UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

YAN KAIC ANTUNES DA SILVA

XADREZ ROBÓTICO LIVRE

VITÓRIA DA CONQUISTA - BA
2017

YAN KAIC ANTUNES DA SILVA

Monografia a ser apresentada ao Curso de Graduação em Ciência da Computação da Universidade Estadual do Sudoeste da Bahia, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Orientadora: Alzira Ferreira da Silva
Coorientadora: Cátia Mesquita Brasil Khouri

VITÓRIA DA CONQUISTA - BA
2017

YAN KAIC ANTUNES DA SILVA

XADREZ ROBÓTICO LIVRE

Estão de acordo com o trabalho:

Yan Kaic Antunes da Silva
Graduando

Alzira Ferreira da Silva

Cátia Mesquita Brasil Khouri

Gidevaldo Novais dos Santos

Agradecimentos

Quero agradecer às minhas orientadoras, por dedicarem o seu tempo, mesmo desempenhando diversas atividades, sempre conseguiram arrumar tempo para ler páginas e páginas do trabalho e sugerir melhorias.

Ao meu chefe e professor, Hélio, pela compreensão e oferta de flexibilidade no trabalho, possibilitando que eu solucionasse vários problemas durante o dia. Quero agradecê-lo também por um dos primeiros a reconhecer o meu trabalho e acreditar no meu potencial.

Ao professor Roque ser a pessoa que planta as ideias nas cabeças dos alunos, confiar na capacidade deles, mais do que cumprindo o seu papel em sala de aula. Agradeço por conseguir um ambiente de trabalho (LINDALVA) para os alunos, onde eles tenham a liberdade de realizar seus trabalhos com autonomia.

Quero agradecer também a toda a galera do LINDALVA por deixar o ambiente acadêmico mais alegre e humano. Eu tenho muito gosto em trabalhar na companhia de vocês.

Quero agradecer em especial ao grupo do braço robótico. A Gustavo, Sales, Dalton, Jackson e Alexandre pela dedicação e persistência no projeto, mesmo que nem sempre eram reconhecidos por isso, não ganhavam salários ou horas complementares na academia. Todo esforço foi valioso. Por mais que não temos certificados de tudo o que fizemos, nunca esquecerei da participação de vocês.

Agradeço também a Teté, Dadai e Rafinha por deixarem o caminho mais fácil de caminhar, com a irmandade, resenhas, incentivo, apoio, que foram elementos essenciais para a continuação desse trabalho.

Aos amigos que foram também a minha turma, Matheus, Iago e Limão por sempre indicam os melhores caminhos, colaborar no crescimento mútuo e fazer parte do crescimento intelectual, com discussões, resoluções de problemas, reivindicações de direitos, etc.

Resumo

Neste trabalho é apresentado um modelo de arquitetura de software que transforma um braço robótico em um jogador de xadrez. A partir de programas que permitem que jogadores de xadrez joguem pelo computador foi desenvolvido um conjunto de tarefas que permitam a ligação desse tipo de programa a um braço robótico. Permitindo que esse sistema interaja com o humano por meio de um tabuleiro físico, dispondo de mecanismos que realize e reconheça jogadas. Tal sistema também permite a troca fácil do motor de xadrez predefinido. A partir da observação das dificuldades enfrentadas por estudantes e pesquisadores da área de computação em desenvolverem um motor de xadrez ou um braço robótico e interagirem com o humano de maneira mais natural, verificou-se a necessidade de um sistema que faça a integração de um motor de xadrez com um braço robótico, onde em cada área pode construir seu produto sem se preocupar com a outra área. Para alcançar os objetivos, os requisitos do sistema são encontrados e descritos, permitindo notar a necessidade de módulos responsáveis por cada etapa do processo e então uma arquitetura é elaborada que interligue tudo. Este trabalho trabalha com processamento de imagens, operações lógicas e conversões de formatos. O reconhecimento de movimentos por processamento de imagem teve um resultado satisfatório, por meio da técnica de seleção por cor. Em linhas gerais, reconhece as jogadas e permite que partidas inteiras sejam realizadas.

Palavras chave: xadrez, braço robótico, moto de xadrez, processamento digital de imagens.

Abstract

In this work presents a model of software architecture that a robotic arm turns a chess player. From programs that support chess players to play through the computer, a set of tasks has been developed, connecting this type of program to a robotic arm. The system interacts with the human through a physical board, having mechanisms that realize and recognize moves. Such a system also supports easy exchange of the default chess engine. Observing the difficulties found by students and researchers in computing area in developing a chess engine or a robotic arm and interacting with the human in a more natural way, there was a need for a system that integrates a chess engine with a robotic arm where in each area it can build its product without worrying about the other area. To achieve the objectives, the system requirements are found and described, allowing to note the need for modules responsible for each step of the process and then an architecture is elaborated that interconnects components. This work uses digital image processing, logical operations and format conversions. The recognition of movements by image processing had a satisfactory result, through the color selection technique. In general terms, it recognizes the moves and allows whole matches to be played.

Keywords: chess, robot arm, chess engine, digital image processing.

Lista de Figuras

Lista de Abreviaturas

LINDALVA	Laboratório de Inteligência em Dispositivos de Arquitetura Livre e Veículos Autônomos
XRL	Xadrez Robótico Livre
UCI	Universal Chess Interface
CECP	Chess Engine Communication Protocol
UESB	Universidade Estatual do Sudoeste da Bahia

Sumário

1. Introdução

O xadrez é um jogo de estratégia em que dois jogadores desafiam suas habilidades lógicas e matemáticas, movimentando suas peças para capturar as peças do adversário. A inclusão do xadrez em escolas, com o objetivo de desenvolver o raciocínio lógico e estratégico de crianças e adolescentes durante a sua formação intelectual, é um acontecimento que está ganhando espaço no cenário nacional. Contudo, é também um jogo que gera interesse em pessoas que procuram lazer, aperfeiçoamento intelectual ou investigação científica.

Este jogo alcançou grande sucesso com sua adaptação para programas de computador. As partidas começaram a ter características diferentes das tradicionais, sendo que pessoas podem jogar e comunicar-se a distância, estando conectadas a internet, ou treinando diretamente com o computador. Este fato foi possível porque o computador é capaz de processar fórmulas e modelos matemáticos que foram adaptados para o xadrez, unindo conhecimentos de inteligência artificial, gerando assim um modelo que é conhecido atualmente como **motor de xadrez**.

Outro avanço tecnológico está na área da robótica, obtendo aceitabilidade na sociedade. É uma área que auxilia humanos a realizarem suas atividades cotidianas por meio de equipamentos mecânicos e elétricos, poupando-os de desgaste físico ou perigos (SANTOS, 2004). E esses equipamentos robóticos podem ser classificados em várias categorias de acordo com sua aplicação ou funcionamento. Em linhas gerais, um robô é caracterizado como um manipulador multipropósito, controlado automaticamente, reprogramável, programável em três ou mais eixos (ISO, 2011). Suas aplicações são encontradas em diversas áreas da sociedade moderna, realocando profissionais e gerando uma nova realidade.

Dentro deste cenário de evolução tecnológica, um manipulador muito utilizado na robótica industrial, o **braço robótico**, é usado como atuador de agentes inteligentes para jogar xadrez em tabuleiros físicos. Trata-se de um sistema munido de um mecanismo que movimenta as peças do tabuleiro de acordo com as jogadas que são computadas em um motor de xadrez. Em geral, são sistemas proprietários que já possuem seu sistema de controle inteligente para definir o lance e controle do movimento do manipulador, não permitindo visualizar o funcionamento interno, assim como não é permitido realizar a troca de componentes.

Existem programas que implementam a Interface Gráfica do Usuário (do inglês, *Graphic User Interface*, *GUI*) para jogar em disputa com um motor de xadrez. São programas que oferecem elementos visuais interativos que permitem que o humano visualize a situação do jogo e realize alguma jogada. Além disso, esse tipo de programa faz a comunicação com uma diversidade de motores de xadrez, utilizando um padrão de linguagem, conhecido por **protocolo de comunicação**. Por meio deste, é possível fazer a seleção do motor de xadrez de maneira livre que será oponente do humano.

A partir destas considerações, surge a seguinte questionamento: é possível fazer a ligação de um braço robótico com um motor de xadrez com a mesma eficácia de um programa de interface gráfica se comunica com vários tipos de motores de xadrez?

Buscando oferecer um produto flexível, este trabalho visa desenvolver um produto de software de arquitetura livre que forneça suporte à utilização de um motor de xadrez e um braço robótico de escolha livre.

1.1. Objetivo Geral

Desenvolver uma arquitetura aberta que sirva de ponte de comunicação entre um modelo genérico de braço robótico e um modelo genérico de motor de xadrez.

1.1.1. Objetivos Específicos

- Definir os requisitos de qualidade para a funcionalidade da arquitetura aberta;
- Identificar quais os principais componentes do sistema;
- Identificar quais os principais componentes da arquitetura, como estes podem ser interligados e/ou substituídos;
- Descrever requisitos de comunicação entre os componentes principais; com base nos protocolos existentes;
- Desenvolver um protótipo com base nos requisitos;
- Analisar aplicação do projeto.

Desta forma, este trabalho contribui para a pesquisa em arquiteturas abertas, proporcionando um ambiente adequado para programação e testes de novos motores, e análise de arquitetura de *software*; como também para o ensino de programação e robótica. A utilização em salas de aula permitirá ao professor mostrar a aplicação de um algoritmo de inteligência artificial a um dispositivo que interaja com o humano e com o computador de uma maneira mais natural. Justifica-se também este trabalho,

por seu cunho social, considerando que haverá suporte para a implantação em braços robóticos de baixo custo, sendo que nem todas as escolas possuem poder aquisitivo para ter um braço robótico industrial, isto tanto em instituições de ensino superior quanto da educação básica.

A monografia está organizada como segue: o capítulo 2 apresenta conceitos e definições relevantes para o entendimento do trabalho como um todo, bem como uma revisão da literatura no que concerne à (ao xadrez) robótica e às subáreas que a interfaceiam; o capítulo 3 apresenta o método de avaliação dos componentes citados no capítulo 2, assim como seus resultados. No capítulo 4, teremos a conclusão e as considerações finais deste trabalho.

Este trabalho reúne conceitos de robótica, sistemas embarcados, Interação Humano-Robô, arquitetura de *software* e operações lógicas.

2. O xadrez e o computador

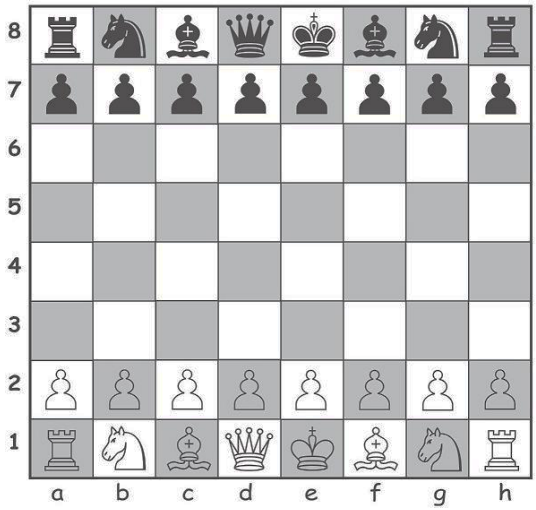
O jogo de xadrez é um esporte intelectual praticado por dois jogadores cujo objetivo principal é ser o primeiro a realizar o xeque-mate (D'AGOSTINI, 2002). O jogo físico é composto por dois tipos de elementos: 1 tabuleiro e 16 peças. As peças são divididas em brancas e pretas, sendo um Rei, uma rainha, duas Torres, dois Bispos, dois Cavalos e oito peões. A Figura 1 mostra a representação atual das peças de xadrez.

Figura 1. Representação das peças do tabuleiro

Fonte: (BECKET, 1974). Adaptada pelo autor.

O tabuleiro é quadrado e dividido em 8 colunas e 8 fileiras, cujas interseções dão origem a 64 células (posições) pintadas de forma alternada, conforme a Figura 2 (D'AGOSTINI, 2002). Para identificação das casas, é utilizado o sistema de coordenadas formado por uma letra (a) para representar as colunas, e um número (n) para representar as fileiras, formando a expressão letra-número (an). Geralmente, em tabuleiros físicos, a marcação de identificação das colunas e fileiras vem impressa nas bordas do tabuleiro.

Figura 2. Posicionamento inicial das peças



Fonte: (BECKET, 1974). Adaptada pelo autor.

Esta é a identificação padrão para jogos convencionais, porém na memória de um computador, a configuração e as características das peças podem ser armazenadas em uma estrutura de dados mais conveniente para os desenvolvedores, proporcionando um melhor gerenciamento da memória, ou que permita realizar análises de posições e cálculo das jogadas de maneira mais eficiente.

A configuração inicial do tabuleiro é sempre a mesma, como apresentada na Figura 2. Todas as peças são dispostas de modo que nenhum jogador tenha vantagem sobre o outro (BECKET, 1974). A segunda e a sétima fileira são preenchidas totalmente por peões, brancos na segunda e pretos na sétima. A primeira e a última linha são preenchidas da seguinte forma: torre, cavalo, bispo, rainha, rei, bispo, cavalo e torre; com peças brancas na primeira e pretas na última fileira.

À medida que as peças vão se movimentando no tabuleiro, elas vão formando configurações de tabuleiro. Uma configuração de tabuleiro é uma instância de um

momento do jogo. Até que o jogo seja finalizado, ele passará por várias configurações diferentes, cada uma delas fornecendo, matematicamente, uma relação de vantagem para um jogador ou outro. Para que humanos saibam qual é a configuração de um momento do jogo, basta visualizar o tabuleiro físico e analisar as peças, junto com suas posições. Para o computador, é necessário armazenar a configuração do tabuleiro em sua memória, e cada modificação efetuada deverá ser informada para que o motor atualize sua configuração virtual. É suficiente que sejam registradas as modificações apenas quando elas acontecerem.

Por ser um jogo bastante complexo e desafiador, muito cientistas começaram a criar modelos matemáticos e computacionais que o modelassem e conseguissem encontrar a melhor estratégia. Depois da criação dos primeiros computadores destinados a uso geral, em 1935 (KEMPEN, 2016), houve uma explosão de pesquisas na área, a fim de criar um algoritmo que fosse eficiente o suficiente para desafiar um humano.

Nas décadas de 1940 e 1950, as pesquisas específicas na área com o avanço na tecnologia voltada à construção do computador, foi possível presenciar o surgimento dos motores de xadrez, que jogam como os humanos. Em maio de 1997, um computador da IBM, chamado de *Deep Blue*, conseguiu vencer o campeão mundial da época, Garry Kasparov, e desde então foi considerado que o computador era melhor em xadrez do que o humano (IBM, 2012).

2.1. Motores de Xadrez

Um motor de xadrez (do inglês, *chess engine*) é um programa de computador capaz de decidir o próximo movimento de uma peça em uma partida de xadrez. O motor recebe uma configuração do tabuleiro, isto é, o conjunto de casas contendo a informação de qual peça está ocupando cada casa; analisa esta informação considerando somente os movimentos válidos; e retorna uma melhor jogada possível no momento, de acordo com algum critério (SANTANA, 2014).

Os motores variam de um para outro e de acordo com a máquina em que serão instalados, ou seja, existem motores específicos para cada tipo de arquitetura de computador, de acordo com a quantidade de núcleos, com a quantidade de memória, etc. Logo, a seleção de um motor de jogo não pode ser realizada de forma aleatória,

é preciso analisar quais serão os equipamentos disponíveis para a aplicação (MARSLAND, 2011).

Os motores “conhecem” o tabuleiro e a posição de suas peças. Para analisar o estado temporário da partida do xadrez, é realizado um cálculo com base nas derivações de estados provenientes do estado imediatamente anterior. Para isso, é necessário utilizar-se de uma estrutura de dados adequada para representar esses estados. A estrutura de dados representa as possíveis jogadas de acordo com o estado do jogo, e por ser uma previsão, ele apresenta uma pontuação que indica quão bom ou ruim uma jogada pode ser. A partir desta estrutura é possível analisar a pontuação que um estado do jogo pode oferecer, gerando assim uma indicação de qual dos dois jogadores está em vantagem no momento, e quais seriam as melhores futuras jogadas (SANTANA, 2014).

Um tipo de representação muito usado é o *bitboard*. Segundo esta abordagem, para cada tipo de peça, usamos uma variável do tipo inteiro de 64 bits. Esta variável está relacionada às 64 casas do tabuleiro de xadrez, como é possível perceber na Figura 3. Como temos 6 tipos de peças para cada lado (preto e branco), precisamos de 12 variáveis para representar todos os tipos de peças. Cada bit da variável representa se uma casa do tabuleiro está ocupada ou não. Para obter informações do tabuleiro, basta fazermos algumas operações binárias (SANTANA, 2014).

Figura 3. Mapeamento de um *bitboard*

8	56	57	58	59	60	61	62	63
7	48	49	50	51	52	53	54	55
6	40	41	42	43	44	45	46	47
5	32	33	34	35	36	37	38	39
4	24	25	26	27	28	29	30	31
3	16	17	18	19	20	21	22	23
2	8	9	10	11	12	13	14	15
1	0	1	2	3	4	5	6	7
	a	b	c	d	e	f	g	h

Fonte: (BECKET, 1974). Adaptada pelo autor.

No mercado, podemos encontrar vários motores de xadrez tanto comerciais como gratuitos. Destacamos o *Strelka*, que funciona muito bem para computadores de 32

bits de até um núcleo. Para computadores com mais de um núcleo encontramos o *Houdini*, que é gratuito para fins não comerciais, e tem versões otimizados para o uso de processadores de 1 a 8 núcleos. Para os que aderem a softwares livres, temos o *Stockfish*, que funciona tanto para computadores de 32 quanto de 64 bits. Outro é o *GNU Chess*, que possui licença livre, permitindo que outros desenvolvedores possam analisar, modificar e redistribuir desde que os produtos derivados tenham a mesma licença. Como temos o requisito de nossa plataforma seja livre, foram analisados dois que se enquadram neste aspecto.

O *GNU Chess* é um motor de xadrez que se comunica com um jogador humano através de comandos de texto. Ele foi criado em conjunto com a interface gráfica *Xboard* para comunicar-se apenas com ela. Com o passar do tempo, e com um maior uso na comunidade, passou a oferecer suporte a outras interfaces gráficas. Por conta da liberdade de sua licença, é um *software* que recebe constantes atualizações por desenvolvedores do mundo inteiro (LETOUZEY, 2014).

O *Stockfish* é um motor de jogo desenvolvido em código aberto (licença GNU), que permite realizar uma análise detalhada sobre o seu funcionamento. Foi criado em 2008, a partir de outro projeto, o *Glaurung engine*, que por motivos técnicos foi preferível seguir uma nova linha. Devido a sua eficiência em diferentes processadores, este motor de xadrez ganhou continuidade por outros desenvolvedores e até hoje recebe atualizações (ROMSTAD, 2016).

Stockfish já foi considerado um dos melhores motores de xadrez do mundo, estando sempre no topo das listas dos motores de xadrez mais fortes, na categoria de código aberto. Em 2013 ganhou um campeonato mundial de xadrez, e vários outros prêmios não oficiais (KEMPEN, 2016). O seu sucesso foi atribuído ao fato de possuir versões compatíveis com computadores de até 128 núcleos e conseguir gerar uma tabela de transposição de até 1 TB (ROMSTAD, 2016).

O problema é que a maioria dos motores não possui interface gráfica própria. Estes recebe comandos em texto simples, realiza processamento do jogo e devolve o próximo movimento em um formato semelhante ao do comando de texto recebido (SANTANA, 2014). Para realizar a interação entre motor e homem, é preciso uma interface que se comunique com ambos os atores na linguagem adequada;

geralmente visual, para o homem e por comandos textuais para o motor de xadrez (MANN, 2009).

Uma interface gráfica para jogo de xadrez com suporte a motores de xadrez é um programa que oferece por um lado elementos visuais interativos que permitam ao humano observar a situação do jogo e realizar alguma jogada. Por outro lado, envia e recebe comandos para o motor de xadrez em uma linguagem textual. Também permite as mesmas funcionalidades para o motor de xadrez, porém esta interação é feita por comandos textuais (SANTANA, 2014). Em geral, esse tipo de programa permite instalar uma série de motores diferentes, alguns oferecem ferramentas para analisar o comportamento do motor para o usuário, exibindo na tela informações de pesquisa que o motor estiver realizando. Com isso é possível que desenvolvedores possam testar e comparar o desempenho de seus protótipos de motores.

Desta categoria, destacamos o *Xboard que* é um programa que implementa a interface gráfica de jogo de xadrez e que participa do projeto GNU. Podemos encontrá-lo em duas versões: *XBoard* para dispositivos que usam um sistema operacional Linux e *WinBoard* para dispositivos que usam o Windows. Para se comunicar com o motor de xadrez, essa interface utiliza o mecanismo do CECP, desenvolvido pelo seu mesmo criador (MANN, 2009). Para interagir com o usuário, ele dispõe de uma tela simples e amigável, como pode ser observado na Figura 4.

Figura 4. Partida de xadrez no *Xboard*

Fonte: (MANN, 2009)

No ramo de interface gráfica, ainda podemos encontrar o programa Arena. O Arena possui compatibilidade com uma diversidade maior de motores de xadrez do que a do *Xboard*, porque além de implementar o protocolo CECP, também é compatível com o protocolo UCI. Este programa apresenta um conjunto maior de ferramentas com informações detalhadas para análises do jogo, e tudo isso com uma interface fácil de usar, permitindo também várias personalizações, como podemos ver na Figura 5 (BLUME, 2016).

Figura 5. Partida de Xadrez no Arena

Fonte: (BLUME, 2016)

Este programa é compatível com aproximadamente 250 motores de xadrez ao redor do mundo, oferecendo diferentes níveis de dificuldade para melhorar a dinâmica do jogo. Também permite partidas entre jogadores geograficamente distantes, mediantes a utilização de um servidor de xadrez, ao invés de um motor (BLUME, 2016).

Durante os anos, alguns programas foram desenvolvidos especificamente para manipular o ambiente visual e repassar os comandos para os motores de xadrez, e com isso, vários motores de xadrez estavam sendo desenvolvidos e nem sempre eles conseguiam se comunicar por incompatibilidade dos comandos (BLUME, 2016).

Para que um organismo partilhe algo, sejam dados ou informações, é preciso ter um processo que seja do conhecimento dos indivíduos que participam da comunicação. Em computação, um protocolo de comunicação é um conjunto de regras formais que descrevem como formar dados. Este pode ser de baixo nível ou de alto nível. No primeiro são definidos os padrões elétricos e físicos a serem observados, ordenação de bits, a transmissão e a detecção de erros do fluxo de bits. Os protocolos de alto nível lidam com a forma dos dados, incluindo sintaxe da mensagem, o diálogo entre o terminal e o computador, conjunto de caracteres, sequência de mensagens, etc.

No caso do xadrez, é preciso estabelecer um processo de comunicação entre o motor e a interface gráfica. A principal razão para o uso de um protocolo de comunicação é

que quando se está desenvolvendo um motor de xadrez não é necessário implementar a interface gráfica do usuário; o contrário também se aplica (MANN, 2009).

Atualmente existem alguns protocolos de comunicação para o motor de xadrez que foram bem adotados na internet. Existem alguns sites que mostram quais motores de xadrez são compatíveis com determinados protocolos. Os mais usados são: a interface universal de xadrez (UCI, sigla em inglês) e o protocolo de comunicação do motor de xadrez (CECP, sigla em inglês).

No ano 2000, foi publicado o protocolo UCI (KAHLEN, 2004), que ficou competindo com o antigo CECP, mas o seu principal objetivo era ser independente de plataforma (sistema operacional) e da quantidade de funções do motor de xadrez, ou da variedade de recursos disponíveis na interface gráfica. Dessa forma, as entidades (motor e GUI) poderiam comunicar-se entre si através de pequenas mensagens recebidas e emitidas pela entrada e saída padrão do receptor, respectivamente (KAHLEN, 2004).

Alguns comandos da UCI permitem verificar o status do jogo, *feedback* de comandos ou forçar que o motor retorne de imediato a melhor jogada computada até o momento (caso esteja demorando demasiadamente). Todos os comandos são documentados e publicados no *site* de wbec-ridderkerk[1], para que futuros desenvolvedores possam criar novos motores baseados no mesmo protocolo de comunicação (KAHLEN, 2004). Na Figura 6, podemos visualizar um exemplo do uso dos comandos UCI.

[1] http://wbec-ridderkerk.nl/html/UCIProtocol.html

Figura 6. Exemplo de troca de mensagens usando o UCI

```
1  uci
2  id name Pulse 1.5-java
3  id author Phokham Nonava
4  uciok
5  position startpos moves e2e4
6  go movetime 10
7  info depth 1 seldepth 3 score cp -17 pv a7a6 nps 0 time 1 nodes 4
8  info depth 1 seldepth 3 score cp -16 pv a7a5 nps 0 time 3 nodes 5
9  info depth 1 seldepth 3 score cp -15 pv b7b6 nps 0 time 3 nodes 6
10 info depth 1 seldepth 3 score cp -2 pv d7d6 nps 0 time 4 nodes 11
11 info depth 1 seldepth 3 score cp -1 pv d7d5 nps 0 time 4 nodes 14
12 info depth 2 seldepth 6 score cp -15 pv d7d5 b1c3 d5e4 c3e4 nps 0 time 8 nodes 82
13 info depth 2 seldepth 6 currmove a7a6 currmovenumber 5 nps 0 time 10 nodes 101
14 bestmove d7d5 ponder b1c3
15 position startpos moves e2e4 d7d5 b1c3
16 go movetime 10
17 info depth 1 seldepth 4 score cp -15 pv d5e4 c3e4 nps 0 time 1 nodes 5
18 info depth 1 seldepth 5 score cp -13 pv e7e6 nps 0 time 2 nodes 23
19 info depth 1 seldepth 5 score cp -1 pv g8f6 nps 0 time 4 nodes 49
20 info depth 2 seldepth 8 score cp -4 pv g8f6 d2d3 nps 0 time 12 nodes 161
21 info depth 2 seldepth 8 currmove e7e6 currmovenumber 2 nps 0 time 20 nodes 162
22 bestmove g8f6 ponder d2d3
```

Fonte: (SANTANA, 2014)

Quando um usuário faz um movimento, um conjunto de caracteres são enviados: position startpos moves e2e4 (linha 5, Figura 6). O comando *startpos* indica que o jogo começou a partir da posição inicial e comando *moves* significa os movimentos realizados. Quando o motor recebe esta linha, deve configurar a posição no seu tabuleiro interno. Esta sequência de posição não solicita que o motor comece a "pensar" ou responder com alguma confirmação, isto é feito com o comando *go*. Quando recebe este, inicia o seu processo de busca por uma melhor jogada de acordo com a configuração das peças do tabuleiro mais recentemente armazenado. O comando *info* envia informações para a GUI sobre o andamento de suas pesquisas. O motor só pode enviar informações para a GUI quando estão previamente descritas pelo protocolo utilizado.

2.2. Braços Robóticos

Já temos um motor e a interface, agora necessitamos de algum dispositivo capaz de efetuar o movimento físico das peças em um tabuleiro também físico. Para isto, utiliza-se braços robóticos. Assim, a robótica vem ganhando espaço no jogo de xadrez, que nos séculos XX e XXI avançou radicalmente, com a criação de máquinas capazes de reunir outras máquinas e até mesmo robôs que podem ser confundidos com seres humanos (CRUZ, 2013). “Um robô é um manipulador multifuncional, programável, projetado para mover materiais, componentes, ferramentas ou dispositivos especiais

através de movimentos programáveis variáveis para a execução de uma variedade de tarefas.” (SANTOS, 2004)

Já braço robótico é um robô constituído por juntas e elos que é capaz de movimentar-se ou manipular objetos de acordo com coordenadas espaciais. Os elos são usualmente blocos alongados rígidos, e são ligados uns aos outros através das juntas. Os elos podem variar a sua posição relativa e estão normalmente associados em série. Existem variadíssimas combinações de elos e juntas de acordo com as aplicações (SANTOS, 2004). A Figura 7 apresenta um exemplo de um braço robótico.

Figura 7. Modelo simples de um braço robótico

Fonte: (SANTOS, 2004)

O termo “grau de liberdade” é utilizado para indicar o número total de movimentos independentes que um dispositivo pode efetuar. Um cubo no espaço a 3 dimensões pode deslocar-se ao longo dos três eixos, e também girar em torno de cada um deles, resultando assim um total de 6 graus de liberdade para a sua movimentação (SANTOS, 2004).

O desenvolvimento de braço robótico para o jogo de xadrez tem, segundo o Xadrez Robótico (ATOS, DOGAN, ATOS, 2014) se mostrado uma área com vários problemas no campo da Interação Humano-Robô. Além de questões referentes a manipuladores, deve-se resolver problemas para a captura dos movimentos realizados pelo humano.

No mercado, é possível encontrar alguns braços robóticos que jogam xadrez, dentre eles, encontramos o *Drexel Human Robot Chess*, uma plataforma de xadrez robótico de baixo custo e versátil (ABHICHANDANI, 2013). Também é possível encontrar o Chiara, um robô educacional de código aberto, desenvolvido pelo laboratório da Universidade de Tekkotsu, em Pitsburgo (EUA). Este robô tem a forma de um inseto

e consegue movimentar-se sobre o plano (chão ou mesa) para alcançar objetos. Ele foi desenvolvido originalmente para que alunos pudessem praticar técnicas de programação inteligente, mas sua arquitetura também permite a manipulação de peças físicas de xadrez

2.3. Processamento de imagens

O processamento digital de imagens é um conjunto de tarefas interconectadas para gerar uma representação de uma imagem com informações relevantes para exibição. Envolve fundamentos de várias ciências Física, Computação, Matemática, destas áreas, conceitos como Óptica, Física do Estado Sólido, Projeto de Circuitos, Teoria dos Grafos, Álgebra, Estatística, entre outros (Queiroz e Gomes, 2014). O objetivo é melhorar o aspecto visual para que o interpretador humano tenha suporte a realizar uma análise, e talvez posteriormente levar a gerar outros processamentos posteriores que melhoram o aspecto visual da imagem continuamente (SPRING, 1996).

Podemos definir que uma imagem é uma matriz bidimensional de pontos da imagem, conhecidos como pixels. Os pixels podem ser representados de várias formas. A forma mais conhecida de representação para exibição em telas colorida é pelo sistema de cores RGB. Neste, cada pixel é representado por uma matriz tridimensional, cada eixo representando as cores vermelho, verde e azul. As misturas dessas 3 cores, em forma de luz e em graus de intensidade, formam as outras cores.

> “No processamento digital de imagens, usamos um modelo de cor RGB simplificado (baseado no sistema de cor CIE de 1931), otimizado e padronizado para exibição de imagens. Contudo, o principal problema de RGB é ter um caráter não linear da percepção. (...). Por exemplo, começando pelo branco, a subtração da componente azul produz amarelo; começando com o vermelho, a adição da componente azul produz roxo.” (SOLOMAN e BRECKON, 2013)

Com conhecimento sobre dimensões de imagens e o comportamento das cores no sistema RGB, iremos realizar algumas transformações para auxiliar a obtenção dos objetivos deste trabalho. O problema central que envolve a visão computacional: como reconhecer as mudanças no tabuleiro físico?

3. Plataforma Robótica de Xadrez

Tendo em vista o objetivo da construção de um sistema robótico para xadrez, o Xadrez Robótico Livre é um braço robótico que joga xadrez e que atende a licença GNU. Logo, consiste de um sistema que envolve entes físicos e de software. Um artefato digital foi criado e para isto foram empregados métodos que garantissem que o processo de concepção e implementação resultassem em um produto que atendesse às restrições do projeto. Foi adotado o paradigma *design science research*, que segundo Dresch, Lacerda e Antunes Júnior,

> "(...) é a ciência que procura desenvolver e projetar soluções para melhorar sistemas existentes, resolver problemas ou, ainda, criar novos artefatos que contribuam para uma melhor atuação humana, seja na sociedade, seja nas organizações." (2015, p. 57)

Ainda segundo os autores, o método *design science research* fundamenta e operaciona a condução da pesquisa quando o objetivo a ser alcançado é um artefato ou a construção. Para o desenvolvimento deste projeto foram realizados os seguintes procedimentos metodológicos:

1. Identificação do problema: para estabelecer o escopo e a identificação do problema foram realizadas reuniões para analisar sua importância para a área, para o pesquisador e sua viabilidade.

Nesta etapa foram encontrados os seguintes problemas:

- Identificação das jogadas;
- Promoção do peão;
- Mecanismo de processamento do jogo.

2. Após esta etapa, foi realizado o levantamento dos requisitos, funcionalidades, performance esperada para funcionamento. Durante esta etapa foram realizadas entrevistas não sistematizadas com professores da área para melhor compreensão do problema e suas limitações. Foram então definidos os seguintes requisitos:

- O sistema precisa jogar xadrez;
- O sistema precisa modificar as peças num tabuleiro físico;
- O sistema precisa reconhecer a jogada que humano realizar.

Em paralelo a estas etapas foi realizada uma revisão sistemática da literatura, buscando: conhecimentos sobre braço robótico de xadrez, obtenção de imagem, tratamento, análise de jogadas obtidas via câmera; protocolos de comunicação e processos de construção de braços robóticos. Para isso, foi realizado o levantamento

de trabalhos publicados relevantes para o projeto. Através de uma análise do conteúdo obtido foi feito um resumo destes e interpretação dos resultados que contribuíram para elaboração do capítulo 2 e no processo de soluções para os problemas.

3. Concepção da arquitetura.

Após análise de artefatos existentes e com o entendimento do problema, foi proposta uma arquitetura modular para o controle do XRL. Nesta, foram definidos os principais componentes, processos de comunicação, formas de obtenção de imagem e seu processamento para definição de jogadas válidas. Os principais componentes são:

- Visão computacional;
- Memória de estados do tabuleiro;
- Controlador de braço robótico;
- Inteligência Artificial;

4. Desenvolvimento do protótipo XRL.

Nesta etapa, baseada no método interativo de Engenharia de Software, o sistema foi implementado. Os componentes físicos foram construídos a partir da planta e especificações geradas na etapa 3. A implementação do software foi feita em linguagem C/C++, pois é uma linguagem que já implementa operações lógicas e é bastante utilizada para sistemas embarcados. Alguns diagramas foram desenvolvidos para a visualização geral dos componentes implementados. As etapas de modelagem e construção dos módulos são descritas nas seções seguintes.

5. Avaliação

Os módulos foram analisados com alguns trechos de código cruciais para o funcionamento do sistema. Discutiremos se as saídas esperadas condizem com os resultados obtidos.

3.1. Arquitetura

Conforme explicado no capítulo 2, é possível realizar uma partida de xadrez contra o computador, utilizando um motor de xadrez e uma interface gráfica que se comunique com o motor e apresente as jogadas na tela de forma amigável escolhido pelo usuário.

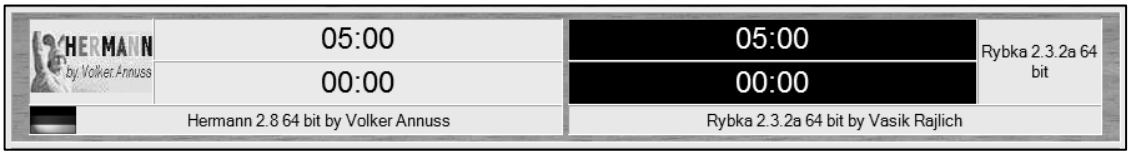
Para este trabalho, foi selecionada a interface gráfica Arena, a qual permite fazer a instalação ou troca de motor de xadrez de modo simples. Na tela apresentada, existe

uma área que informa ou faz a alteração de quais serão os jogadores em competição, conforme a Figura 8. Nesta, é possível fazer a seleção de dois agentes, que estão com um objetivo em comum, ganhar a partida de xadrez. Estes agentes podem ser humanos ou máquinas, então podemos ter 3 tipos de partida:

1. Motor de xadrez contra motor de xadrez;
2. Motor de xadrez contra humano;
3. Humano contra humano.

Na primeira opção, temos dois motores de xadrez jogando um contra o outro. Geralmente é usada para testar algoritmos e verificar qual é o mais eficiente. A segunda, é um objetivo deste trabalho, em que um humano consegue jogar xadrez com a máquina acessando a interface gráfica. Por fim, podemos configurar o Arena para dois jogadores humanos, sendo nessa modalidade, podendo ser configurada para jogo local ou remoto (internet).

Figura 8. Visor de competidores do Arena



Fonte: (BLUME, 2016)

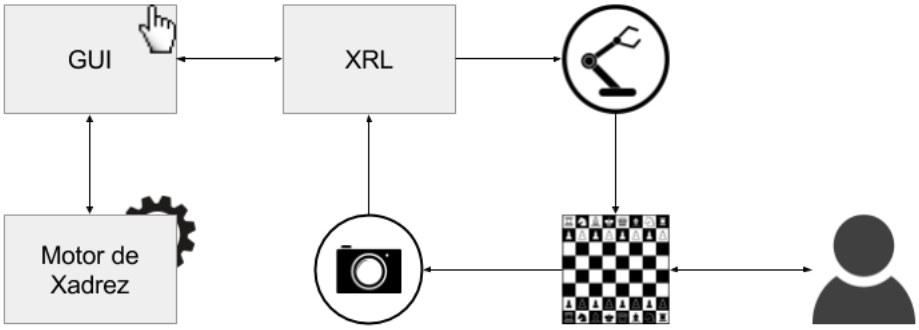
Para a primeira opção, utilizaremos a primeira modalidade, para dois motores de xadrez jogarem entre si. O primeiro será referido aqui como engine1 e o segundo como engine2. O processo de comunicação será mediado pelo programa Arena, e instalaremos um motor de xadrez qualquer, neste trabalho, utilizaremos o Stockfish para desempenhar o papel de "pensar" e ocupará o papel engine1. Para o papel de engine2 utilizaremos o produto desenvolvido neste trabalho.

Este trabalho propõe o Xadrez Robótico Livre - XRL, um programa que terá duas funções: interagir com o tabuleiro físico e interagir com a *GUI*. Com isso, o XRL fará interação com o humano através de um atuador, o braço robótico, e com um sensor, a câmera, conforme a Figura 9. Como o XRL se comunica com duas entidades, dependendo da visão, este pode ser interpretada de duas formas:

- Na visão do humano, o braço robótico é o oponente no jogo de xadrez;
- Na visão da GUI, o XRL é um motor de xadrez.

Nas duas visões, os detalhes referentes à outra entidade, não serão visíveis. O trabalho do XRL é traduzir as informações geradas pelas entidades da partida: o motor de xadrez e o humano, e passar para a outra, cada um na sua forma de linguagem.

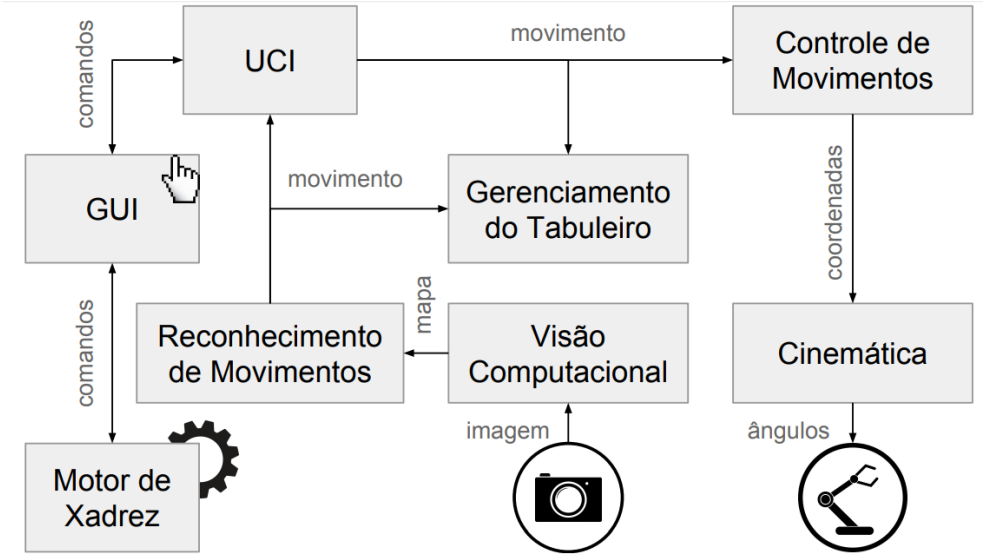
Figura 9. Processo de comunicação das entidades envolvidas



Fonte: Autor

Para que este processo tenha sucesso, é necessário uma série de controles e conversões de informações. Analisando a Figura 9, percebe-se que o XRL terá comunicação com 3 componentes distintas com linguagens distintas. A interface gráfica realiza troca de mensagens por comandos de textos previstos no protocolo UCI; o braço robótico realiza movimentos quando recebe um conjunto de ângulos formados em seus elos; e a câmera entrega uma imagem, que é uma matriz de pontos de cores que representa o tabuleiro de xadrez. Por isso, o XRL é dividido em vários módulos, para gerenciar e tramitar essas informações, conforme visto no capítulo 3 e representado na Figura 10

Figura 10. Diagrama de Fluxo, representando a arquitetura do trabalho



Fonte: Autor

Assim, o XRL trabalha com 3 meios de comunicação: comunicação padrão entre programas utilizando o protocolo UCI, comunicação serial por Bluetooth e comunicação serial por USB, respectivamente para comunicar com a interface gráfica, com microcontrolador do braço robótico e com a câmera.

O primeiro componente interno, representado na Figura 10, é o módulo **UCI**, responsável pela troca de mensagens com a interface gráfica. No início de uma interação, o interpretador recebe um comando oriundo do motor de xadrez num formato de texto conforme protocolo descrito nos Anexos A e B. O interpretador traduz o comando e repassa o movimento para os módulos de gerenciamento do tabuleiro e de controle de movimentos. Alguns comandos que a GUI transmite a um motor de xadrez são solicitações de informações sobre o motor ou se está pronto para iniciar a partida. Nestes casos simples, o módulo UCI responde diretamente para a GUI, sem que mensagens sejam passadas para os demais módulos internos do XRL.

O **gerenciamento do tabuleiro** contém o estado atual do tabuleiro e estruturas de dados necessárias para o funcionamento do programa. Ele é responsável por analisar o movimento recebido do módulo UCI e atualizar o estado interno do tabuleiro para futuras inferências. Assim como também é responsável por registrar o movimento feito pelo humano, que é interpretado pelo módulo **Reconhecimento de Movimentos**.

Após receber o movimento realizado pelo motor de xadrez, é necessário traduzir a jogada digital em uma jogada correspondente no tabuleiro físico. Para isso, vamos precisar definir onde a garra do braço robótico deva se localizar para alcançar as peças e coloca-la no lugar certo. Para que um único movimento lógico seja realizado no tabuleiro físico, o braço robótico precisa realizar diversos movimentos em sequência. Assim, o módulo **Controle de Movimento**s é responsável por converter um movimento de xadrez em um conjunto de coordenadas, que correspondem às posições que a ponta da garra deverá estar para realizar a jogada.

Dependendo do modelo de braço robótico que estiver utilizando, este não terá suporte a comandos por coordenadas, trabalhando, na verdade, com controle de ângulos dos elos. Nessa arquitetura, o módulo **Cinemática** é responsável por converter coordenadas solicitadas pelo módulo Controle de Movimentos para ângulos. Este módulo também estará responsável por executar o movimento gradativamente, pois pode haver acidentes se o braço robótico realizar um movimento muito brusco.

Com as etapas citadas acima, temos um modelo de como transferir um movimento gerado pelo motor de xadrez para o braço robótico. Assim, completa a primeira etapa do funcionamento, fazendo então a movimentação da peça no tabuleiro. O humano fará a sua jogada e então o sistema deve verificar qual foi o movimento realizado e passar para o motor de xadrez. Para verificar o movimento é necessário um dispositivo que faça a captura do estado do tabuleiro após a jogada do humano. No capítulo 4 encontra-se a descrição de alguns mecanismos possíveis para essa detecção.

Nesse trabalho foi utilizada uma câmera para capturar o estado do tabuleiro. A imagem gerada passará por um tratamento, utilizando técnicas de processamento de imagem, com o objetivo de mapear a localização das peças do tabuleiro. A estrutura de dados será um mapa de ocupação, informação se a casa está livre ou ocupada. No capítulo 4 tem a descrição completa do **Visão Computacional**.

Após realizar o mapeamento das peças, realiza-se o reconhecimento das peças que foram alteradas. O reconhecimento é feito por inferências lógicas, fazendo interseção dos dados anteriores com os dados recebidos das peças modificadas do módulo Visão Computacional, porém nem todos os casos serão contemplados. Veremos adiante os principais problemas gerados para inferir a movimentação a partir de um reconhecimento de imagem. Com a inferência realizada, este módulo gera um movimento correspondido pelo humano, que então é repassado para o módulo de gerenciamento do tabuleiro e para o módulo interpretador. O módulo interpretador envia o movimento para o motor de xadrez que analisa e realiza o próximo movimento e fecha a rodada.

3.2. UCI

O módulo **UCI** terá a função de comunicar-se com a interface gráfica para motores de xadrez. Como a interface gráfica estará apenas repassando a jogada realizada por outro motor de xadrez, é como se este motor de xadrez desenvolvido estivesse comunicando-se diretamente com o oponente. Para que os comandos sejam "entendidos" por ambas as partes, é necessário adotar um protocolo de comunicação. Neste trabalho, utilizamos a Interface de Comunicação Universal (em inglês, UCI), por isso, o nome deste módulo recebe o mesmo nome.

A primeira etapa do processo de comunicação é a apresentação. A interface gráfica tem o papel de mediador entre os motores, então envia uma mensagem para cada

componente informando o protocolo de comunicação utilizado, neste caso o UCI, e fica aguardando uma resposta sobre a compatibilidade. Caso ambos motores respondam "*uciok*", significa que os motores têm o UCI implementado e demais procedimentos podem ser continuados.

Posteriormente, quando todos os componentes entendem que podem comunicar-se por um mesmo protocolo, eles apresentam seus nomes, realizam alguns comandos de controle sobre as restrições do jogo, como tempo para realizar a jogada ou profundidade da pesquisa. Todas essas configurações são originadas pela interface gráfica que media o jogo. Quando tudo estiver configurado, então a partida pode ser iniciada. Cada jogada é realizada pelo comando "move" no protocolo UCI. Cada jogada é passada para a interface gráfica e posteriormente para o oponente, gerando então um ciclo.

3.3. Controle de Movimentos

Este módulo é responsável por 3 etapas da composição do movimento: cálculo das dimensões do tabuleiro, controle de trajetória e controle de animação. Utilizaremos coordenadas espaciais com dimensões medidas em centímetros.

Quando o braço robótico vai realizar algum movimento, este não pode realizar diretamente do ponto que o braço se encontra; é necessário um conjunto de movimentos para não haver catástrofes no jogo. Por exemplo, quando a ponta da garra acaba de deixar uma peça no tabuleiro, este está numa altura baixa e se for tentar alcançar a peça seguinte com a mesma altura que estiver no momento, pode acabar derrubando algumas peças antes de pegar a peça desejada. Existem alguns algoritmos de controle de rota para desviar dos obstáculos, porém, este sistema de tabuleiro tem uma complexidade pequena, já que o braço robótico consegue elevar a garra em uma altura maior do que a maior peça. Sendo assim, quando o braço estiver na altura adequada, este pode movimentar-se em longitude ou latitude sem derrubar nenhuma peça.

O braço robótico precisa ficar numa determinada posição quando estiver em inatividade, para não atrapalhar a captura de imagem pela câmera ou o usuário a jogar. Quando uma posição for solicitada, o ponto efetivado terá a mesma latitude e longitude solicitada, modificando a altitude para uma constante de altitude alta. No

passo seguinte, mantemos Ox e Oy e modificaremos Oz para a constante altitude baixa, que é uma altura suficiente para a garra capturar uma peça.

Com isso, basicamente teremos 4 pontos que indicarão a trajetória da garra: ponto atual da garra, sem modificações; ponto atual da garra com modificação da altura para a constante "alturaMaior"; ponto destino, com altura modificada para a constante "alturaMaior"; e ponto destino, sem modificações. Esses pontos serão efetivados por interpolação linear, realizada também por este módulo.

A interpolação linear é a transição do movimento, permitindo a troca de posição de maneira gradual. É um processo parecido com animações gráficas. Quando um elo possui a abertura de 60º e for solicitado a abertura de 15º, este módulo irá configurar o braço em várias etapas, para não haver mudanças bruscas: 60º, 55º, 50, (...), 25º, 20º e 15º. Porém, este módulo não faz operações com ângulos, então modificaremos gradualmente as coordenadas espaciais, onde o controle dos ângulos será descrito no capítulo 3.5.

3.4. Gerenciamento do Tabuleiro

A arquitetura precisa salvar na memória o estado atual do tabuleiro para fazer inferências posteriores de jogadas humanas, uma vez que no mapeamento das peças, não há identificação de qual peça está em determinada casa, apenas se há ou não alguma peça. Este módulo possui estruturas simples, mas é muito importante para o funcionamento geral.

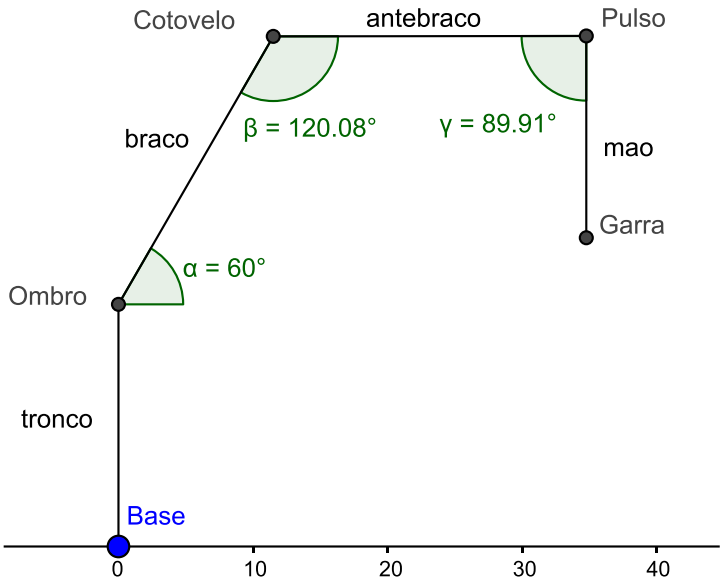
Sua estrutura simples é composta por uma matriz de caracteres 8x8, que representa as peças do xadrez em suas casas. A posição das peças é análoga às posições da matriz, porém contendo 8 bits para cada célula. As peças são identificadas da seguinte forma: rei (r), dama (d), torre (t), bispo (b), cavalo (c) e peão (p). Porém, sua estrutura não pode ser alterada externamente, nem por outros módulos.

A memória é manipulada da seguinte forma: é possível ler todas as posições do tabuleiro, mas a escrita é feita em formato de movimento. O movimento é um conjunto de dados que contém a posição da peça de origem com a posição desejada. Como este módulo receberá comandos originados de ambos jogadores, será necessário uma *flag* para controle de qual jogador está atuando no momento, sendo alternado a cada jogada.

3.5. Cinemática

A posição que o braço robótico apresenta é configurada a partir do controle dos ângulos que formam na base, ombro, cotovelo, pulso e garra. Essas configurações são facilmente definidas por um programa, desde que seja manipulado por um humano. Este módulo é responsável por converter coordenadas espaciais em um conjunto de ângulos, que correspondem às juntas do braço robótico. Para isto, é necessário adotar o estudo da cinemática direta. Este trabalho não possui o objetivo de descrever detalhadamente o estudo da cinemática, mas sim a sua aplicação.

Figura 11. Modelo esquelético da lateral do braço robótico no Geogebra

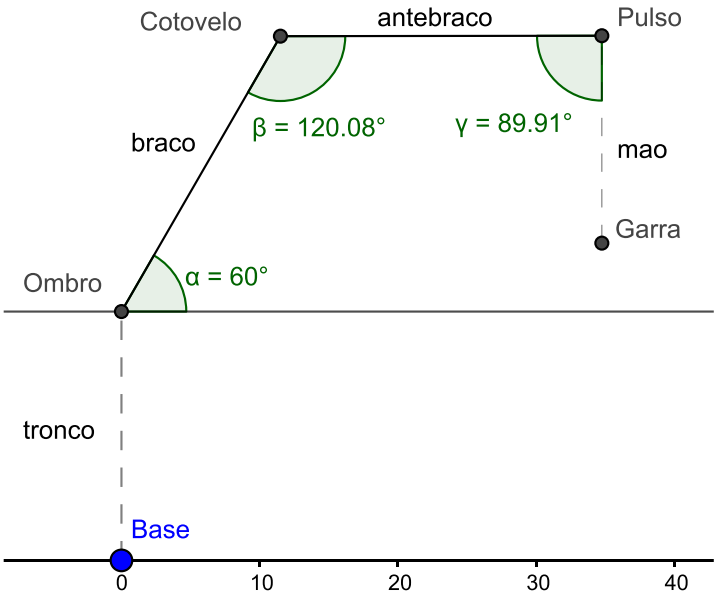


Fonte: Autor

Ao observar um braço robótico de 3 graus de liberdade como na Figura 11, vamos ter que levar em conta algumas considerações para posteriormente fazer as adaptações para a aplicação da fórmula de cinemática. Esta ilustração mostra o braço robótico por uma visão lateral. Essa visão é importante para entender sua anatomia e deixar mais fácil o entendimento de controle, sendo que deste modo podemos ignorar o ângulo formado pela base, considerando-o como fixo. Supondo que o braço robótico está disposto fixamente de frente para o tabuleiro, obteremos o ângulo da base do braço igual a zero e a distância da projeção da garra com o chão e a base será exatamente igual à distância y solicitada pelo módulo anterior.

Para a garra alcançar uma peça sem derrubar as demais, é necessário que a captura seja feita de cima para baixo, num movimento vertical. Com isso, o ângulo que a garra fará com o chão deverá ser de 90 graus. Com isso, a diferença entre o ponto Pulso e o ponto Garra é apenas na altura, onde o Pulso tem a altura maior, com o comprimento da mão de diferença. Este fato facilita nos cálculos, pois essa variação de altura é um valor constante (tamanho da mão). Então, virtualmente, vamos considerar que o ponto que pega qualquer objeto no mundo físico será o pulso. Porém, ao invés de solicitar que o ponto do Pulso desça mais um pouco para alcançar a peça, vamos considerar que a peça já está mais alta, no valor da diferença de altura entre a Garra e o Pulso. Com isso, podemos desconsiderar a mão e a Garra dos cálculos.

Figura 12. Visualização lateral do braço robótico virtual, com aplicação da poda para os cálculos



Fonte: Autor

Além disso, a altura do ombro possui um valor constante, o que permite que façamos o mesmo processo da Mão. Portanto, o cenário virtual sofrerá algumas podas para facilitar os cálculos, resultando numa estrutura virtual que tenha um braço robótico com apenas braço e antebraço, como mostrado na *Figura 12*.

Com a redução dos componentes do braço robótico é possível utilizar a fórmula de cinemática direta, mostrada pela Figura 13. Com essa fórmula encontraremos os valores dos ângulos do ombro (α) e cotovelo (β), a partir de um ponto com coordenadas espaciais (x, y, z). Os valores de L1 e L2 correspondem ao comprimento dos elos braço e antebraço.

Figura 13. Fórmula de Cinemática Direta

$$\alpha = ArcCos\left(\frac{x}{\sqrt{x^2 + y^2}}\right) - ArcCos\left(\frac{x^2 + y^2 + L_1^2 - L_2^2}{2L_1\sqrt{x^2 + y^2}}\right)$$

$$\beta = 180 - ArcCos\ \frac{L_1^2 + L_2^2 - (x^2 + y^2)}{2L_1\sqrt{x^2 + y^2}}$$

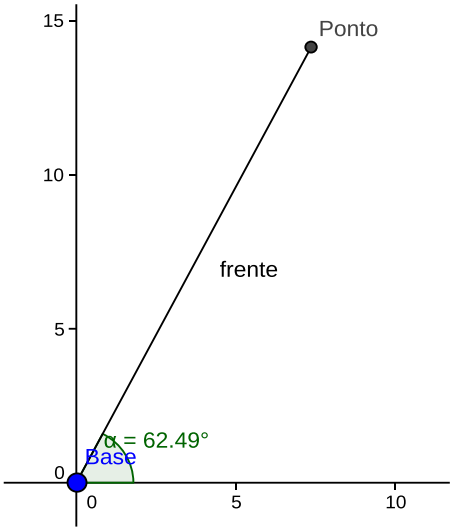
$$\gamma = 270 - \alpha - \beta$$

Fonte: (SANTOS, 2004)

Observe que essa fórmula possui apenas duas coordenadas: eixo Ox e Oy. Esta é aplicada com a visão de um braço em duas dimensões com visão lateral. Considerando que a base é fixa e não permite fazer rotações, teremos a seguinte correspondência: o eixo Ox da fórmula é direcionada à referência do eixo Oy do braço robótico; e o eixo Oy é direcionado à referência do eixo Oz do braço robótico. Neste caso, o eixo Ox do braço é ignorado considerando que a base não faz rotação, então não possível alcançar qualquer valor no eixo Ox diferente de zero.

A visão que temos do modelo esquelético na Figura 11, é uma visão lateral. Como o braço robótico move-se nas três dimensões, é preciso inserir no cálculo as dimensões latitudinais. Após compreendermos a aplicação da fórmula com o braço fixo sem rotação da base, temos que considerar a adaptação da fórmula com este recurso de rotação. Considerando a rotação da base, obteremos valores para o eixo Ox do braço robótico.

Figura 14. Visão por cima do braço robótico



Fonte: Autor

Quando for solicitado um ponto espacial (x, y, z), primeiramente vamos calcular o valor da rotação da base para posteriormente encontrarmos as demais angulações. Para o ângulo da base utilizaremos apenas os eixos Ox e Oy e calcularemos com trigonometria básica, como mostrado na Figura 14. Após isso, calcularemos a distância do ponto ao centro da base. Quando formos aplicar a fórmula de cinemática, substituiremos as variáveis y por essa distância.

3.6. Visão Computacional

Como visto, o sistema vai dispor de uma câmera para fazer a captura do tabuleiro após uma jogada humana, gerando assim uma imagem mostrando onde as peças se encontram. O XRL utiliza algumas técnicas de visão computacional para reconhecer a localização dessas peças a partir da imagem.

No momento em que o humano concluir a jogada, ele precisa sinalizar para o sistema. Essa sinalização pode ser um toque em alguma tecla do teclado, ou um botão separado, acompanhando o braço robótico. Essa sinalização pausa o tempo de jogada do humano e inicia o tempo de jogada do motor de xadrez e aciona a câmera. O sistema faz a captura do tabuleiro e armazena a imagem para iniciar o devido tratamento dela.

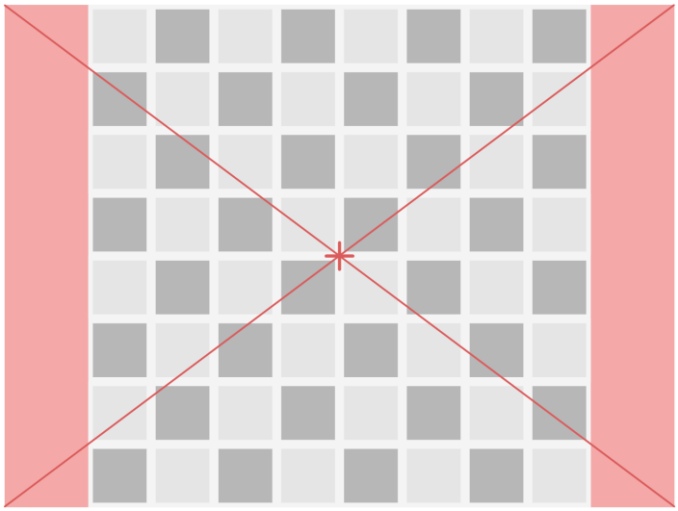
O sistema precisa transformar essa imagem em uma matriz de pontos binários, apresentando quais casas estão ocupadas no tabuleiro. Para isto, é necessário realizar uma série de operações na imagem:

- Cortar a imagem em proporção quadrada, recortando o tabuleiro;
- Dividir a imagem do tabuleiro em 64 partes, recortando cada casa;
- Dividir a imagem de cada casa em 9 partes, recortando o centro da imagem;
- Redimensionar o centro para a dimensão 1x1 pixels, gerando um ponto de cor;
- Verificar a proximidade das cores com as predefinições do tabuleiro;

Como mostra a Figura 15, a imagem deve atender aos seguintes requisitos:

1. O centro da imagem tem que coincidir com o centro do tabuleiro, ou próximo disso;
2. As bordas do tabuleiro devem coincidir com as bordas da imagem, ou próximo disso;
3. A casa *a1* deve aparecer na parte inferior esquerda.
4. As cores de casa clara, casa escura, peça clara e peça escura devem ser diferentes.

Figura 15. Modelo de imagem ideal para processamento



Fonte: Autor

A primeira modificação na imagem será o corte da imagem em um formato quadrado, independentemente do tamanho da imagem. Com isso, obtém-se a imagem ajustada nas proximidades das quatro bordas do tabuleiro. Portanto é possível realizar a divisão das casas de maneira uniforme.

Dividindo a imagem do tabuleiro em 8 fileiras e 8 colunas, obteremos 64 imagens de mesmo tamanho, com cada imagem portando uma casa. Nem sempre essas imagens terão as casas preenchidas perfeitamente, principalmente se não seguir as regras

descritas. Porém, a imagem pode sofrer algumas distorções por conta da lente da câmera e pelo fato de que a foto é tirada de apenas um ponto, podendo gerar efeito de perspectiva.

Além do erro que a imagem pode gerar, a parte mais significativa da imagem é a central, para indicar se tem peça ou não. Cortando a imagem em 9 partes, 3 fileiras e 3 colunas, e considerarmos apenas a quinta imagem, contando de baixo para cima e da esquerda para a direita, obteremos na imagem do centro e o erro da amostragem também será evitado, porque o desalinhamento acontece nas bordas das imagens.

Contudo, obteremos uma imagem representando cada casa, com suas bordas cortadas. Se a imagem seguir o quarto requisito, esta apresentará uma imagem com uma cor quase uniforme. Feito isso, iremos redimensionar cada imagem dessas para a resolução de 1x1 pixels. Essa transformação irá calcular a média de todos os pontos, resultando em uma só cor. Iremos definir as constantes de cores do tabuleiro com os seguintes nomes:

- Cor da peça clara;
- Cor da peça escura;
- Cor da casa clara;
- Cor da casa escura.

Como o resultado das operações de imagem resultam num pixel representando uma cor, podemos fazer a comparação dessa cor com as constantes de cores padrões. Geralmente, uma cor é representada pelo sistema RGB, que armazena os valores de intensidade das cores vermelho, verde e azul numa matriz tridimensional, numa escala de 0 a 255, numa representação de 24 bits.

Figura 16. Fórmula de distância entre duas cores

$$d = \sqrt{(R_b - R_a)^2 + (B_b - B_a)^2 + (G_b - G_a)^2}$$

O valor da distância entre duas cores (A e B), podem ser encontradas usando a formula de distância entre dois pontos tridimensionais, como mostra a Figura 16. A cor constante mais próxima da amostra resultante do processamento anterior é a que possui menor distância.

Como a saída esperada é uma matriz de posições ocupadas, para todas as posições que estiverem com proximidade maior de cor de peça (clara ou escura), será retornado o valor um, caso contrário, valor zero.

3.7. Reconhecimento de Movimentos

Como já temos o estado do jogo, só precisamos identificar qual foi a peça movida. O problema aqui é determinar pela posição qual lance foi efetuado e informar ao módulo UCI e Gerenciamento do Tabuleiro para que a jogada seja registrada.

Todas as partidas de xadrez iniciam-se exatamente iguais, como mostrado na Figura 2, e cada jogador pode movimentar uma vez por rodada. Tais condições contribuem para que o sistema inferir a última modificação do tabuleiro com apenas duas matrizes de alocação.

Uma matriz de alocação é uma estrutura que marca quais casas estão ocupadas, assumindo verdade (true) na célula que contiver peça na casa correspondente ou negação (false) quando não houver.

Figura 17. Matriz de modificações

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Matriz anterior

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1

Matriz atual

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0

Matriz modificações (XOR)

Fonte: Autor

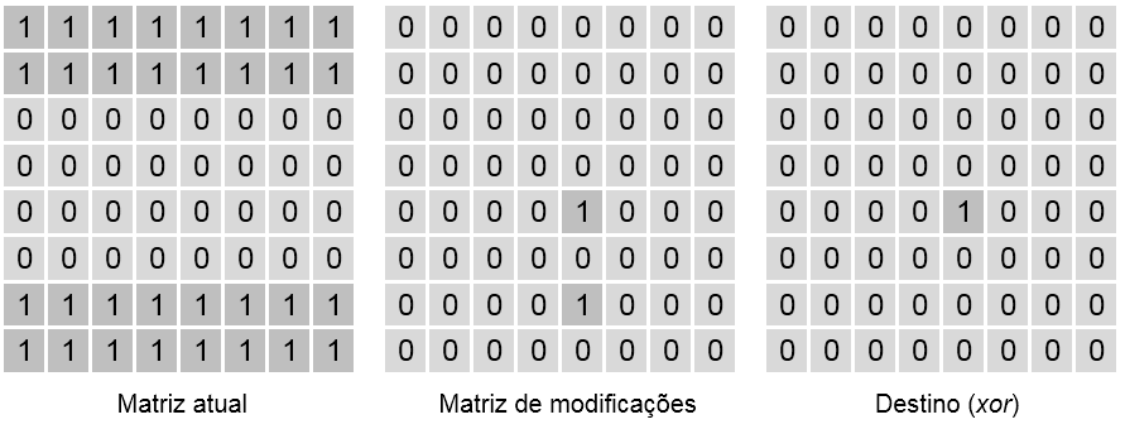
Quando um novo estado é gerado (matriz), ele pode ser comparado com o antigo estado utilizando a operação lógica *XOR*, que é uma operação que deixa ativo o que tem de diferente nas duas matrizes, gerando então uma matriz de modificações, como mostra a Figura 17. Apenas com essa matriz resultante não é possível definir a peça de origem ou destino do movimento.

Para obter o movimento de destino é realizado outra operação lógica, o *and,* que resulta uma matriz que representa a intercessão das duas matrizes de entrada, que

no exemplo são: a matriz atual e a matriz de modificações, como é exemplificado na Figura 18.

Para obter o movimento de origem basta fazer a mesma operação, porém, ao invés de usar a matriz atual para a comparação, vamos usar a matriz anterior. O resultado será bastante parecido com o movimento de destino, gerando uma matriz com um único elemento na matriz marcado como verdadeiro, alterando apenas pelo seu significado, sendo a identificação do movimento de origem.

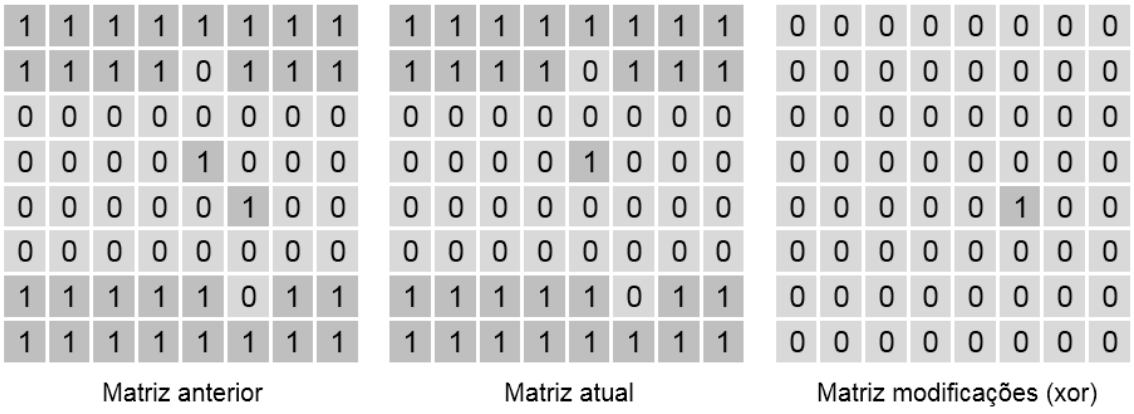
Figura 18. Identificação do movimento de destino



Fonte: Autor

Logo, quando um movimento é realizado sem que uma peça fique sobre a outra, obtemos da matriz de modificações uma matriz com duas células ativas. Quando aplicando as operações lógicas de interseção obtemos a origem e o destino do movimento, como é indicado na Figura 19. Caso o movimento realizado seja do tipo captura, a situação muda, resultando que uma peça vai ocupar o lugar de outra peça. Neste caso, a matriz de modificações conterá apenas uma célula ativa, a posição anterior da peça. Como no jogo de xadrez, as peças possuem movimentos específicos relacionados ao tipo, podemos inferir o destino da peça.

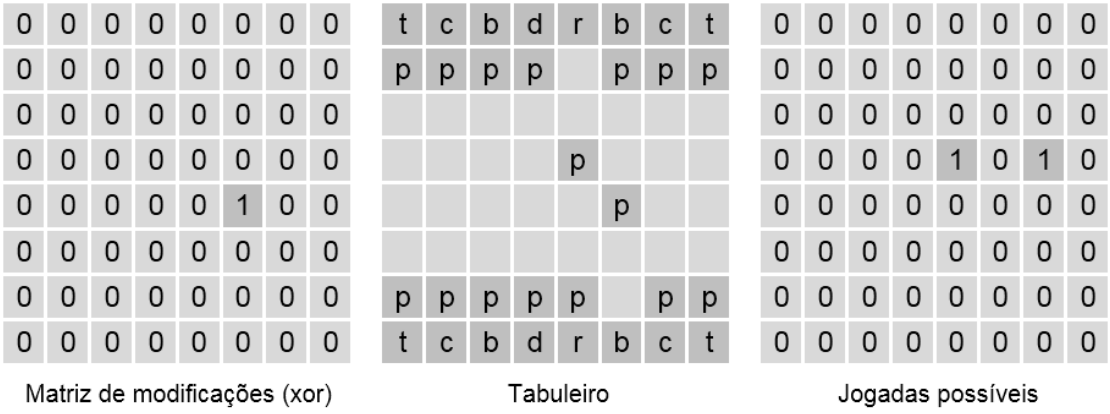
Figura 19. Identificação de movimento de ataque



Fonte: Autor

Para descobrir o destino de uma peça em uma jogada de ataque, é necessário descobrir primeiramente o tipo de peça que foi identificada na origem do ataque. Para isso, basta realizar a intercessão da matriz anterior com matriz de modificações. Suponha que a localização e identificação de todas as peças do tabuleiro são informações conhecidas, como indicado na Figura 20. Podemos aliar estas informações com uma matriz de movimentos possíveis da peça e obter então uma matriz de modificações.

Figura 20. Identificação das possíveis jogadas de um estado



Fonte: Autor

Na Figura 20 encontramos a matriz de modificações da movimentação de ataque e uma segunda matriz que representa a identificação das peças armazenadas na memória do sistema. Utilizando a operação lógica *and*, encontramos a localização e identificação e localização da peça ofensiva. Com a identificação da peça, podemos saber o comportamento desta e então fazer a verificação dos movimentos possíveis. Utilizando funções específicas das peças, podemos obter todos os movimentos

possíveis de uma peça de acordo com sua localização e neste caso, vamos precisar apenas dos movimentos de ataque – o peão é a única peça que diferencia movimentos comuns de movimentos de ataque – para então inferir qual foi o destino da peça.

No exemplo da Figura 20, foi identificada a peça de ataque (peão na posição f4). De acordo com as regras modernas do xadrez, o peão pode atacar uma peça à frente em uma de suas diagonais [e5, g5]. Como foi um movimento de ataque, basta realizar uma interseção entre a matriz atual e a matriz de jogadas possíveis da peça, que vai resultar numa matriz contendo apenas a célula e5 marcada como verdadeira. Um movimento em um motor de xadrez é caraterizado por um comando de movimento identificando apenas origem e destino, então todos os elementos necessários para enviar o comando já estão disponíveis. Exemplo de comando de movimento para motor de xadrez: “move f4e5”.

Figura 21. Captura não identificada

Fonte: Autor

Observando a situação da Figura 21 acima, a operação lógica aplicada pode gerar ambiguidade quando houve um movimento de captura. Isso acontece porque uma peça ficará sobreposta a outra e a operação *xor* não identifica a modificação. Em alguns casos, esse problema pode ser resolvido identificando a peça que realizou o movimento e analisar quais os possíveis movimentos da peça e então inferir o movimento realizado. Contudo, nem todos os casos estarão livres de ambiguidade, como é o caso do exemplo anterior. De acordo com a matriz anterior, podemos identificar que foram realizados movimentos apenas com os peões e a peça que foi movimentada foi o peão que estava na célula e5, deixando a incógnita sobre qual dos movimentos destino da peça. De acordo com a movimentação válida do peão, ele pode mover-se uma casa à frente ou uma diagonal para frente se for um movimento

de captura. Note que o peão pode realizar o movimento de captura a qualquer uma das suas diagonais da frente. Nenhuma dessas capturas serão identificadas pela matriz de modificações, necessitando de interferências externas para a informação ser encaminhada.

4. Avaliação da plataforma por meio de um protótipo

O trabalho foi dividido em vários módulos, em cada módulo são aceitas determinadas entradas e do qual devem retornar saídas que deverão ser usadas como entradas de outros módulos. As conexões entre módulos estão descritas na Figura 10. Para apresentação dos testes, serão mostrados alguns trechos de código mostrando o funcionamento, em seguida será verificado se as saídas são compatíveis as entradas e o objetivo da função. Todos os módulos foram testados.

4.1. Gerenciamento do Tabuleiro

Para mapear é preciso definir qual estrutura de dados será usada. Considerando que teremos mapas de bits para localização de peças vindas do módulo que recebe a imagem do tabuleiro real, então toda a estrutura do trabalho será com orientação centrada nas peças com mapas de bits. Trata-se de uma estrutura relativamente simples, onde 64 bits são suficientes para mapear as peças em suas casas, porém, sem nenhuma informação sobre o tipo ou coloração da peça.

Figura 22. Estrutura de Dados do tabuleiro

```
#define BRANCO  0
#define PRETO   1
#define PEAO    2
#define TORRE   3
#define CAVALO  4
#define BISPO   5
#define RAINHA  6
#define REI     7
#define casasOcupadas ((pecas[BRANCO] | pecas[PRETO]))

typedef unsigned long Mapa;

Mapa pecas[8] = {
  0x000000000000FFFF, //brancas
  0xFFFF000000000000, //pretas
  0x00FF00000000FF00, //peoes
  0x8100000000000081, //torres
  0x4200000000000042, //cavalos
  0x2400000000000024, //bispos
  0x0800000000000008, //rainhas
  0x1000000000000010  //reis
};

struct Casa{
  int fileira;
  int coluna;
};

struct Movimento{
  Casa origem;
  Casa destino;
};
```

Fonte: Autor

A estrutura de dados que possui 64 bits na linguagem de programação C/C++ é o long, que no nosso programa foi reescrito como "Mapa", como é indicado na Figura 22. Com isso, foi usada uma lista para representar todas as peças do tabuleiro, onde o índice 0 desse *array* representa o mapa de peças brancas; o índice 1, o mapa das peças pretas; o índice 2 o mapa das peças do tipo Peão; o índice 3 com o mapa das peças do tipo Torre; e assim por diante.

Para fazer o registro dos movimentos, foram utilizadas as estruturas de dados Casa e Movimento, como mostra a *Figura 23*.

Figura 23. Código do registro de jogada

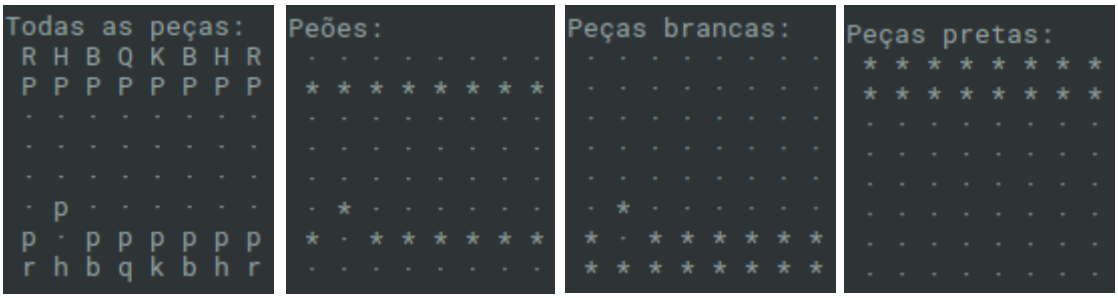
```
void registrarMovimento(Movimento movimento){
  Mapa origem = mapear(movimento.origem);
  Mapa destino = mapear(movimento.destino);

  int atacante = tipoPeca(movimento.origem);
  int cor = corPeca(movimento.origem);

  pecas[atacante] |= destino;   //uniao
  pecas[cor] |= destino;        //uniao

  for(int tipo = BRANCO; tipo <= REI; tipo++){
    if(!(tipo == atacante || tipo == cor))
      pecas[tipo] &= ~destino;    // diferenca (A-B)
    pecas[tipo] &= ~origem;         // diferenca (A-B)
  }
}
```

Fonte: Autor

Na função "registrarMovimento", nota-se que são realizadas várias operações binárias para gerar o novo tabuleiro em função do movimento de entrada. Nessa estrutura, caso o movimento em questão seja de um peão do time branco, é necessário alterar o mapa dos peões e o mapa das peças brancas como peça ativa na posição do movimento de destino. Para fazer essa alteração, basta fazer uma operação de união entre os o mapa da peça que ataca e o mapa que indica a posição de destino. Em todo o mapeamento de peças (peões, bispos, torres, etc.) terá o bit que representa a posição de origem limpo (zero), pois quando uma peça se movimento, ela deixa a sua antiga posição desocupada. Após a função de registrar o movimento, é possível verificar duas funções auxiliares que identificam qual é a cor e tipo de uma peça em determinada casa do tabuleiro (parâmetro).

Para testar o funcionamento dessa função, foi criado um movimento predefinido, com jogada de origem e destino selecionadas manualmente, e posteriormente analisados os mapeamentos de cada tipo. Na Figura 23 mostramos o resultado do código da Figura 22, com seus respectivos mapeamentos.

Figura 24. Resultado do teste de um movimento



Fonte: Autor

Para que os mapas fossem impressos na forma de uma matriz, já que sua estrutura é um "long", foi utilizada a função "*printMap*", marcando os pontos que não tem peça com ponto centralizado simples (•) e os pontos que têm peça com asteriscos (*). A orientação do tabuleiro segue conforme a Figura 1.

4.2. Cinemática

Para muitos, o módulo de cinemática pode ser o mais difícil de entender porque trabalha diretamente com as fórmulas descritas no capítulo 3.5. Para a ativação de um movimento, primeiramente é verificado se o braço robótico tem capacidade de alcançar o objeto desejado, pois todos os braços terão suas limitações físicas. Para isto, utilizaremos a função “dimensõesSuportadas”, como mostrado na Figura 25.

Figura 25. Código da verificação da capacidade do braço em alcançar uma casa

```
bool dimensoesSuportadas (Coordenada posicao) {
  float modulo = sqrt(pow(posicao.x, 2) + pow(posicao.y,2));
  base = acos(posicao.x / modulo) * 180 / M_PI;

  float y = modulo; //comprimento da frente, observando de lado
  float alturaGarra = posicao.z + COMPRIMENTO_MAO;
  float distancia = sqrt(pow(y, 2) + pow(alturaGarra - ALTURA_BASE, 2));

  //distancia base ao pulso.
  float distanciaBasePulso = sqrt(pow(alturaGarra, 2) + pow(y, 2));

  float parte1 =
    acos(( -pow(COMPRIMENTO_ANTEBRACO, 2)
    + pow(distancia, 2) + pow(COMPRIMENTO_BRACO, 2))
    / (2 * distancia * COMPRIMENTO_BRACO));
  float parte2 =
    acos((-pow(distanciaBasePulso, 2) + pow(distancia, 2)
    + pow(ALTURA_BASE, 2)) / (2 * distancia * ALTURA_BASE));

  ombro = (parte1 + parte2) * 180 / M_PI -90;

  float cimaVirtual = posicao.z - (ALTURA_BASE - COMPRIMENTO_MAO);

  cotovelo = acos(
    (pow(COMPRIMENTO_BRACO, 2) + pow(COMPRIMENTO_ANTEBRACO, 2)
      - (pow(y,2) + pow(cimaVirtual, 2)))
    / (2 * COMPRIMENTO_BRACO * COMPRIMENTO_ANTEBRACO) ) * 180 / M_PI;
  pulso = 270 - ombro - cotovelo;

  return !(isnan(ombro) || isnan(cotovelo) || isnan(pulso) || isnan(base));
}
```

Fonte: Autor

Esta função apenas mostra se o braço robótico é capaz de realizar o movimento, salvando as configurações dos ângulos em variáveis globais. Caso o movimento seja possível, é necessário chamar outra função que retorna os ângulos calculados. Caso

o braço não seja capaz de realizar o movimento, é retornado a última configuração válida.

4.3. Controle de Movimentos

Nesse módulo serão realizadas 3 etapas: identificar a localização das casas com coordenadas baseadas no mundo físico, definir a rota que o braço irá percorrer e definir pontos de intercalação para que o braço não faça movimento bruscos.

Primeiramente, é necessário definir as constantes referentes às dimensões dos componentes do braço robótico, e assim poder definir o quê o braço alcança. Após isso, é necessário definir as dimensões do tabuleiro, assim como sua distância para com o braço robótico. Como as dimensões estão sendo escritas como constantes no código, uma nova compilação será necessária, caso as dimensões do braço robótico ou do tabuleiro sejam alteradas.

Para definir a posição de cada casa do tabuleiro, primeiro é necessário calcular a localização da casa "a1" e as demais serão calculadas de acordo com a posição desta. Na *Figura 26*, é mostrado como encontrar a localização da casa a0, e posteriormente selecionando qualquer uma das demais casas, adicionando a comprimento de uma casa do tabuleiro. Neste caso, estamos considerando um tabuleiro com casas de 5.5 x 5.5 cm.

Figura 26. Calculo da posição de acordo com uma casa do tabuleiro.

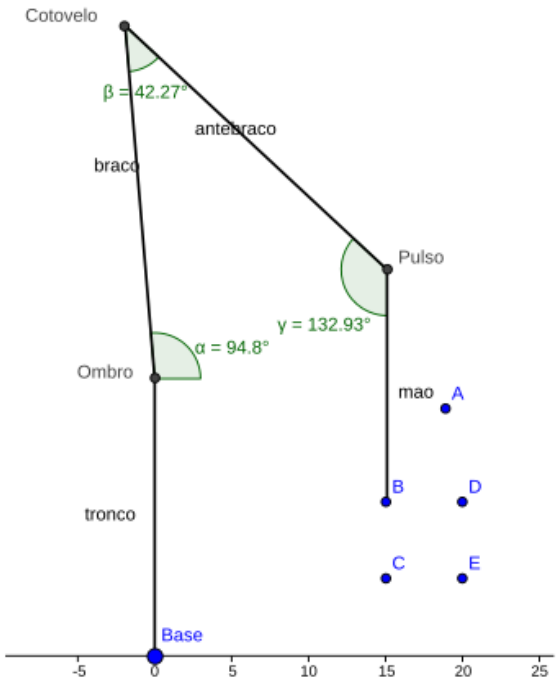
```
Coordenada localizar(Casa casa, bool high){
  Coordenada posicao; //colocando a peca na posicao a0
  posicao.x = TAMANHO_CASA * -3.5;
  posicao.y = DISTANCIA_TABULEIRO_EO_ROBO + TAMANHO_BORDA + TAMANHO_CASA / 2;
  posicao.z = high ? ALTURA_CIMA : ALTURA_BAIXO;

  //colocando na posicao de acordo com casa
  posicao.x += casa.coluna * TAMANHO_CASA;
  posicao.y += casa.fileira * TAMANHO_CASA;
  return posicao;
}
```

Fonte: Autor

Em seguida é necessário encontrar a rota que o braço precisa fazer para realizar a jogada vinda do motor de xadrez sem derrubar as peças. Com um braço robótico com a estrutura descrita nesse trabalho, ele sempre irá pegar as peças por cima e descer verticalmente até alcançar. Depois que a peça é capturada, o braço deve subir a uma altura suficiente para movimentar-se horizontalmente sem colidir com nenhuma peça. O movimento que a ponta da garra fará seguirá uma rota semelhante com a

representada na Figura 27, seguindo os seguintes cursos: AB, BC, CB, BD, DE, ED, DA.

Figura 27. Visualização do esqueleto do xadrez, com a ponta da garra no ponto B



Fonte: Autor

Essa representação da *Figura 27*, é adaptada para o código abaixo, com as variáveis posição inicial, origem alta, origem, destino alto e destino, representando respectivamente os pontos A, B, C, D e E.

Figura 28. Definição dos pontos da rota

```
void mover(Movimento movimento){
  Coordenada posicaoInicial, origem,
   origemAlta, destino, DestinoAlto;

  posicaoInicial.x = X_INICIAL;
  posicaoInicial.y = Y_INICIAL;
  posicaoInicial.z = Z_INICIAL;

  origem = localizar(movimento.origem, false);
  origemAlta = origem;
  origemAlta.z = ALTURA_CIMA;

  destino = localizar(movimento.destino, false);
  DestinoAlto = destino;
  DestinoAlto.z = ALTURA_CIMA;
```

Fonte: Autor

Depois de gerados os 5 pontos do movimento, é necessário gerar os pontos que estão entre dois vértices para gerar o movimento gradual. E para cada ponto que gerar, para

fazê-lo executar para o braço, basta usar a função descrita na *Figura 28* considerando os ângulos de cada junta do braço robótico.

4.4. Visão Computacional

Como descrito na Seção 3.6, é necessário realizar 5 etapas para que uma imagem seja transformada em um mapeamento de peças. A Seção descreve, também, uma série de requisitos para a imagem. O processo de tratamento na imagem será todo realizado por meio das ferramentas disponibilizadas no *opencv*.

A primeira etapa consiste em cortar a imagem em uma proporção quadrada, ajustando suas dimensões às dimensões menores. Por exemplo, se a imagem estiver no formato paisagem (largura maior que altura), a largura terá sua nova dimensão conforme o valor da altura da imagem. Se a imagem estiver no formato retrato (altura maior que largura), a altura terá as dimensões modificadas para o valor da largura.

Figura 29. Código enquadramento de Imagem

```
Mat enquadrarImagem(Mat imagem){
  Rect corte;
  corte.width = imagem.size().width;
  corte.height = imagem.size().height;

  corte.x = corte.width - corte.height ;
  corte.x = max(corte.x / 2, 0);
  corte.y = corte.height - corte.width;
  corte.y = max(corte.y / 2, 0);

  corte.width = min(imagem.size().width, imagem.size().height);
  corte.height = min(imagem.size().width, imagem.size().height);
  return imagem(corte);
}
```

Fonte: Autor

Mat é o tipo de dados usado no *opencv* para representar uma imagem. O código mostrado na Figura 29 apresenta como é feita a identificação da dimensão maior da imagem e configura a máscara de corte e sua aplicação. Esta imagem irá passar um processo de divisão em partes iguais. Num tabuleiro na proporção, gera casas com proporções quadradas. Quando é feito esse corte, as bordas da imagem se aproximam das bordas do tabuleiro, em todos os lados, mas nem sempre esse corte irá gerar um encaixe perfeito, dependendo da posição da câmera.

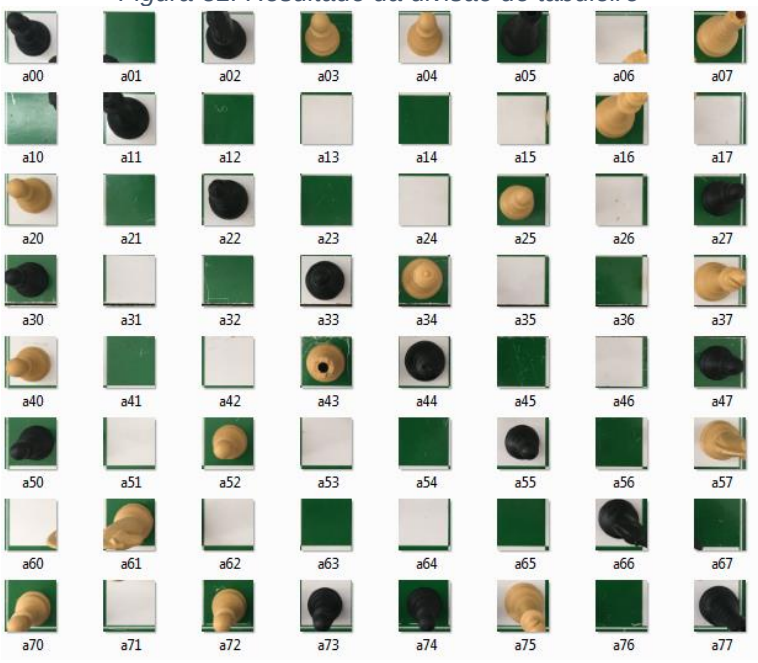
Figura 30. Código de divisão de uma imagem

```
Mat dividirImagem(Mat imagem, int quantidade, int fileira, int coluna){
  Rect corte;
  fileira = quantidade - 1 - fileira;
  corte.width = imagem.size().width / quantidade;
  corte.height = imagem.size().height / quantidade;
  corte.x = coluna * corte.width;
  corte.y = fileira * corte.height;
  return imagem(corte);
}
```

Fonte: Autor

Após o corte da imagem em formato quadrado, é preciso dividir a imagem em 64 partes para representar as casas. A divisão é simétrica, então se o corte do tabuleiro não for perfeito, o resultado implicará no resultado dos cortes de cada casa, podendo gerar um corte a invasão de outra casa não correspondente, como é possível visualizar na Figura 31. Porém, este problema pode ser resolvido se desconsiderarmos as bordas de cada casa, obtendo apenas o centro.

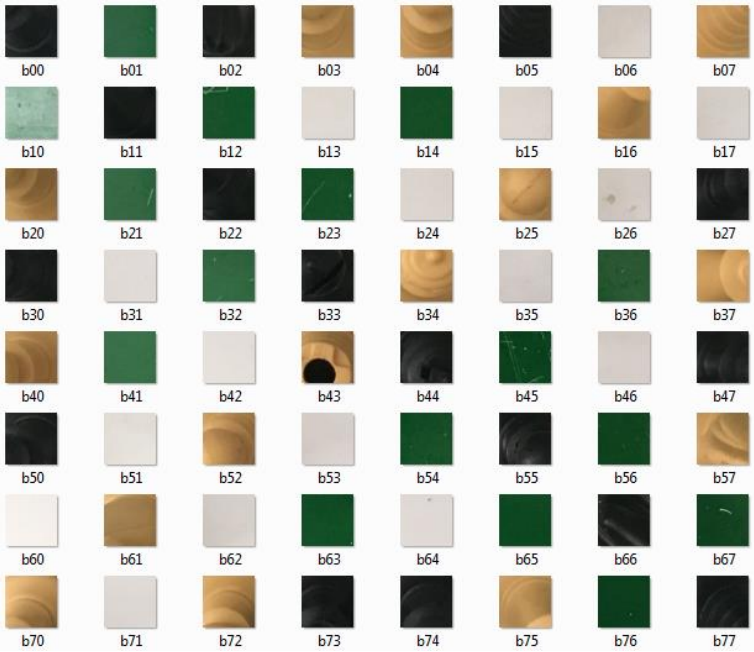
Figura 31. Resultado da divisão do tabuleiro



Fonte: Autor

Capturando o centro da imagem da casa, não é possível identificar a forma da peça, mas é possível identificar a cor da peça, ou da casa, caso não tenha peça. Com as informações da cor, podemos gerar 3 mapas: mapa das peças brancas, mapa das peças pretas e em razão das anteriores, o mapa das casas ocupadas. Esses mapas serão utilizados no módulo Reconhecimento de Movimentos, por isso não abordaremos nesta seção o seu uso.

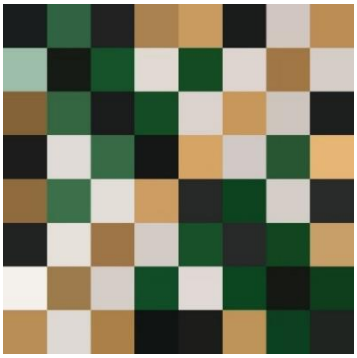
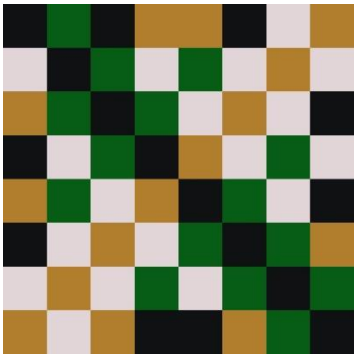
Figura 32. Resultado de um recorte do centro das casas



Fonte: Autor

Para a identificação da peça, é necessário classificá-la com base em alguma característica. Como em cada casa houve uma aproximação, não é possível identificar a forma da peça, essa classificação será feita com base nas cores, sendo que neste tabuleiro, usaremos o dourado e preto como cores que significam que há alguma peça na casa. As demais cores (verde e branco) serão considerados como ausência de peça.

Figura 33. Média de Cores das casas/peças do tabuleiro

(a)  (b) 

Fonte: Autor

As cores capturadas pelo tabuleiro geram uma média por casa, conforme a Figura 33-a. Note que as cores da amostragem não apresentam uniformidade. É necessário passar um filtro para aproximação de cores, com base nas constantes que definimos para casa clara, casa escura, peça clara e peça escura. A Figura 33-b mostra o

resultado após a aplicação do filtro da aproximação das cores, utilizando a distância entre dois pontos como barema de classificação. Na figura, podemos perceber que nem todas as peças foram identificadas como deveriam, como a casa h5, que aproximou a cor para o branco, invés de aproximar para o dourado. Nota-se que a aproximação por média resolve boa parte dos problemas, porém uma variação muito grande de iluminação no tabuleiro, pode ocasionar alguns erros. Para resolver isso, podemos melhorar a cor de amostragem de seleção das cores por uma que represente melhor as peças do tabuleiro. Toda vez que for iniciar uma partida, é importante fazer uma média das quatro cores significativas do tabuleiro para ter melhor aproximação.

Após essa identificação de peças claras e escuras, será gerado um mapeamento, na estrutura de um long *int*, para repassar para o módulo Reconhecimento dos Movimentos.

4.5. Reconhecimento dos Movimentos

No módulo de reconhecimento de movimentos, ocorre o processo de inferência das peças que foram movimentadas. Como visto na seção 3.7, precisamos das informações sobre os possíveis ataques de cada peça. Cada tipo de peça do tabuleiro movimenta-se de forma diferente. Logo, para gerar inferências, é necessário um mapeamento das possíveis jogadas das peças.

As peças do tipo Torre, Bispo e Rainha movimentam-se de forma de corredor. Elas percorrem um percurso até encontrar um obstáculo. Para isto, foram criadas as funções auxiliares para posteriormente serem unidas de acordo com o tipo de peça:

- Ataques ao norte;
- Ataques ao nordeste;
- Ataques ao leste;
- Ataques ao sudeste;
- Ataques ao sul;
- Ataques ao sudoeste;
- Ataques ao oeste;
- Ataques ao noroeste.

Essas funções percorrem um caminho, de acordo a sua orientação e param quando encontrar algum obstáculo ou fim do tabuleiro. Cada uma dessas funções gera um mapeamento de jogadas possíveis. As jogadas possíveis da Torre são a união dos mapeamentos adquiridos das funções de ataque a: norte, oeste, sul e leste; o Bispo: nordeste, sudeste, sudoeste e noroeste; a rainha une os ataques da torre e do Bispo. A Figura 34 mostra a implementação dessas

jogadas utilizando um vetor para direcionar a jogada. Caso a direção seja para o norte, é preciso criar um vetor v (0,1), para o nordeste, um vetor v(1,1), para o oeste, v(1,0) e assim por diante. As demais peças são criadas com funções individuais.

Figura 34. Implementação do ataque direcionado

```
Mapa ataquesDirecionado (Mapa ocupadas, Casa casa, Vetor vector){
  int escala = 1;
  Mapa proximoAtaque, ataques = 0;

  do{
    proximoAtaque = mapear(casa.fileira + vector.y * escala,
                           casa.coluna + vector.x * escala);
    ataques |= proximoAtaque;
    escala++;
  }while((proximoAtaque & ocupadas) != proximoAtaque);
  //continua enquanto não encontrar obstaculo ou não sair do tabuleiro

  return ataques;
}
```

Fonte: Autor

Quando uma jogada for realizada pelo humano, este módulo receberá um mapeamento das peças. Quando ocorrer jogadas comuns sem ataques a matriz de modificações terá dois bits ativos. Neste caso, basta verificar a conjunção das modificações com as casas livres. Caso seja um movimento de ataque, precisamos verificar as jogadas possíveis de uma peça e fazer uma conjunção com as peças do oponente, como mostra a Figura 35.

Figura 35.Código de inferir movimento

```
Movimento descobrirMovimento(Mapa novaOcupacao){
  Mapa modificacoes = casasOcupadas ^ novaOcupacao;
  Mapa origem = modificacoes & casasOcupadas;
  Mapa destino;
  if(contar(modificacoes) == 2){
    destino = modificacoes & ~casasOcupadas;
  }
  else{
    int cor = corPeca(origem);
    int oponente = 1 - cor;
    destino = ataquesDisponiveis(origem) & pecas[oponente];
  }

  Movimento movimento;
  movimento.origem = localizacao(origem);
  movimento.destino = localizacao(destino);
  return movimento;
}
```

Fonte: Autor

Após a descoberta da jogada, o programa deve repassar esse movimento para o módulo Controle do Tabuleiro, para registrar a jogada internamente, e para o módulo UCI para fazer a conversão do movimento para texto, e então ser transmitido para a GUI.

5. Considerações Finais

O desenvolvimento deste trabalho permitiu integrar motores de xadrez com braços robóticos genéricos. Também foi possível verificar as várias etapas que compõem esse processo, com os algoritmos para a conversão das diferentes linguagens através das quais os componentes se comunicam. Além disso, permitiu indicar novas frentes de pesquisa, já que o trabalho deixa alguns campos em aberto para futuras melhorias, mas oferecendo uma base mais consistente das etapas do processo, melhorando o grau de conhecimento em computação dos profissionais que o utilizarão.

A comunidade acadêmica de computação ganha uma ferramenta para testar e executar algoritmos de inteligência artificial para motores de xadrez. Para o aluno, é uma plataforma para visualizar o fenômeno do xadrez na prática, com mais interação do que com uma interface gráfica.

De outro lado, a comunidade de robótica também ganha uma ótima ferramenta para aplicações para braços robóticos. Estudantes que aprenderem a controlar um braço robótico, podem usar a plataforma para colocar o braço robótico para jogar xadrez e ainda escolher um motor de xadrez que for conveniente.

O fato de o jogador de xadrez ser um agente robótico proporciona um maior interesse em jovens cientistas, pelo fato de movimentar peças do mundo físico. O interesse desses jovens pode aumentar a comunidade de cientistas em computação, onde, com mais pessoas trabalhando no tema pode trazer avanços para a área.

Ao fazer um teste em ambiente fechado, verificou-se que as partes mais complexas e desgastantes do processo são as que envolvem o processamento de imagem, pois é a parte do projeto que não oferece certeza no processamento. As imagens podem apresentar ruídos que impeçam o programa de identificar corretamente as peças do tabuleiro. Os demais componentes do programa

oferecem determinismo, por isso, as dificuldades apresentadas nesses módulos foram de implementação.

Analisando os resultados gerados no projeto, foi possível perceber que esse produto com essa arquitetura é incapaz de reconhecer a promoção do peão. A promoção do peão acontece quando um peão alcança o outro lado do tabuleiro, podendo trocar o tipo da peça para uma Dama, Torre, Bispo ou Cavalo. Como o processamento de imagem reconhece a localização topológica da peça quando ocorre um movimento, no caso da promoção, não haverá alteração topológica no tabuleiro (a casa que era ocupada pelo peão, continuará ocupada por uma peça do mesmo time), impossibilitando então uma inferência. Fica sugerido que o problema da promoção do peão seja solucionado com intervenção humana ou realizando a identificação da peça por processamento de imagem, baseando-se na forma da peça para fazer a diferenciação.

Apesar da arquitetura oferecer suporte, este trabalho deixa como sugestão para trabalhos futuros, o mecanismo de identificação de jogadas inválidas por parte do humano, por não haver tempo hábil de implementar. O sistema poderia identificar quando o humano realizasse uma jogada que não condiz com o comportamento das peças do xadrez e solicitar um comando para que o braço fizesse a correção no tabuleiro indicando que a jogada foi inválida. Esta funcionalidade permitiria que jogadores aprendizes jogassem com este sistema, permitindo a implantação em escolas de xadrez.

No módulo da cinemática, ocorreram alguns problemas quanto à conversão de graus para radianos e vice-versa. Na descrição da fórmula, não são exibidos quais serão os resultados específicos que retornarão, como se tudo fosse trabalhado com graus, porém, a maioria é trabalhada em radianos. Em algumas operações era necessário realizar a conversão antes de multiplicar com outro número. Por isso, na aplicação da fórmula para o código, cada etapa foi realizada com muito cuidado, pois a fórmula é complexa demais para identificar erros se inserida inteira de uma vez só.

Por outro lado, trabalhar com operações binárias foi de imensa facilidade. Na linguagem de programação C/C++ existem as operações binárias clássicas,

permitindo que essas operações sejam feitas em variáveis inteiras de 64 bits num único ciclo de *clock* do processador (em processadores de 64 bits).

Com os resultados também é possível perceber que a identificação topológica não é totalmente correta. Dependendo das cores de amostra, a aproximação das cores usando distância entre pontos pode gerar uma aproximação de uma cor não desejada. Por exemplo, se tivermos duas amostras, cor preta e verde, e a cor em teste seja um verde escuro, há uma grande possibilidade dessa cor se aproximar do preto. Como os nossos olhos veem uma cor, é mais fácil perceber que mesmo escura, a cor ainda é verde. Isso acontece por que processamos a cor como ela realmente é: raios de luz. Cada cor oferece um comprimento de onda diferente. No computador, geralmente as cores são representadas pela mistura das cores vermelho, verde e azul.

As cores no sistema RGB formam um cubo de espectro de cores. Para solucionar o problema da aproximação das cores, é sugerido que essa aproximação seja feita com a comparação de outro componente: distância de um ponto com uma reta. Analisando o ponto da cor de amostra, ela pode formar uma reta com a base do cubo (cor preta). Assim, um tom de verde sempre terá a mesma distância do verde puro, mesmo se mudarem as condições de iluminação. Com esse mecanismo, quando as cores estiverem mais próximas do preto, serão mais difíceis de ser identificadas, porém ele resultará em mais eficiência do que a distância crua entre os pontos.

De modo geral, o XRL consegue desempenhar bem com os requisitos propostos. O reconhecimento de imagem consegue perceber o movimento que o humano realizou com a iluminação vindo da mesma direção da câmera. Nesse caso, a comunicação com a GUI apresenta ótimo funcionamento. A comunicação com o motor de xadrez funciona bem, como previsto em tempo de projeto. Problemas de precisão foram encontrados, porém este é um requisito a ser atendido pelo braço robótico escolhido, por este motivo, não foram tratados neste trabalho.

Referências

ABHICHANDANI, P. Electrical and Computer Engineering. **Drexel University**, 2013. Disponivel em: <http://drexel.edu/ece/academics/undergrad/senior-design/project-archive/2012-2013/>. Acesso em: 01 Agosto 2016.

BECKET, I. **Manual do Xadrez**. [S.l.]: NBL Editora, 1974.

BLUME, M. Welcome to Arena. **Arena Chess GUI 3.5.1**, 2016. Disponivel em: <http://www.playwitharena.com/?Welcome_to_Arena>. Acesso em: 05 Setembro 2016.

CRUZ, G. Ciência e Tecnologias. **A História da Robótica até os dias de hoje**, 2013. Disponivel em: <https://cienciaetecnologias.com/robotica-historia/>. Acesso em: 19 Julho 2016.

D'AGOSTINI, O. **Xadrez Básico**. São Paulo: Ediouro, 2002.

IBM. Deep Blue. **Overview of Deep Blue**, 2012. Disponivel em: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>. Acesso em: 07 Julho 2016.

KAHLEN, S. M. Wbec. **UCI protocol**, 2004. Disponivel em: <http://wbec-ridderkerk.nl/html/UCIProtocol.html>. Acesso em: 22 Agosto 2016.

KEMPEN, H. V. Chess Engines Grand Tournament. **CEGT Best Versions 40/20**, 2016. Disponivel em: <http://www.husvankempen.de/nunn/40_40%20Rating%20List/40_40%20Best Version/rangliste.html>. Acesso em: 06 Setembro 2016.

KEMPF, K. The Eniac. In: KEMPF, K. **Historical Monograph:** Electronic Computers Within the Ordinance Corps. Estados Unidos da América: Thelen, v. I, 1961. Cap. 2, p. 19-39.

LETOUZEY, F. GNU Chess 6.1.2. **GNU**, 2014. Disponivel em: <https://www.gnu.org/software/chess/manual/gnuchess.html#Overview>. Acesso em: 29 Agosto 2016.

MANN, T. Chess Engine Communication Protocol. **GNU**, 2009. Disponivel em: <https://www.gnu.org/software/xboard/engine-intf.html>. Acesso em: 27 Agosto 2016.

MARSLAND, T. A. Computer Chess and Search. **EDMONTON**, Canad, 2011.

ROMSTAD, T. Stockfish: About. **Stockfish**, 2016. Disponivel em: <https://stockfishchess.org/about/>. Acesso em: 25 Julho 2016.

SANTANA, H. V. M. D. Anatomia de um Motor de Xadrez, São Paulo, 2014.

SANTOS, V. M. F. **Robótica Industrial**. Aveiro, Portugal: Universidade de Aveiro, 2004.

SOLOMAN, C.; BRECKON, T. **Fundamentos de Processamento Digital de Imagens - Uma abordagem Prática**. [S.l.]: [s.n.], v. LTC Editora, 2013.

STALLMAN, R. Sobre o projeto GNU. **GNU**, 2015. Disponivel em: <https://www.gnu.org/gnu/thegnuproject.html>. Acesso em: 29 Agosto 2016.