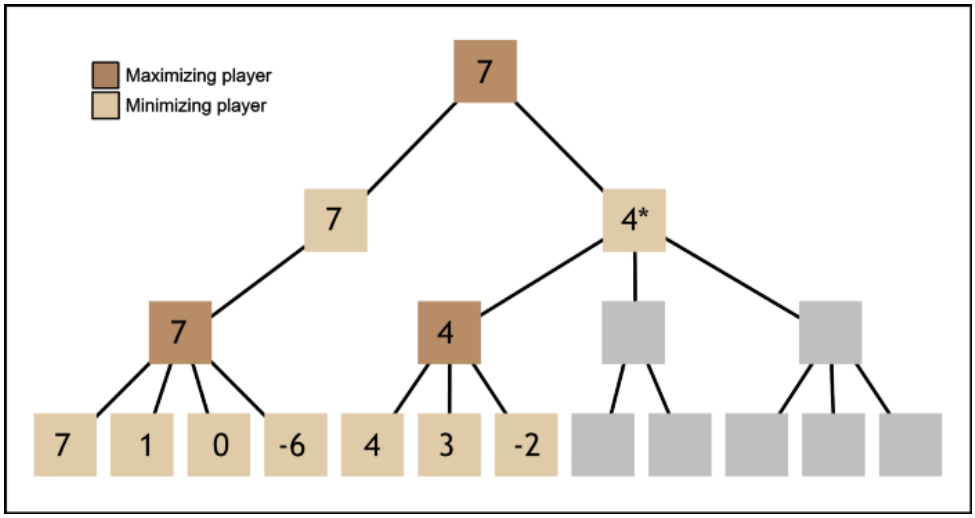
Fonte: PANDOLFINI (1997)

3.2.3 Algoritmo Alpha-Beta (1966)

O algoritmo alpha-beta parece ter sido implementado primeiramente no programa Nss, porém também foi utilizado nos projetos KOTOK/McCARTHY e ITEP por volta desta época. Assim, como falado anteriormente, ele tira vantagem do fato de que ao realizar uma pesquisa MinMax em profundidade na árvore do jogo, nem todos os ramos precisam ser examinados para determinar a continuação principal, ou o mais vantajoso (PANDOLFINI, 1997). A vantagem dessa abordagem é que com poucas modificações no algoritmo do MinMax, já se torna possível realizar essas podas.

O alfa-beta acelera o minmax ao ignorar os movimentos simplesmente "ruins" de uma árvore de pesquisa. Isso pode ser feito adicionando dados extras a cada nó, que representam o pior resultado para cada jogador daquele nó. Mais uma vez tomando as brancas como o lado que busca maximizar os valores da árvore, uma vez que elas saibam que as pretas escolherão uma resposta que minimiza a avaliação, elas também sabem que devem evitar movimentos que permitam ao jogador minimizador tornar as coisas ainda piores do que já estão. Esses movimentos são então imediatamente “podados" da pesquisa e ignorados (WALKER, 2020). A Figura 19 contém um exemplo visual desta estrutura.

Figura 19 – Algoritmo Alpha-Beta



Fonte: WALKER (2020)

A poda alfa-beta torna viável a busca por força bruta, produzindo exatamente os mesmos resultados que o MinMax convencional. Na teoria, o ganho em performance e custo computacional é da ordem da raiz quadrada. Se o MinMax leva 100 segundos para processar, por exemplo, o alpha-beta ideal levaria apenas 10. Isso na prática, porém, quase nunca acontece (WALKER, 2020).

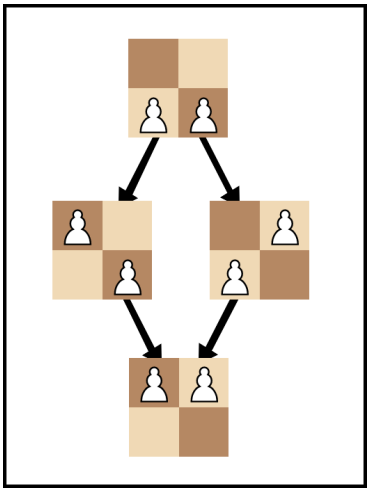
A razão para isso é porque não há nenhum mecanismo para decidir quais nós processar primeiro, então o algoritmo apenas os "sorteia" aleatoriamente. Isso pode levar a uma situação em que o programa analisa do pior movimento ao melhor, pois ainda não sabe que este é o pior (WALKER, 2020).

3.2.4 Tabelas de Transposição (1967)

Tabelas de transposição são um meio de armazenar informações sobre cada posição à medida em que ela é pesquisada para uso posterior. Se a mesma posição for encontrada mais tarde, as informações nesta tabela tornam desnecessária uma nova pesquisa a partir desta posição (PANDOLFINI, 1997). Ela nada mais é do que um cache para a busca em profundidade.

A partir da posição inicial existem muitos ramos que, através de uma transposição de movimentos, conduzem às mesmas posições. Duas posições são idênticas se e apenas se: possuírem configurações de peças idênticas, direitos de roque idênticos, opções *en passant* idênticas e o mesmo lado para mover (Figura 20) (PANDOLFINI, 1997).

Figura 20 – Equivalência de posições



Fonte: WALKER (2020)

Estas tabelas de transposição, essencialmente *hashtables* enormes, são consultadas a cada posição encontrada pelo programa. Caso ele encontre a posição, e se as informações na tabela forem suficientes, essa posição não precisa ser pesquisada novamente e pode ser considerada uma posição terminal. Neste caso, a pontuação utilizada é a própria pontuação armazenada na tabela, ou em caso contrário, continuará a busca a partir dela. Porém, se nenhuma correspondência for encontrada, evidentemente, será dada continuidade à avaliação de forma convencional. Em ambas as situações, cada nova posição encontrada será adicionada à tabela (PANDOLFINI, 1997).

Este cache do programa consegue aumentar a velocidade da pesquisa consideravelmente, principalmente na fase de final de jogo quando são extremamente eficazes. No entanto, esta estratégia possui seus problemas, sendo o maior deste o espaço de armazenamento. Como dito anteriormente, basta apenas alguns lances para alcançar mais de um milhão de posições possíveis, esgotando rapidamente os recursos do computador.

Por causa disso, várias estratégias de substituição foram desenvolvidas, com o principal fator determinante sendo a profundidade da entrada. Quanto maior a profundidade, provavelmente mais útil será a entrada. Outra métrica relevante é o tamanho da subárvore, quanto maior a subárvore, mais útil é a entrada (PANDOLFINI, 1997).

3.2.5 Aprofundamento Iterativo (*Iteratively-deepening*) (1975)

A pesquisa de aprofundamento iterativo é um conceito simples. Ela basicamente corrigiu o problema que os computadores estavam tendo com a pesquisa em profundidade: ao fazer uma pesquisa em profundidade, é necessário decidir, antes de iniciar a pesquisa, exatamente em que profundidade pesquisar (PANDOLFINI, 1997).

Acontece que na prática esta é uma decisão muito difícil de tomar. Se a profundidade definida for muito grande, a pesquisa demorará muito. Se for muito pequena, a busca perderá muitas informações. Mesmo que uma profundidade de seis níveis em um determinado movimento tenha funcionado bem, isso não significa que será apropriada para o próximo. Por exemplo, se uma rainha é capturada, muitas vezes é possível pesquisar mais profundamente em movimentos subsequentes porque o número médio de ramos em cada posição na árvore de pesquisa é menor. Por outro lado, se um peão for promovido a rainha, a profundidade da busca deve ser reduzida em movimentos subsequentes pelo motivo oposto (PANDOLFINI, 1997).

O aprofundamento iterativo funciona da seguinte maneira: em vez de decidir inicialmente a profundidade da pesquisa, o computador realiza uma sequência de pesquisas em profundidades rasas, em recursão e cada vez mais profundas, enquanto o limite de tempo permitir (PANDOLFINI, 1997).

A primeira "iteração" realiza a busca apenas até o próximo movimento, após isso ordena os resultados através de uma pesquisa "superficial" e segue para a próxima iteração. Esta próxima iteração irá começar a pesquisar a partir dos ramos mais promissores encontrados anteriormente, olhando para todas as continuações de dois movimentos para encontrar o melhor ramo. Se o tempo permitir, uma terceira busca em profundidade é realizada olhando para todas as continuações de três movimentos, e assim por diante. Quando o tempo se esgota, a busca para e o que quer que seja considerado a melhor continuação até agora é aquele selecionado como melhor jogada. (PANDOLFINI, 1997). Em outras palavras, esta busca irá olhar para os movimentos mais promissores primeiro, deixando os "ruins" para o final.

O critério de ordenação pode ser algo simples como ordenar as capturas primeiro e considerar todo o resto depois. Porém algumas heurísticas de escala mais precisa precisam ser usadas, como primeiro olhar para capturar a última peça movida etc. (WALKER, 2020)

3.2.6 StockFish

O StockFish é um motor de xadrez *opensource*, disponibilizado gratuitamente e desenvolvido para as plataformas Windows, Mac, Linux, Android e iOS (STOCKFISH, 2020). Ele é considerado um dos melhores competidores do prestigiado *Top Chess Engines Competition* (TCEC), alcançando as finais desde a 4ª temporada. Estabeleceu seu status de número um mundial ao vencer vários TCECs, deixando seus rivais comerciais *Komodo* e *Houdini* para trás (PROGRAMMING, 2020).

O StockFish possui duas funções de avaliação para o xadrez: a clássica, baseada em termos convencionais, e a recém adicionada avaliação *Efficiently Updatable Neural Networks* (NNUE), mais poderosa, baseada em uma eficiente auto atualizável rede neural (MCOSTALBA, 2020).

Apesar de ser amplamente conhecido no mundo do xadrez, o StockFish infelizmente não possui uma documentação oficial a respeito, ao menos até a data de escrita deste trabalho. Na falta desta documentação, será utilizado como referência o artigo do site PROGRAMMING. Este artigo realizou uma síntese do funcionamento do StockFish, a partir do seu código fonte, onde destacou seus principais pontos (PROGRAMMING, 2020):

3.2.6.1 Avaliação do posicionamento

As métricas do StockFish seguem mais ou menos a linha de raciocínio já abordada até então. Algumas delas são:

1. Ter mais peças é melhor do que ter menos.
2. Pontuação de cada peça:
    a) Peão no meio do jogo = 198, Peão no final do jogo = 258
    b) Cavalo no meio do jogo = 817, Cavalo no final do jogo = 846
    c) Bispo no meio do jogo = 836, Bispo no final do jogo = 857
    d) Torre no meio do jogo = 1270, Torre no final do jogo = 1281
    e) Dama no meio do jogo = 2521, Dama no final do jogo = 2558
3. Posicionamento Geral:
    a) Controlar (atacar / defender) o centro é melhor do que controlar casas desocupadas.
    b) Defender peças amigas importantes (que controlam casas importantes) é bom.
    c) Atacar peças inimigas importantes (que por sua vez controlam casas importantes) é bom.
4. Posicionamento Específico:
    a) **Cavalo**:
        i. Prefere postos avançados (casas onde são suficientemente protegidos por um peão), perto do centro do tabuleiro.
    b) **Bispo**:
        i. Prefere o controle das principais diagonais.

ii. Prefere estar emparelhado (tendo o outro bispo ainda no jogo) em posições abertas (onde a maioria das peças não são bloqueadas).

c) **Rei**:

i. Prefere estar na segurança de casas difíceis para o oponente dar xeque-mate (ou controle todas as outras casas ao redor).

d) **Peões**:

i. Peões duplos e triplos (que estão na mesma coluna) são ruins, pois têm mobilidade limitada e não podem se defender.

ii. Peões isolados são ruins, já que não podem ser defendidos por outras peças.

iii. Peões passados protegidos são bons, eles podem ser promovidos a rainha em breve.

3.2.6.2 Técnicas de avaliação

As principais técnicas que o StockFish utiliza já foram explicadas durante o trabalho: *Alpha-beta pruning*, tabelas de transposição, bancos de dados de abertura e final de jogo e o Aprofundamento Iterativo (PROGRAMMING, 2020).

Claro que, outras abordagens como processamento paralelo também são utilizadas, mas elas estão fora do escopo deste trabalho e não serão aprofundadas. Mais detalhes sobre a implementação do StockFish podem ser encontrados no artigo do PROGRAMMING.

3.2.6.3 StockFish.js

Para este trabalho não será utilizada a implementação original do StockFish, mas sim um porte (reescrito para outra linguagem) para JavaScript conhecido como StockFish.js, desenvolvido pelo usuário NMRUGG. Esta biblioteca está, na data de criação deste trabalho, em sua versão 10.0.2 e sua principal função é prover para jogos de navegador o suporte para motor StockFish (NMRUGG, 2020).

Sua proposta é bastante simples: ser apenas um motor de xadrez e nada mais. A aplicação, portanto, deverá construir a interface com o usuário e apenas chamar a api do StockFish.js para calcular as jogadas do computador. Esta é exatamente a funcionalidade desejada por este trabalho, visto que a "interface de usuário" será o próprio tabuleiro físico.

Esta arquitetura, onde é separado a interface de usuário do motor de xadrez, é na verdade tão comum que foi criado um padrão de comunicação para esta finalidade. O protocolo implementado pelo StockFish.js é o *Universal Chess Interface* (UCI), descrito em sua forma modelo por KAHLEN. Os comandos deste protocolo são descritos na página <http://wbec-ridderkerk.nl/html/UCIProtocol.html>.

No caso do StockFish.js, a comunicação com o motor é mediada através da api *Workers* do JavaScript (os *workers* podem ser considerados o equivalente a *threads* no JavaScript, apesar de não ser exatamente como uma *thread* convencional). Cada comando deve ser enviado para este *Worker*, não como uma chamada de função mas sim através de uma mensagem *string*, seguindo o protocolo UCI (NMRUGG, 2020). A resposta do motor, por ser de natureza assíncrona, também será dada por uma mensagem *string* enviada através de uma função *callback* (função que será chamada quando o processamento solicitado terminar).

Mais informações sobre o StockFish.js podem ser encontradas em sua página no GitHub (<https://github.com/nmrugg/stockfish.js>).

4 HARDWARE

Neste capítulo será realizado um breve comentário sobre a plataforma utilizada para o desenvolvimento da aplicação, como também os principais componentes da estrutura física do tabuleiro responsável por realizar a movimentação das peças.

4.1 RASPBERRY PI

O Raspberry Pi é um microcomputador que roda distribuições Linux como o Raspbian e Ubuntu. Possuindo o tamanho de um cartão de crédito, ele consegue prover recursos comparáveis a um smartphone, mantendo tudo a um baixo custo (FOUNDATION, 2020). Este tipo de produto é bastante usado para projetos eletrônicos, videogames retrô, servidores de baixo custo (arquivo, mídia etc.), entre muitos outros.

No Quadro 3 estão os principais modelos do Raspberry PI para comparação (FOUNDATION, 2020):

Quadro 3 – Revisões do Raspberry Pi.

Produto	Processador	Clock	RAM	USB	Ethernet	Wireless
Rasp Pi 1 A+	BCM2835	700MHz	512MB	1	-	-
Rasp Pi 1 B+	BCM2835	700MHz	512MB	4	100Base-T	-
Rasp Pi 2 B	BCM2836	900MHz	1GB	4	100Base-T	-
Rasp Pi 3 B	BCM2837	1200MHz	1GB	4	100Base-T	802.11n
Rasp Pi 3 A+	BCM2837	1400MHz	512MB	1	-	802.11ac
Rasp Pi 3 B+	BCM2837	1400MHz	1GB	4	1000Base-T	802.11ac
Rasp Pi 4 B	BCM2711	1500MHz	2GB	2xUSB2, 2xUSB3	1000Base-T	802.11ac
Rasp Pi 4 B	BCM2711	1500MHz	4GB	2x2.0, 2x3.0	1000Base-T	802.11ac
Rasp Pi 4 B	BCM2711	1500MHz	8GB	2x2.0, 2x3.0	1000Base-T	802.11ac

Para este trabalho, foi selecionada a versão *Rasp Pi 3 B+* (Figura 21). Ela é uma versão um pouco mais antiga (2018) (FOUNDATION, 2020), porém possui poder de processamento suficiente para esta aplicação. A seguir, está a especificação completa do produto (FILIPEFLOP, 2020):

- Processador Broadcom BCM2837B0 64bits ARM Cortex-A53 Quad-Core
- Clock de 1.4 GHz
- Memória RAM: 1GB
- Adaptador Wifi 802.11 b/g/n/AC 2.4GHz e 5GHz integrado
- Bluetooth 4.2 BLE integrado

- Conector de vídeo HDMI
- 4 portas USB 2.0
- Conector Gigabit Ethernet over USB 2.0 (throughput máximo de 300 Mbps)
- Alimentação: fonte DC chaveada 5V 3A
- Interface para câmera (CSI)
- Interface para display (DSI)
- Slot para cartão microSD
- Conector de áudio e vídeo
- GPIO de 40 pinos
- Certificado de homologação Anatel: 01598-18-10629
- Dimensões: 85 x 56 x 17mm

Figura 21 – Raspberry PI 3 b+

Fonte: FILIPEFLOP (2020)

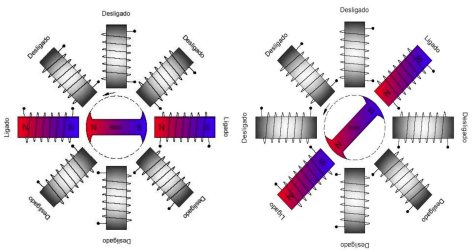
4.2 MOTORES DE PASSO

Os motores elétricos são dispositivos muito conhecidos hoje em dia. Eles conseguem transformar a energia elétrica em movimento rotativo, por meio de ímãs e indutores em seu interior. E, dentro da classe de motores elétricos, existem diversas categorias diferentes, tais como: motores de corrente contínua, corrente alternada, universais (funcionam com ambas), servomotores e motores de passo (PATSKO, 2006).

Os motores de passo funcionam não por permitir livremente o giro do rotor, mas por realizar este giro passo a passo. No seu interior, os estatores (parte de um motor ou gerador elétrico que se mantém fixa à carcaça) contém bobinas elétricas que, quando percorridos por uma corrente elétrica, viram um eletroímã, gerando a força necessária para o movimento do rotor que é embutido por ímãs permanentes (PATSKO, 2006). Esse tipo de motor garante muita precisão no seu movimento e são essenciais em aplicações como impressoras, drives de cd/dvd, braços robóticos, impressoras 3d etc., situações nas quais o controle da movimentação deve ser muito preciso.

Para este motor funcionar, no entanto, não basta apenas ligar as bobinas na fonte de energia. Caso isso seja feito, não só ele não irá se mover, como ficará travado no lugar. Para dar movimento ao rotor, é necessário ligar cada conjunto/par de bobinas em ordem, um de cada vez. A cada par de bobinas que é magnetizada, ocorre um passo no eixo do motor (PATSKO, 2006).

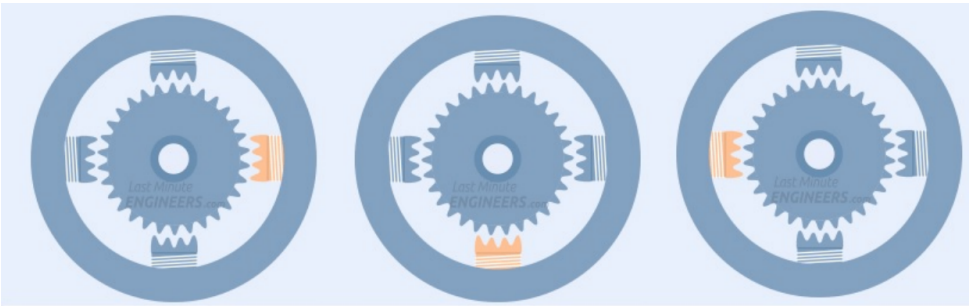
Figura 22 – Diagrama de um motor de passo



Fonte: PATSKO (2006)

Na Figura 22 é possível ver o diagrama para um motor de passo com um estator de 4 pares e um rotor com 1 polo, gerando uma precisão de 45 graus de rotação. Porém, enquanto este exemplo é bom para entender o conceito, na prática, os motores de passo são mais parecidos com a Figura 23.

Figura 23 – Motor de passo mais próximo dos encontrados hoje



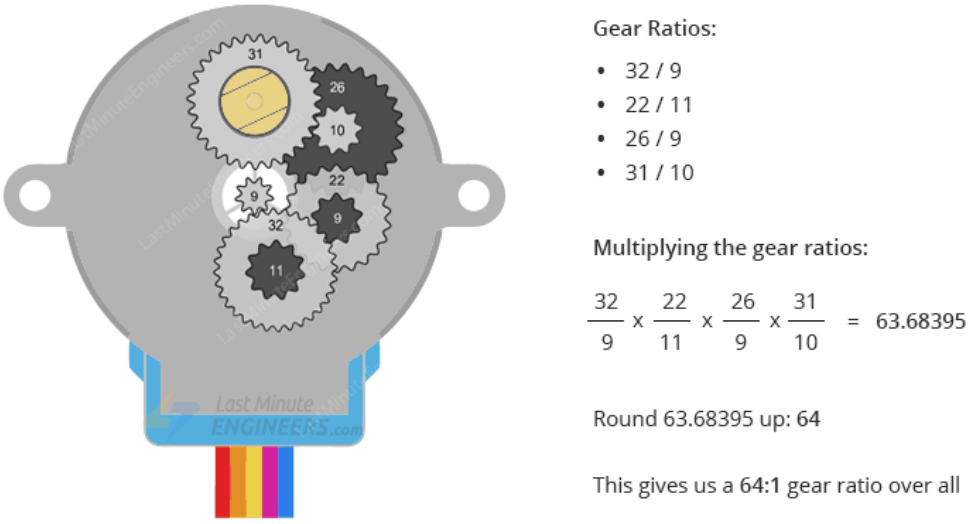
Fonte: ENGINEERS (2020)

Um detalhe importante a se notar neste tipo de motor dentado é que os dentes do rotor e do estator não são alinháveis. Pelo contrário, quando uma bobina é ligada e o rotor se ajusta a

ela, todas as outras ficam "desalinhadas". Ele é projetado desta forma porque é justamente este desalinhamento que permite o movimento do rotor durante a troca das bobinas (ENGINEERS, 2020).

Tomando por exemplo o motor da Figura 23, ele possui 32 dentes e 4 eletroímãs para rotacionar um passo de cada vez, gerando uma resolução de 11.25°, ou 32 passos. Entretanto, é possível reduzir este número ainda mais ao utilizar outra técnica conhecida como "caixa de redução", exemplificada na Figura 24.

Figura 24 – Caixa de redução



Fonte: ENGINEERS (2020)

Acoplada diretamente ao eixo do motor, ela reduz a rotação do rotor em aproximadamente 64 vezes. Isto quer dizer que além de um torque muito maior, o motor possui uma resolução de saída de aproximadamente 2038 passos (ENGINEERS, 2020).

Outra técnica possível de ser empregada para aumentar a resolução da rotação é a técnica chamada de "micro passo". O micro passo é realizado pelo controlador de modulação de largura de pulso, equipamento que recebe o sinal do controlador (Raspberry por exemplo) e ativa as respectivas bobinas do motor. Este controlador irá, portanto, ativar duas bobinas ou mais ao mesmo tempo, conduzindo o rotor para parar no "meio" do passo. Com esta técnica, é possível atingir uma resolução teoricamente infinita com o motor (PATSKO, 2006).

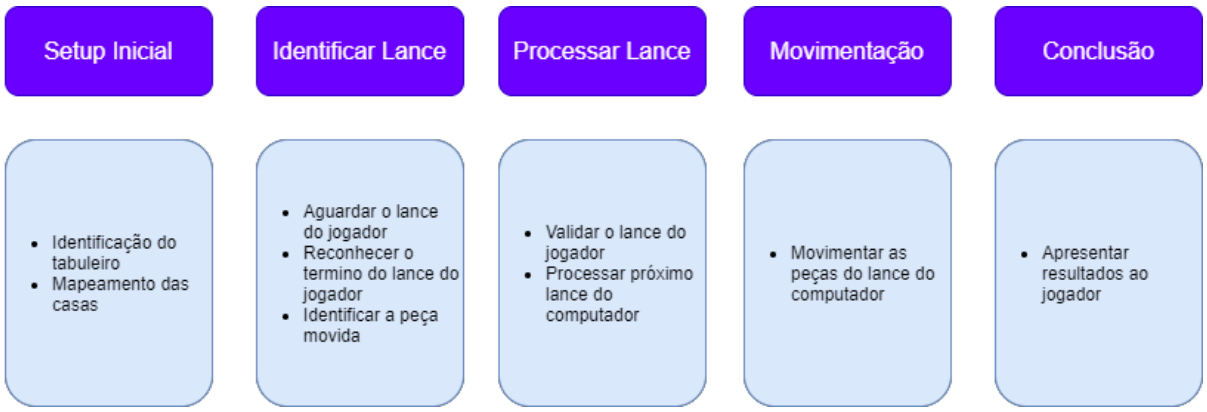
5 APLICAÇÃO

A aplicação desenvolvida segue três etapas distintas no decorrer da partida: identificar a jogada do adversário, processar a jogada do computador e realizar o movimento das peças. Este ciclo de leitura-processamento-movimentação se repete até a partida ser encerrada, seja por xeque-mate, empate ou mesmo desistência do jogador.

Para preparar a execução deste ciclo, no entanto, é necessário realizar um passo inicial antes de iniciar a partida que é localizar o tabuleiro e suas casas. Este passo irá extrair o tabuleiro da imagem, permitindo o software trabalhar mais facilmente com os elementos de interesse da imagem, ou seja, suas peças. Já no final da partida, o jogo irá entrar na fase de conclusão, na qual o jogo informa o ganhador e finaliza a execução da aplicação.

Cada fase do jogo pode ser mapeada de acordo com a Figura 25:

Figura 25 – Etapas do software



Fonte: O Autor

Para realizar tudo isso, foi criado um tabuleiro físico especificamente para este trabalho. Ele foi construído em madeira e utiliza dois motores de passo, cada um responsável por movimentar seu respectivo eixo e juntos posicionar um eletroímã imediatamente abaixo da peça de xadrez. A captura de movimentos é realizada com o auxílio de uma câmera posicionada ligeiramente acima do tabuleiro e uma lâmpada led para iluminar a cena (Figura 26).

A seguir será explicado como o algoritmo do jogo funciona, e ao final uma análise do desempenho e confiabilidade do projeto como um todo, assim como uma seção com as limitações encontradas desta implementação.

Figura 26 – Foto do projeto final

Fonte: O Autor

5.1 DETECÇÃO DO TABULEIRO

A detecção do tabuleiro é realizada seguindo o algoritmo de detecção de objetos abordado no capítulo de visão computacional (HANOğLU, 2020). A diferença entre o sugerido pelo autor e o utilizado aqui é que ele foi adaptado para localizar exclusivamente o contorno do tabuleiro e apenas ele neste momento. Será descartado qualquer contorno que não configure um polígono fechado de 4 lados (o retângulo externo do tabuleiro), que não seja um contorno raiz e que não represente uma área mínima na tela (para descartar as casas das peças). Desta forma, qualquer desalinhamento da câmera ou do próprio vidro com o centro do tabuleiro pode ser corrigido durante a inicialização do software.

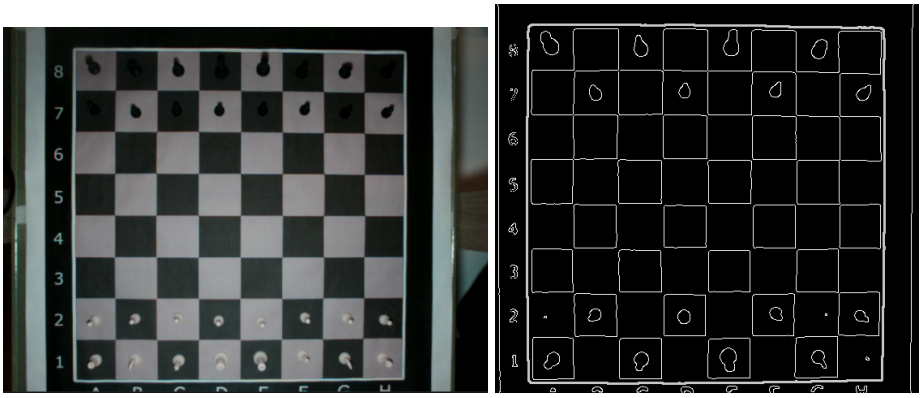
A Figura 27a contém a imagem original da câmera, sem cortes ou tratamento. Já a Figura 27b é o resultado deste primeiro processo, em que o tabuleiro detectado está em destaque no *frame*.

Para detectar a posição de cada casa foi realizado um simples mapeamento para sua respectiva região da imagem, utilizando uma proporção estimada de altura e largura das casas previamente calculadas (Figura 28). Este método é bastante sensível a distorção de captura nas bordas do tabuleiro, porém como as peças do jogo são muito menores que a casa, a precisão atingida com este método demonstrou ser suficiente para esta aplicação.

5.2 JOGADA DO USUÁRIO

A detecção da jogada do usuário é realizada em três momentos distintos: a "entrada" da mão na imagem, sinal de que o adversário está para mover uma peça; a "saída" dela da imagem indicando que a jogada pode ter sido realizada e a detecção das partes alteradas da imagem, localizando a peça alterada, caso houver.

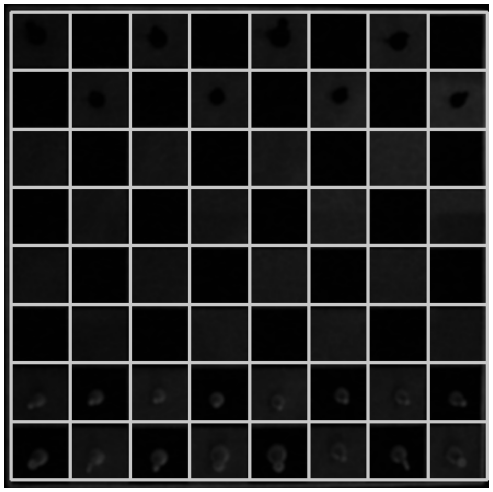
Figura 27 – Etapas do mapeamento



(a) Frame original (b) Frame processado

Fonte: O Autor

Figura 28 – Mapeamento das casas do tabuleiro



Fonte: O Autor

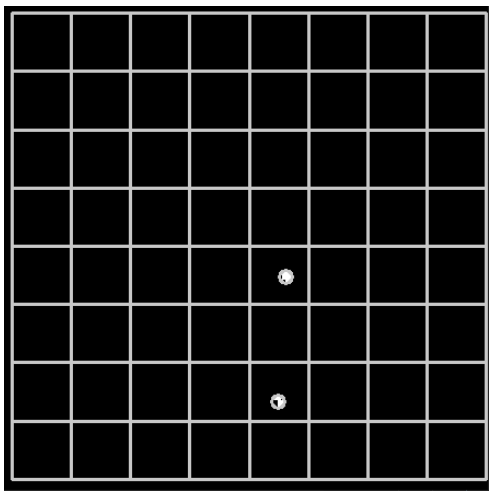
Como a câmera está posicionada de forma fixa, isto possibilita a utilização de uma técnica chamada subtração de fundo. A subtração de fundo utiliza uma imagem estática como referência do tabuleiro e, a partir dela, subtrai seus *pixels* pelos *pixels* de um novo *frame*. Com isto, caso as duas imagens sejam iguais ou muito próximas, o resultado será uma nova imagem predominantemente preta (valor 0). Caso contrário, será possível obter o valor da diferença ("branco") exatamente no local onde ocorreu a mudança (ROSEBROCK, 2020). Para isolar e suprimir ruídos da operação, a subtração é utilizada em conjunto com um filtro de borramento, uma operação de *threshold* binário e de fechamento.

Para identificar a jogada, o software utiliza a porcentagem de branco em relação ao preto na diferença entre os *frames*. Quando esta porcentagem passar de um certo limiar, devido a mão do usuário acima do tabuleiro, será identificado como "jogando". E, quando ela diminuir de um segundo limiar devido a retirada da mão, será considerado "fim de jogada".

Com a jogada terminada, e caso tudo tenha ocorrido com sucesso, o resultado da subtração mostrará dois pontos distintos, um deles é onde está a peça movida e o outro onde estava antes do movimento. Aqui, a operação de contornos é utilizada mais uma vez, agora para selecionar e extrair o centro de massa dos dois maiores círculos da imagem com a função *minEnclosingCircle*. Esta localização é então traduzida na notação do xadrez e encaminhada para a IA.

A Figura 29 representa a detecção do movimento 'e2e4' do xadrez. Nesta figura, os pontos encontrados foram destacados com pequenos círculos ao redor.

Figura 29 – Detecção da jogada



Fonte: O Autor

No entanto, nem sempre a detecção funciona corretamente. Para o tratamento de exceções de detecção do software, foram utilizadas as seguintes estratégias:

1. Caso a detecção fique "presa" no estágio 2, esperando a diferença da subtração diminuir, por mais de 30 segundos, o *frame* de referência é considerado inválido e o processo é reiniciado.

2. Caso não seja detectado nenhum ponto na subtração, o algoritmo apenas reinicia a detecção com um novo *frame* de referência;

3. Caso seja detectado apenas 1 ponto, o jogo identifica como uma remoção/reposicionamento de peça, ignora o movimento e reinicia a detecção com um novo *frame* de referência;

4. Caso sejam 3 ou mais pontos é identificado imediatamente como uma jogada inválida (ou falha da detecção) e é solicitado ao usuário para desfazer o movimento, ignorando a próxima detecção inteira.

5.3 PROCESSAMENTO DA JOGADA DO COMPUTADOR

Para o processamento da jogada, foram implementadas duas abordagens diferentes: utilizar o motor de xadrez StockFish.js e implementar um algoritmo MinMax próprio.

É também neste momento que a jogada do usuário é validada. O passo de detecção consegue apenas identificar as casas alteradas, delegando ao jogo procurar algum movimento válido com essas duas casas e prosseguir com ele. Caso o movimento seja inválido, o usuário será solicitado a desfazê-lo.

5.3.1 Implementando com o StockFish

A implementação com o StockFish.js será utilizada para as partidas sérias contra o tabuleiro. Ele oferece um desempenho consistente e confiável no decorrer da partida, além de vários níveis de dificuldade.

A comunicação com o Stockfish.js se dá através de *Workers*. Apesar de sua utilização não ser exatamente igual ao *multithreads* de outras linguagens, como em C, Java ou Python, explicar o funcionamento dos *Workers* do JavaScript está fora do escopo deste trabalho. Dito isto, esta comunicação com o Stockfish.js não é complexa. Será enviado o comando UCI (KAHLEN, 2004) para o motor juntamente com uma função de *callback*. Quando o motor terminar o processamento este chamará a função *callback* e sinalizará o programa principal para realizar a tratativa adequada.

5.3.2 Implementando o MinMax

Para fins didáticos deste trabalho também foi criada uma implementação própria do algoritmo MinMax de xadrez. Este algoritmo não tem o objetivo de se tornar uma inteligência artificial particularmente forte no jogo, ou mesmo rápida e otimizada, mas sim ser capaz de realizar uma partida completa, de configurar e/ou desativar recursos específicos e registrar os seus tempos de execução em um log de dados para análise. No Capítulo 6 serão apresentadas as estatísticas de tempo de execução da IA, assim como os resultados de colocar uma contra a outra.

A implementação do MinMax foi escrita puramente em JavaScript e baseada na literatura apresentada até agora, envolvendo o MinMax, alpha-beta pruning e tabelas de transposição. Utiliza a mesma forma de comunicação que o StockFish.js, trabalhando com comandos UCI via *Workers* (apenas os comandos utilizados pela aplicação foram implementados) e, portanto, pode acoplar-se ao módulo de xadrez do jogo e substituir o StockFish.js durante sua execução.

Como a implementação é idêntica à apresentada nos capítulos anteriores, os únicos pontos abordados serão as divergências das técnicas e as métricas utilizadas no algoritmo.

Para avaliar o tabuleiro, foram utilizadas as seguintes métricas:

1. Pontuação de cada peça:
    a) Peão: 198 pontos
    b) Cavalo: 817 pontos
    c) Bispo: 836 pontos
    d) Torre: 1270 pontos
    e) Dama: 2521 pontos
    f) Rei: 200000 pontos
2. Mobilidade: 10 pontos
3. Xeque: 3000 pontos

Para implementar as tabelas de transposição foi utilizada uma biblioteca chamada "lru-cache". Esta biblioteca de cache, como o nome sugere, utiliza a política de substituição *Least recently used* (LRU). Esta política também é conhecida como "primeiro a entrar, primeiro a sair", ou seja, ao inserir uma nova entrada no cache as entradas mais antigas serão descartadas primeiro.

A chave de identificação do estado do tabuleiro é gerada a partir de uma função hash inspirada no *hashcode* da classe *String* do Java, adaptada à necessidade deste projeto. Ela leva em conta o estado do tabuleiro e do jogo (possibilidade de roque), gerando um número de 32 bits ao final da execução. Os dados armazenados no cache são os movimentos possíveis deste nó e a verificação de xeques/xeque-mate, visto que são as operações mais custosas deste algoritmo.

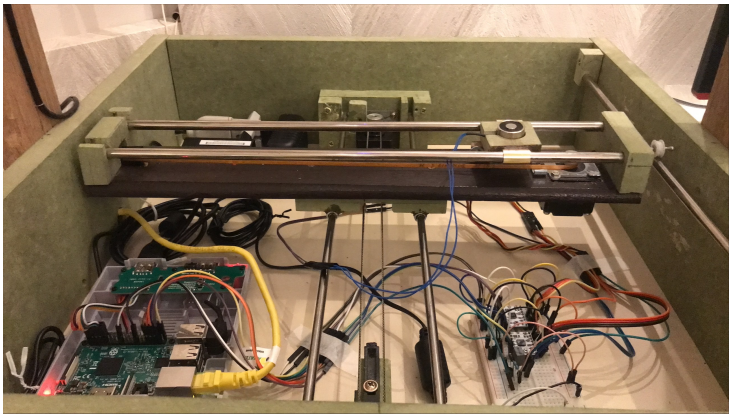
5.4 TABULEIRO E MOVIMENTAÇÃO DAS PEÇAS

Para esta etapa, foi construído um tabuleiro físico especialmente para o projeto. Em seu interior ele possui dois eixos perpendiculares, X e Y respectivamente, responsáveis por movimentar o eletroímã por toda a área do tabuleiro (Figura 30).

Acima dele, posicionado sobre a borda da caixa, está um painel de vidro com 2mm de espessura, o qual é a superfície do tabuleiro. Foi escolhido um painel de vidro pois ao mesmo tempo que é altamente rígido, também é bastante fino e não interfere com as propriedades magnéticas do eletroímã. Já o tabuleiro em si foi impresso em papel sulfite e posicionado no lado de cima do vidro, mostrando ser o melhor material para difundir o reflexo da luz da lâmpada na câmera. Por último, todas as peças de xadrez possuem um pequeno ímã em sua base, o qual fornece uma força extra na hora da movimentação (Figura 31).

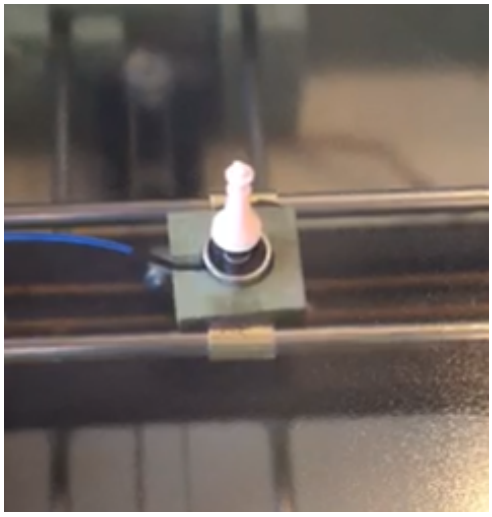
O código que controla os motores foi desenvolvido para permitir que cada motor seja controlado de forma individual ou conjunta, podendo ainda variar a velocidade de movimento

Figura 30 – Estrutura dentro do tabuleiro



Fonte: O Autor

Figura 31 – Tabuleiro com apenas o vidro



Fonte: O Autor

ao longo do trajeto. Isto permite chegar o mais rápido possível na peça e carregá-la lentamente. Para fazer a ligação do Raspberry com o motor de passo foi utilizado o *driver A4988*.

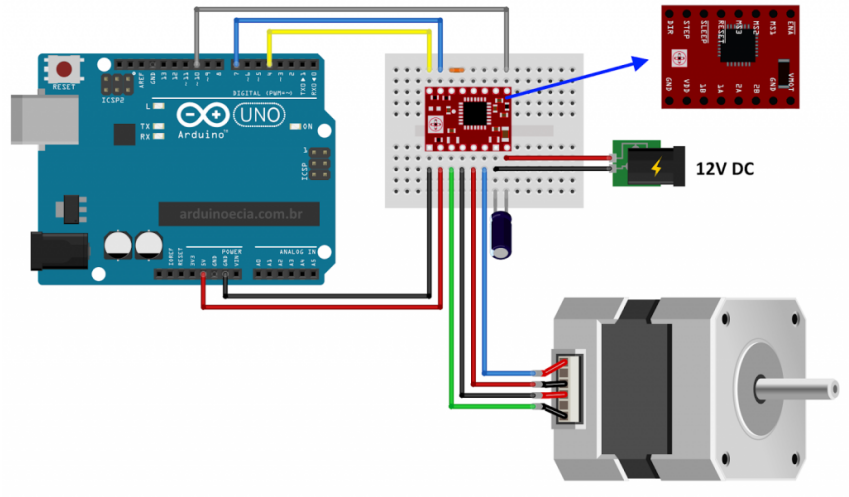
A Figura 32 contém um diagrama da ligação elétrica de um Arduino Uno com o motor de passo. Neste trabalho, no entanto, o Raspberry Pi foi conectado diretamente aos *drivers* (um driver para cada motor), sem necessitar a utilização do Arduíno.

Para evitar a colisão de duas peças o algoritmo de rotas trabalha de duas formas distintas:

1. Caso o caminho reto esteja livre (vertical e horizontal), a peça é movida diretamente à casa alvo.

2. Caso o caminho seja na diagonal (bispos ou captura do peão) ou na movimentação do cavalo, a peça é deslizada pelos vértices das casas.

Figura 32 – Diagrama da ligação elétrica do motor de passo



Fonte: CIA (2015)

6 RESULTADOS

Neste capítulo serão, inicialmente, apresentadas as limitações encontradas com as ferramentas e materiais utilizados a fim de esclarecer algumas decisões de projeto e depois mostrar os resultados obtidos com esse *design* durante o processo de 5 partidas completas de xadrez. A avaliação destas partidas irá abordar as fases de detecção dos movimentos e movimentação das peças, estudando sua taxa de confiabilidade, assim como a integração dos módulos do sistema.

Para avaliar a IA desenvolvida, no entanto, foi criado um programa à parte para colocá-la contra o stockfish.js e utilizado o perfilador do VisualStudio Code para verificar os tempos de execução de cada parte de sua implementação. O objetivo é localizar pontos fracos dessa construção, os quais podem ser trabalhados ou mesmo evitados em uma implementação posterior.

6.1 LIMITAÇÕES DO PROJETO

Nesta sessão estão as limitações encontradas no desenvolvimento do projeto, seja pela abordagem escolhida ou mesmo das ferramentas utilizadas.

6.1.1 Suporte à configuração da câmera pelo OpenCV

Para a detecção de movimentos do jogador, é necessário que exista um certo contraste entre todas as peças (brancas e pretas) e as casas do tabuleiro. Para isto, o controle de iluminação é fundamental. Um *frame* muito claro vai esconder as peças brancas, enquanto um muito escuro vai esconder as pretas.

Durante a captura da imagem, o OpenCV fornece diversas configurações como tamanho de *buffer*, contraste, brilho, saturação etc. (OPENCV, 2021). Dentre estas configurações, duas delas muito importantes para este projeto são a exposição do sensor e o ganho de iluminação, os quais são responsáveis por controlar a iluminação da imagem capturada.

É possível configurar e persistir esta configuração de imagem no sistema operacional com ajuda de outros aplicativos, como o *GUVCVIEW*, porém o OpenCV utilizado pelo *opencv4nodejs* não dá suporte a esta configuração. Inicialmente, a etapa de *setup* seria também responsável por testar e escolher a configuração adequada de exposição e ganho, porém com esta limitação é necessário desativar o auto ganho da câmera e calibrá-la manualmente antes de executar o projeto.

6.1.2 Detecção de jogadas inválidas

Devido à natureza do método de detecção de movimento utilizado neste projeto, é possível dizer apenas as casas que sofreram alterações durante a detecção. Essas alterações podem

ser tanto uma jogada legal, um movimento inválido do jogador, uma remoção da peça do tabuleiro ou mesmo um movimento acidental na peça (o usuário pegou a, peça mas desistiu do movimento).

Durante o desenvolvimento da aplicação foi evidenciado que não é possível distinguir com acuracidade os casos acima, utilizando apenas o método de subtração de fundo. Este problema, por exemplo, impede a implementação de mecanismos para corrigir uma jogada inválida automaticamente, pois o algoritmo não sabe a localização atual da peça.

Por causa dessa ambiguidade, foi optado por deixar a correção das jogadas por parte do usuário e apenas informar que a jogada é inválida. Caso o tabuleiro esteja conectado a um monitor, é possível também mostrar o estado esperado do tabuleiro para o usuário.

6.2 VALIDAÇÃO DO TABULEIRO

Para a validação dos módulos desenvolvidos como um sistema, e a integração deles entre si, foram realizados 5 jogos completos contra o computador. Nestes 5 jogos, foram anotados o número de lances realizados e o número de falhas detectadas durante o jogo. Os resultados estão na Tabela 1, dos quais o primeiro e quinto jogo foram feitas pelo próprio autor, enquanto as jogadas 2, 3 e 4 foram feitas por uma segunda pessoa que auxiliou nos testes.

Tabela 1 – Resultados dos jogos contra o tabuleiro físico

Jogada	Lances	Iluminação	FG	FD	FP	FE
1	27	Ambiente (solar)	7	0	0	4
2	15	Ambiente (solar)	2	0	0	0
3	19	Apenas Interna	6	1	0	1
4	54	Apenas Interna	9	2	0	11
5	31	Apenas Interna	0	0	0	8

Fonte: O Autor (2021).

FG: Falha no gatilho de detecção da "entrada/saída" da mão na imagem

FD: Falha grave na detecção de movimento que interrompeu o processo do jogo

FP: Falha na detecção da posição correta da peça jogada

FE: Falha do eletroímã na hora de carregar a peça

6.2.1 Detecção das jogadas

A detecção das jogadas ocorreu sem problemas aparentes. Em todas as jogadas não houve falsas detecções e sempre foram retornadas as posições corretas das peças jogadas, seja com iluminação ambiente (externa) ou com iluminação controlada (interna). Para suprimir as sombras das peças geradas pela iluminação em ambos os testes foi utilizada uma lâmpada led.

O sucesso foi, em grande parte, devido à superfície de papel. Embora ele gere mais atrito na hora da movimentação, foi o melhor dentre os materiais testados no quesito difusão de luz, permitindo a utilização da lâmpada e ao mesmo tempo uma captura adequada da imagem. Os materiais testados foram a própria superfície de vidro (vidro antirreflexo), papel fotográfico e superfície plástica.

O único problema encontrado foi em relação ao gatilho de detecção, o qual gerou duas situações distintas. A primeira é quando a jogada é feita, mas o gatilho de "entrada" não é disparado, gerando a necessidade de "recolocar" a mão na frente da câmera para ativá-lo. O segundo, mais grave, é o gatilho de "saída" disparar mesmo com a mão na frente da câmera, geralmente nos movimentos mais perto das bordas do tabuleiro ou ao tentar rapidamente reposicionar melhor a peça ao final da jogada, causando uma comparação errada e necessitando o *reset* forçado da detecção. Porém, mesmo nestes casos mais graves, foi possível retomar o jogo após o algoritmo de detecção expirar o *frame* de referência.

Ambos os casos foram resolvidos na quinta jogada, quando os limiares de entrada e saída foram alterados de, respectivamente, 7% e 3% para 4% e 1%. Tomando o devido cuidado na hora da jogada e com essas modificações, estes problemas não voltaram a acontecer.

6.2.2 Movimentação das peças

Durante os testes, em nenhum momento ocorreram problemas com os motores de passo, com o algoritmo de rotas ou mesmo a parte de hardware de ligação elétrica, todos funcionaram perfeitamente durante a duração da partida. A proximidade de uma peça em movimento com outras peças também não representou problemas, onde nenhuma das peças foi pega por engano, por mais perto que tenham passado uma da outra.

Apesar do sucesso da execução do movimento, o eletroímã muitas vezes não conseguiu gerar atração suficiente na peça para movê-la por todo o trajeto e acabou perdendo-a no meio do caminho. Ao investigar a localização em que elas ficaram para trás, geralmente ocorreram em locais onde o papel estava mais enrugado devido a umidade do ambiente. Mesmo ao aproximar o eletroímã o máximo possível do vidro, o problema continuou acontecendo.

Foram utilizados ímãs de ferrite, que possuem baixa atração entre si, em conjunto com um pequeno eletroímã de 12 volts para o conjunto das peças. Não foi possível fazer a substituição do ímã nas peças, mas possivelmente um ímã um pouco maior resolveria o problema.

6.3 VALIDAÇÃO DA IA

Para a validação da IA de xadrez desenvolvida neste trabalho, foi criado um programa à parte do software principal que é responsável por chamá-la em sucessão contra o stockfish.js, até que um dos dois obtenha o xeque-mate. Durante a execução do programa, é monitorado os tempos de resposta de cada lance da IA e ao final é obtido o tempo total de execução, a média,

o número de lances e o histórico do jogo para análise. Os tempos de execução de cada função da IA foram obtidos através do perfilador JavaScript do VisualStudio Code.

Foram executadas 6 sessões de testes neste algoritmo, variando as configurações de profundidade da árvore e tamanho do cache utilizado, com cada sessão contendo a execução de 3 testes idênticos, totalizando 18 execuções ao todo. Em todos os testes a configuração do stockfish.js foi mantida igual para não influenciar na acuracidade da coleta.

Na Tabela 2 está a síntese destes dados, contendo a configuração utilizada, a média de lances executados e tempos de processamento. Os dados foram gerados a partir de um computador com as seguintes características: **Processador:** Intel Core 5 7600k; **Memória**: 16GB de RAM; Linux Ubuntu WSL 2.

Tabela 2 – Estatísticas dos jogos contra o tabuleiro físico

Sessão	Profundidade	Cache	Lances	Tempo médio IA	Tempo IA	Tempo Total
1	3	5000	15 18	674+-51ms	10753+-738ms	11075ms
2	3	100.000	15	771+-48ms	12278+-512ms	12659ms
3	3	200.000	15 18	829+-64ms	13178+-126ms	13493ms
4	4	5000	23 24	8036+-456ms	187297+-7033ms	187742ms
5	4	100.000	23 24	8575+-192ms	202857+-1422ms	203324ms
6	4	200.000	23 24	10604+-890ms	243886+-20485ms	244420ms

Fonte: O Autor (2021).

Tempo médio IA: Tempo médio de cada jogada da IA

Tempo IA: Tempo de execução da IA durante o jogo

Tempo Total: Tempo total do jogo

Na Tabela 3 estão listados, do maior para o menor, as três áreas mais significativas de cada sessão dos mesmos testes anteriores. Os dados são a média de tempo por lance.

Tabela 3 – Perfilação da sessão de jogos

Sessão	expansão do movimento	verificação do xeque	garbage collector
1	222,37+-18,02ms	68,57+-22,42ms	210.93+-17,47ms
2	236,77+-13,45ms	72,04+-9,4ms	334,37+-57,15ms
3	229,14+-79,54ms	71,70+-21,08ms	353,74+-48,05ms
4	3245+-50,94ms	2404+-52,97ms	2329+-424 ms
5	2202+-52,82 ms	1725+-35,22ms	4138+-165 ms
6	2276+-108 ms	1793+-45 ms	5954+-1167 ms

Fonte: O Autor (2021).

(garbage collector): Função interna do NodeJS, responsável por reciclar as áreas de memória alocada

6.3.1 Resultados da IA

Os resultados gerados a partir deste experimento apontam para uma IA de xadrez totalmente funcional, gerando e interpretando todos os movimentos do xadrez, inclusive movimentos especiais como o roque e o *en passant*. Porém, para torná-la pronta para uso prático faz-se necessário um trabalho extra na implementação, principalmente na parte de avaliação e otimização.

Não foi apresentado nas tabelas anteriores os resultados das partidas, mas em todos os jogos o resultado foi um xeque-mate para o StockFish.js, embora a IA tenha conseguido resistir a um número considerável de lances. O próximo passo para melhorar nesta área seria adicionar outras métricas de avaliação, como por exemplo o melhor posicionamento de cada peça, punição para peças desprotegidas etc.

O grande problema encontrado, no entanto, não foi a dificuldade de adicionar novas funcionalidades à implementação, mas sim de otimizá-la. Apesar de funcional, esta implementação foi insuficiente no quesito tempo de resposta, principalmente comparando ao Stockfish.js, que também utiliza JavaScript como linguagem e ainda assim é capaz de entregar a resposta em uma fração significativa de tempo.

Analisando os resultados obtidos com o perfilador foi possível identificar dois grandes problemas nesta implementação: acesso à memória, mais especificamente ao acessar as casas do tabuleiro, e grande quantidade de alocações dinâmicas ao percorrer os nós da árvore de movimentos.

A estrutura utilizada para armazenar os dados é um vetor de vetores, o qual é recriado a cada novo movimento (nó) da árvore para representar o novo estado do tabuleiro. Esta estrutura, porém, não é muito favorecida pelo JavaScript. O acesso indireto a informação, ou mais especificamente ao dado do vetor filho a partir do vetor pai, parece ser o mais impactante. A função *expandMovePawn* é a mais prejudicada nesse quesito, pois precisa verificar todo o entorno do peão (verificar o *en passant*) e acaba não conseguindo aproveitar o *loop* principal muito bem.

Voltando à árvore de movimentos, o outro problema é a alocação dinâmica gerada neste momento. De acordo com o perfilador, a função mais significativa do algoritmo não é sequer uma função implementada para o jogo, mas sim o *garbage colletor* do próprio JavaScript. *Garbage colletor* é um processo interno de linguagens de alto nível, geralmente orientadas a objetos, que realiza a liberação de memória automaticamente para o usuário, semelhante ao *free()* do C (SILVAW *et al.*, 2021).

Este problema, ao contrário do que era inicialmente suposto, acabou se agravando cada vez mais que o cache era aumentado. Isto indica que caches convencionais não são adequados a esta aplicação, sendo cogitada uma implementação nova específica para este caso.

7 CONSIDERAÇÕES FINAIS

De forma geral, o resultado do projeto foi bastante satisfatório. Apresentou alguns problemas ao longo do caminho que com algum trabalho adicional e materiais mais adequados, possui um potencial de melhoria bastante alto com investimento adicional pequeno.

A detecção dos movimentos foi construída utilizando a ferramenta de contornos do OpenCV como pivô central, tanto para a detecção do tabuleiro quanto de cada movimento da peça. Embora não seja possível reconhecer a peça na imagem com este método, os resultados gerais durante a partida foram excelentes.

O motor de xadrez escolhido foi o StockFish porque na data de criação deste trabalho ele era um dos únicos que possuía um porte para JavaScript, a linguagem escolhida para ele. Esta integração funcionou perfeitamente bem, e ela ainda permite a integração com outros motores de xadrez, como foi o caso da *IA* implementada. Porém, isso não impede a utilização de outros motores que compartilham o protocolo UCI. Durante o desenvolvimento do trabalho foi utilizado o programa **Arena Chess GUI** para inspecionar a comunicação da interface de usuário com o motor e, esta aplicação possui outras dezenas de motores prontas para utilização.

Para movimentar as peças, a ideia inicial era utilizar um braço robótico, porém os custos ficariam muito altos para este projeto. A segunda alternativa foi os motores de passo, bastante utilizados em projetos de máquinas Controle Numérico Computadorizado (CNC) com arduinos, juntamente com um eletroímã para mover a peça. Os resultados foram satisfatórios, com a movimentação funcionando adequadamente, mas deixando um pouco a desejar na hora de levar a peça até sua casa.

7.1 SUGESTÕES DE TRABALHOS FUTUROS

Como trabalhos futuros sugere-se:

- Incrementar as métricas de gatilho utilizadas na detecção de movimentos para torná-la menos suscetível a erros.
- Realizar a identificação das peças não com visão computacional, mas sim com *tags* de *RFID* colocados em cada peça.
- Utilizar outra combinação de eletroímã e peças para melhorar a confiabilidade do movimento ou mesmo trocar a estrutura de motores de passo por um braço robótico (ou outra forma de movimentação).

- Continuar a implementação da IA deste projeto, utilizar outros motores de xadrez para diversificar as configurações ou mesmo conectar o projeto a algum servidor público de jogos de xadrez.

REFERÊNCIAS

BECKER, I. **Manual de Xadrez**. NBL Editora, 2004. ISBN 8521307586,9788521307587. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=b69e910e13017301cceddc5f9d22726a>.

CARVALHO, T. **Espectro Eletromagnético**. 2020. <https://www.infoescola.com/fisica/espectro-eletromagnetico/>. Acesso em: 20 de set. de 2020.

CIA, A. E. **Como usar o driver A4988 com motor de passo Nema 17**. 2015. <https://www.arduinoecia.com.br/driver-a4988-com-motor-de-passo-nema-17/>. Acesso em: 12 de jun. de 2021.

COMMONS, W. **Chaturanga**. 2005. <https://commons.wikimedia.org/wiki/File:Chaturanga.png>. Acesso em: 20 de out. de 2020.

ENGINEERS, L. M. **Control 28BYJ-48 Stepper Motor with ULN2003 Driver & Arduino**. 2020. <https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/>. Acesso em: 19 de out. de 2020.

FILIPEFLOP. **Raspberry Pi 3 Model B+ Anatel**. 2020. <https://www.filipeflop.com/produto/raspberry-pi-3-model-b/>. Acesso em: 27 de set. de 2020.

FOUNDATION, R. P. **R. P. FAQs.** 2020. <https://www.raspberrypi.org/documentation/faqs/>. Acesso em: 27 de set. de 2020.

GIUSTI, P. **Historia Ilustrada Do Xadrez**. Annablume, 2002. Consultado em 14 de setembro de 2020. Disponível em: <https://books.google.com.br/books?id=Cx6Ikto38fUC&printsec=frontcover&hl=pt-BR&source=gbs_atb#v=onepage&q&f=false>.

GONZALEZ, R. C. **Processamento Digital de Imagens**. 3rd. ed. Pearson, 2009. ISBN 978-8576054016. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=bad9f7e77103e98987a2af8a668c24a4>.

HANOğLU, M. **OpenCV ile Dikdörtgen Algılama (Python)**. 2020. <https://mehmethanoglu.com.tr/blog/6-opencv-ile-dikdortgen-algilama-python.html>. Acesso em: 20 de out. de 2020.

HSU, F.-H. **Behind Deep Blue: Building the Computer that Defeated the World Chess Champion**. [S.l.]: Princeton University Press, 2002.

JUSTADUDEWHOHACKS. **opencv4nodejs**. 2020. <https://github.com/justadudewhohacks/opencv4nodejs>. Acesso em: 19 de set. de 2020.

KAEHLER, G. B. A. **Learning OpenCV, 2nd Edition: Computer Vision in C++ with the OpenCV Library**. O'Reilly Media, 2013. ISBN 978-1-44931-465-1. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=d34db9034f737e1ebf18de61bf426796>.

KAHLEN, S.-M. **UCIProtocol**. 2004. <http://wbec-ridderkerk.nl/html/UCIProtocol.html>. Acesso em: 19 de out. de 2020.

LEVITT, G. M. **The Turk, Chess Automaton**. [S.l.]: McFarland & Company, 2000. ISBN 0786429038,9780786429035.

MCOSTALBA. **Stock Fish**. 2020. <https://github.com/official-stockfish/Stockfish>. Acesso em: 19 de out. de 2020.

NMRUGG. **Stockfish.js**. 2020. <https://github.com/nmrugg/stockfish.js>. Acesso em: 19 de out. de 2020.

OPENCV. **Canny Edge Detection**. 2020. <https://docs.opencv.org/master/da/d22/tutorial_py_canny.html>. Acesso em: 25 de out. de 2020.

____. **Contours in OpenCV**. 2020. <https://docs.opencv.org/master/d0/d43/tutorial_js_table_of_contents_contours.html>. Acesso em: 25 de out. de 2020.

____. **Morphological Transformations**. 2020. <https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html>. Acesso em: 25 de out. de 2020.

____. **OpenCV**. 2020. <https://opencv.org/>. Acesso em: 19 de set. de 2020.

____. **OpenCV modules**. 2020. <https://docs.opencv.org/4.4.0>. Acesso em: 19 de set. de 2020.

____. **Flags for video I/O**. 2021. <https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gaeb8dd9c89c10a5c63c139bf7c4f5704d>. Acesso em: 29 de maio de 2021.

PANDOLFINI, B. **Kasparov and Deep Blue**. Original. Fireside, 1997. ISBN 068484852X,9780684848525. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=11a9f9b54f827d4eba1f7f1d7403312e>.

PATSKO, L. F. **TUTORIAL Controle de Motor de Passo**. 2006. <https://www.academia.edu/download/46176773/tutorial_eletronica_-_motor_de_passo.pdf>. Acesso em: 27 de set. de 2020.

POE, E. A. **MAELZEL'S CHESS-PLAYER.** Southern Literary Journal, 1836. Consultado em 14 de setembro de 2020. Disponível em: <https://www.eapoe.org/works/essays/maelzel.htm>.

PROGRAMMING, C. **Stockfish**. 2020. <https://www.chessprogramming.org/Stockfish>. Acesso em: 19 de out. de 2020.

ROSEBROCK, A. **Basic motion detection and tracking with Python and OpenCV**. 2020. <https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/>. Acesso em: 14 de nov. de 2020.

SILVAW *et al.* **Gerenciamento de Memória**. 2021. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Memory_Management>. Acesso em: 12 de jun. de 2021.

STOCKFISH. **StockFish**. 2020. <https://stockfishchess.org/>. Acesso em: 19 de out. de 2020.

WALKER, L. **The Anatomy of a Chess AI**. 2020. <https://medium.com/the-innovation/the-anatomy-of-a-chess-ai-2087d0d565>. Acesso em: 17 de out. de 2020.

ANEXO A – DESCRIÇÃO DO JOGO DE XADREZ

Neste anexo será realizada uma breve explicação sobre o jogo do ponto de vista de um iniciante no xadrez, ensinando o necessário para realizar uma partida completa (suas peças, nomenclaturas relevantes, regras e movimentos básicos).

Para isso, será utilizada formalização trazida por BECKER, que realizou uma compilação de todas as regras e termos do jogo oficial e as traduziu para o português. Mais detalhes sobre as regras do xadrez podem ser verificados em seu livro (BECKER, 2004).

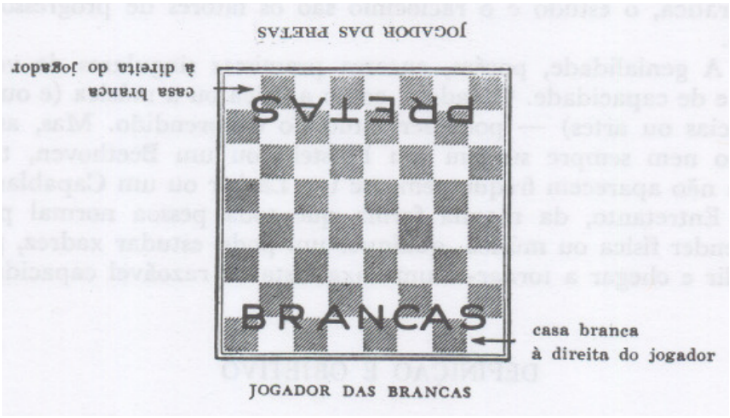
O JOGO

O xadrez é jogado por dois jogadores num tabuleiro quadrado de 64 casas, alternadamente claras e escuras. Cada jogador possui 16 peças (brancas para um e pretas para o outro), cujo objetivo é dar *xeque-mate* no rei adversário.

Tabuleiro

O tabuleiro deve ser posicionado de modo que a casa no canto inferior direito seja branca (Figura 33):

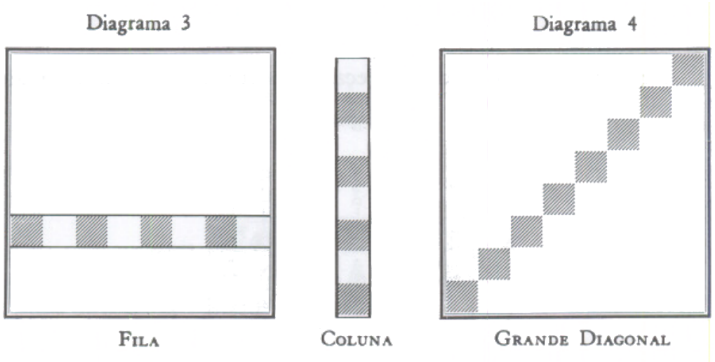
Figura 33 – Tabuleiro



Fonte: BECKER (2004), adaptado

As oito fileiras verticais são chamadas de *colunas*, enquanto as horizontais de *filas*. O conjunto de casas da mesma cor em direção oblíqua é chamado de *diagonal*. A diagonal principal do tabuleiro possui um nome especial de *Grande Diagonal* (Figura 34).

Figura 34 – Fila e Coluna

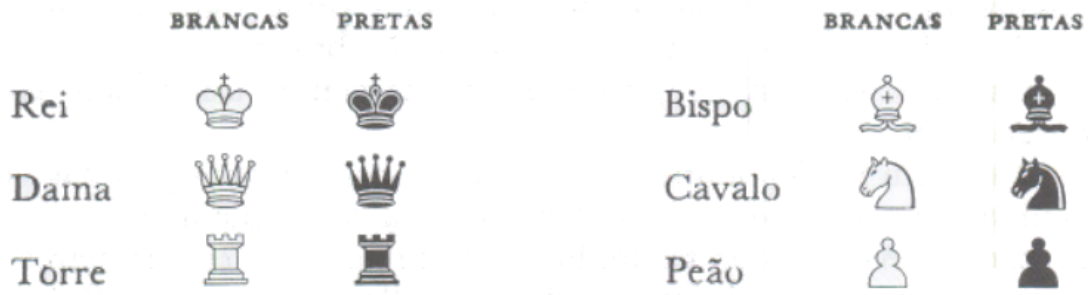


Fonte: BECKER (2004), adaptado

Peças

Cada jogador começa com 16 peças, posicionadas nas duas últimas filas de cada lado do tabuleiro. Estas peças são: um rei, uma dama, duas torres, dois bispos, dois cavalos e oito peões (Figura 35).

Figura 35 – Representação das Peças



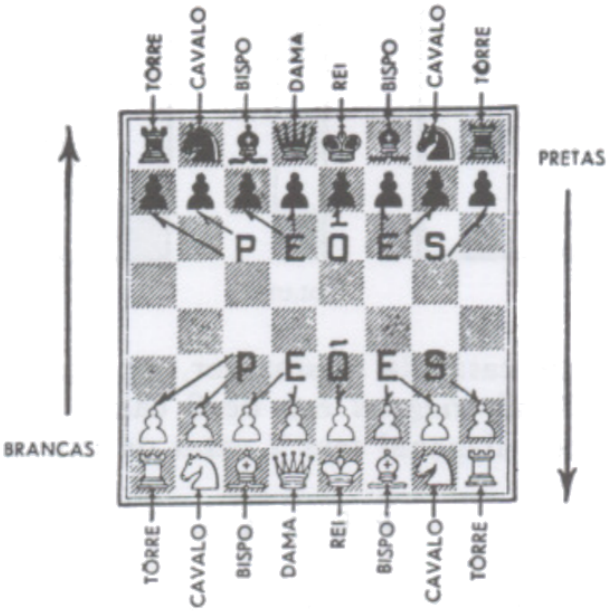
Fonte: BECKER (2004), adaptado

A posição inicial de cada peça deve seguir o padrão da Figura 36. Deve-se observar com cuidado as casas em que as damas estão posicionadas, que devem corresponder a sua respectiva cor (dama branca na casa branca e dama preta na casa preta).

A primeira jogada é sempre realizada pelas brancas. No caso de uma competição com vários jogos entre a mesma dupla, geralmente é decidido via sorteio quem será as brancas no primeiro jogo, rotacionando as cores para os demais.

Pode-se também denominar *lance* para o conjunto de duas jogadas consecutivas: o movimento das brancas e a resposta das pretas. Quando se menciona "os 10 primeiros lances da partida", logo, refere-se "aos 10 primeiros lances completos". Para evitar confusão entre a nomenclatura, duas jogadas consecutivas serão chamadas de *lances completos* no decorrer do trabalho.

Figura 36 – Posição inicial das peças

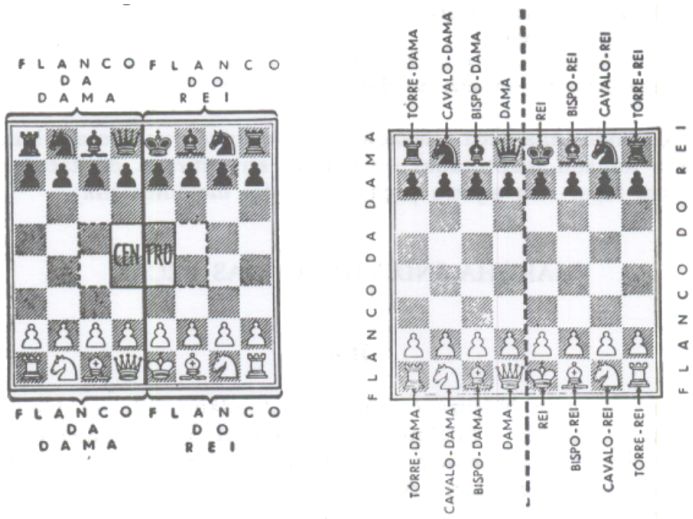


Fonte: BECKER (2004), adaptado

Nomenclatura Geral

Cada metade lateral do tabuleiro é denominada de *flanco*. Cada *flanco* é chamado, respectivamente, *FLANCO DA DAMA* e *FLANCO DO REI* (Figura 37).

Figura 37 – Flancos e nomenclatura das peças



Fonte: BECKER (2004), adaptado

As oito casas centrais do tabuleiro, sendo estas as quatro no centro mais 2 de cada lado, formam o *centro*.

A dama e a torre denominam-se *peças maiores*, enquanto o cavalo e bispo são *peças*

menores. Para identificar cada peça individualmente, dá-se o nome do flanco de sua colocação inicial: *torre do rei*, *cavalo da dama* etc. Já para os peões, referem-se ao mesmo tempo a peça e flanco respectivos, como *peão da torre da dama*.

Movimentação Geral das Peças

Toda peça pode ser jogada apenas em uma casa livre ou ocupada por uma peça inimiga.

Quando uma peça é levada a uma casa ocupada por uma peça inimiga, esta é capturada e removida do jogo.

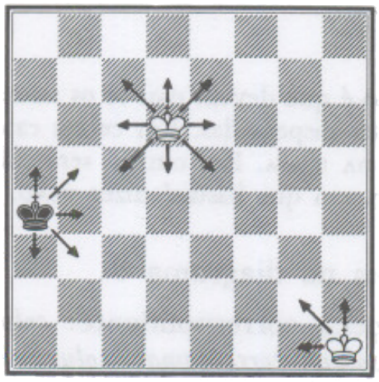
Nenhuma peça pode atravessar casas ocupadas, pulando outra(s) peça(s), com exceção do *cavalo* e *roque* da torre. Esta última será explicada mais para frente.

Movimentação do rei

O *rei* pode se mover para qualquer casa imediatamente ao seu redor. Seu movimento, portanto, se dá de casa em casa, uma de cada vez, em qualquer direção: para cima e para baixo, para os lados e todas as diagonais (Figura 38).

O movimento do rei possui apenas uma exceção: o *roque*.

Figura 38 – Movimentação do rei



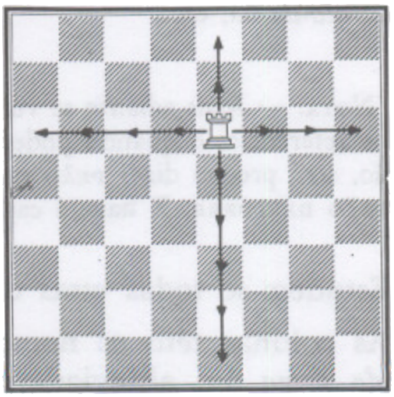
MOVIMENTO DO REI

Fonte: BECKER (2004), adaptado

Movimentação da torre

A *torre* pode se mover para qualquer casa da fila ou coluna em que se encontra. Seu movimento acontece, portanto, em sentido horizontal ou vertical (Figura 39).

Figura 39 – Movimentação da torre



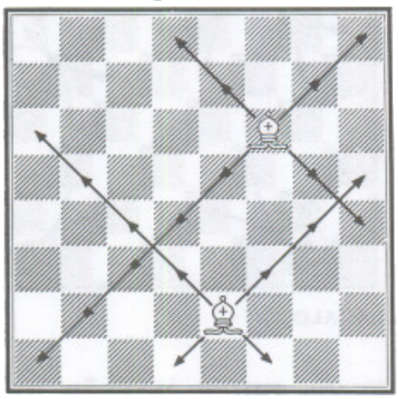
MOVIMENTO DA TORRE

Fonte: BECKER (2004), adaptado

Movimentação do bispo

O *bispo* se movimenta para qualquer casa em sentido oblíquo, ou seja, correndo ao longo das diagonais. Com isso, deve-se notar que cada bispo sempre anda por casas de mesma cor (Figura 40).

Figura 40 – Movimentação do bispo



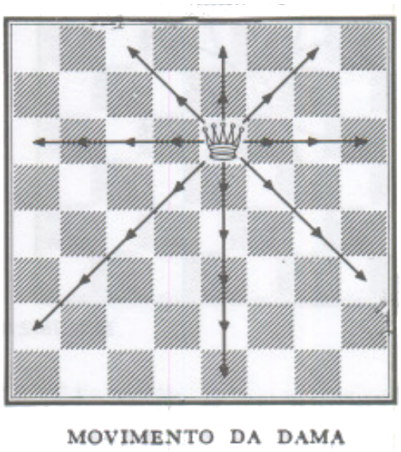
MOVIMENTO DO BISPO

Fonte: BECKER (2004), adaptado

Movimentação da dama

A *dama* combina, simultaneamente, a movimentação da torre e do bispo. Seu movimento, logo, ocorre a partir de qualquer fila, coluna ou diagonal da casa em que se encontra (Figura 41).

Figura 41 – Movimentação da dama



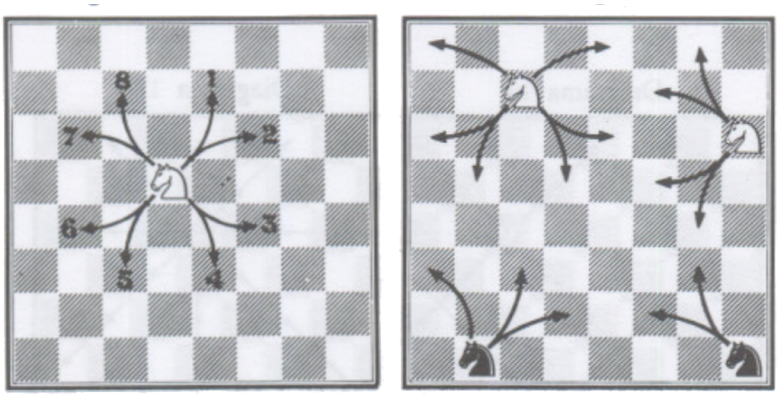
MOVIMENTO DA DAMA

Fonte: BECKER (2004), adaptado

Movimentação do cavalo

O *cavalo* possui um movimento, de forma resumida, parecido com um '**L**' pelo tabuleiro, avançando duas casas à "frente" e outra "para um dos lados". Além disso, ele é a única peça do xadrez que pode pular outras peças, seja ela de sua cor ou da cor contrária (Figura 42).

Figura 42 – Movimentação do cavalo



MOVIMENTO DO CAVALO

Fonte: BECKER (2004), adaptado

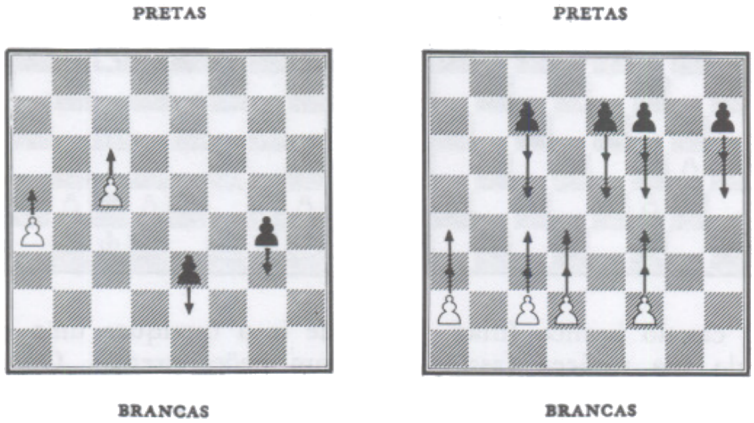
Movimentação do peão

O *peão* é a única peça do xadrez que **não pode retroceder**. Ele avança sempre para a casa imediatamente à sua frente, no sentido vertical, uma casa de cada vez (Figura 43).

A única exceção, porém, é o seu lance inicial. Na sua primeira jogada, ele pode avançar apenas uma casa, como normalmente, ou avançar duas casas. Sempre para a frente.

Outra singularidade do peão é a sua captura. Eles se movem apenas para frente, porém capturam apenas para a sua diagonal.

Figura 43 – Movimentação do peão



Fonte: BECKER (2004), adaptado

Movimentações especiais

A seguir estão os movimentos especiais do xadrez:

Roque

O *roque* é o único movimento do xadrez que pode mover duas peças ao mesmo tempo. Ele permite mover o rei e a torre de forma conjunta, sendo possível apenas caso as seguintes condições sejam atendidas:

- Tanto o rei quanto a torre que vai rocar não podem ter saído de suas casas iniciais, mesmo que tenham saído e voltado para a casa original. Caso a peça tenha se movido, o roque não pode mais ser realizado com ela.
- Não pode haver nenhuma outra peça entre o rei e a torre, seja aliada ou inimiga.
- O rei não pode estar em xeque.
- O rei, durante o movimento, não pode passar por uma casa dominada por uma peça inimiga, ou seja, passar por uma casa onde poderia ser atacado por esta.
- No final do movimento, o rei não pode ficar em xeque.

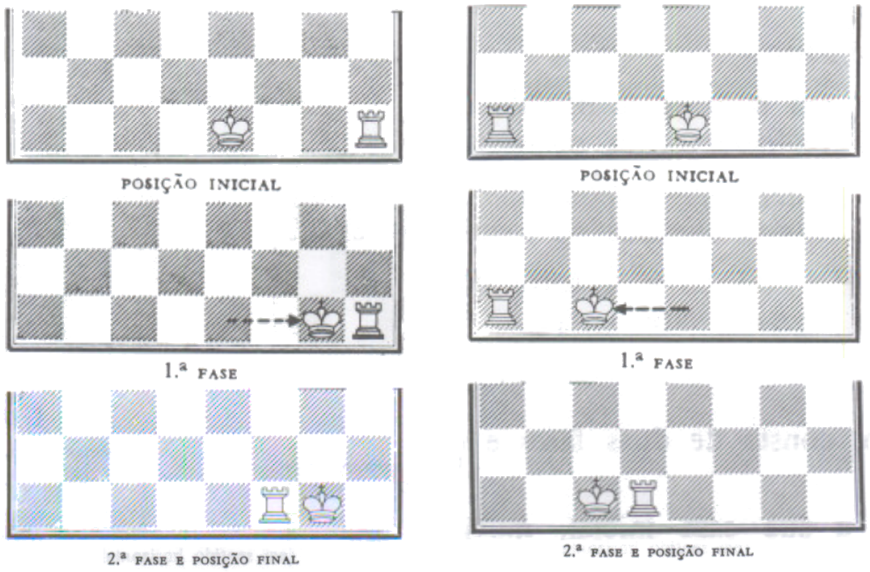
O *roque* efetua-se em duas fases:

1. O rei, de sua casa inicial, movimenta-se duas casas à direita ou esquerda na direção da torre a ser rocada.

2. A torre desloca-se para o lado correspondente, pula por cima do rei, e coloca-se ao lado deste.

Quando o *roque* é realizado no flanco do *rei*, é chamado de *PEQUENO ROQUE*, ou *roque menor* (Figura 44a). Já o realizado do lado oposto, no flanco da *dama*, é chamado de *GRANDE ROQUE*, ou *roque maior* (Figura 44b).

Figura 44 – Passos do roque



(a) Roque Menor (b) Roque Maior

Fonte: BECKER (2004), adaptado

Promoção

No xadrez, a promoção acontece quando um peão chega à oitava fila do tabuleiro. Neste momento, este deve ser substituído por uma peça da mesma cor (dama, torre, bispo ou cavalo), à vontade do jogador.

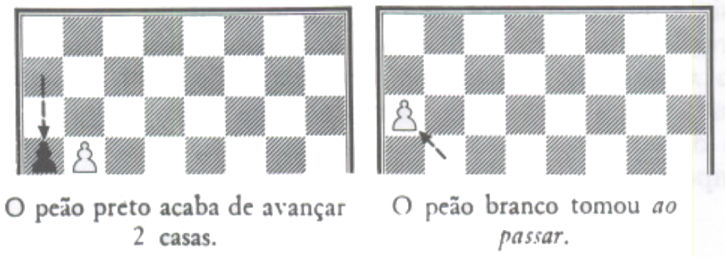
Teoricamente, é possível no decorrer da partida um jogador acabar com todos os oito peões sendo promovidos. Na prática, porém, isso não acontece. Promoções no xadrez são pouco frequentes, com a maioria das partidas terminando antes de ocorrer qualquer promoção.

En passant

En passant, ou *ao passar* em português, pode ser executado quando um peão avança duas casas em seu lance inicial passando ao lado de um peão adversário, este podendo estar tanto à direita quanto à esquerda. Procede-se então, como se o peão tivesse avançado apenas uma única casa, permitindo o peão inimigo capturá-lo (Figura 45).

O *En passant* também pode ser chamado de "tomar o peão de passagem", passando, na passagem ou à passagem. Porém, na maioria dos casos é utilizada a expressão do francês *En passant* (que se abrevia e.p.).

Figura 45 – *En passant*



Fonte: BECKER (2004), adaptado

Xeque

O rei está em *xeque* quando sua casa atual pode ser atacada por uma peça adversária. Quando o *xeque* acontece, o mesmo deve ser desfeito pelo atacado imediatamente no lance seguinte. Por causa disso, infere-se que não é possível realizar uma jogada que coloque seu próprio rei em xeque.

Caso o rei em *xeque* não possa fugir ou capturar a peça atacante de forma alguma, é considerado *xeque-mate* e o lado atacado **perde**.

Xeque-Mate

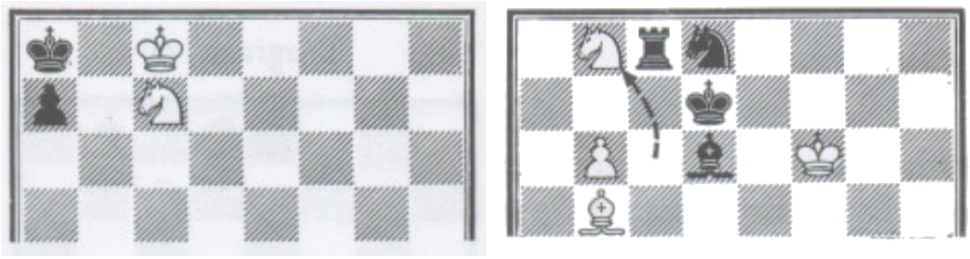
O *xeque-mate* é o xeque decisivo, irremediável, que **conclui a partida**. Há *xeque-mate* quando:

1. O rei não pode fugir para qualquer casa onde não fique em *xeque*
2. Não é possível posicionar nenhuma peça entre o rei e a peça que dá *xeque*
3. Não é possível capturar a peça que dá *xeque*.

Na Figura 46a o rei preto está em *xeque-mate*. O cavalo dá xeque e as pretas não podem capturar o cavalo, nem o rei tem para onde fugir. Já na Figura 46b, o cavalo branco moveu-se, desobstruiu o bispo e deu xeque-duplo. O cavalo preto poderia capturar o bispo branco, o bispo preto tomar o cavalo branco, ou mesmo a torre anular o ataque de ambas as peças. Mas o ataque ao rei é duplo e, num único lance, as pretas não podem anular os dois xeques.

Dentre os tipos de xeque-mate, há dois casos interessantes que vale a pena mencionar: xeque-mate *abafado* e xeque-mate de *banda*.

Figura 46 – Xeque mate

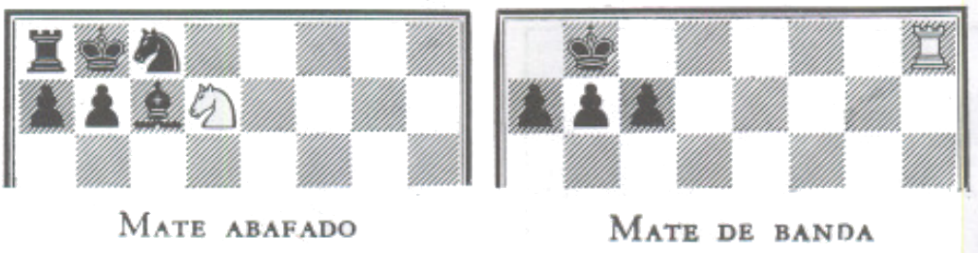


(a) Xeque mate 1 (b) Xeque mate 2

Fonte: BECKER (2004), adaptado

A Figura 47a é um exemplo de xeque-mate *abafado*, quando o rei está bloqueado por suas próprias peças. Quando isso acontece, denomina-se *autobloqueio*. Já na Figura 47b é possível ver um exemplo de xeque-mate de *banda*. Neste caso, o rei está bloqueado/preso ao longo de uma fila, por suas próprias peças. É um xeque que se ameaça com relativa frequência no decorrer das partidas.

Figura 47 – Xeque mate abafado e de banda



(a) Xeque Abafado (b) Xeque banda

Fonte: BECKER (2004), adaptado

Partidas Empatadas

A partida está empatada quando nenhum dos lados pode vencer. Ou seja, quando nenhum jogador pode dar *xeque-mate*. Esse tipo de situação geralmente acontece por empates matemáticos, nos quais já é provado que não há mais nenhuma chance de dar *xeque-mate*. Estes casos são:

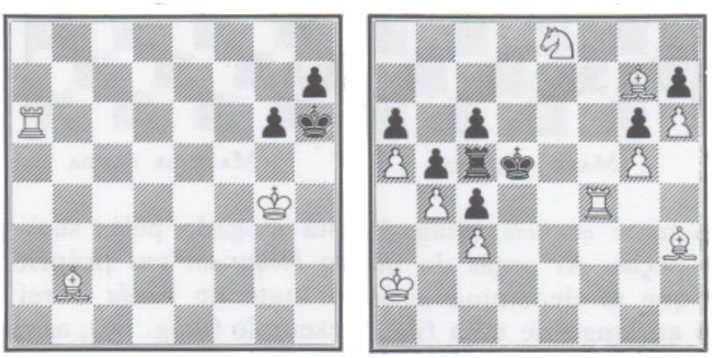
1. Quando ficam apenas os dois reis sozinhos.
2. Quando restam um rei e um cavalo contra um rei sozinho
3. Quando restam um rei e um bispo contra um rei sozinho

4. Quando restam um rei e dois cavalos contra um rei sozinho. Enquanto este não é impossível, o *xeque-mate* só pode acontecer caso o adversário não se defenda corretamente.

Há ainda dois casos especiais, porém não matemáticos, em que se pode declarar empate: *Rei Afogado* e *Xeque perpétuo*

O *rei afogado* é um caso em que jogador que está fazendo o lance, não está em xeque e, ao mesmo tempo, não pode realizar nenhuma jogada (Figura 48). Neste caso, diz-se que o rei está *afogado* e a partida termina em empate.

Figura 48 – Rei Afogado



JOGAM AS PRETAS: EMPATE. JOGAM AS PRETAS: EMPATE.

Fonte: BECKER (2004), adaptado

Já o *xeque perpétuo* é quando um dos jogadores demonstra que pode dar xeque ao rei, continuamente, ininterruptamente, porém sem configurar *xeque-mate*. Neste caso, a partida também termina em empate. A Figura 49 representa bem essa situação, caso a dama branca se mova para a fila do rei preto.

Figura 49 – Xeque perpétuo

AS BRANCAS JOGAM E EMPATAM.

Fonte: BECKER (2004), adaptado

Empates por Regulamento

Além dos empates técnicos, como por *rei afogado* e *xeque perpétuo*, há outros casos estabelecidos pelo Regulamento da Federação Internacional de xadrez. Enquanto os dois primeiros são os que mais interessam o principiante, todo o enxadrista deve conhecer os empates definidos pelo regulamento, principalmente para os jogos oficiais.

Haverá empate, portanto:

1. Quando a mesma posição se repetir 3 vezes. Denomina-se empate pela *repetição de lances*.

2. Quando um jogador demonstrar que foram realizados 50 *lances completos* (50 jogadas de cada lado) sem ter havido captura ou movimento de peão. Após a captura ou movimento, a contagem recomeça.

3. Cada jogador pode, em qualquer momento da partida, exigir do seu adversário um *xeque-mate* em 50 lances. Se o *xeque-mate* não acontecer dentro destes 50 lances, será declarado empate.

    a) A contagem irá recomeçar a cada movimento ou captura de um peão.

    b) Para certas posições (rei, bispo e cavalo contra rei), a teoria demonstrou que são precisos mais de 50 lances para forçar o *xeque-mate*. Nesse caso a contagem deverá ser de 100 lances ao invés de 50.