Universidade Federal
do Rio de Janeiro
Escola Politécnica

Projeto e Desenvolvimento de um Tabuleiro Autônomo de Xadrez

Fabio Oliveira Baptista da Silva e Marcos Paulo Oliveira Silva

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Carlos José Ribas D'Ávila

Rio de Janeiro

Agosto de 2014

Projeto e Desenvolvimento de um Tabuleiro Autônomo de Xadrez

Fabio Oliveira Baptista da Silva e Marcos Paulo Oliveira Silva

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Autor: _____
Fabio Oliveira Baptista da Silva

Autor: _____
Marcos Paulo Oliveira Silva

Orientador: _____
Carlos José Ribas D'Ávila

Examinador: _____
Joarez Bastos Monteiro

Examinador: _____
Heraldo Luís Silveira de Almeida

Rio de Janeiro – RJ, Brasil
Agosto de 2014

Aos nossos familiares e amigos.

“O que prevemos raramente ocorre; o que menos esperamos geralmente acontece.” - **Benjamin Disraeli**

Agradecimentos

Agradeço primeiramente a minha irmã, minha mãe e meu pai, por todo o suporte, ajuda e conforto ao longo de toda a graduação. Ao meu tio Marcio por toda a ajuda na construção deste protótipo. Aos meus amigos envolvidos nesses cinco anos de graduação: Tati, Caio, Leonardo, Stephanie, Ewerton, Thays e Vitor. Aos meus amigos de fora da faculdade por me fazer esquecer dela de vez em quando, afinal foi numa dessas que surgiu a ideia deste protótipo, Tati, Caio, Lucas, Luis, Bruna e Pedro. Um agradecimento especial aos meus amigos que tiveram influência direta sobre este projeto como Gabriela e Pedro Paulo.

Ao professor Casé que além de orientador nos acompanha desde o nosso primeiro dia de aula. Ao professor Joarez por todo o suporte ao longo da construção da primeira versão deste protótipo na disciplina de projeto integrado. E principalmente ao Marcos por ter topado essa ideia no mínimo trabalhosa, por todo companheirismo desses anos de faculdade, principalmente nesse final, desculpe por te meter nessa.

Fábio

Bom, primeiramente queria agradecer a Deus, por todos os benefícios que me tem feito, pela segurança na adversidade, pela direção na incerteza, pela provisão no momento oportuno e pelo consolo na angústia. Toda glória a Ele.

Quero agradecer também aos meus pais por todo carinho, dedicação, sustento e exemplo que me permitiram chegar até aqui. Embora, a maturidade tenha me feito vê-los como de fato são, carne e osso, vocês sempre serão meus heróis, mesmo sem capa ou superpoderes, vocês são minha inspiração para seguir lutando. Não posso esquecer do meu irmão, que embora seja um grande chato, tem minha admiração pela sua força de vontade e alegria. Obrigado pelas risadas e pelos momentos de entretenimento.

Também gostaria de agradecer aos meus amigos do curso de Eletrônica: Gabriela, Nadinne, Léo, Joyce, Stephanie, Tati, Leonardo e Pedro Paulo pela amizade de cada um de vocês, pelo apoio, seja material ou afetivo, e incentivo nesses anos difíceis. Também dedico aos meus amigos e irmãos na fé: André Souteiro, Samuel Adonai, Priscila e Albert Maia, Felipe Milepe e Priscila Ribeiro por sua amizade, compreensão, consolo e orações.

Quero também agradecer ao professor Casé pela orientação neste trabalho e todos os conselhos ministrados nestes 5 anos de curso. Ao professor Joarez pela orientação no início desta empreitada. Ao professor Maurício Aredes por disponibilizar o LEMT para o desenvolvimento deste projeto e pelo apoio durante minha estadia no mesmo. Ao professor Mamour Sop Ndiaye por toda orientação, apoio e amizade que desenvolvemos nestes 3 anos de iniciação científica no qual trabalhamos. Também não posso deixar de agradecer a todo o pessoal do LEMT pelos momentos de descontração e pelo apoio e incentivo que me proporcionaram.

E por fim, porém não menos importante, agradeço ao Fábio, que embora tenha me metido nesta fria, me acompanhou praticamente todos estes 5 anos do curso. Obrigado por todo esclarecimento, companheirismo, descontração e paciência (ou não) com minhas insistentes brincadeiras.

Marcos Paulo

Resumo

Este trabalho consiste na elaboração e construção de um protótipo de um tabuleiro de xadrez autônomo. Integrando um arcabouço mecânico de três dimensões ortogonais, que realiza a movimentação por debaixo do tabuleiro com um servomecanismo que realiza a aproximação ou afastamento de um conjunto de imãs das peças no tabuleiro. Cada peça possui também um imã que serve tanto para sua movimentação quanto para identificação de sua localização espacial através de sensores de efeito *Hall*. Para controlar toda esse aparato e gerir a inteligência artificial deste projeto é utilizado um Arduino que se comunica através de uma interface *Bluetooth* com uma aplicação para Android.

Palavras-chave: Xadrez, Inteligência Artificial, Automação, Jogos, Android, Arduino, Comunicação Serial e Bluetooth

Abstract

This work is the development and construction of a prototype of an autonomous chessboard. Integrating a mechanical framework of three orthogonal dimensions, which performs the movement underneath the board with a servo which modifies the distance between a set of magnets and the pieces on the board. Each piece also has a magnet which serves both as movement as identification of their spatial location using Hall effect sensors. To control all this structure and manage the artificial intelligence of this Project is used an Arduino that communicates via a *Bluetooth* interface with an application for Android.

Keywords: Chess, Artificial Intelligence, Automation, Games, Android, Arduino, Serial Communication and Bluetooth

Siglas

ADT – *Android Development Tools*

AOSP – *Android Open Source Program*

AOT – *Ahead-Of-Time*

API – *Application Programming Interface*

EEPROM – *Electrical Erasable Programable Read-Only Memory*

FSF – *Free Software Foundation*

GPS – *Global Positioning System*

GSM – *Global System for Mobiles communications*

IDE – *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronics Engineers*

JIT – *Just-In-Time*

LCD – *Liquid Crystal Display*

MIPS – *Microprocessor without Interlocked Pipeline Stages*

OEM – *Original Equipment Manufacturer*

OHA – *Open Handset Alliance*

OSHW – *Open Source Hardware*

SDK – *Software Development Kit*

SMD – *Surface-Mount Device*

TSCP – *Tom Kerrigan's Chess Program*

USB – *Universal Serial Bus*

VM – *Virtual Machine*

Índices do Texto

Índices de Figura

Índices de Tabelas

1. Introdução

1.1. Conteúdo dos capítulos

Este trabalho foi construído de forma que o leitor possa acompanhar o desenvolvimento do projeto sob uma perspectiva de contrapor as demandas e as decisões. No primeiro capítulo é feita uma introdução sobre o projeto e sobre a inserção do projeto dentro da engenharia eletrônica e de computação, de forma a dar ao leitor uma visão geral.

Dado que o projeto é o desenvolvimento de um tabuleiro de xadrez, a primeira pergunta que vem à tona é sobre a inteligência artificial, assunto que será discutido no capitulo 2. Neste capitulo serão destacados os três principais algoritmos utilizados no projeto: algoritmo de xadrez, algoritmo de melhor caminho e o algoritmo de desvio.

No capítulo 3 será discutida a plataforma principal do projeto, o Arduino, dando uma noção do funcionamento geral e das interfaces do Arduino com o sistema mecânico além das interfaces com o usuário.

No capítulo 4 será abordado o Android com uma breve introdução explicando seu papel no projeto, características, origens e *o* kit de desenvolvimento de aplicativos, o ADT Bundle, assim como a aplicação desenvolvida para o projeto.

No capítulo 5 será abordada a comunicação entre o Arduino e o Android.

No capítulo 6 serão apresentados alguns resultados práticos, como a análise da movimentação dos eixos, teste dos algoritmos de movimentação, e uma breve análise da placa de leitura.

E por fim, no capítulo 7, serão apresentadas as conclusões acerca do projeto e os trabalhos futuros.

1.2. Tema

O tema do trabalho é o projeto e desenvolvimento de um tabuleiro de xadrez autônomo capaz de interagir com o usuário e jogar contra ele. A interação deve ser feita através de uma interface na qual o usuário possa definir algumas configurações básicas, tais como, a escolha de quais são suas peças. Pretende-se apresentar as etapas de elaboração e construção deste trabalho.

1.3. Delimitação

O objeto de estudo é a automação de um tabuleiro de xadrez. Este projeto contempla diversas áreas de formação do curso de Engenharia Eletrônica e de Computação, tais como:

- Circuitos de acionamento dos motores
- Circuito de sensores para a leitura das posições das peças
- Programação em ambientes embarcados e de baixo recurso
- Programação para dispositivos móveis
- Os algoritmos de inteligência artificial

1.4. Justificativa

O trabalho de construção de um protótipo de um xadrez automatizado permeia as noções de gerência de projetos, automatização de processos, projeto de sistemas e a integração de diversos conceitos apresentados ao longo do curso.

1.5. Objetivos

O objetivo geral é construir um tabuleiro de xadrez automatizado capaz de interagir com os usuários através de uma interface simples e que sirva de entretenimento, característica intrínseca aos jogos. Além disso, apresentar todo o processo decisório e das etapas de criação do projeto.

1.6. Descrição

Este trabalho utiliza um sistema mecânico com três graus de liberdade, sendo os eixos horizontal e vertical controlados por motores de passo, devido a sua confiabilidade e simplicidade (não há necessidade de sensores para a orientação quanto à posição das casas do tabuleiro, dada uma posição inicial).

Um servomecanismo, como terceiro grau de liberdade, responsável por segurar ou não peças através de imãs de neodímio-ferro-boro (imãs de terras raras) localizados tanto na base das peças quanto no eixo do servomecanismo, conforme pode ser visto na Figura 1 e na Figura 2.

Combinando o movimento dos eixos com a possibilidade das peças estarem ou não presas ao eixo do servomecanismo, pode-se realizar a movimentação das peças por baixo do tabuleiro de forma que nenhum aparato precise ficar exposto, aumentando assim a robustez do projeto.

Todo esse sistema é controlado por um Arduino. Arduino é um projeto aberto que integra *hardware* e *software* e facilita a implementação de projetos utilizando microcontroladores. O Arduino deve não só controlar os motores dos três eixos e gerir a inteligência artificial, mas também se comunicar com a interface disponível ao usuário em um dispositivo móvel com Android via *Bluetooth*.

O tabuleiro de referência possui 10x10 casas, ao invés do normal do xadrez que são 8x8, para que as peças que saiam do jogo possam permanecer no tabuleiro, sendo acumuladas ao redor do tabuleiro normal.

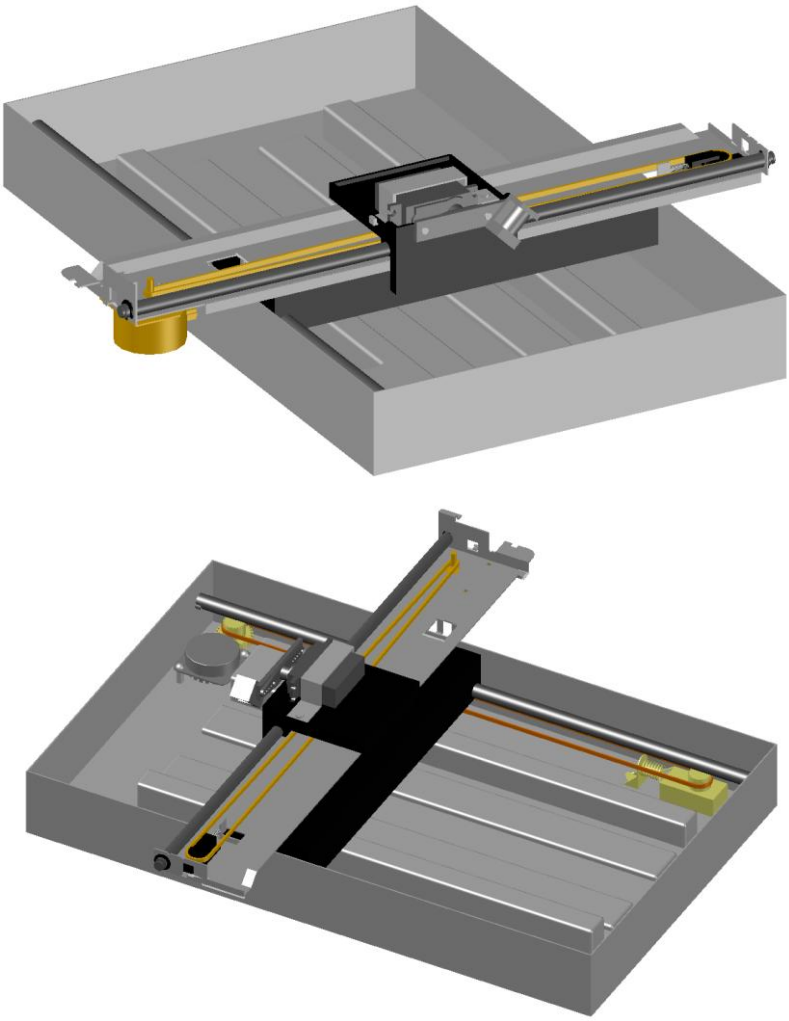


Figura 1: Modelo tridimensional do arcabouço mecânico

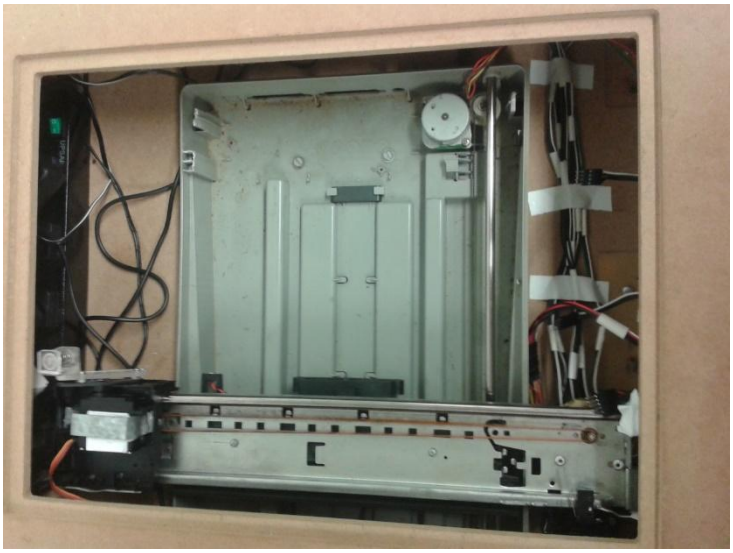


Figura 2: Arcabouço mecânico

2. Inteligência Artificial

Nas etapas iniciais de construção deste trabalho, o primeiro problema a ser resolvido, devido a sua natureza indispensável e central nesta aplicação, foi a necessidade de uma inteligência artificial capaz de jogar xadrez, o que será abordado na seção 2.1.

Dada a descrição do projeto, seção 1.6, o sistema mecânico a ser utilizado se move em dois eixos alinhados ao tabuleiro e sem movimentos em diagonal. O fato de saber a jogada a ser realizada (casa inicial e casa final) não é suficiente, é necessário calcular um caminho que execute a jogada sem interferir no restante do tabuleiro. Para este fim, foi implementado um algoritmo de cálculo do melhor caminho que executa a jogada, o qual será abordado na seção 2.2.

No entanto, em certos casos, não existe um caminho livre. Portanto o melhor caminho implicaria em colidir com uma ou mais peças do tabuleiro. Para evitar essas colisões e manter a integridade do restante do tabuleiro ao executar uma jogada, foi desenvolvido o algoritmo de desvio abordado na seção 2.3.

Dos três algoritmos deste capítulo o primeiro foi apenas adaptado, o segundo implementado utilizando uma heurística já conhecida e bem estabelecida e o terceiro foi idealizado e implementado.

2.1. Algoritmo de xadrez

A história dos algoritmos de xadrez começa com a publicação de Claude Shannon em 1949, na qual, apesar de não ser apresentado um programa específico, foram analisados vários dos problemas envolvidos na construção de um [1].

Sua proposta foi tão bem sucedida que o núcleo da arquitetura proposta por Shannon é usada até hoje [2]. A finitude do jogo de xadrez é facilmente comprovada ao seguir as regras do jogo, podendo assim ser demonstrado que eventualmente o resultado será sempre uma vitória, derrota ou empate [3].

Os programas desenvolvidos para este jogo utilizam uma estrutura em árvore onde cada nó corresponde a configuração do tabuleiro e cada ramo a uma jogada [1].

O funcionamento básico proposto de um algoritmo de xadrez se divide em três partes: o gerador de jogadas, no qual se analisam quais jogadas devem ser consideradas; o ponderador, que tem a função de avaliar as posições finais do tabuleiro (nós folhas) e dar pesos para elas; e o arbitrador, responsável pela análise da estrutura em árvore e decisão de qual jogada deve ser feita [1].

O gerador de jogadas, como seu nome sugere, é o responsável por gerar o conjunto de movimentos válidos que será analisado. Existem diferentes implementações possíveis, porém destacam-se: considerar todos os movimentos válidos, ou então apenas um conjunto limitado fixo ou variável, segundo um critério desejado. Para cada jogada possível, devem ser geradas as jogadas do adversário e posteriormente a próxima jogada da máquina, formando a estrutura em árvore já descrita [1].

Dada a árvore de movimentos criada pelo gerador de jogadas, o ponderador quantifica cada um dos cenários finais (nós folhas). A função de quantificação é o cerne da inteligência da máquina, pois a decisão será embasada nesses pesos. Uma função de exemplo para um ponderador é:

$$f(P) = 200(K - K') + 9(Q - Q') + 5(R - R') + 3(B - B' + N - N') + (P - P')$$

Onde K, Q, R, B, N e P, indicam a quantidade no tabuleiro de rei, rainha, torre, bispo, cavalo e peão, respectivamente, e a presença do " ' " indica uma peça adversária ainda no tabuleiro. Nessa função de exemplo considera-se como um cenário promissor aquele onde se possui mais peças que o adversário, considerando certas peças mais importantes que outras conforme os pesos [3].

O arbitrador decide qual a jogada que será feita, através da análise da árvore de pesos criada até aqui. A avaliação se baseia no método do *minimax, que será* discutido na seção 2.1.1

Segundo [3] uma máquina capaz de executar uma operação por microssegundo levaria 10^{90} anos para calcular o primeiro movimento seguindo as estratégias abordadas. Para diminuir tamanha complexidade computacional

é acrescentado um livro de aberturas (termo traduzido do inglês *opening book*), que é uma base de dados que contém centenas de jogadas iniciais e possíveis respostas [2].

Como o objetivo deste trabalho não era o desenvolvimento de um programa de inteligência artificial para xadrez, não foi desenvolvido um código seguindo esses moldes. Para o trabalho foi adaptado um código aberto disponibilizado por Tom Kerrigan em sua *homepage*. Esse código necessita de menos que 64kB de memória RAM para ser executado, sendo ideal para um ambiente limitado como o do Arduino. O mesmo utiliza o método de *negamax* com poda α-β, uma variação do método do *minimax* (mais detalhes na seção 2.1.1) e possui livro de aberturas [4].

2.1.1. Análise em árvores *minimax*

Essa seção será escrita baseada em [5], [6] e [1].

Inicialmente, considere a árvore criada e avaliada conforme o exemplo da Figura 3. Os níveis em branco representam decisões da máquina enquanto os níveis em cinza a decisão do oponente, ou seja, o nível em branco assume o maior dentre os pesos dos níveis inferiores, enquanto o nível em cinza assume o menor dentre os pesos dos níveis inferiores e o valor de cada nó quão promissor é aquele cenário.

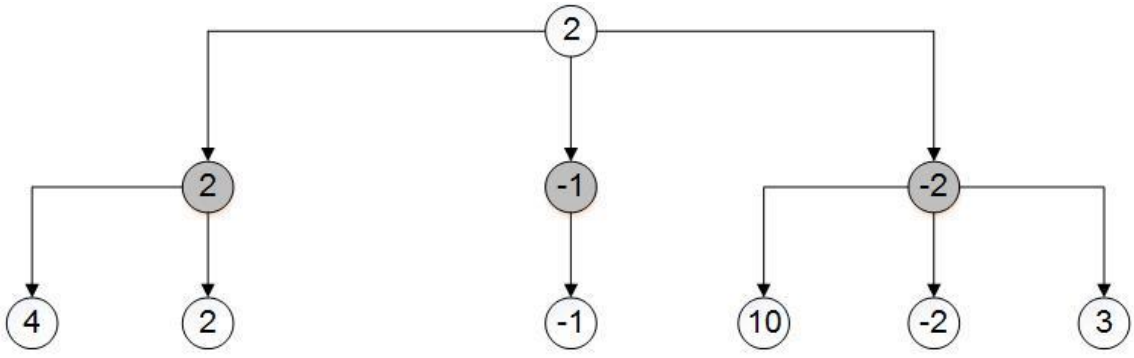


Figura 3: Árvore de pesos

A análise ilustrada na Figura 3 demonstra o princípio simples do método *minimax*, considerando que quanto mais promissor o cenário para a máquina

maior o peso do nó. É intuitivo que a máquina maximize os cenários criados por cada uma das possíveis jogadas analisadas (ramos), assim como o adversário irá minimizá-lo.

Este método, apesar de simples, não é comumente implementado, pois considerando que cada cenário (nó) possui um conjunto grande de possíveis jogadas e que para uma análise utiliza-se um número grande de níveis na árvore, essa análise acaba sendo demasiadamente demorada.

O método de poda α-β é uma variação do método *minimax* que elimina a análise de certos ramos podendo reduzir drasticamente o nível de processamento necessário. Este método analisa a árvore, conforme a Figura 4.

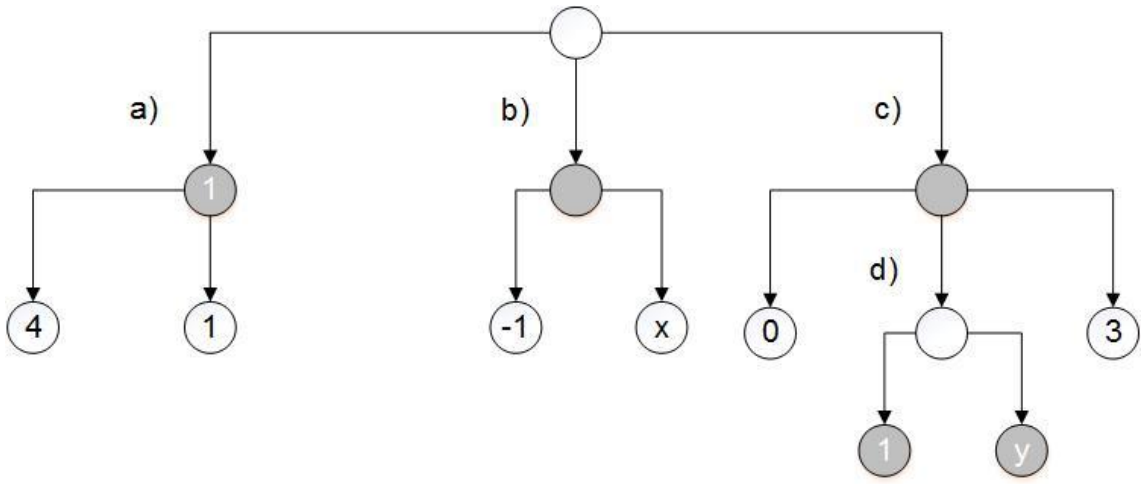


Figura 4: Poda α-β

Partindo da suposição que a análise dos nós filhos mais à esquerda seja realizada antes dos nós filhos mais à direita, tem-se que, na análise do nó **b**, o nó **x** e toda e qualquer ramificação de **x** não precisa ser analisada. Dados os axiomas:

- O nó raiz irá escolher o maior dentre os nós **a**, **b** e **c**
- O nó **b** irá escolher o menor dentre seus filhos
- Um filho de **b** possui um valor menor que o nó **a**

Baseado nos dois últimos axiomas, tem-se que o valor de **b** será menor que o valor de **a**, e portanto conforme o primeiro axioma, o nó **b** não será escolhido independentemente dos outros filhos de **b**. Assim, o nó **x** não precisa ser analisado. Essa eliminação é a poda α.

Mais especificamente, α é o parâmetro que representa o menor dentre os filhos de um nó *max*. Se um nó *min* filho deste possuir um filho menor que α sua análise é interrompida e descartada.

A poda β é o análogo inverso. No exemplo, despreza-se o ramo **y** e todas as suas possíveis ramificações. Dados os axiomas:

- O nó c irá escolher o menor dentre seus filhos
- O nó d irá escolher o maior dentre os seus filhos
- Um filho de d (no caso o primeiro) possui um valor maior que o primeiro filho de c

Baseado nos dois últimos axiomas, tem-se que o valor de **d** será menor que o valor do primeiro filho de **c**, e portanto conforme o primeiro axioma o nó **d** não será escolhido independentemente dos outros filhos de **d**. Assim, o nó **y** não precisa ser analisado. Essa eliminação é a poda β.

Mais especificamente, β é o parâmetro que representa o maior dentre os filhos de um nó *min.* Se um nó *max* filho deste possuir um filho maior que β, sua análise é interrompida e descartada.

Como dito na seção 2.1, o método de análise utilizado pelo TSCP (do inglês Tom Kerrigan's Chess Program) é o método do *negamax*, uma variação do método da poda α-β que utiliza a inversão de sinal. Ou seja, o nó analisa os pesos dos nós filhos com o sinal invertido. Essa lógica permite que tanto os nós *max* quanto os nós *min* tentem maximizar o peso, descartando a necessidade da minimização feita pelos nós *min.* Afinal o menor dentre os pesos dos filhos é equivalente ao maior dentre os pesos dos filhos com sinais trocados.

2.2. Algoritmo de melhor caminho

O algoritmo de melhor caminho deve ser capaz de calcular o menor caminho com o menor número de colisões possíveis, no cenário onde cada casa do tabuleiro representa um nó que possui ligação (ramo) com as casas de cima, de baixo, da direita e da esquerda, se existirem. Será considerado que é melhor, do ponto de vista do algoritmo, percorrer todo o tabuleiro do que colidir com uma peça. Na análise desse grafo em busca do melhor caminho caímos no clássico algoritmo de Dijkstra. As explicações desta seção de baseiam em [7].

O algoritmo de Dijkstra consiste em analisar o caminho com menor peso para ir de um nó de origem a um nó de destino. Para isso, utiliza-se uma lista de prioridade, na qual o elemento de menor prioridade é o próximo a ser chamado. Nesse algoritmo a prioridade utilizada é a soma dos pesos necessários para se chegar até aquele nó, ressaltando que não podem haver pesos negativos na rede.

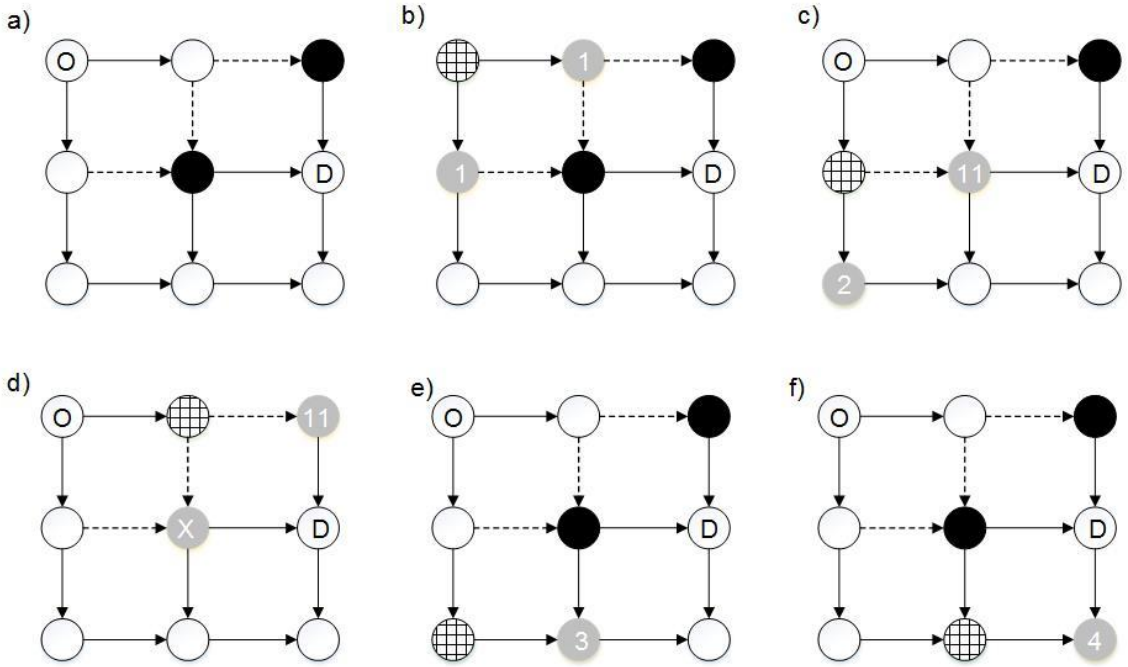


Figura 5: Algoritmo de Dijkstra.

Por simplicidade será considerado o tabuleiro Figura 5, com apenas nove casas (nós). O algoritmo se inicia pelo nó **O** até atingir o nó **D**, ou na

notação matricial do nó (1,1) ao nó (2,3). O peso inicial é zero. Sempre que se vai para um nó vazio adiciona-se um ao peso e sempre que se vai a um nó ocupado adiciona-se 10 (o número total de nós mais um garante que será preferível percorrer todo tabuleiro do que passar por um nó ocupado).

Começando a execução do algoritmo seguindo a Figura 5, na parte **a** temos a situação inicial indicando em preto as posições ocupadas e em branco as posições livres. Na parte **b** o inicio do algoritmo propriamente dito, o nó quadriculado está em análise e os nós cinzas serão inseridos na pilha. Como ambos são nós livres, ambos serão inseridos na lista com prioridade um.

O próximo nó a ser analisado é o de menor prioridade na lista. Como só temos os nós (1,2) e (2,1), ambos com peso 1, o último que foi inserido será o primeiro a ser analisado. Na parte **c**, o nó (2,1) é retirado da pilha inserindo seus vizinhos que ainda não foram inseridos na pilha, o nó (3,1) com peso 2, pois a casa está livre, e o nó (2,2) com peso 11, já que a casa está ocupada.

Na pilha tem-se agora os nós (1,2), (3,1) e (2,2) com prioridades 1, 2 e 11, respectivamente. Portanto, conforme a parte **d**, o próximo nó a ser analisado é o (1,2). Vale ressaltar que a análise dos nós (1,2) e (2,1), nas partes **c** e **d** podem ser feitas em qualquer ordem e o algoritmo continuará funcionando, mas o incremento de peso garante que nenhum nó inserido por um deles será analisado antes do outro. Continuando a análise da parte **d**, ao ser retirado da pilha o nó (1,2) insere seus vizinhos que ainda não foram inseridos na pilha. Como o nó (2,2) já foi inserido, apenas o nó (1,3) entra, mas como a casa está ocupada, o nó (1,2) que possuía peso 1 agora é inserido com peso 11.

Não reinserir o nó (2,2) significa que para o algoritmo, se o melhor caminho passar pelo nó (2,2) será feito pela sequência (1,1)->(2,1)->(2,2). Vale observar que o caminho (1,1)->(1,2)->(2,2) é equivalente ao escolhido pelo algoritmo. Portanto, Dijkstra encontra o melhor caminho, mas apenas um caminho, mesmo que exista outro com o mesmo peso.

Nesse ponto do algoritmo temos na pilha três nós (1,3), (2,2) e (3,1) com pesos 2, 11 e 11. O restante do algoritmo fica bem simples. Na parte **e** o nó (3,1) é retirado inserindo o nó (3,2) com peso 3, o único vizinho não visitado. Na parte **f**, o nó (3,2) é retirado e é inserido o nó (3,3) com peso 4. O algoritmo se encerra quando o nó (3,3) é retirado e alcança o nó de destino com peso 5.

Para calcular o caminho basta ter uma rede auxiliar onde cada nó armazena o nó que o inseriu na lista de prioridades.

2.3. Algoritmo de desvio

O Algoritmo de desvio é utilizado para resolver casos onde o Algoritmo de melhor caminho não encontra um caminho livre. Essa situação é bem fácil de acontecer em um jogo de xadrez. Considere, por exemplo, a situação da Figura 6, onde o cavalo branco é movido da casa G1 para a E2.

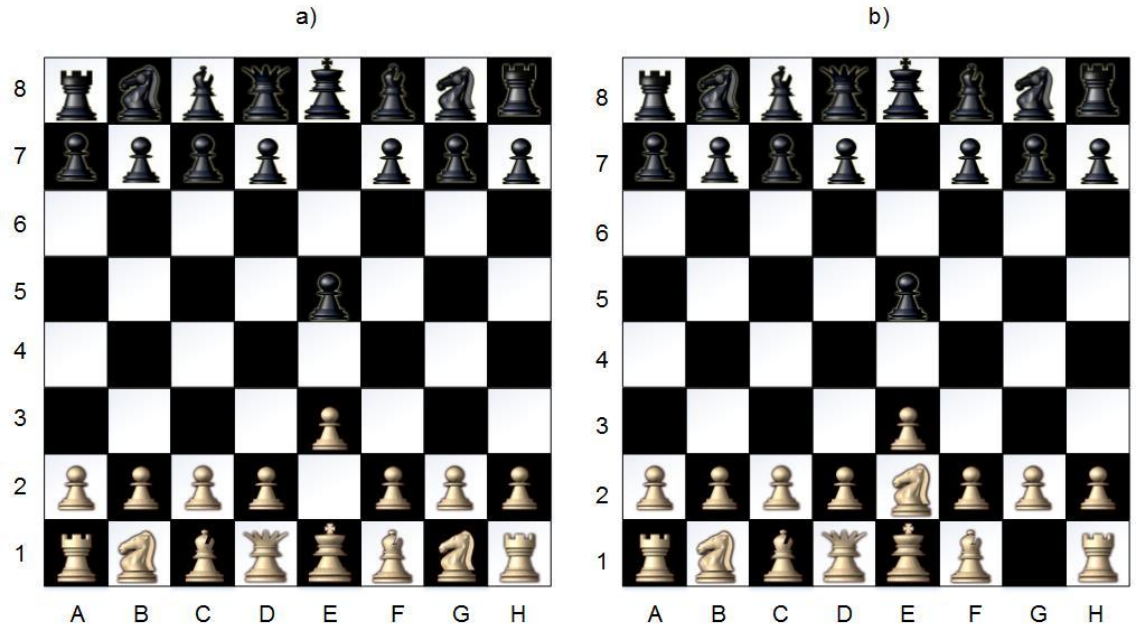


Figura 6: Aplicação do algoritmo de desvio

Nessa situação considerando-se o tabuleiro 10 por 10, conforme especificado na seção 1.6, o Algoritmo de melhor caminho faria o caminho indicado na Figura 7 em casas cinzas.

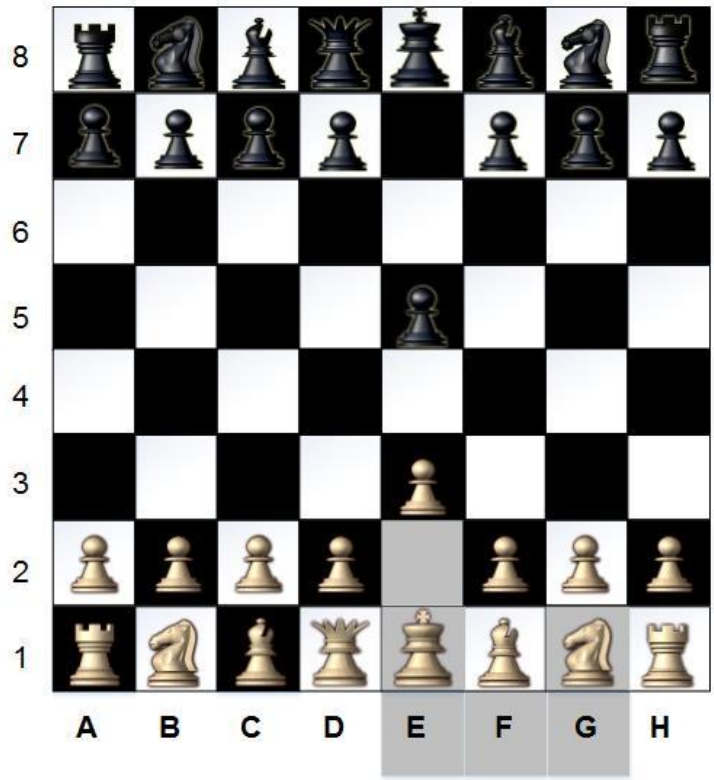


Figura 7: Melhor caminho com necessidade do algoritmo de desvio

Dada, portanto, a necessidade de se retirar o rei da casa E1 antes de mover o cavalo para manter o tabuleiro na posição correta, foi desenvolvido o algoritmo de desvio. A Figura 8, apresenta uma visão geral das partes do código e da sua interação.

Nesta figura podemos analisar o processo em três partes. Primeiramente o programa principal que em algum ponto do código deseja executar uma nova jogada da posição (a,b) para posição (c,d). Para tal, basta chamar a segunda parte que calcula o melhor caminho e retorna um vetor de movimentos e, por fim, executá-los.

O cálculo do melhor caminho utiliza a lógica de implementação do algoritmo de Dijkstra, explicada na seção 2.2. Dado que o menor caminho está calculado, basta analisar cada casa e verificar se existe colisão, ou seja, se a casa possui uma peça. Não havendo uma peça em algum lugar da trajetória o algoritmo encerra e o vetor de movimentos nada mais é do que o próprio melhor caminho já calculado. No entanto, se existir alguma colisão é calculado um desvio para cada colisão. O desvio é um vetor de movimentos que devem

ser realizados para que a jogada principal possa ser realizada. Portanto os movimentos de desvios devem compor o vetor de movimentos que será retornado pelo algoritmo e devem preceder o caminho calculado por Dijkstra nesta execução. Entretanto, se for executado apenas o desvio e o caminho principal, as peças desviadas ficarão deslocadas. Para corrigir isto, basta inserir no vetor os movimentos de desvios invertidos em ordem inversa, ou seja, se para fazer um movimento são necessários dois desvios o primeiro desvio feito será o último a ser desfeito e cada movimento consiste no oposto do movimento original.

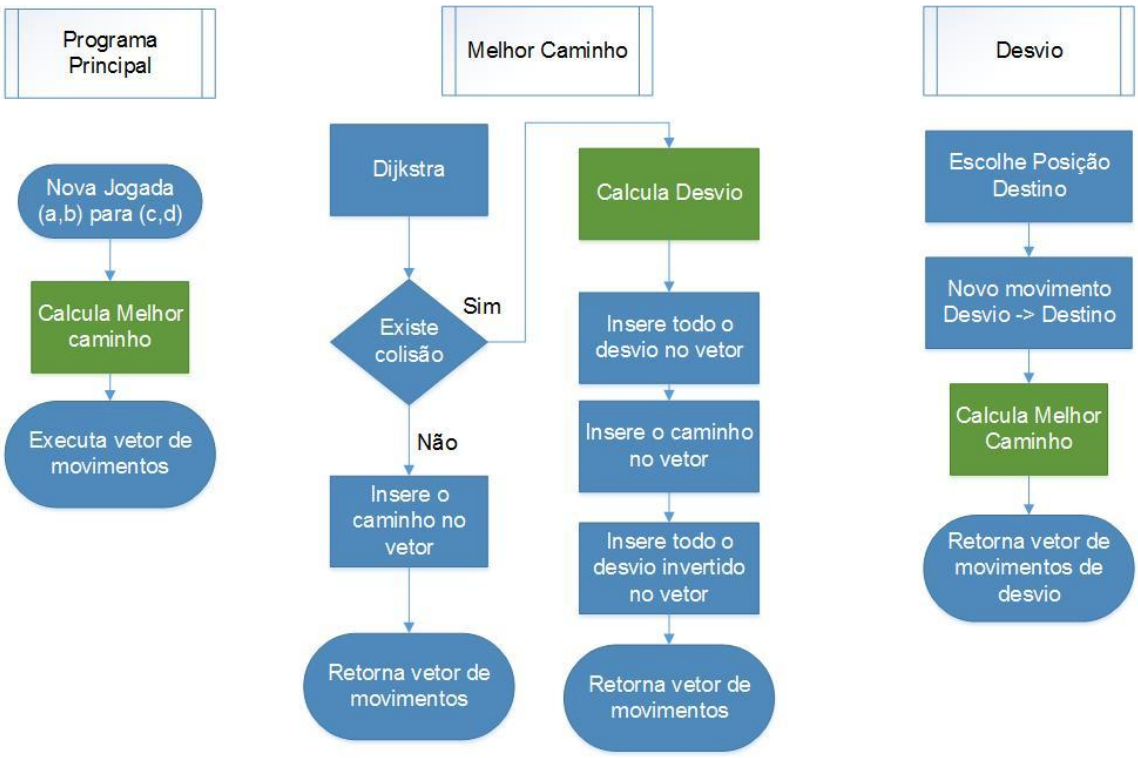


Figura 8: Fluxograma do algoritmo de desvio

Essa lógica não é a mais eficiente possível, ou seja, não é o algoritmo ótimo, mas é um algoritmo robusto, e portanto capaz de resolver toda e qualquer situação enfrentada em um jogo real. Para mais detalhes de testes sobre este algoritmo, ver a seção 6.2.

O núcleo da lógica de desvio é o cálculo da posição para onde será desviada a peça que sofreria a colisão da peça em deslocamento. O algoritmo desenvolvido procura a casa mais próxima em um raio que esteja vazia e que

não pertença ao caminho principal, conforme mostrado na Figura 9. A princípio bastaria mover a peça para essa posição que a integridade do tabuleiro seria mantida. Na Figura 9, pode-se ver o caminho desejado em cinza, as casas quadriculadas são as livres com busca em raio 1 e as casas listradas são as livres com busca em raio 2.

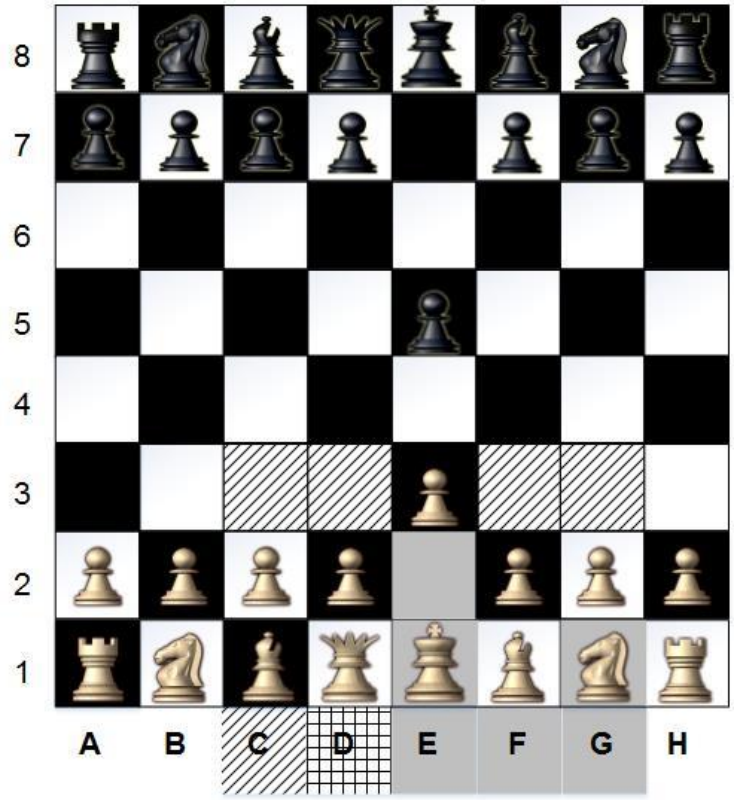


Figura 9: Raio de busca do algoritmo de desvio.

Entretanto, existe uma situação adversa que ainda precisa ser tratada. Caso o melhor caminho do desvio passe pela posição de destino do caminho principal, no momento de desfazer o desvio, ou seja, ao realizar o movimento inverso, ocorreria uma colisão. Para maior detalhamento do problema e da análise completa da solução ver a seção 6.2. Para resolver isso, basta inserir uma condição na qual o melhor caminho não possa passar pelo fim do caminho principal. Essa modificação equivale a inserir peso infinito nesta casa para o algoritmo de Dijkstra, ou impedir que essa casa seja inserida na pilha de prioridades.

3. Arduino

3.1. Introdução

Esta seção foi baseada no site oficial do Arduino [8].

Após a abordagem da questão referente à inteligência artificial, foi levantada a questão de como se daria a sua implementação física. Devido a experiências anteriores de utilização e sua flexibilidade, a opção foi feita pela plataforma de desenvolvimento *open-source* Arduino.

Havendo diversos modelos, com diferentes características, a decisão foi tomada em função da grande quantidade de portas digitais, antevendo a necessidade de acionamento e sensoriamento de diversos sinais, e do maior poder de processamento. O modelo que melhor se adequou foi o Due (vide Figura 10). Segue na Tabela 1 suas especificações técnicas:

Tabela 1: Especificações Arduino Due, adaptado de [9]

Microcontrolador	AT91SAM3X8E
Tensão de operação	3.3V
Tensão de Entrada (recomendada)	7-12V
Tensão de Entrada (limite)	6-16V
E/S Digitais	54 (das quais 12 PWM)
Entradas Analógicas	12
Saídas Analógicas	2 (DAC)
Fornecimento Total de Corrente CC (todas I/O)	130 mA
Corrente CC para o pino 3.3V	800 mA
Corrente CC para o pino 5V	800 mA
Memória *Flash*	512 KB (toda disponível para aplicações do usuário)
SRAM	96 KB (twobanks: 64KB and 32KB)
Velocidade do *clock*	84 MHz

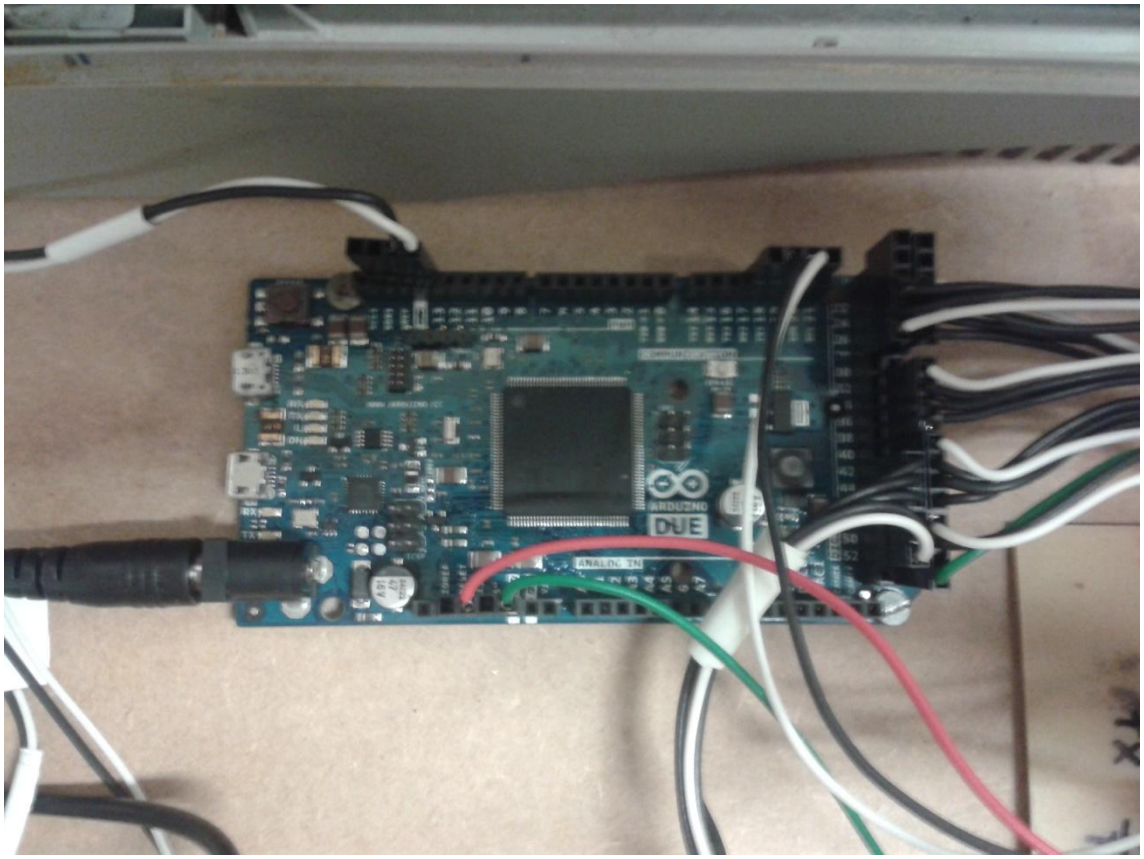


Figura 10: Arduino Due

3.1.1. Descrição

O Arduino é uma plataforma eletrônica de desenvolvimento fabricada pela companhia italiana SmartProjects (embora alguns modelos sejam fabricados pela SparkFunElectronics), que está sob a licença de *hardware* livre da CreativeCommons. Portanto, seus projetos e esquemas estão disponíveis ao público, o que favorece a criação de diversas versões para as mais diversas utilidades.

Os modelos são basicamente compostos por um microcontrolador Atmel AVR de 8 bits ou um Atmel ARM de 32 bits (caso do Arduino Due) pré-programado com um *bootloader*, o que facilita a sua gravação. Também possuem conectores dispostos de uma maneira padronizada para a conexão de outros módulos conhecidos como *shields*, assim como a possibilidade de alimentação via USB ou fonte DC, sendo o canal de gravação, normalmente, a

porta USB. Porém em versões mais antigas era comum encontrar uma conexão serial RS-232.

A filosofia adotada pelo projeto é a de facilitar o acesso de pessoas leigas, amadores e até mesmo artistas que venham querer se aventurar no desenvolvimento de artefatos interativos. Portanto, para isso, o baixo custo, flexibilidade e simplicidade de desenvolvimento são essenciais.

3.1.2. Hardware livre

Esta seção será discutida baseada em: [10], [11] e [12]

Seguindo a tendência dos movimentos *software* livre e código aberto, que pregam a utilização da inteligência coletiva como meio de criar e melhorar ideias, o movimento *open-source hardware* (OSHW) também vem crescendo amplamente no que tange a construção colaborativa.

Em 1975, Dennis Allison, tido como um dos precursores do movimento *software* livre, no *release* do TINY Basic afirmou: “Apoiemo-nos nos ombros um dos outros, em vez de pisar no pé”. Esta frase influenciou muitos no desenvolvimento desta cultura de compartilhamento, que agora vem se expandido para o *hardware*. Porém, diferentemente do *software*, o *hardware* não é feito de *bits*, portanto, por melhores intenções que alguém possua, dificilmente poderia distribuir gratuitamente produtos físicos para todos que desejassem. Entretanto, um produto físico é a implementação de um *design*, o qual pode ser compartilhado gratuitamente entre a comunidade com uma licença aberta, *copyright* ou patenteada. Assim, possibilitando que outros possam trabalhar a partir deste *design*, com o intuito de modificar, melhorar, adaptar para os mais diversos fins ou até mesmo desenvolver dispositivos compatíveis que venham a se comunicar com o original, criando uma grande variedade de ferramentas e atribuindo versatilidade ao projeto.

3.1.3. Arduino IDE

Como abordado nas seções anteriores, o projeto Arduino tem por objetivo permitir o acesso à tecnologia do público em geral. Para isso, uma interface de desenvolvimento amigável é essencial.

O Arduino IDE é uma aplicação *open-source* escrita em Java derivada de projetos como o *Processing* e o *Wiring*, além de ser multiplataforma, podendo ser executado tanto no Windows quanto no Linux ou no Macintosh OSX. Nele, é possível programar em uma linguagem muito semelhante ao C/C++, porém com pequenas modificações estruturais.

Na Figura 11 abaixo, é possível ver como se apresenta a interface ao usuário.

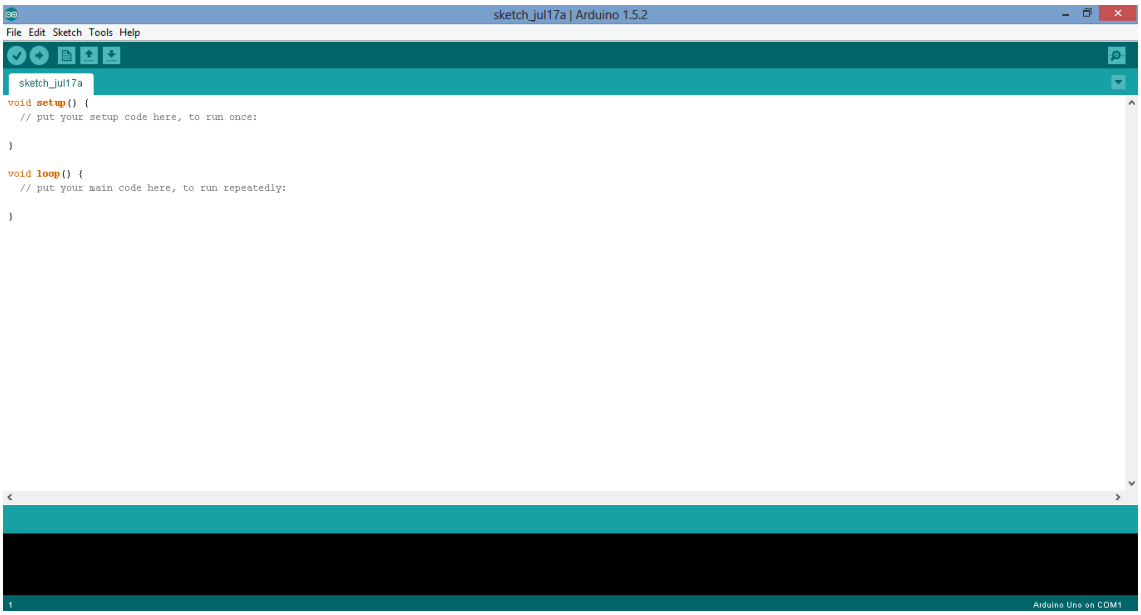


Figura 11: Arduino IDE

Basicamente, todo programa gerado é estruturado dentro de duas funções padrões: *setup()* e *loop()*, de forma que toda linha de código escrita dentro da função *setup()* é executada apenas uma vez no início da execução, como uma espécie de configuração inicial. Dentro da função *loop()* é escrito o programa principal, que será executado em *loop* infinitamente ou até quando o usuário desejar parar. Essa implementação em *setup()* e *loop()* equivale ao código em C da Figura 12.

```
int main () {

    setup();

    while( true )
        loop();

    return (0);
}
```

Figura 12: Código em C equivalente a lógica da IDE

A interface prima pela simplicidade. Gravar no Arduino é simples e intuitivo, basta clicar no botão *upload* e a interface se encarrega da gravação. Além de contar com um bom acervo de bibliotecas como: EEPROM, Ethernet, GSM, Servo, SPI, entre outras.

3.2. Visão geral do funcionamento

No *loop* principal do programa duas situações são esperadas: uma jogada ser executada no tabuleiro ou uma mensagem vinda via *Bluetooth*, ou seja, o *loop* principal responde as interfaces do usuário, como se o programa fosse orientado a evento. O fluxograma do programa que é executado no Arduino pode ser visto na Figura 13.

Existem, resumidamente, dois eventos, uma nova jogada, seja ela feita pelo celular ou no tabuleiro, ou uma mensagem de comunicação. Para mais detalhes sobre esse tipo de mensagem e da forma com que essas mensagens são trocadas ver a seção 5.

No caso de uma jogada ser recebida pelo Arduino essa jogada deve ser executada no tabuleiro virtual. Ou seja, executada dentro do módulo de inteligência artificial adaptado do TSCP [4] de forma que a jogada seja validada. Sendo esse evento iniciado por mensagem, a jogada é então executada no tabuleiro.

Sempre que uma jogada, seja ela do usuário ou da inteligência artificial, é executada, verifica-se a condição de fim de jogo (Xeque-mate). Não sendo o caso, o programa continua normalmente.

Ao final de uma jogada recebida, uma nova jogada é gerada pela máquina e executada no tabuleiro, reiniciando o ciclo de espera pela próxima jogada do usuário.

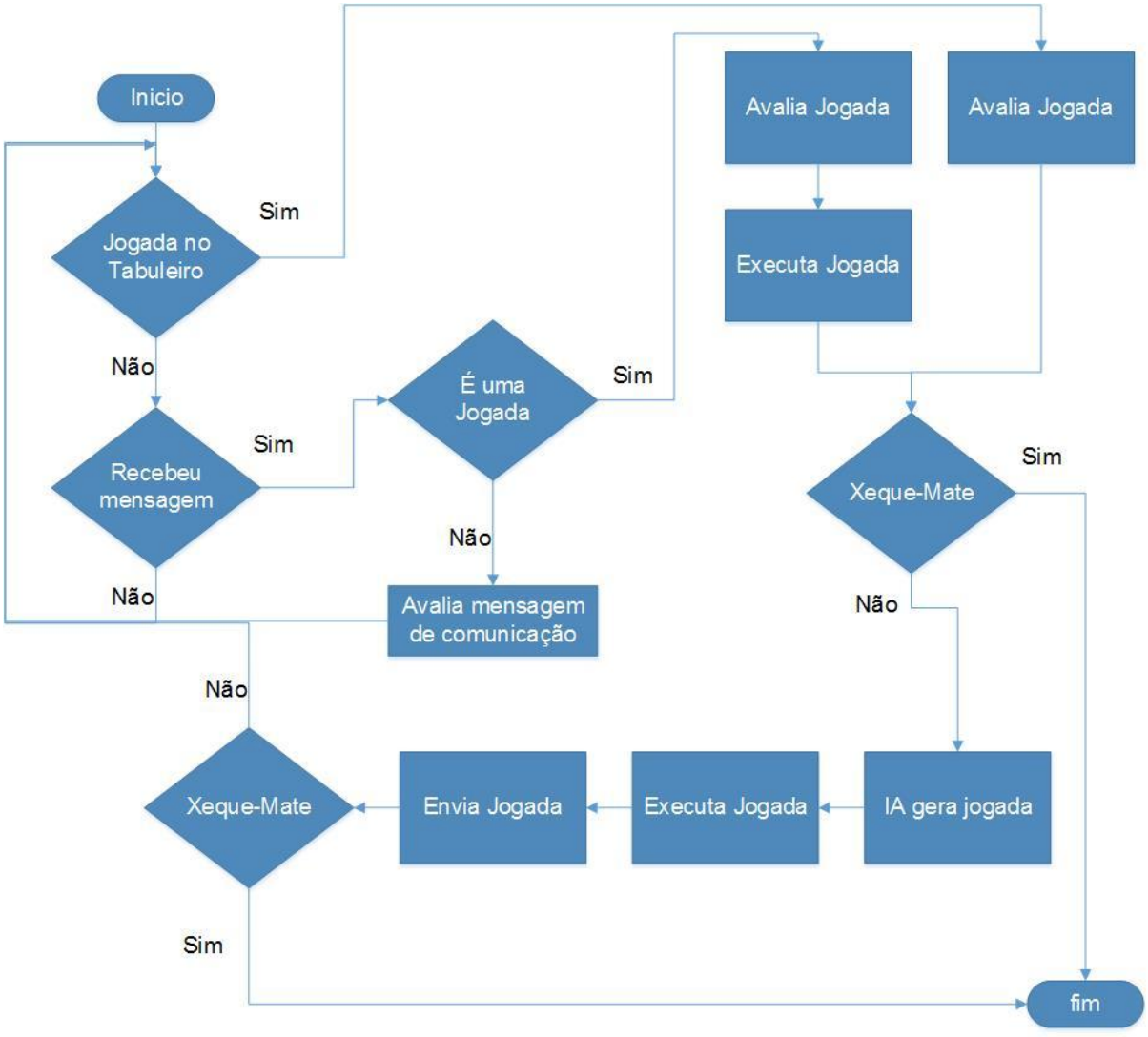


Figura 13: Fluxograma simplificado do programa

3.2.1. Interfaces com o sistema

A plataforma Arduino Due desempenha um papel central neste projeto, pois ela é a responsável por traduzir em impulsos elétricos os comandos processados pelo programa. Na Figura 14 pode-se ver as interfaces que o Arduino tem com os dispositivos que o rodeiam, além dos dispositivos com os quais o Arduino só tem contato indireto.

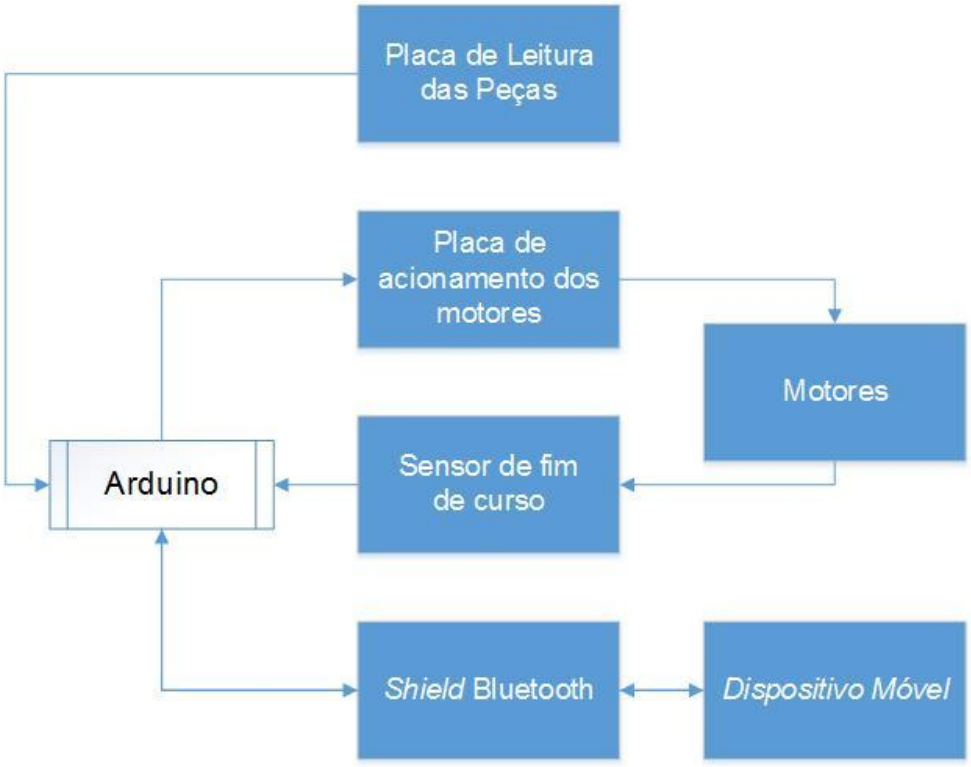


Figura 14: Arduino e suas interfaces

Para haver uma comunicação com o usuário do sistema, de forma que ele possa receber um *feedback* de suas ações ou até mesmo um alerta quanto às regras do jogo, situações de cheque, etc.; foi escolhido o sistema operacional *open-source* para dispositivos móveis Android, que será tratado com mais detalhes na seção 4. Para a comunicação do dispositivo remoto com o Android, a solução adotada foi utilizar a tecnologia *Bluetooth* através de um módulo (*shield*) compatível com a plataforma. Mais detalhes na seção 5.

Para controle do tabuleiro e execução das jogadas, o Arduino deve se comunicar com os motores através da placa de acionamento. E por fim, para que o controle seja efetivo, também é necessário que o Due interprete

adequadamente os sinais provenientes da placa de leitura das peças, assim como os de fim de curso para alinhamento dos eixos.

A placa de acionamento dos motores (vide Figura 15) é basicamente composta por dois *drivers*, que além do ganho de corrente, também isolam o Arduino do circuito onde circula a corrente de acionamento dos motores de passo, a qual poderia danificá-lo. Esse circuito é formado por dois relés para evitar consumo desnecessário de corrente, dois transistores operando como chave de acionamento para os relés e dois resistores de polarização do transistor.

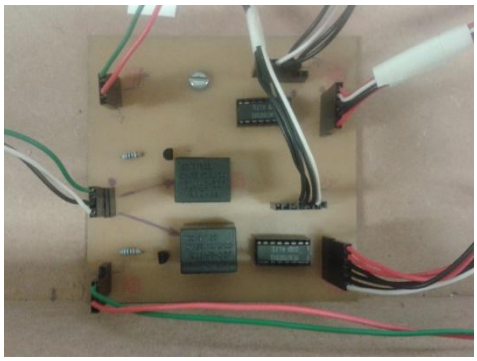


Figura 15: Placa de acionamento dos motores

Já a placa de leitura (vide Figura 16) é uma placa 30x40 cm dupla face que possui 64 sensores de efeito *hall* SMD (vide Figura 17) dispostos na placa de forma a estarem debaixo do centro de cada casa do tabuleiro de jogo, 64 capacitores para a estabilização da tensão e filtragem de ruídos, 64 resistores de *pull-up* na saída de cada sensor e também 8 multiplexadores 8x1 (vide Figura 18) para diminuir o número de entradas necessárias para o Arduino poder varrer o tabuleiro em busca de variações. Para mais detalhes acerca dos *layouts,* componentes, etc., verificar o Apêndice A.

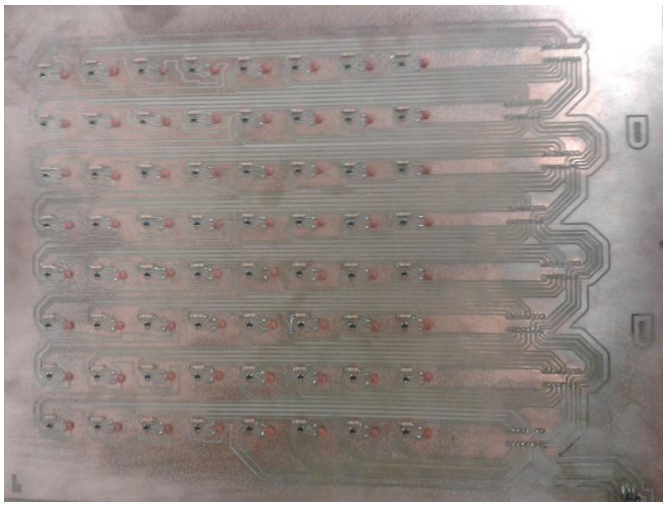


Figura 16: Placa de leitura

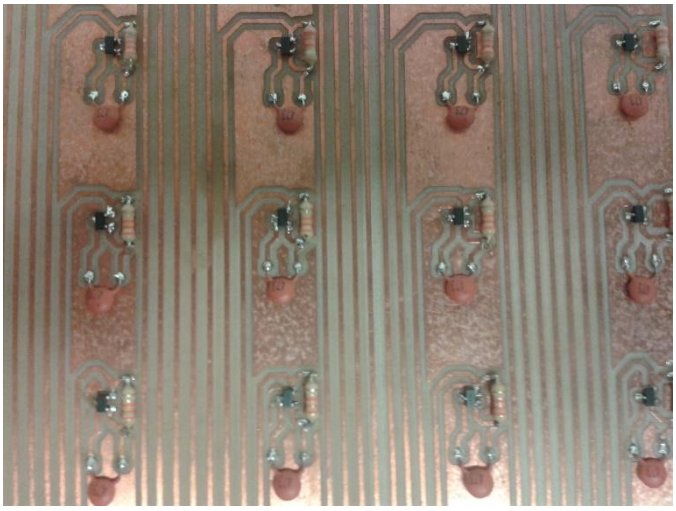


Figura 17: Sensores de efeito *hall*

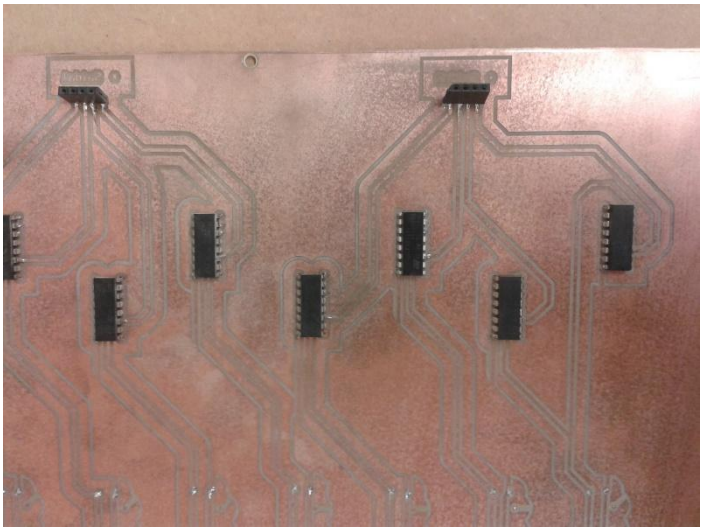


Figura 18: Multiplexadores

4. Android

4.1. Introdução

Tendo sida estabelecida a plataforma central do projeto, foi aventada a ideia de uma interface de comunicação com usuário, sendo primeiramente considerada a utilização de uma tela LCD. Muito embora, a tela LCD fosse suficiente na questão envolvendo o *feedback* para o usuário, já mencionada na seção 3.2.1, esta apresenta pouca flexibilidade. Portanto, surgiu a ideia de utilizar um aplicativo Android como interface, que a tornaria flexível não apenas no âmbito da interatividade, como também na possibilidade de experimentar a utilização do sistema com qualquer aparelho que possua Android, seja *tablets, smartphones,* ou até mesmo *smart TVs*, se assim o usuário desejar, enriquecendo sua experiência.

Além disso, na primeira placa construída, optou-se por utilizar *reed switches* como sensores de campo magnético. Porém, essa solução apresentou diversos problemas, tais como, o alinhamento dos sensores com os imãs das peças, e portanto se fez necessário mudar os sensores. A solução proposta foi a utilização de sensores de efeito *hall*. Para isso, era preciso construir uma nova placa, entretanto, não era certo se haveria tempo hábil e tampouco sucesso. Portanto a ideia de utilizar uma interface Android ganhou ainda mais força visto que também poderia servir como uma forma alternativa do usuário jogar. Pois, independentemente do sucesso ou não da construção da nova placa de leitura, teríamos uma interface apta para mediar um jogo entre o usuário e o sistema.

As subseções deste capitulo foram baseadas em: [13],[14],[15],[16],[17] e[18].

4.1.1. Descrição e Origens

O Android é um sistema operacional para dispositivos móveis baseado no *kernel* Linux desenvolvido pela Open Handset Alliance (OHA), uma aliança que envolve atualmente 84 empresas com o propósito de desenvolver padrões abertos para dispositivos móveis. Liderada pela Google, a OHA conta com empresas dos ramos de *hardware*, *software* e telecomunicações como: Sony, Dell, LG, Samsung, Acer, Intel, Texas Instruments, Atheros, eBay, Nvidia, Accenture, Vodafone, etc.

A Android, Inc. foi fundada em 2003 em Palo Alto, cidade localizada no Vale do Silício na Califórnia, por Andy Rubin, Rich Miner, Nick Sears e Chris White com o objetivo inicial de desenvolver um sistema operacional para câmeras digitais, porém foi percebido que o mercado não era grande para estes dispositivos, e portanto foi decidido pela produção de um sistema operacional para *smartphones*. Em agosto de 2005, a Google comprou a Android, Inc. Com isso, especulações acerca da entrada da Google no ramo de sistemas operacionais surgiram. Até que em novembro de 2007, a OHA se revelou e apresentou o Android, sendo que, o primeiro *smartphone* disponível com o sistema foi introduzido no mercado em outubro de 2008, o HTC Dream.

A plataforma de *hardware* principal que suporta o Android é baseada na arquitetura ARM de 32-bits, embora também seja possível encontrar Android embarcado em processadores com a tecnologia x86 e MIPS. O sistema também pode incorporar uma série de dispositivos, incluindo: câmeras digitais, GPS, sensores de orientação, *Bluetooth*, Wi-Fi, controle de game dedicados, microfones, acelerômetros, giroscópios, barômetros, magnetômetros, sensores de proximidade, sensores de pressão, termômetros e telas *touchscreen*.

O Android, como já dito anteriormente, é baseado no *kernel* Linux, sendo este sua camada base. “Por cima” desta camada temos bibliotecas, *middleware* e APIs escritas em C, que por sua vez interagem com aplicações de *software* que rodam num *framework* com bibliotecas Java baseadas no Apache Harmony. Levando em consideração a variedade de dispositivos que suportam o Android, para que os aplicativos não sofram com a diferente gama

de peculiaridades de cada arquitetura, a presença de uma máquina virtual como a Dalvik, é essencial.

A Dalvik se notabiliza por ser uma VM (*virtual machine*) otimizada para ambientes de baixa memória, o que é importante em aparelhos onde sua alimentação é suprida por baterias, pois resulta em economia de energia. Além disso, seu *design* permite executar várias instâncias de si mesma, contribuindo para uma arquitetura baseada em componentes onde cada aplicação é executada num processo próprio. A desvantagem fica por conta de sua característica *just-in-time* (JIT) de compilação, que depende da compilação do bytecode toda vez que um aplicativo é lançado. Entretanto, a última versão, o Android 4.4 Kit-Kat, permite experimentar uma nova VM, a Android Runtime, que apresenta como principal característica a compilação *ahead-of-time* (AOT), ou seja, a compilação é efetuada uma única vez no momento da instalação do aplicativo no aparelho.

4.1.2. Código Aberto (*Open-Source)*

O Android apresenta-se como um sistema operacional de código aberto (em inglês, *open-source*) com o propósito de incentivar desenvolvedores a buscarem adaptar o código às mais diversas situações, construindo assim uma comunidade em torno da plataforma. Para evitar possíveis incompatibilidades o *Android Open Source Program* (AOSP), criou o *Android Compatibility Program*, onde estabelece os detalhes técnicos necessários e provê ferramentas aos desenvolvedores que desejam customizar o sistema de forma a manterem sua versão compatível [19]. Assim, os desenvolvedores de aplicativos podem ficar despreocupados quanto a compatibilidade de seus *softwares* nessas versões customizadas do Android. E desta forma, fica assegurada uma rede de cooperação, onde OEMs (*Original Equipment* Manufacturers) podem customizar o sistema para os mais diversos dispositivos, garantindo flexibilidade, além da certeza de poder contar com os aplicativos dos desenvolvedores para dar ao público-alvo o que eles querem em termos de funcionalidade, entretenimento, interação e acesso.

Portanto, através da construção colaborativa é possível criar um produto poderoso, confiável e flexível, que é de fato pregado pelo movimento *open-source*, o qual procura focar na questão prática e técnica, se distanciando de questões éticas e morais pregadas pelo movimento do *software* livre. O que, inclusive, gerou críticas do fundador do movimento *Free Software Foundation* (FSF), Richard Stallman, pois embora qualquer um tenha a liberdade de acesso e de alteração do código-fonte, *firmwares* e *drivers* proprietários são requeridos para o funcionamento adequado, além da possibilidade do *Google Play* instalar ou desinstalar aplicativos sem o desejo do usuário e por último, a presença de aplicativos não-livres (proprietários) no mesmo [20].

4.1.3. ADT Bundle

Como dito na seção anterior, a filosofia adotada pelo Android é a da construção colaborativa, seja quanto ao próprio sistema, quanto aos aplicativos que enriquecem a experiência do usuário. Para incentivar cada vez mais desenvolvedores para sua comunidade, é importante uma boa ferramenta de desenvolvimento.

O ADT Bundle é um pacote com ferramentas de desenvolvimento disponível para desenvolvedores novatos[21]. Consta neste pacote: a IDE adotada pelo Android de forma oficial, o Eclipse com o Android Development Tools (ADT) *built-in*, o Android SDK, a versão mais atual da plataforma e uma imagem para o emulador.

O Eclipse é uma IDE *open-source* escrita basicamente em Java desenvolvida pela Eclipse Foundation, que conta com um sistema expansível de *plug-ins* e um *workspace* base. Embora a linguagem nativa seja Java, através da instalação de *plug-ins* é possível escrever em outras linguagens como: Perl, Ruby, PHP, C/C++, Python, etc. Com a adição do *plug-in* Android Development Tools (ADT), o desenvolvimento de aplicações Android fica bastante simples, sendo possível criar todo o *framework* necessário para o desenvolvedor na inicialização do projeto, assim como também permite fazer o *debug* da aplicação utilizando as ferramentas de depuração do Android

SDK. Além disso também possibilita a exportação de um arquivo .apk para distribuição do aplicativo.

O Android SDK é conjunto de ferramentas de desenvolvimento que inclui exemplos de projeto Android com seus respectivos códigos-fonte, um emulador e bibliotecas API necessárias para construir um aplicativo.

Com este conjunto de ferramentas, o desenvolvimento torna-se bastante simplificado e diversificado. É possível lançar mão de diversas ferramentas, além das facilidades já disponíveis no Eclipse como os recursos de complementação, correção e até geração de trechos de código usuais como *getters* e *setters*, porém sempre com a confirmação do desenvolvedor, impedindo assim que o sistema em vez de fornecer apoio se torne um inconveniente.

4.2. Modelagem da Aplicação

O aplicativo do projeto é basicamente dividido em três partes: os pacotes (**SCREENS**, **BLUETOOTH**, **CHESSBOARD**), os *layouts* e os *drawables*. Essa divisão é induzida pela própria IDE que já separa esses blocos. Antes de descrever as principais características do código desenvolvido, algumas ressalvas sobre a forma de programar para Android são importantes.

Toda aplicação Android se baseia numa sucessão de telas exibidas aos usuários, onde uma tela é a associação de uma *Activity* e de um *layout*. O *layout* é responsável por definir como será visualizada a aplicação, enquanto que a *Activity* se responsabiliza pelo código que será acionado em cada ação na interface gráfica. Um *drawable* é um arquivo que será acessado por um layout ou por uma *Activity* e será exibido para o usuário, como um vídeo, uma imagem ou um arquivo de som [21].

A aplicação desenvolvida se baseia em uma aplicação chamada *BluetoothChat* [22], cujo código está disponível como exemplo na própria IDE. Possui duas diferentes telas, conforme a Figura 19, uma inicial, a da esquerda, com objetivo de especificar condições iniciais e configurações, e a segunda, no centro, que é a tela de jogo propriamente dita. A terceira tela, da direita, é herdada da aplicação *BluetoothChat*.

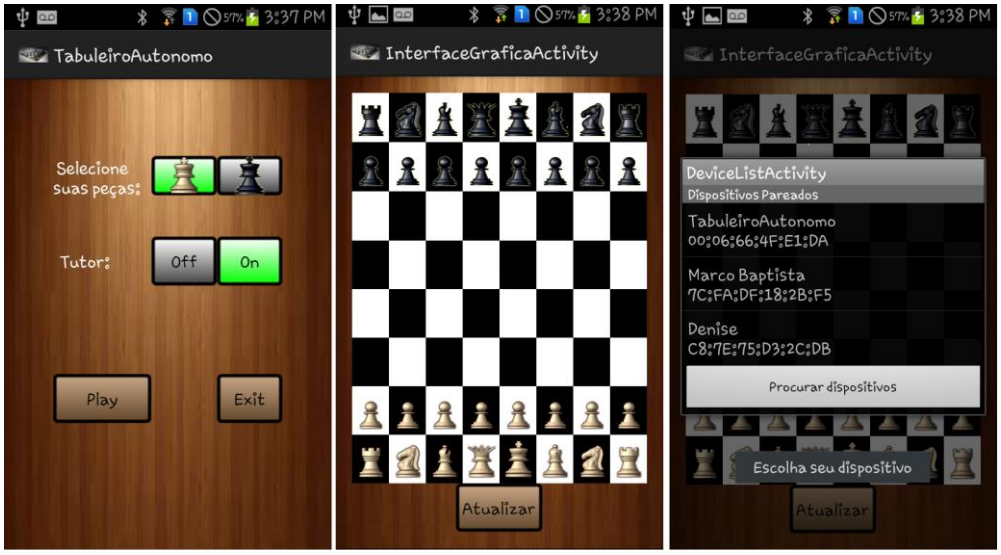


Figura 19: Telas da aplicação

Na Figura 20, pode-se ver as classes **HomeActivity** e **MainActivity**, que estendem a classe *Activity* e representam, respectivamente, as telas na esquerda e do centro da Figura 19. Além dessas duas telas, só existe mais uma que pode ser exibida ao usuário é uma tela já previamente implementada no *BluetoothChat* representada pela **DeviceListActivity** no diagrama de classes pertencente ao pacote **BLUETOOTH**.

Os três pacotes dividem a aplicação em: **SCREENS**, que são as herdeiras da classe *Activity* já descritas; **BLUETOOTH**, que são as classes herdadas ou adaptadas do *BluetoothChat*; e **CHESSBOARD**, que contém as classes responsáveis pela parte gráfica referente ao tabuleiro.

A parte visível do pacote **BLUETOOTH** é a classe abstrata **BluetoothManagerActivity**, dado que para implementar troca de mensagens por *Bluetooth,* a *Activity* precisa ter diversas características. Em função disso, foi criada uma extensão da classe *Activity* com todas essas implementações já estabelecidas, para permitir que qualquer aplicação que necessite de *Bluetooth* possa utilizar esse pacote. É necessário implementar três diferentes funções para criar um herdeiro do **BluetoothManagerActivity**: ***receiveMessage***, chamada quando uma nova mensagem for recebida; ***updateState***, chamada a cada alteração no estado da conexão; e ***onConnectionConclusion***, chamada a cada vez que uma nova comunicação é por fim estabelecida. Além disso, já

está implementada a função ***sendMessageToBuffer*** que permite às classes filhas enviar uma mensagem diretamente.

O pacote **CHESSBOARD** tem como partes visíveis uma interface denominada **Playable** e a classe **BoardManager**, que gerencia o tabuleiro. Os eventos de cliques no tabuleiro determinam quais os movimentos que o usuário está tentando fazer. No construtor do **BoardManager** é necessário passar como parâmetro um objeto **Playable,** que tratará as mensagens, e uma *Activity* para que ele possa gerir o tabuleiro na tela. Na aplicação, ambos os objetos são a instância de **MainActivity**. Quando o usuário realiza uma tentativa de movimento, a **BoardManager** chama o método especificado na interface **Playable** através do objeto recebido no construtor. A **BoardManager** possui também uma função para a atualização de tabuleiro discutida em mais detalhes na seção 5.1.

A **MainActivity** é basicamente, além de tratamento para as mensagens especiais (vide seção 5.1), uma conexão entre o **BLUETOOTH** e o **CHESSBOARD**. Quando a **BoardManager** chama a função ***makeMove*** do objeto **Playable,** a **MainActivity** redireciona essa jogada para o **BluetoothManagerActivity** pelo **sendMessageToBuffer**. O inverso também é válido, quando se recebe uma jogada por *Bluetooth*, ela é passada para a classe **BoardManager** pela função ***newMove***.

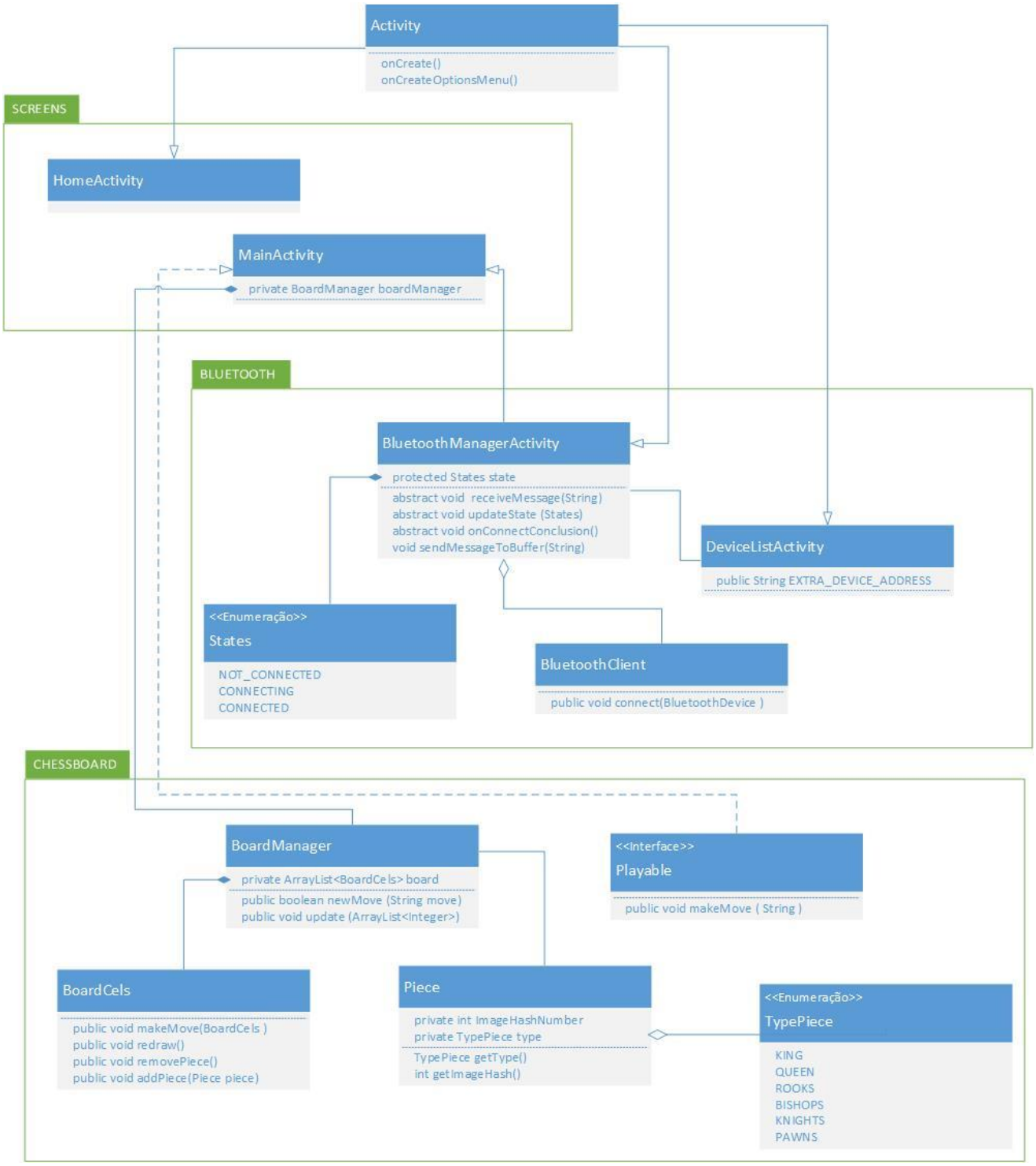


Figura 20: Diagrama de classes da aplicação para Android

5. Comunicação Serial

Existem diversas opções para implementar a comunicação entre o Arduino e o Android, tais como: infravermelho, USB, Wi-Fi, *Bluetooth*, entre outras. Do ponto de vista da aplicação, a comunicação tanto no aplicativo Android quanto no Arduino é feita por um módulo ou uma classe na qual é acessível uma forma de enviar e receber mensagens. Essa abstração permite que independentemente da opção feita ou de alguma eventual modificação na forma como a comunicação é feita, não haja interferência no restante do código.

No projeto foi utilizada, como já mencionado ao longo deste trabalho, uma comunicação via *Bluetooth*. Isso se deu por alguns motivos, tais como: não existe a necessidade de se jogar e interagir com o tabuleiro de longe, portanto, uma comunicação de curto alcance é suficiente; a necessidade de confiabilidade, mas não extrema; e a simples implementação em ambas as plataformas. Uma outra implementação possível que atenderia a essas condições até com mais segurança seria a comunicação via Wi-Fi, entretanto, o custo desta implementação no Arduino é muito superior ao do *Bluetooth*.

O *Bluetooth* é uma tecnologia para conexão de dispositivos num raio de até 15 metros, especificado pelo padrão da IEEE 802.15.1, utiliza a técnica de espectro-espalhado em 2,4GHz, com velocidade de comunicação de 432,6kB/s no caso de configuração simétrica (mesma velocidade em ambos os sentidos), e suporta até três canais *full-duplex* (comunicação bidirecional e simultânea) [23].

5.1. Protocolo adotado

Durante os teste iniciais do sistema de comunicação Android-Arduino, foi observado que mesmo a curta distância em ambiente de pouca interferência uma em cada cinco mensagens em média vinha com erro (um bit trocado), para contornar esse problema foi adotado o protocolo de comunicação demonstrado na Figura 21. Por padronização, toda mensagem, seja ela de

comunicação, de confirmação, ou qualquer outra será composta de quatro caracteres ASCII.

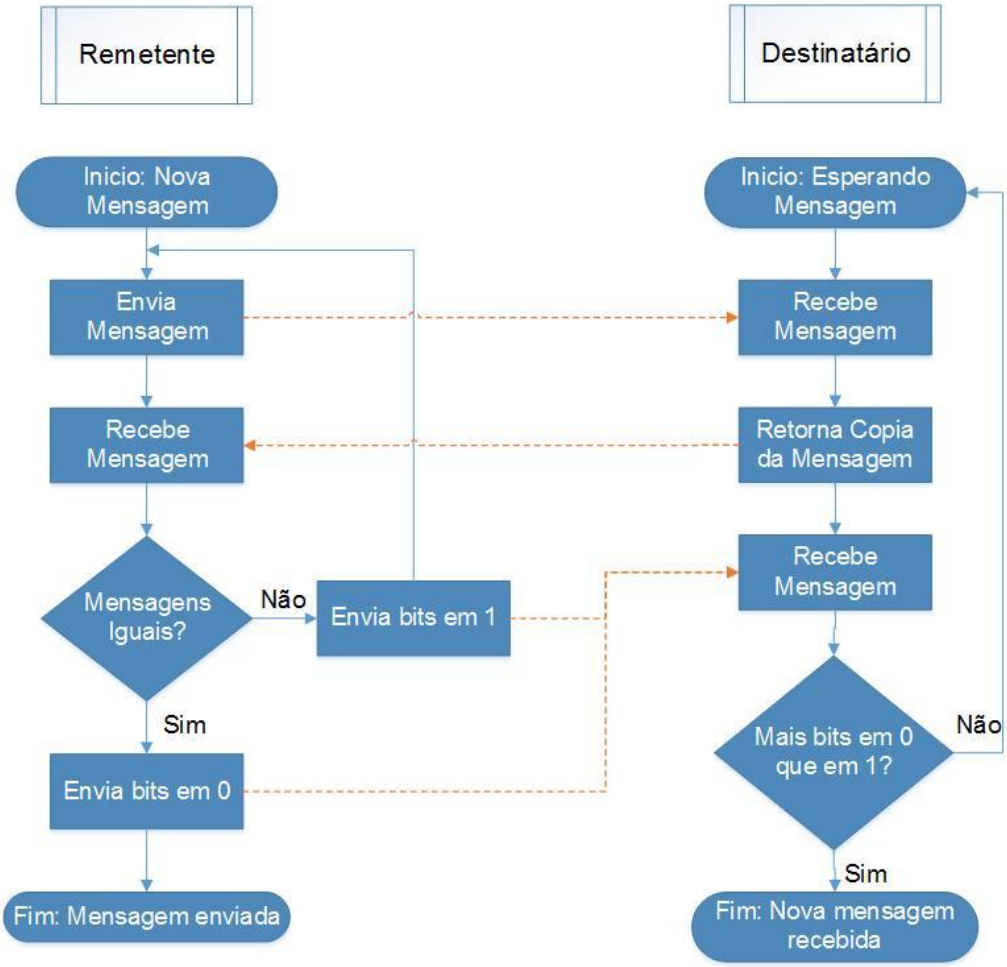


Figura 21: Protocolo de comunicação adotado

Como o erro poderia ocorrer em qualquer passo da troca de mensagens, para garantir que a mensagem recebida seja a mesma que a enviada, o destinatário ao receber uma mensagem retorna a mesma mensagem e aguarda uma terceira mensagem (uma confirmação). O núcleo da lógica deste protocolo se baseia na mensagem de confirmação, pois caso o remetente receba uma cópia idêntica à mensagem original, significa que a mensagem foi enviada em ambos os sentidos sem ocorrer erro, então é enviado um sinal de confirmação, caso difira é enviado um sinal de negação.

Como pode haver erro mesmo na mensagem de confirmação foi adotado o método que supõe que ocorreram menos *bits* trocados que *bits* mantidos, a mensagem de confirmação é o complemento da mensagem de

negação. Na confirmação, todos os *bits* dos quatros caracteres se encontram em 0, enquanto na negação todos os *bits* em 1. Assim para verificar se a mensagem foi confirmada ou negada, verifica-se a quantidade de bits em 0 e em 1, havendo mais *bits* em 0 é uma confirmação, havendo mais *bits* em 1 uma negação. Caso seja uma negação o processo é reiniciado.

Considerando que todas as mensagens são testadas por esse processo, na Tabela 2 estão as mensagens de comunicação e suas funções. Por padrão toda mensagem de comunicação deve começar com o caráter '?'. A mensagem que representa uma jogada é a notação padrão do xadrez em código ASCII. Por exemplo, para mover o peão do rei no primeiro movimento do jogo, a mensagem gerada é E2E4, onde a letra representa a coluna da direita para a esquerda e as letras representam as linhas de baixo para cima.

Tabela 2: Mensagens do protocolo

Mensagem	Significado	Sentido
?099	Jogada Válida	Arduino -> Android
?100	Jogada Inválida	Arduino -> Android
?101	Requisição de Atualização de Tabuleiro	Android -> Arduino
?101	Confirmação de Atualização de Tabuleiro	Arduino -> Android
?102	Fim de Atualização de Tabuleiro	Arduino -> Android
?300	Peças pretas venceram	Arduino -> Android
?301	Peças brancas venceram	Arduino -> Android
?302	Empate	Arduino -> Android
?500	Solicitação para jogar com as pretas	Android -> Arduino
?501	Solicitação para maquina jogar pelo usuário	Android -> Arduino
?555	Negação para a solicitação ?500	Arduino -> Android
?900	Solicitação de desconexão	Android -> Arduino
?901	Confirmação de desconexão	Arduino -> Android
?910	Solicitação de conexão	Android -> Arduino

As mensagens tem funções bem claras, não sendo necessário nenhuma colocação adicional sobre o protocolo. A única exceção é o sistema de atualização do tabuleiro. Nessa situação, o usuário do aplicativo para Android

deseja que a aplicação atualize as posições das peças para ficar igual as peças do tabuleiro, enviando a mensagem ‘?101’.

O Arduino ao receber essa requisição envia um código de aceitação, no caso a mesma mensagem ‘?101’, e uma sequência de mensagens que representam o conteúdo de cada uma das casas percorridas, conforme a Figura 22, da direita para a esquerda de cima para baixo, analisando cada linha de uma vez e varrendo todos os elementos de cada linha antes de ir para a próxima. O conteúdo das casas é associado às mensagens pela Tabela 3, ao final do processo é enviado a mensagem de fim ‘?102’.

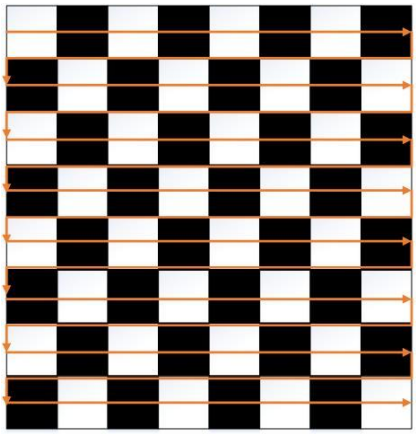


Figura 22: Varredura de Atualização

Tabela 3: Mensagem de atualização

Mensagem	Significado	Sentido
?000	Casa Vazia	Arduino -> Android
?001	Rei Branco	Arduino -> Android
?002	Rainha Branca	Arduino -> Android
?003	Torre Branca	Arduino -> Android
?004	Bispo Branco	Arduino -> Android
?005	Cavalo Branco	Arduino -> Android
?006	Peão Branco	Arduino -> Android
?007	Rei Preto	Arduino -> Android
?008	Rainha Preta	Arduino -> Android
?009	Torre Preta	Arduino -> Android
?010	Bispo Preto	Arduino -> Android
?011	Cavalo Preto	Arduino -> Android
?012	Peão Preto	Arduino -> Android

Com essas mensagens, só existe uma sinalização não prevista, que ocorre quando a placa de leitura identifica que uma determinada peça está fora do lugar. Como a placa apenas identifica aonde as peças estão, o código não é capaz de corrigir uma situação inesperada. Para tal, é enviado um sinal para o Arduino informando qual ou quais casas estão diferentes do que deveriam, com código ?7XY, onde X varia de 1 a 8 representando as colunas de A à H, respectivamente, e Y de 1 a 8 representando as linhas. A cada ciclo de leitura é enviado um sinal de reset para que caso alguma casa seja corrigida ela deixe de ser sinalizada. A mensagem de reset é ?700. Quando todas a posições forem corrigidas é enviado um sinal de encerramento da varredura que habilita uma nova jogada, a mensagem ?800.

5.2. Android

Conforme descrito na seção 4.2 e demonstrado no diagrama de classes da Figura 20, a aplicação desenvolvida se baseia na aplicação *BluetoothChat* disponibilizado pela própria Google. Primeiramente, a *Activity* testa se o *Bluetooth* está disponível no aparelho. Caso não esteja, já encerra avisando ao usuário da incompatibilidade com a aplicação e caso esteja disponível, pede autorização ao usuário para habilitar o *Bluetooth.* Essas inicializações estão implementadas na classe abstrata **BluetoothManagerActivity.** Quando o *Bluetooth* já está iniciado é invocada a **DeviceListActivity**, utilizando um recurso da estrutura da programação para Android. Essa classe é chamada e ao ser encerrada o método **onActivityResult** da *Activity* que a chamou é acionado passando como parâmetro um identificador e inserindo na *Activity* as informações acessadas através de uma chave. No caso, conforme o diagrama de classes, a chave é EXTRA_DEVICE_ADRESS**.**

Essa chave dá acesso às informações necessárias sobre qual dispositivo foi selecionado na **DeviceListActivity**. Essas informações serão utilizadas para fazer o pareamento entre os dispositivos.

Quando o dispositivo é conectado, a *Activity* abstrata encapsula todos os tratamentos e a forma de envio das mensagens, conforme já discutido na seção 4.2.

5.3. Arduino

Para utilizar o *Bluetooth* com o Arduino foi necessária à aquisição de um *shield* específico, no caso, o *Bluetooth Mate Silver* (Figura 23*)*, disponível no site da Sparkfun [24]. Sua utilização é bastante simplificada. Do ponto de vista do Arduino, a comunicação *Bluetooth* é uma comunicação serial como outra qualquer, sendo suportado por código nativo da IDE, com portas já especificas de transmissor e receptor serial.

Figura 23: Bluetooth Mate Silver, retirado de [24]

A troca de mensagens é feita pelos comandos da Tabela 4, dado que todo o protocolo de descoberta e conexão característicos do *Bluetooth* são implementados e de responsabilidade do *shield*.

Tabela 4: Mensagens de comunicação do Arduino

	Comando
Enviar Mensagem	Serial1.print(char)
Receber Mensagem	charSerial1.read();

6. Resultados

6.1. Análise da movimentação dos eixos

Para análise da movimentação mecânica que será utilizada nos testes dos algoritmos de movimentação na seção 6.2 é importante observar a movimentação dos eixos isoladamente. Nesta seção serão analisadas a movimentação dos eixos considerando a execução de uma jogada que vá apenas para a casa vizinha, como forma de verificar o movimento de uma única casa.

Dois testes, um para cada eixo, foram executados e os resultados se encontram na Tabela 5. O eixo X movimenta de uma coluna para outra e o eixo Y de uma linha para outra. O teste do eixo X consiste em percorrer o tabuleiro todo movendo-se de casa em casa pelas colunas e repetindo o teste para cada uma das linhas. O inverso análogo foi realizado para o eixo Y se movendo de uma linha para outra repetindo o teste para cada uma das colunas.

Tabela 5: Tempo médio de movimentação de uma casa

	Eixo X (ms)	Eixo Y (ms)
Média	2187,071	3761,786
Desvio padrão	2,801669	2,19681
Máximo	2191	3766
Mínimo	2185	3760
Mediana	2185	3760,5
Moda	2185	3760

6.2. Teste dos algoritmos de movimentação

Para testar os algoritmos de movimentação, ou seja, tanto o algoritmo de melhor caminho quanto o algoritmo de desvio, três diferentes situações foram analisadas onde o nível de dificuldade aumenta em cada uma delas, vale ressaltar que os cenários não são característicos do jogo e não se atêm as regras de movimentação, são testes genéricos para os algoritmos de movimentação.

Os tempos de execução do algoritmo com e sem execução mecânica se encontram na Tabela 6. Nenhum dos cenários se baseia nas posições características do jogo. A análise do algoritmo é genérica e, portanto, o teste é feito de forma a demonstrar as características dos algoritmos. As figuras que representam os cenários adotaram o seguinte padrão:

- Peão branco representa uma casa ocupada
- Rainha branca representa peça que será movimentada
- Peão preto representa uma peça que foi retirada do lugar
- A casa em destaque representa a posição final do movimento

Tabela 6: Tempos de execução dos algoritmos de movimentação

	Sem execução (ms)	Com execução (s)
Cenário 1	2	58,132
Cenário 2	5	160,355
Cenário 3	35	536,519

Cenário 1

Na Figura 24, pode-se ver o primeiro cenário de testes, a movimentação é feita da casa (0,0) para a casa (2,2), com o objetivo de visualizar como o algoritmo lida com o contorno de obstáculos.

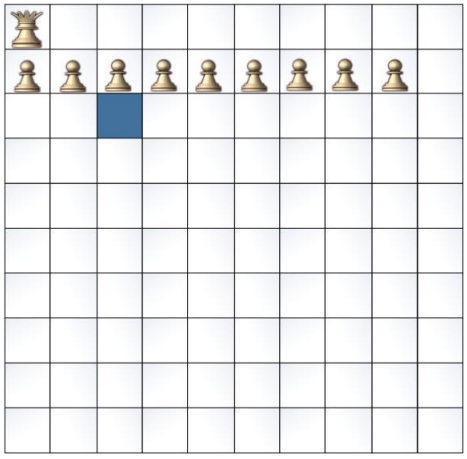


Figura 24: Cenário de teste

Pode-se ver na Figura 25 que a execução deste algoritmo é bem simples, não sendo necessário nenhum desvio este é o cenário mais simples e mais comum durante o jogo.

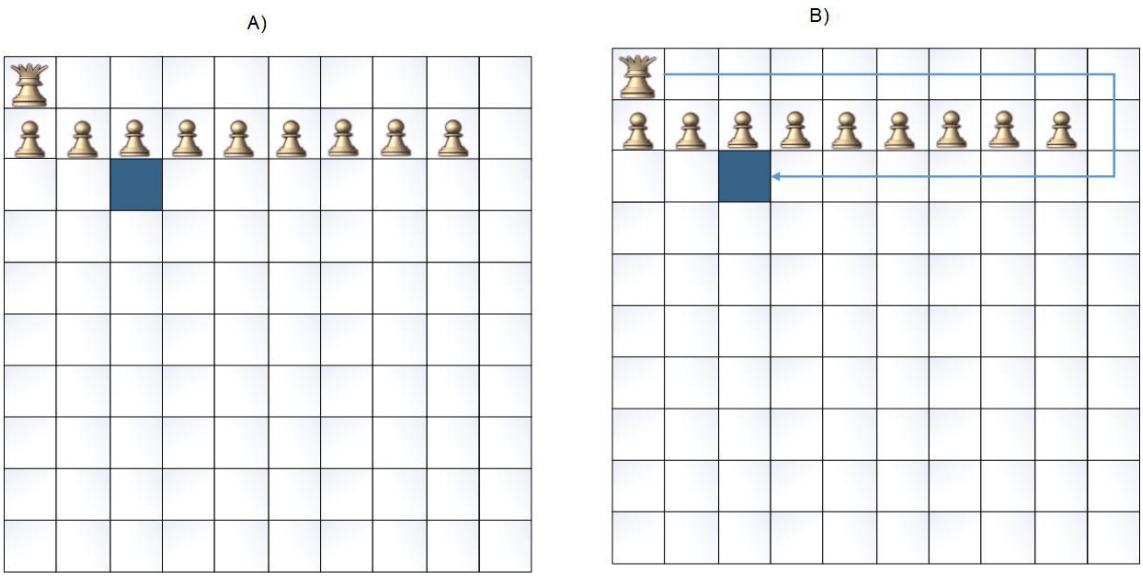


Figura 25: Resultado do algoritmo para o cenário de teste 1

Cenário 2

Na Figura 26, é possível ver o segundo cenário de testes, a movimentação é feita da casa (2,2) para a casa (9,8), com o objetivo de visualizar como o algoritmo lida com desvios simples.

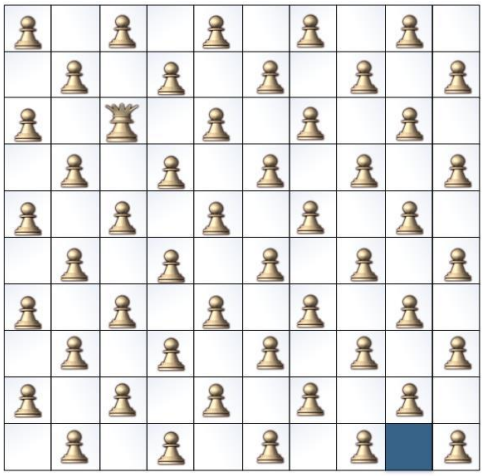
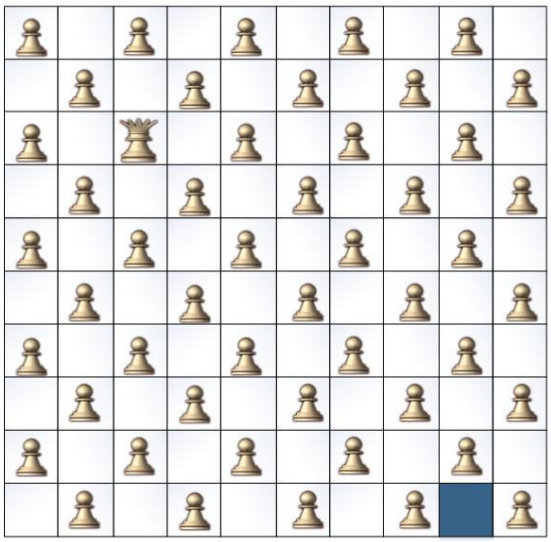


Figura 26: Cenário de teste 2

Pode-se ver na Figura 27 que a execução deste algoritmo é mais complexa, sendo necessários seis desvios, mas todos os desvios são simples e de apenas uma casa. Vale observar que a movimentação total não é

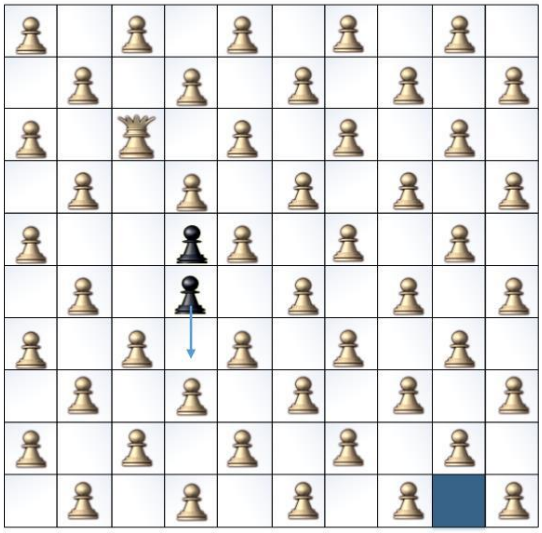
otimizada. Para otimizar o movimento dever-se-ia fragmentar o movimento principal e os desvios e calcular qual a sequência de execuções que resultariam em um menor tempo de execução. Considerando os resultados da Tabela 6, vale destacar que o tempo necessário para calcular o caminho não é tão superior mesmo sendo necessário calcular os desvios de 2ms para 5ms. Entretanto, na execução esse tempo se torna mais significativo.

A)

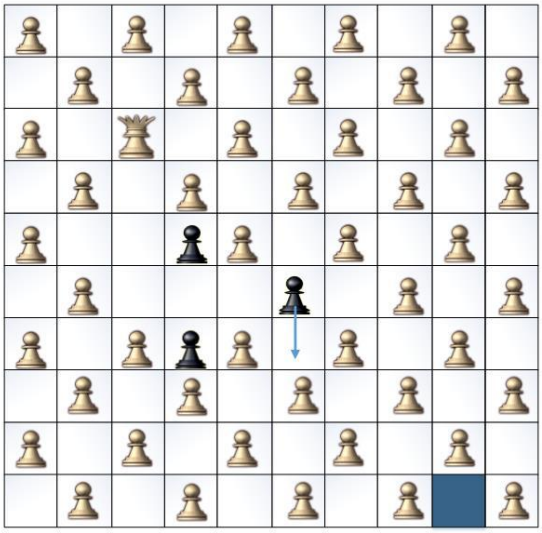


B)

C)

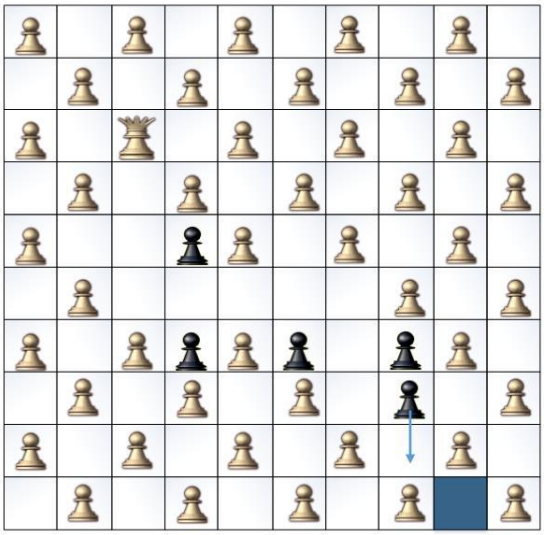


D)

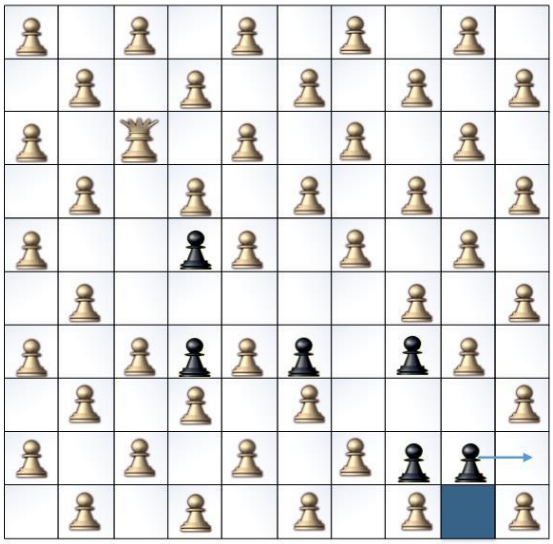


E)

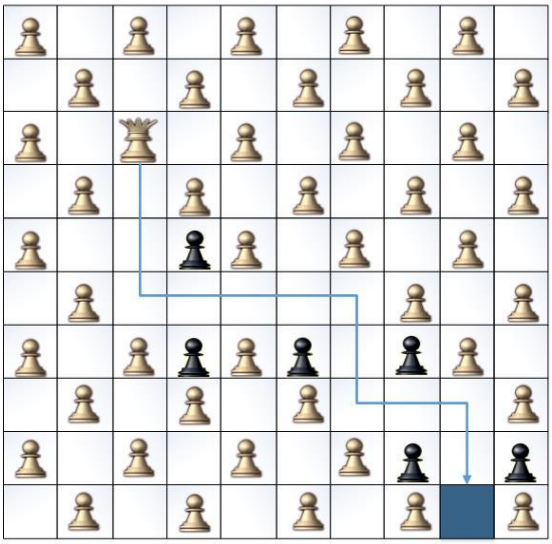
F)



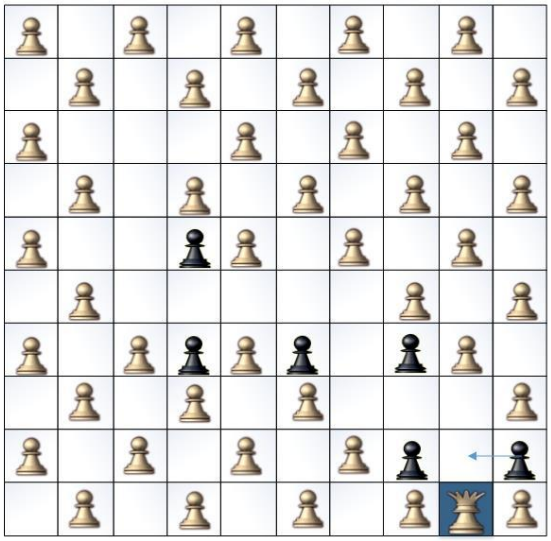
G)



H)



I)



J)

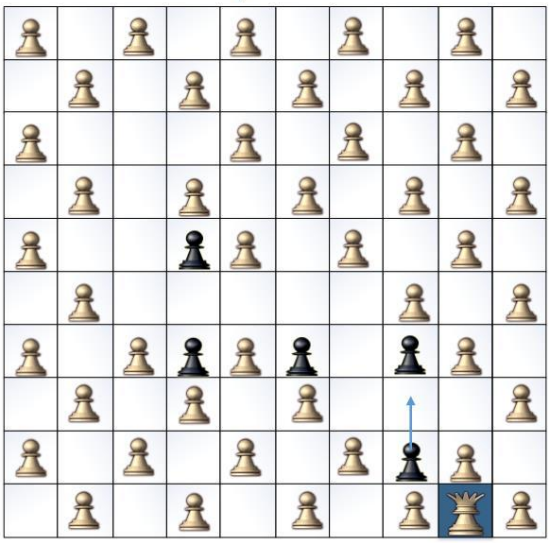


Figura 27: Resultado do algoritmo para o cenário de teste 2

Cenário 3

Na Figura 28, está demonstrado o terceiro cenário de testes. A movimentação é feita da casa (0,0) para a casa (9,9), com o objetivo de visualizar como o algoritmo lida com desvios complexos, ou seja, com desvios que necessitam de desvios. A presença da casa (6,7) vazia vai causar isso no algoritmo.

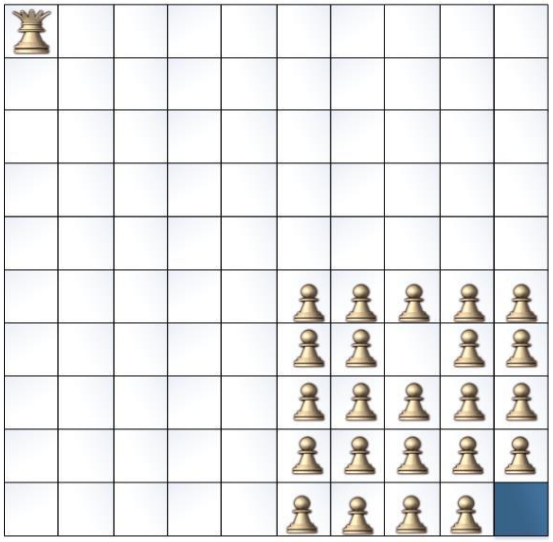
Figura 28: Cenário de teste 3

Pode-se ver na Figura 29, que a execução deste algoritmo é mais complexa, sendo necessários dezoito desvios, alguns em cascata. Por isso só foi mostrada a parte de ida do algoritmo sendo ainda necessário desfazer os movimentos da imagem 2 à 19. Vale ressaltar que uma situação como essa é muito fora do esperado para um jogo de xadrez. A quantidade de desvios para o cenário 2 já é bastante fora do padrão para o jogo, dado os teste já realizados. Na verdade em situação de jogo só foi presenciado a necessidade de um desvio, no caso de um roque. Este cenário deixa clara a não otimização do algoritmo de movimentação. Observa-se, por exemplo, que o movimento (4,8) para (3,8) é feito e desfeito dezesseis vezes ao longo da solução mostrada.

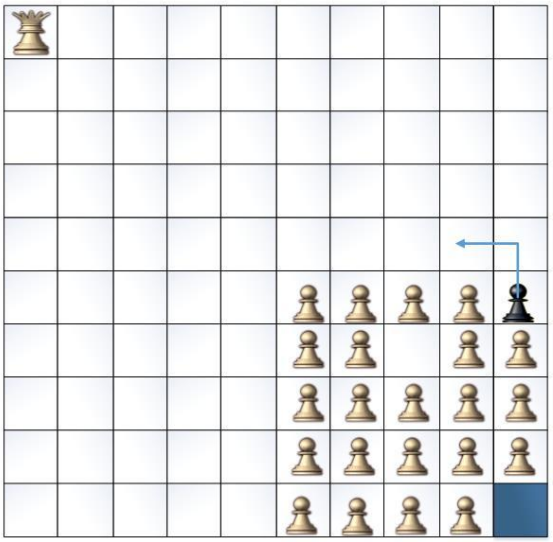
Analisando o resultado do algoritmo passo a passo, consideremos o caminho desejado para mover a rainha na imagem 20. Observa-se que para execução deste movimento seriam necessários apenas quatro desvios, casas (5,9), (6,9), (7,9) e (8,9). O algoritmo se estende porque quando se vai desviar a peça da casa (6,9) a casa vazia mais próxima em raio é a (6,7). Isto desencadeia os movimentos da imagens 3 até a 17. Devido ao caráter recursivo do algoritmo e ao fato de que para realizar um movimento se faz e desfaz os desvios, o algoritmo se estende. Observa-se que para realizar o movimento (6,9) para (6,7) é necessário mover a peça da casa (6,8). Como a casa vazia mais próxima é a (5,9), entretanto, para fazer esse movimento é necessário por sua vez mover a peça da casa (5,8). Como a casa (5,9) é o

destino do movimento anterior, a casa mais próxima para mover a (5,8) é a casa (4,9). Novamente, no movimento da (5,8) para a (4,9) é necessário desviar a peça da casa (4,8), ou seja, o algoritmo caiu no caso de quatro desvios em cascata. Esse é o pior caso para o algoritmo, uma vez que algoritmo de desvio não é otimizado, ou seja, a escolha da casa é feita por proximidade e não por eficiência.

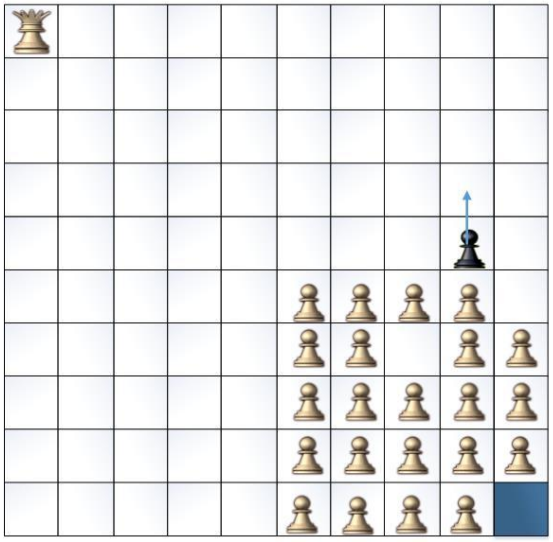
1)



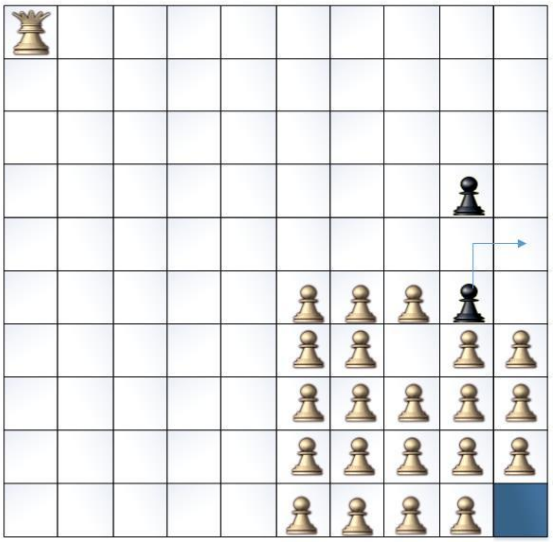
2)



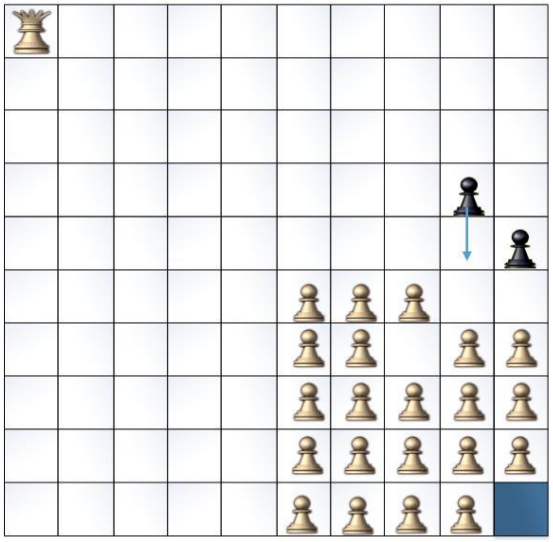
3)



4)

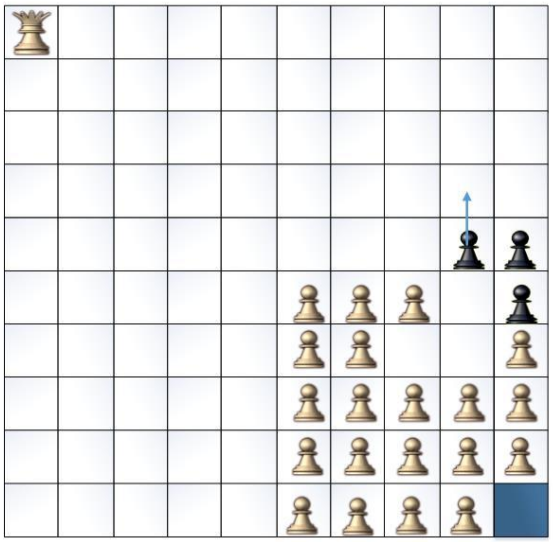


5)



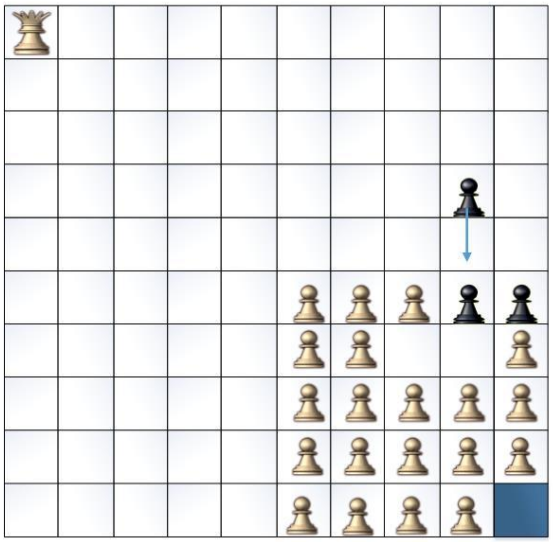
6)

7)

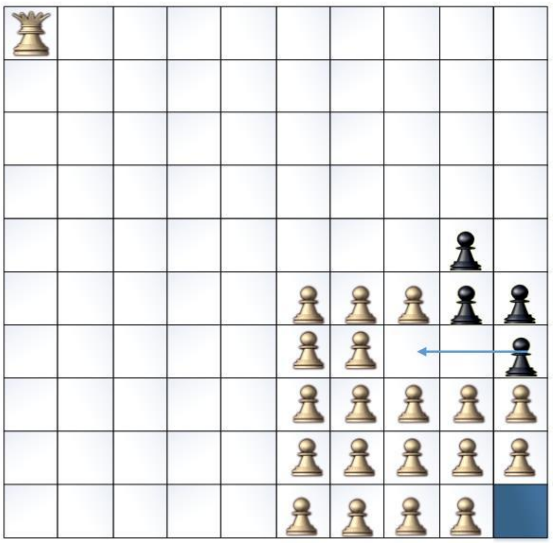


8)

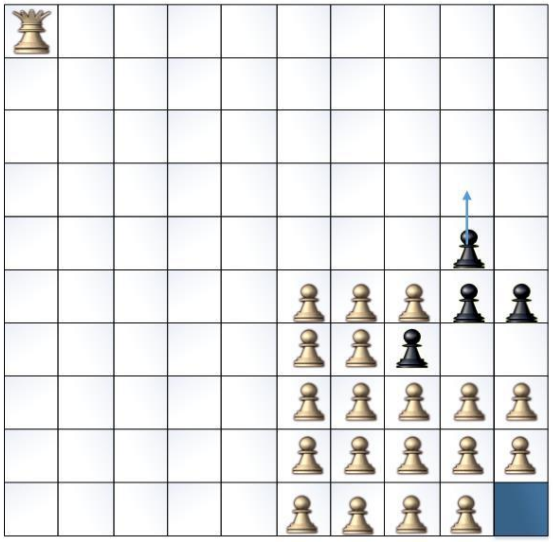
9)



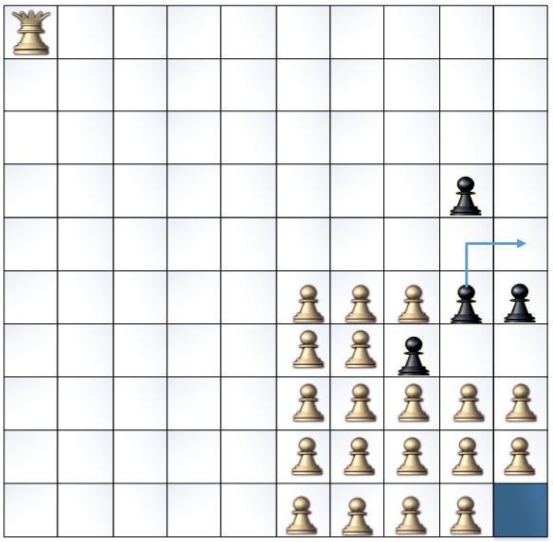
10)



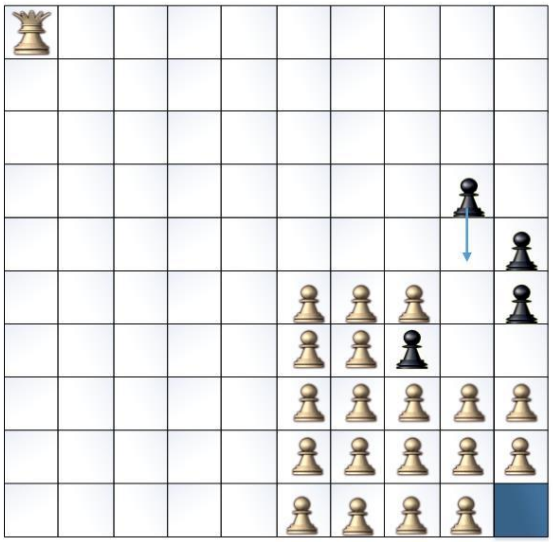
11)



12)

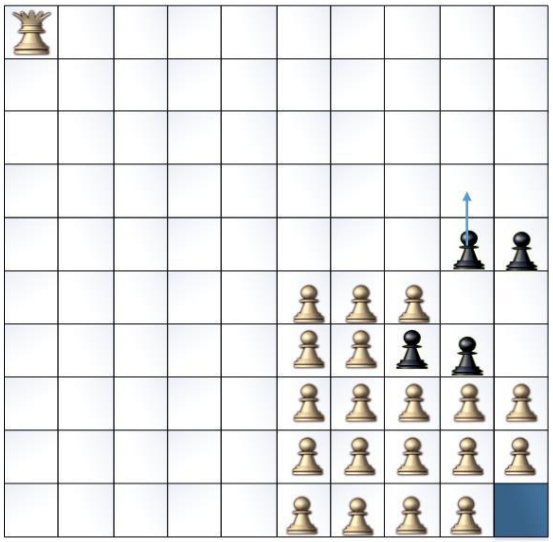


13)

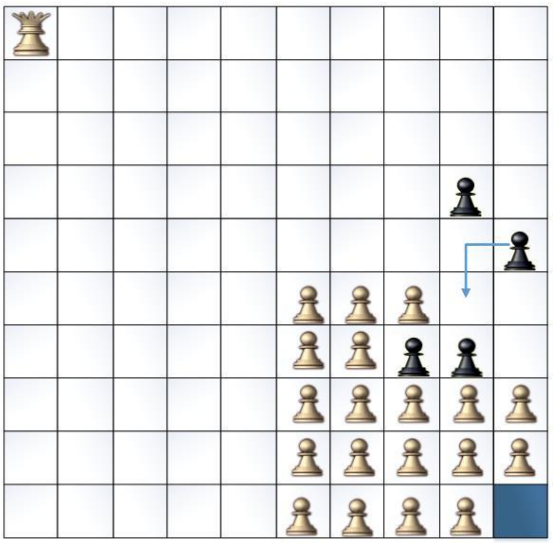


14)

15)



16)



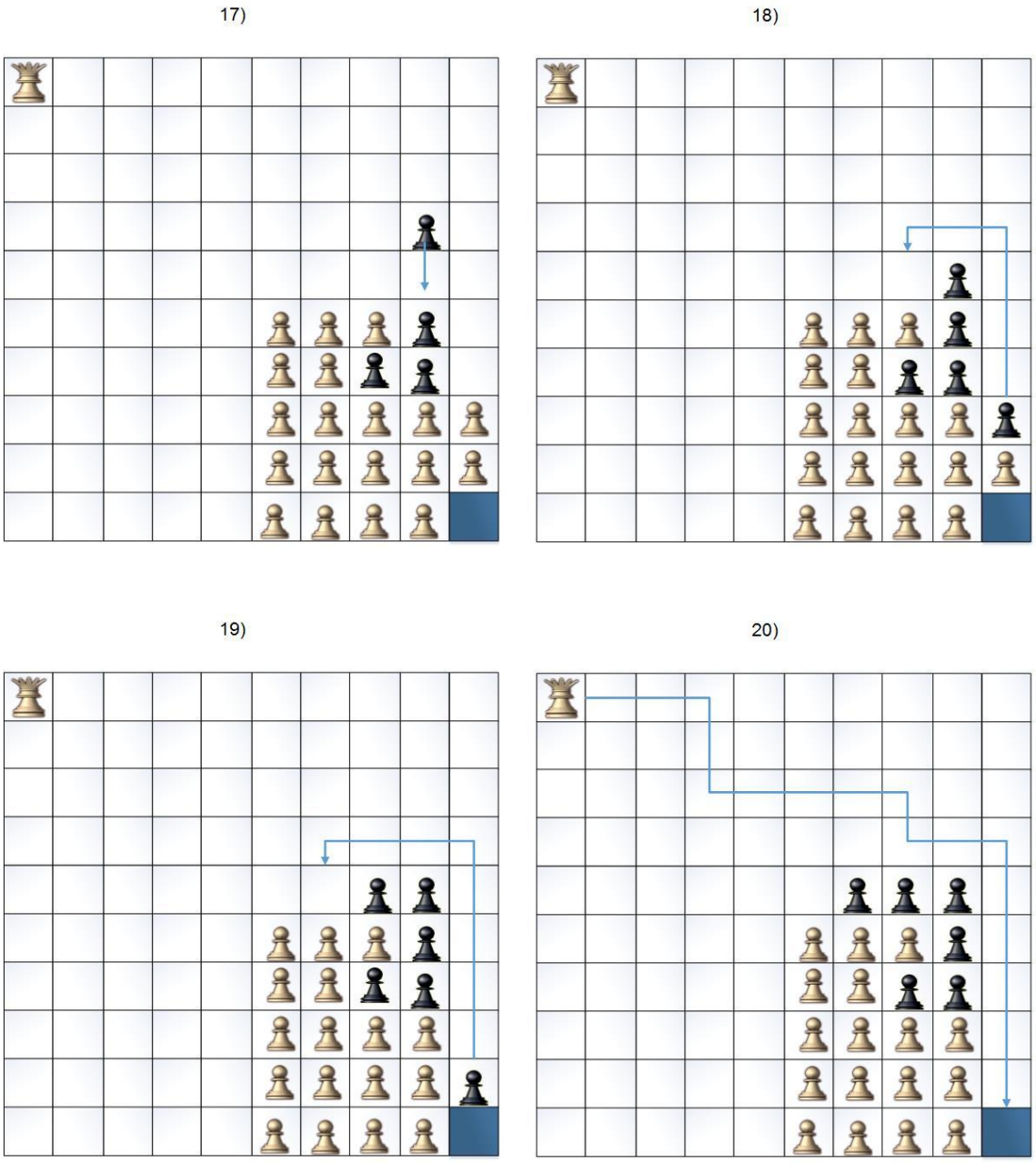


Figura 29: Resultado do algoritmo para o cenário de teste 3

6.3. Análise da placa de leitura

A placa de leitura é um ponto crítico do projeto, inicialmente foi testada casa a casa, sendo corrigidas as imperfeições das soldas. Concluída a etapa inicial de análise com todas as casas funcionando corretamente, foi dado prosseguimento na elaboração de uma lógica capaz de identificar uma jogada. Para simplificar, foi assumido que em uma jogada na qual o usuário captura

uma peça, primeiramente ele deve retirar a peça capturada e posteriormente mover a peça que a capturou para o local dela. Dessa forma, seria possível detectar a captura comparando uma varredura inicial do tabuleiro com uma varredura posterior. Dessa forma seria percebido três modificações: uma peça sumiu, uma segunda peça sumiu e uma peça apareceu no local de onde a primeira sumiu. Sendo esta a situação, com peça capturada, a mais difícil de identificar como uma jogada. Isso funcionou bem em etapas de teste e portanto essa lógica foi integrada ao projeto com a inteligência artificial e o *Bluetooth*.

Nesse ponto começaram os maiores problemas, nos primeiros testes o eixo mecânico colidiu algumas vezes com os conectores o que culminou na ruptura da solda dos mesmos e no descolamento de algumas trilhas. Mas esses problemas foram resolvidos. Entretanto, um dos conectores, ocasionalmente, dava problema e o Arduino perdia o controle de uma das colunas, a H, pois essa foi a conexão onde a trilha soltou com mais violência. Mas, do ponto de vista prático, foi um problema resolvido.

O maior problema, na verdade, tem uma causa desconhecida, os sensores funcionam perfeitamente, individualmente, entretanto, três casas (mais especificamente a A4, B8 e E7) tem um comportamento estranho durante o jogo. Tipicamente, quando uma peça é removida de uma dessas casas o sensor continua indicando sua presença. Isso ocorre sem um padrão correto, mas observou-se que existe uma possível correlação com a presença de peças em volta da casa. Ou seja, como se esses sensores especificamente, fossem capazes de detectar, como se houvesse uma peça, quando um conjunto de peças é posta ao redor. Entretanto, esse fenômeno não é regular e acontece aleatoriamente. Esse fenômeno foi observado em outras casas, com menor frequência e com menor sensibilidade, muitas vezes relacionados a pequenos deslocamentos de uma peça para fora de uma casa influenciando a casa vizinha.

Na prática, este problema atrapalha muito a fluidez do jogo. Como, a cada nova jogada realizada, o tabuleiro é varrido em busca de diferenças entre o tabuleiro interno (tabuleiro virtual de referência no qual baseia-se a inteligência artificial) e o tabuleiro físico, de forma que essas diferenças possam ser mostradas na interface gráfica do Android, é frequente que a cada ciclo de

iteração seja necessário ajeitar uma ou mais peças ao redor dessas casas para que o jogo possa prosseguir.

Como as outras sessenta e uma casas se comportam bem, é de se admitir a viabilidade do projeto. Entretanto, com o objetivo de eliminar esse empecilho na comparação entre o medido e o interno, foi ignorada a comparação nessas três casas. Mas, elas ainda são consideradas como criadoras de eventos para novas jogadas. Entretanto, é possível que ocorram erros em jogadas físicas que envolvam essas casas, sendo recomendado que essa jogadas sejam executadas no aplicativo Android.

7. Conclusões e Trabalhos Futuros

A elaboração deste trabalho envolveu diversas áreas de conhecimento abordadas no curso, indicando o teor multidisciplinar do mesmo. Neste projeto houve a integração entre *hardware* e *software* e até mesmo um arcabouço mecânico para suportar o movimento das peças. Foram utilizadas duas tecnologias de fácil acesso e desenvolvimento, porém bastante poderosas e com futuro cada vez mais promissor, o Android e o Arduino. O fato de serem dois projetos *open-source* contribuiu bastante para sua entrada no projeto, pois acreditamos que projetos como esses viabilizam, capacitam e encorajam novos desenvolvedores a se aventurarem na criação de novos artefatos tecnológicos.

Do ponto de vista da gerência do projeto, tivemos várias experiências associadas à administração dos problemas previstos e imprevistos para um projeto de engenharia que envolva a integração de tantos artefatos como este apresentado, tais como: escolha de um sensor adequado, problemas mecânicos com o carro do eixo, peças com a base praticamente do tamanho da casa exigindo maior precisão no movimento das peças, eixo que colidia com o conector da placa de leitura, alinhamento dos eixos, ruptura insistente de alguns pontos de solda, disposição dos fios dentro da caixa do tabuleiro de forma que não houvesse travamento das partes móveis (vide Figura 30 e Figura 31), etc.

Figura 30: Tabuleiro Autônomo

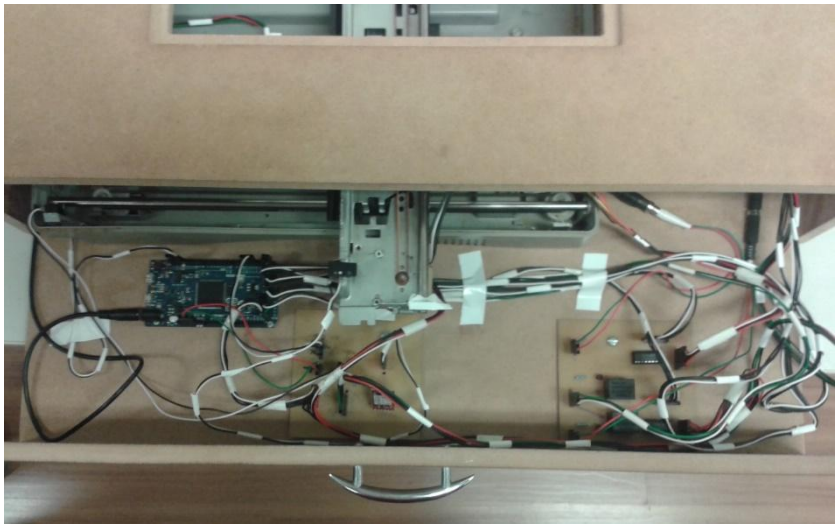


Figura 31: Disposição dos elementos de *hardware* dentro da caixa

Dentre os problemas citados acima, alguns foram gerados por decisões tomadas no início do projeto, como, por exemplo, o tamanho das peças. Como foram utilizadas peças de um tabuleiro de madeira que já possuíamos, isto acabou por ser um fator determinante, pois as mesmas tem a base com o diâmetro praticamente do tamanho da casa (vide Figura 32). Isso exigiu maior precisão na movimentação dos eixos e do alinhamento das peças dentro das casas para que não houvesse grandes colisões (embora ainda ocorram eventualmente colisões pequenas, ou seja, sem que haja derrubada de peças). Para solucionar esse problema é possível utilizar peças menores ou aumentar o tamanho das casas, entretanto eram as peças que possuíamos e o tamanho

das casas foi limitado pelo tamanho do menor dos eixos. A utilização dessas peças apesar da dificuldade de alinhamento se deve em parte ao fato de que o início deste projeto ocorreu na matéria Projeto Integrado, onde tínhamos um pouco mais de 3 meses para realizá-lo.

Figura 32: Peças utilizadas

Outro problema de destaque dentro do projeto foi a adaptação do arcabouço mecânico, composto por um eixo de impressora sobre um eixo de *scanner*. Foi bem custoso encontrá-los, pois era necessário que possuísse motor de passo, dado que a implementação com motor DC aumentaria muito o grau de complexidade do controle necessário para realizar o movimento e a grande maioria dos eixos de impressora modernos o utilizam.

Foi necessário, então, adaptar um motor de passo em um eixo que utilizava um motor DC o que é bem difícil de funcionar devido a incompatibilidade de tamanho dos motores e dos eixos, sendo os motores DC geralmente menor e com eixos mais finos. Além disso, um dos eixos utilizados possui região útil muito próxima a distância que ele deve percorrer, o que dificultou o alinhamento e reduziu a flexibilidade do projeto como um todo durante seu desenvolvimento.

Apesar dos problemas citados, é possível dizer que os objetivos traçados foram devidamente atingidos e que o projeto é funcional, sendo possível jogar uma partida inteira contra a inteligência artificial, seja movimentando manualmente as peças ou através do aplicativo Android. É possível, inclusive alternar entre os dois modos durante um jogo, automaticamente, sem necessidade de qualquer acionamento de mudança de modo de jogo.

Para trabalhos futuros, podemos citar o melhor tratamento ou até a implementação do suporte a certas jogadas especiais, como por exemplo:

- **Roque** – Embora o sistema execute esta jogada no tabuleiro do jogo, para observar a realização da jogada no aplicativo é necessário que o usuário atualize o mesmo.
- ***En Passant*** – Dada sua particularidade única que a peça que captura se movimenta para uma casa onde previamente não havia nenhuma peça adversária, essa jogada não foi implementada.
- **Promoção** – É implementada pela inteligência artificial, mas dificulta a implementação física no tabuleiro dado que um peão pode virar, por exemplo, uma rainha mesmo que a rainha ainda esteja em jogo. Em termos práticos, se o usuário atualizar o tabuleiro ele terá acesso ao jogo com a peça promovida, entretanto, fisicamente a peça ainda será um peão.

Apesar dessas situações de jogo não implementadas ou parcialmente implementadas, vale ressaltar que no sentido prático pouca coisa mudaria para implementação dessas jogadas, não sendo necessária nenhuma modificação física no protótipo, apenas adição de um conjunto de testes para essas condições.

8. Referências Bibliográficas

[1] A. Newell, J. C. Shaw e H. A. Simon, “Chess-playing programs and the problem of complexity,” *IBM Journal of Research and Development,* pp. 320-335, 1958.

[2] C. Donninger e U. Lorenz, “Innovative Opening-Book Handling,” *Lecture Notes in Computer Science,* pp. 1-10, 2006.

[3] C. E. Shannon, “Programming a Computer for Playing Chess,” *Philosophical Magazine,,* pp. 256-275, 1950.

[4] T. Kerrigan. [Online]. Available: http://www.tckerrigan.com/Chess/TSCP. [Acesso em 10 julho 2014].

[5] R. E. Korf e D. M. Chickering, “Best-First Minimax Search: First Results,” *AAAI Technical Report,* pp. 39-47, 1993.

[6] J. Pearl, “HEURISTIC SEARCH THEORY: SURVEY OF RECENT RESULTS,” em *International Joint Conference on Artificial Intelligence (IJCAI)*, Vancouver, 1981.

[7] T. H. Cormem, C. E. Leiserson, R. L. Rivest e C. Stein, Introduction to algorithms, Cambridge: MIT Press 2013, 2009.

[8] Arduino, “What is Arduino?,” 2014. [Online]. Available: http://arduino.cc/en/Guide/Introduction.

[9] Arduino, “Arduino Due,” 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardDue.

[10] J. M. Osier-Mixon, “Hardware Aberto: Como e Quando Funciona,” 06 dezembro 2010. [Online]. Available: http://www.ibm.com/developerworks/br/library/os-openhardware/.

[11] Gentequecooperacresce, “Hardware livre: a tecnologia a alcance de todos,” 24 Setembro 2013. [Online]. Available: http://www.gentequecooperacresce.com.br/site/post.php?t=hardware-livre-a-tecnologia-ao-alcance-de-todos&id=1402.

[12] J. M. Pearce, “Building Research Equipment with Free, Open-Source Hardware,” *Science,* pp. 1303-1304, 19 09 2012.

[13] Alliance, Open Handset, “FAQ,” Novembro 2007. [Online]. Available:

http://www.openhandsetalliance.com/oha_faq.html.

[14] R. Rogers, J. Lombardo, Z. Mednieckse B. Meike, Android Application Development, Sebastopol: O'Reilly, 2009.

[15] SourceAndroid, “ADT Plugin,” 2014. [Online]. Available: http://developer.android.com/tools/sdk/eclipse-adt.html.

[16] SourceAndroid, “Introducing ART,” 2014. [Online]. Available: https://source.android.com/devices/tech/dalvik/art.html.

[17] C. Welch, “Before it took over smartphones, Android was originally destined for cameras,” 16 Abril 2013. [Online]. Available: http://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones.

[18] M. Wilson, “T-Mobile G1: Full Details of the HTC Dream Android Phone,” 23 Setembro 2008. [Online]. Available: http://gizmodo.com/5053264/t-mobile-g1-full-details-of-the-htc-dream-android-phone.

[19] SourceAndroid, “The Android Source Code,” 2014. [Online]. Available: https://source.android.com/source/index.html.

[20] R. Stallman, “Is Android really free software?,” 19 Setembro 2011. [Online]. Available: http://www.theguardian.com/technology/2011/sep/19/android-free-software-stallman.

[21] DeveloperAndroid, “Introduction to Android,” 2014. [Online]. Available: https://developer.android.com/guide/index.html.

[22] GooglePlay, “Bluetooth Chat,” 14 11 2013. [Online]. Available: https://play.google.com/store/apps/details?id=com.blong.bluetoothchat&hl=pt_BR.

[23] S. Dhawan, “Analogy of Promising Wireless Technologies on Different Frequencies:,” em *Wireless Broadband and Ultra Wideband Communications (AusWireless 2007). The 2nd International Conference on* , Sydney, 2007.

[24] sparkfun, “Bluetooth Mate Silver,” 2014. [Online]. Available: https://www.sparkfun.com/products/12576. [Acesso em 22 07 2014].

Apêndice A – Layouts e componentes

Este apêndice a título de ilustração apresenta os *layouts* utilizados e a descrição dos componentes. Na Figura 33 e Figura 34 encontram-se os *layouts bottom* e *top* da placa de leitura, o esquemático simplificado exibindo apenas um sensor, um multiplexador e um conector de saída se encontra na Figura 36.

Na Figura 35 encontra-se o *layout* da placa de acionamento dos motores e a Figura 37, o seu esquemático. Na Figura 38, o layout da placa auxiliar, essa placa visa apenas permitir a conexão do Arduino com os fim-de-curso de ambos os eixos, o *shield Bluetooth* e a placa de leitura, na Figura 39, seu esquemático.

Na Tabela 7 contém uma lista dos componentes utilizados na construção do projeto. Os imãs de neodímio-ferro-boro cilíndricos de diâmetro 8mm e altura 3mm. Os sensores de efeito *Hall* AH1802. Os *multiplexadores* analógicos da família 4051. Os conectores *headers* e os cabos *headers* de 0,1 polegadas. Eixos devem ser acoplados e ortogonais com área efetiva de no mínimo 27cm, dado que cada casa possui 3cm.

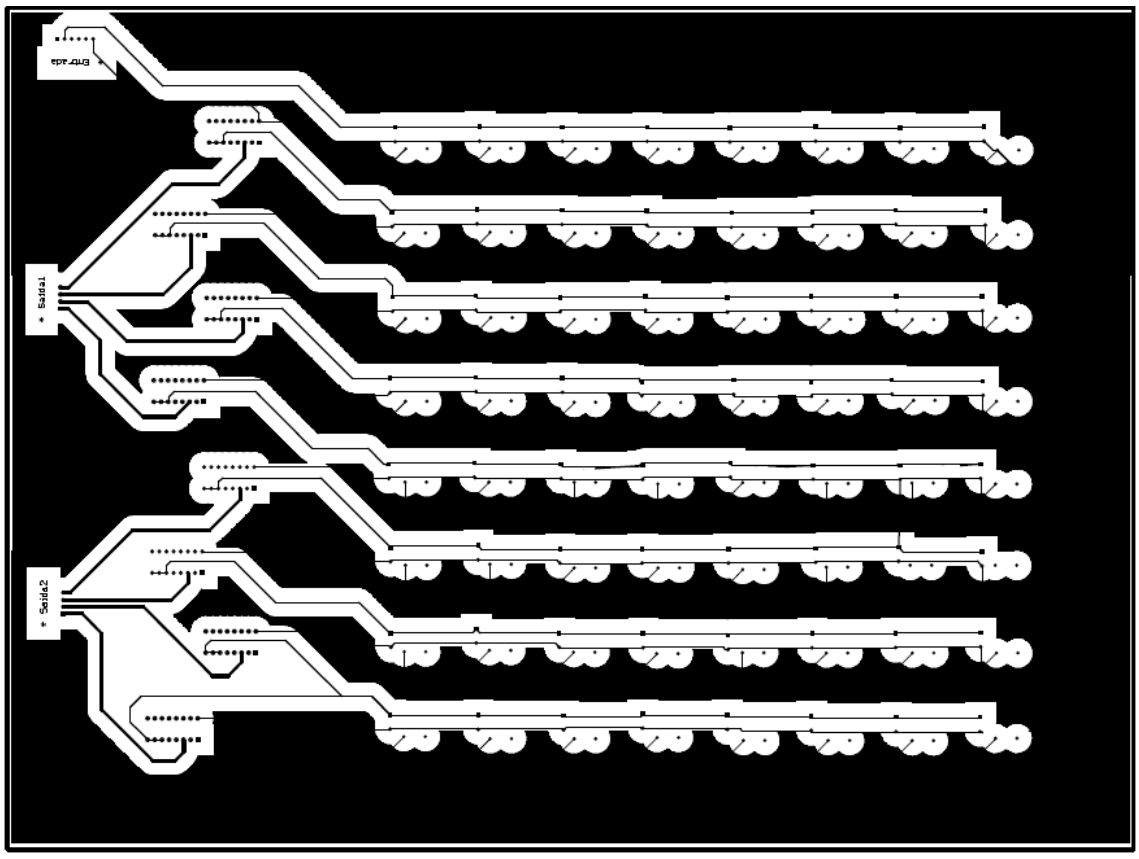


Figura 33: *Layout bottom* da placa de leitura

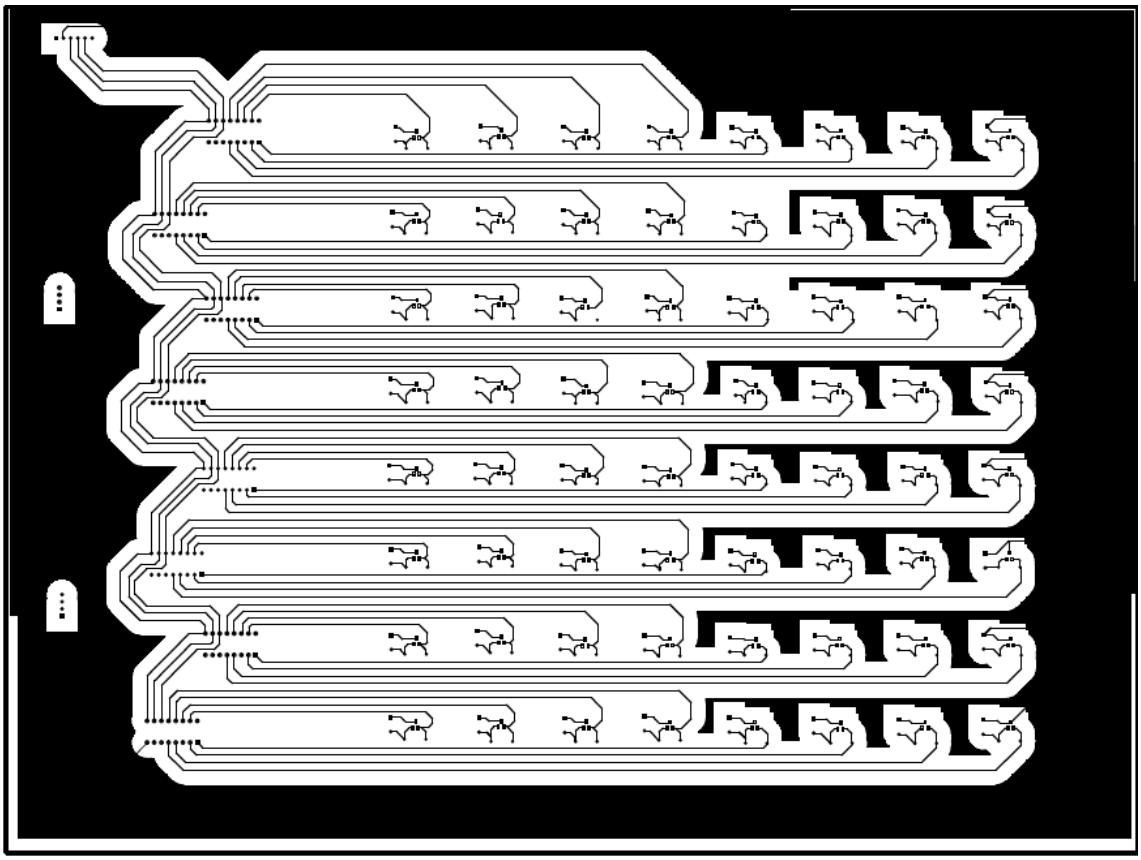


Figura 34: *Layout top* da placa de leitura

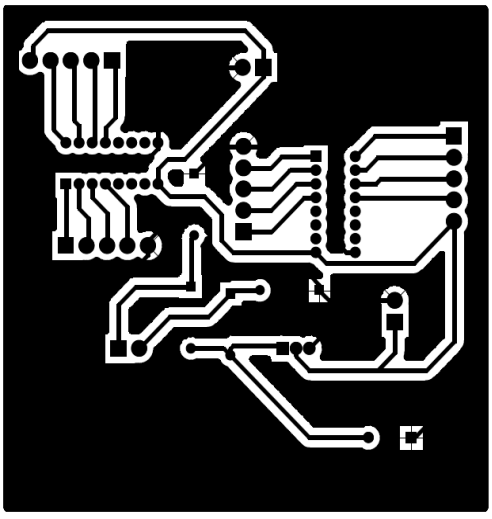


Figura 35: *Layout* da placa de acionamento dos motores

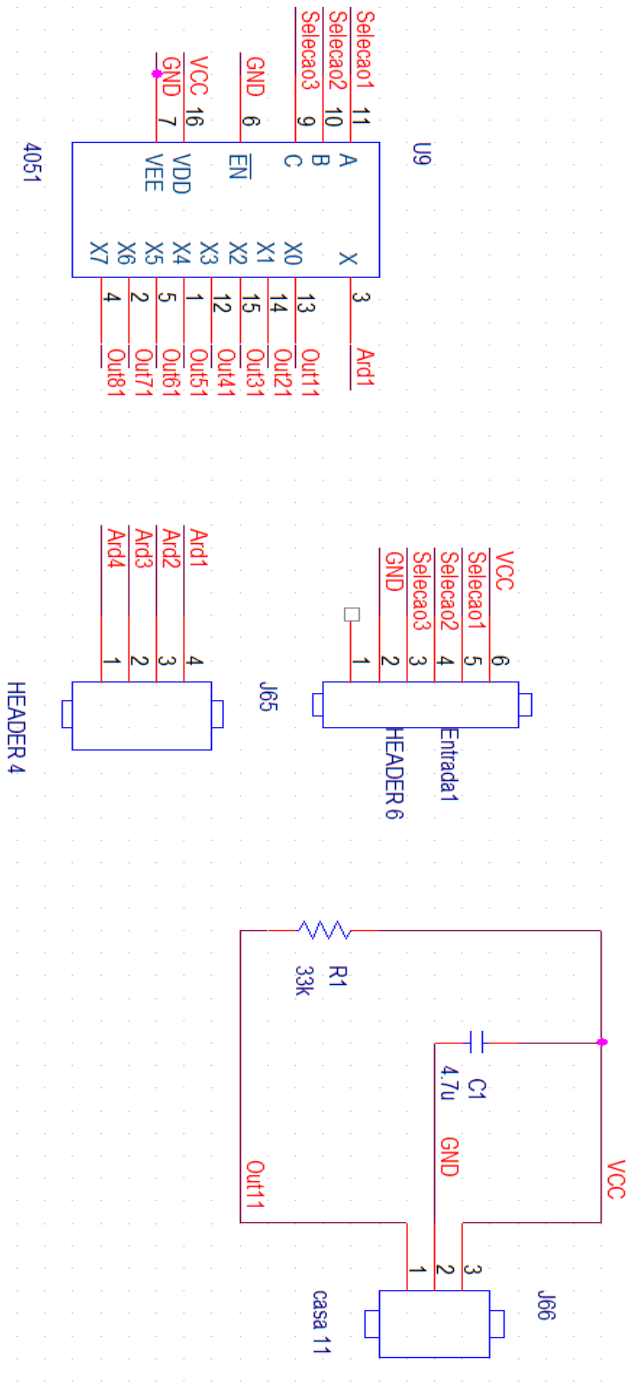


Figura 36: Esquemático de um elemento da placa de leitura

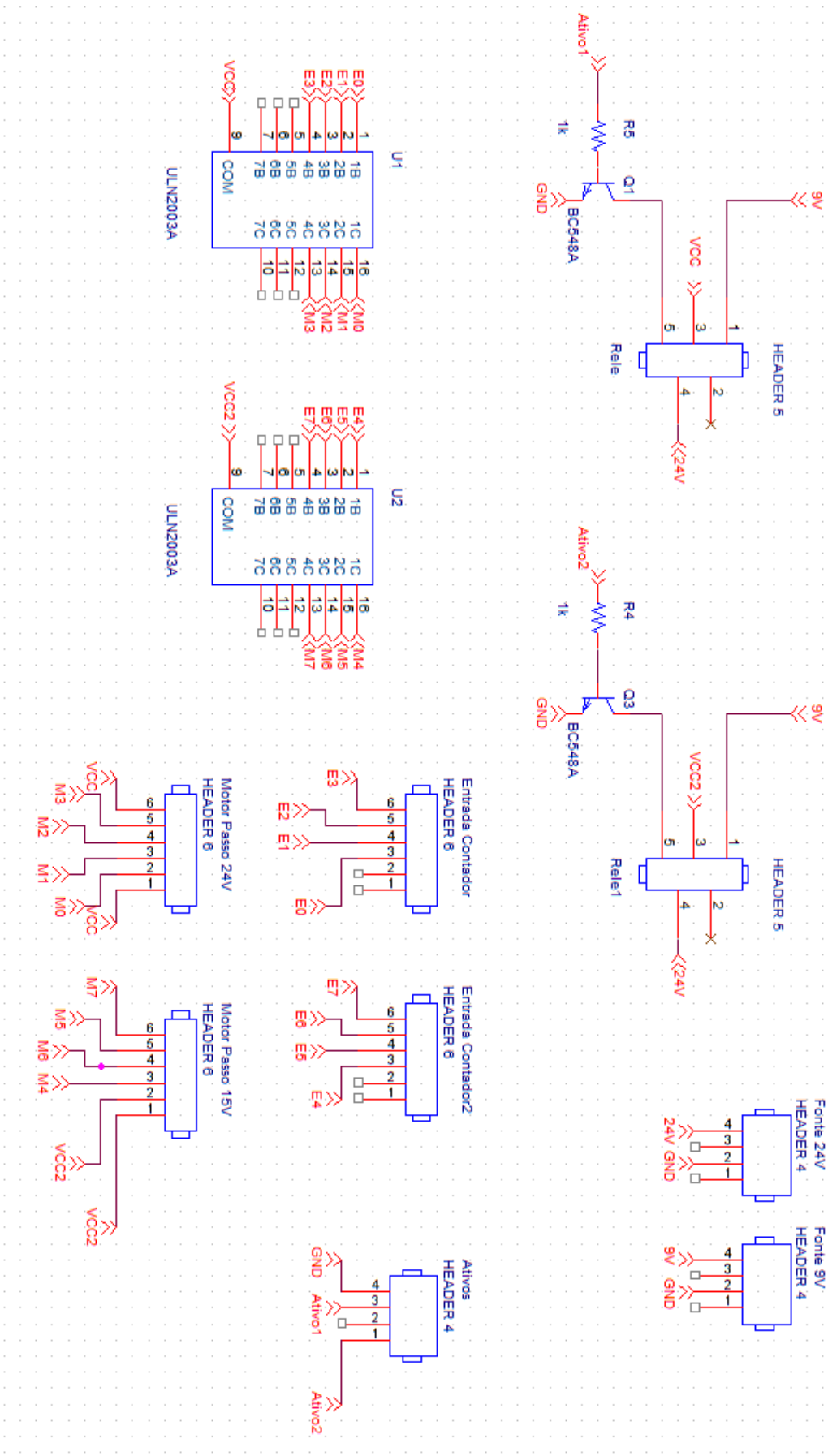


Figura 37: Esquemático da placa de acionamento dos motores

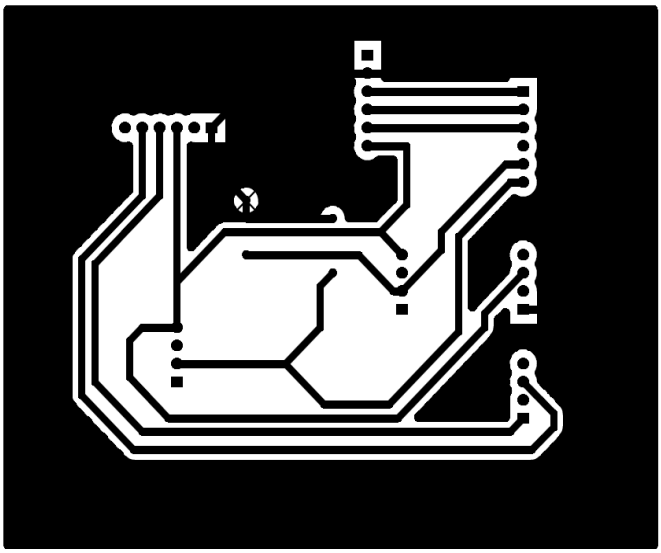


Figura 38: *Layout* da placa auxiliar

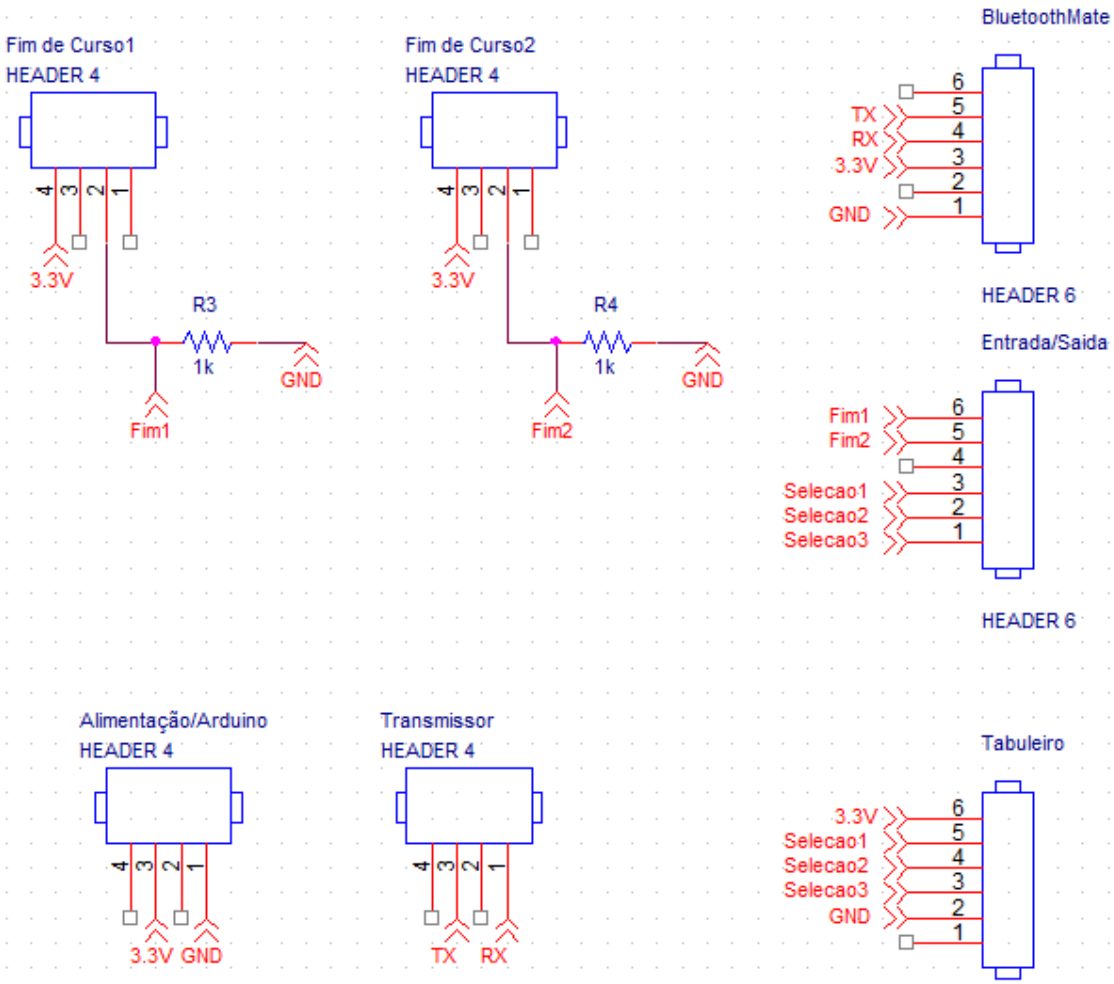


Figura 39: Esquemático da placa auxiliar

Tabela 7: Lista de componentes por utilização

Utilização	Componentes	Quantidade
Peças	Peças de xadrez	32
	Imãs de neodímio-ferro-boro	32
Placa de Leitura	Sensores de Efeito *Hall*	64
	Capacitores 4,7µF	64
	Resistores 33KΩ	64
	Multiplexador Analógico	8
	Conector *Header* Fêmea 4 entradas	2
	Conector *Header* Fêmea 6 entradas	1
Arcabouço Mecânico	Eixos com motor de passo	2
	Servo	1
	Imãs de neodímio-ferro-boro	4
	Acoplador de imãs para o servo	1
	Fim de curso	2
Placa auxiliar	Resistores 1kΩ	2
	Conector *Header* Fêmea 4 entradas	4
	Conector *Header* Fêmea 6 entradas	3
Placa de acionamento dos motores	Multiplexador analógico	2
	Relés 5 pinos	2
	Transistores BC548	2
	Conector *Header* Fêmea 4 entradas	5
	Conector *Header* Fêmea 6 entradas	2
Conexão	Cabos Macho/Macho *Header* 4 entradas	7
	Cabos Macho/Macho *Header* 6 entradas	3
	Cabos P4 Fêmea/Macho *Header* 4 entradas	2
Alimentação	Fonte DC 5V	2
	Fonte DC 9V	1
	Fonte DC 24V	1