**Unioeste - Universidade Estadual do Oeste do Paraná**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
***Curso de Bacharelado em Ciência da Computação***

# Implementando um oponente para o jogo de damas utilizando árvores de decisão

*Andrey William Ribeiro*

**CASCAVEL**
**2019**

**Andrey William Ribeiro**

# Implementando um oponente para o jogo de damas utilizando árvores de decisão

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientadora: Adriana Postal

CASCAVEL
2019

**Andrey William Ribeiro**

# Implementando um oponente para o jogo de damas utilizando árvores de decisão

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Adriana Postal (Orientadora)
Colegiado de Ciência da Computação,
UNIOESTE

Josué Pereira de Castro
Colegiado de Ciência da Computação,
UNIOESTE

Suzan Kelly Borges Piovesan
Colegiado de Ciência da Computação,
UTFPR-Santa Helena

Cascavel, 19 de dezembro de 2019

# EPÍGRAFE

*“The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom.” - Isaac Asimov*

# AGRADECIMENTOS

A minha família pelo apoio e a minha orientadora Adrina por toda a ajuda no desenvolvimento do trabalho.

# Lista de Figuras

# Sumário

# Resumo

Este trabalho tem como objetivo o desenvolvimento de um oponente de damas. Para cada estágio do jogo, é utilizada uma estratégia diferente que visa maximizar o desempenho do jogador. Para o início do jogo utiliza-se a estratégia de se escolher aleatoriamente 3 jogadas, para o meio de jogo é possível alternar entre duas heurísticas, a posicional e a do triângulo defensivo. Já para o final de jogo, quando há 8 peças ou menos, é feita a pesquisa na base de dados Chinook para escolha da melhor jogada, tais jogadas são armazenadas na árvore de decisão, onde é realizada a busca. Foi feita ainda uma avaliação de finais de jogos, fazendo a consulta na base Chinook tentando obter o melhor resultado possível, e comparando esse resultado obtido com o resultado ideal que se poderia obter. Essa avaliação mostrou que o oponente em finais de jogos apresenta um bom desempenho, muito parecido com o resultado ideal da base de dados, e em algumas situações apresenta resultado superior ao esperado.

**Palavras-chave:** Base de dados Chinook, Jogo, Minimax, alfa-beta.

# Capítulo 1

# Introdução

O jogo de damas tem raízes antigas, e acredita-se que a primeira forma do jogo foi descoberta em uma escavação arqueológica em Ur, no Iraque. A datação por carbono estima que o jogo seja de aproximadamente 3000 a.C. porém usava um tabuleiro ligeiramente diferente, um número de peças ligeiramente diferente e ninguém tem certeza das regras exatas.

No antigo Egito havia um jogo muito comum chamado Alquerque, com um tabuleiro 5x5. Historiadores datam que o jogo é de 1800 a.C. Foi um jogo muito popular em todo mundo ocidental por milhares de anos.

Por volta de 1000 e 1500 o jogo de damas era muito popular na França, havia versões diferentes baseadas no jogo. Tal popularidade levou a uma inovação onde os jogadores franceses passaram a jogar damas em tabuleiros de xadrez, o que permitiu que o número de peças fosse expandido, levando a 12 peças para cada lado. Foi chamado de "*Fierge*" ou "*Ferses*". Logo descobriu-se que fazer saltos obrigatórios tornava o jogo mais desafiador. Os franceses chamavam essa versão "*Jeu Force*". A versão mais antiga era considerada um jogo social para mulheres chamado "*Le Jeu Plaisant De Dames*".

As regras para damas foram definidas e o jogo foi exportado para Inglaterra e América. Na Grã-Bretanha foi chamado de *Draughts*. Livros foram escritos na Espanha por volta de 1500 e na Inglaterra um matemático chamado William Payne escreveu seu próprio tratado sobre damas (RAYMENT, 2019).

Ao longo dos anos o jogo manteve sua popularidade. Em 1847 foi realizado o primeiro campeonato mundial. Porém com o passar do tempo percebeu-se que algumas aberturas sempre davam vantagem a um dos lados. Assim a restrição de dois movimentos foi desenvolvida para jogadores experientes, forçando assim jogadas mais ousadas e estilos de jogos mais variados.

Hoje até restrições de três movimentos são usadas em torneios de damas (RAYMENT, 2019).

Na restrição de dois movimentos o primeiro movimento de cada lado é definido por um lote de 47 combinações possíveis. A restrição de 3 movimentos, ou restrição Americana, é uma extensão da restrição de dois movimentos para o segundo movimento das pretas, com cerca de 300 aberturas pré-definidas.

O jogo de damas como esporte teve início no Brasil entre 1935 e 1940, através de Geraldino Izidoro. A partir de 1940 a prática do jogo de forma organizada entrou em recesso, e só veio a ter registros de um movimento nesse sentido em 1954, com o advento do mestre russo W. Bakumenko, quando houve uma popularização do tabuleiro de 64 casas. Existem várias versões do jogo ao redor do mundo, como por exemplo a dama russa, a italiana, a inglesa e a internacional (SARCEDO, 2019).

## 1.1 Inteligência artificial e jogos

A inteligência artificial (IA) é uma das ciências mais recentes e mais proeminentes dos últimos tempos e tem seu início logo após a Segunda Guerra Mundial, tendo seu nome cunhado em 1956. Apesar de recente, suas bases têm ramificações em várias disciplinas que contribuíram e contribuem para seu desenvolvimento, dentre elas estão: Filosofia, Matemática, Economia, Neurociência, Psicologia, Engenharia de computadores, Teoria de controle e cibernética e linguística. As participações de jogos foram uma das primeiras tarefas empreendidas em IA. Ao mesmo tempo que os computadores começavam a ser programáveis, por volta de 1950, o xadrez vinha sendo estudado por Konrad Zuse, Norbert Wiener e por Alan Turing (RUSSEL; NORVIG, 2004).

Em IA os jogos normalmente são de um tipo específico, chamados pelos especialistas de jogos de revezamento de dois jogadores de soma zero com informações perfeitas. Isso significa ambientes determinísticos completamente observáveis, onde existem dois agentes cujas ações são alternadas e que os valores de utilidade no fim do jogo são sempre iguais e opostos (RUSSEL; NORVIG, 2004).

Em 1952 Arthur L. Samuel desenvolveu o primeiro programa de damas para um computador, um IBM 701. Em 1954 ele reescreveu o programa para um IBM 704 com a diferença que o programa passou a ter um mecanismo de aprendizado. O que o fez ter destaque na história da

inteligência artificial é que o programa era capaz de aprender sua própria função de avaliação (KENDALL, 2001).

Para Russel e Norvig (2004) os jogos, como o mundo real, exigem a habilidade de tomar alguma decisão, até mesmo quando o cálculo da decisão ótima é inviável. Assim, técnicas para a busca de uma resposta mais satisfatória são amplamente utilizadas. Nesse sentido, o problema trata de desenvolver um adversário para o jogo de damas que melhore seu desempenho durante as partidas com as entradas do jogador, se adapte a isso, tendo em vista a situação citada por Carvalho, Neto e Lopes (2009), em que o oponente virtual é passível de cometer erros em sua estratégia ou mesmo na avaliação do seu adversário, o que se deve ao foco da aplicação não estar em gerar um jogador virtual *expert*, ou mesmo imbatível.

Schaeffer et al. (2019, p. 01) fortalece essa visão de que o oponente virtual não é imbatível pois afirma que, "embora os programas aplicados aos jogos sejam muito fortes, eles ainda podem perder um jogo para um humano, eles são fortes mas não são perfeitos, a perfeição requer que se resolva um jogo". Isso reforça a afirmação de Chellapilla e Fogel (2001, p. 422), dizendo que "se deve reconhecer, no entanto, que descrever qualquer ação particular como correta ou incorreta pode ser vazia, porque o resultado final é tipicamente uma função não-linear da interação dos movimentos jogados por ambos os jogadores, explicitando assim a dificuldade de uma aplicação no cenário de jogos e suas nuances."

A escolha do jogo de damas como domínio de aplicação deve-se ao fato de que ele apresenta semelhanças com problemas práticos (CAEXÊTA, 2008):

1. Problema de navegação em que os mapas são obtidos de maneira autônoma por um robô móvel;

2. Problema de interação com humanos por meio de um diálogo;

3. Problema do controle de tráfego veicular urbano.

Isso demanda a utilização de técnicas de inteligência artificial. Apesar do jogo ter sido anunciado pela equipe do projeto Chinook (SCHAEFFER et al., 2007), liderado por Jonathan Schaeffer como resolvido (*weakly solved*) e que termina em empate quando jogado perfeitamente pelos dois adversários, para Caexêta (2008), talvez a maior contribuição de

se utilizar técnicas de inteligência artificial no desenvolvimento de jogadores automáticos seja a concretização do seguinte fato: o uso da abordagem de busca intensa com força bruta pode resultar em sistemas com altíssimo nível de desempenho, construídos com mínimo conhecimento dependente do domínio de uma aplicação (SCHAEFFER et al., 2007). Assim, o autor afirma que mesmo uma abordagem de busca intensa com força bruta na base de dados pode levar a sistemas cuja performance em relação aos resultados das partidas é alta, porque encontra sempre uma posição de vitória ou empate (no pior dos casos), enquanto em outras estratégias não há garantia de resultado parecido. O projeto Chinook começou em 1989 como uma tentativa de entender melhor a pesquisa heurística usando um domínio experimental que é mais simples que o xadrez (SCHAEFFER et al., 1996). Assim, será implementada a técnica de árvore de decisão para desenvolver o oponente no jogo de damas.

## 1.2 Trabalhos Correlatos

Dentre os trabalhos que apresentam sistemas de aprendizagem de jogos de damas há o desenvolvido por Neto (2007), que utiliza algoritmos genéticos para proporcionar um conjunto de características mínimas necessárias e essenciais de um jogo de damas para otimizar o treino de um agente que aprende a jogar damas.

O aprendizado desse agente se dá por meio da aproximação de uma rede neural MLP (*Multilayer Perceptron*), uma rede perceptron com a presença de uma ou mais camadas intermediárias de neurônios artificiais, nessas camadas os neurônios são unidades processadoras mas não correspondem à camada de saída, através do método de aprendizagem por reforço, aliado a uma busca minimax, com o mapeamento de tabuleiro NET-FEATUREMAP e a técnica *self-play* com clonagem.

Essa técnica funciona da seguinte forma: o agente é treinado para um determinado número de jogos contra si mesmo e após atingir uma determinada pontuação, que indica uma melhora no seu nível de jogo, é feito uma clonagem da sua função de avaliação (NETO, 2007).

Outro trabalho com objetivo semelhante é o desenvolvido por Caexêta (2008) baseado no trabalho de Neto (2007). O *VisionDraughts* utiliza a técnica de aprendizagem por diferenças temporais para ajustar os pesos de uma rede neural e ele acrescenta dois módulos a mais em

relação a arquitetura desenvolvida por Neto (2007): o primeiro é um módulo de busca eficiente em árvores de jogo baseado em algoritmo alfa-beta, no aprofundamento iterativo e nas tabelas de transposição, que fornece ao jogador uma capacidade de analisar jogadas futuras; e um módulo para acessar bases de dados de finais de jogos que permite obter informações perfeitas de combinações com 8 peças ou menos.

Duarte (2009) desenvolveu um sistema multiagentes onde um deles, conhecido como IIGA (*Initial/Intermediate Game Agent*) é desenvolvido e treinado para ser especializado em fases iniciais e intermediárias do jogo, e outros 25 agentes em fases finais. Cada agente é uma rede neural que aprende a jogar com o mínimo de intervenção humana. Cada um dos 25 agentes especializados em finais de jogo é treinado para lidar com um determinado tipo de *cluster* de tabuleiro de final de jogo. Esses *clusters* são treinados por redes Konohen-SOM. Após treinado, o *MP-Draughts* usa uma versão aprimorada do *VisionDraughts* com IIGA, depois um agente de final de jogo que representa o cluster que mais se aproxima do estado corrente do tabuleiro do jogo substituirá o IIGA e conduzirá o jogo até o final.

Além desses há ainda o trabalho de Barcelos (2011), um agente jogador de damas distribuído, baseado em redes neurais que aprende por reforço, que é treinado em um ambiente de processamento distribuído com o objetivo de alcançar um alto nível de jogo com o mínimo de intervenção humana possível.

O trabalho de Tomaz (2013) é um sistema de aprendizagem de damas, um sistema multiagente jogador de damas que atua em ambiente de alto desempenho. Ele é baseado na integração e refinamento de dois outros trabalhos, o de Barcelos (2011) e o de Duarte (2009).

Todos esses trabalhos foram desenvolvidos no programa de mestrado da Universidade Federal de Uberlândia, com a orientação da professora Profa. Dra. Rita Maria da Silva Julia, e mostram a grande evolução nesse campo.

Um projeto correlato com base parecida é o desenvolvido por Justus (2006), onde o software é feito de forma embarcada, e o computador responde às movimentações das peças no tabuleiro por meio de acionamento de leds. O autor utiliza técnicas como o algoritmo minimax para implementar níveis de dificuldade diferentes para o computador.

## 1.3 Organização do Texto

Essa monografia está organizada da seguinte forma:

**Capítulo 2:** descreve como funciona as regras do jogo de dama e apresenta estratégias utilizadas por jogadores experientes como aberturas de jogos.

**Capítulo 3:** apresenta a técnica de árvore de decisão e algoritmos usados para reduzir seu custo, o minimax e a poda alfa-beta.

**Capítulo 4:** descreve quais ferramentas foram utilizadas para desenvolver o projeto, desde softwares a estratégias implementadas a nível de software.

**Capítulo 5:** descreve como foi feita a implementação e como foram realizados os testes.

**Capítulo 6:** apresenta as considerações finais sobre o trabalho, além de indicar trabalhos futuros.

# Capítulo 2

# O adversário no jogo de damas

## 2.1 Definição

O problema consiste em encontrar a melhor jogada que o computador pode fazer, dada uma disposição do tabuleiro, levando em consideração que o oponente sempre fará as jogadas que o favorecerão.

## 2.2 Regras

Será utilizada a versão brasileira que possui as seguintes regras (SOUSA, 2019):

1. Um tabuleiro 8x8 como no xadrez, com casas claras e escuras.
2. A diagonal mais escura sempre a esquerda dos jogadores.
3. 12 peças claras de um lado e 12 peças escuras de outro lado.
4. O lance inicial cabe às peças claras.
5. A peça só anda para frente, nas diagonais das casas escuras.
6. Ao atingir a oitava linha do tabuleiro, a peça é promovida a dama.
7. A dama pode andar para qualquer direção, quantas casas quiser, mas não pode saltar peças da mesma cor.
8. A captura de uma peça é obrigatória.
9. Peça normal e dama tem o mesmo valor, uma pode capturar a outra.

10. Não é possível capturar duas peças juntas na mesma diagonal.

11. A dama e a peça normal capturam tanto para frente quanto para trás.

12. Se um lance apresenta mais de um modo de captura, é obrigatório executar o lance de maior captura de peças (Lei da maioria).

13. Se uma peça durante a captura de várias peças apenas passa pela casa de coroação e não para, ela não é promovida.

14. Durante o lance de captura de várias peças é permitido passar pela mesma casa vazia mais de uma vez, porém só é permitido capturar a mesma peça uma vez.

15. As peças capturadas não podem ser retiradas do tabuleiro antes da conclusão do lance de captura.

É declarado empate quando, após 20 lances sucessivos de damas, não há captura ou deslocamento de peça normal. O empate também é declarado em finais de jogos, após 5 lances feitos em que não há captura, nas seguintes situações:

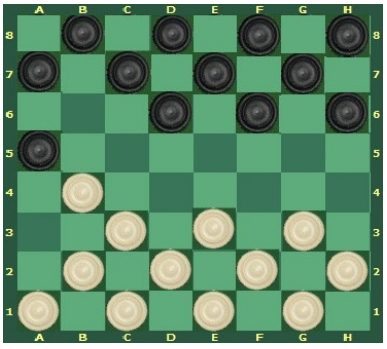
- 2 damas contra 2 damas;
- 2 damas contra uma;
- 2 damas contra uma dama e uma peça normal;
- 1 dama contra 1 dama;
- 1 dama contra 1 dama e uma peça normal.

## 2.3 Estratégias

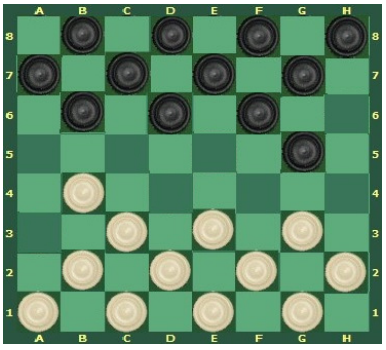
Os jogadores de damas fazem uso de várias estratégias com o objetivo de atingir estados em que suas peças ocupem posições melhores no tabuleiro e consequentemente acabem levando a vitória. Tais estratégias podem ser aplicadas em qualquer estágio do jogo, seja na abertura,

onde o jogador busca assumir uma posição de domínio em relação ao outro, seja no meio jogo, buscando reverter uma posição desfavorável ou encurralar ainda mais o adversário ou no fim de jogo, onde o jogador buscará sacramentar a vitória.

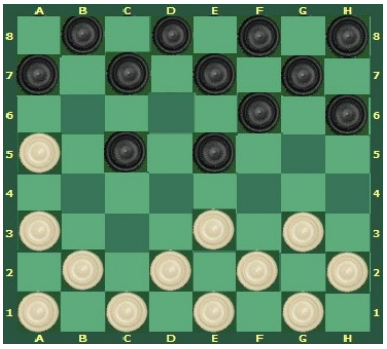
Existem 48 aberturas mais tradicionais usadas no jogo de damas (FILHO, 2015), apresentadas nas figuras 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 e 2.8.



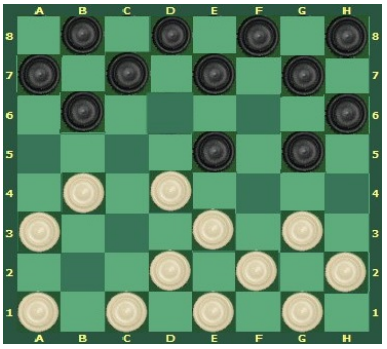
(a) Abertura bodianski



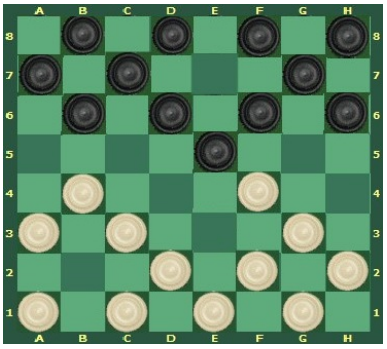
(b) Abertura bodianski recusada



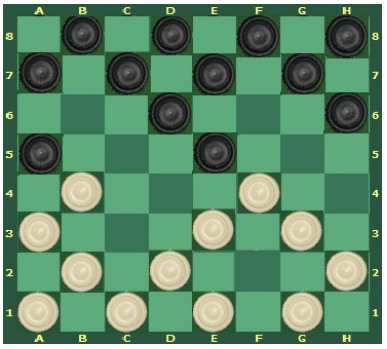
(c) Abertura espanhola



(d) Abertura alma

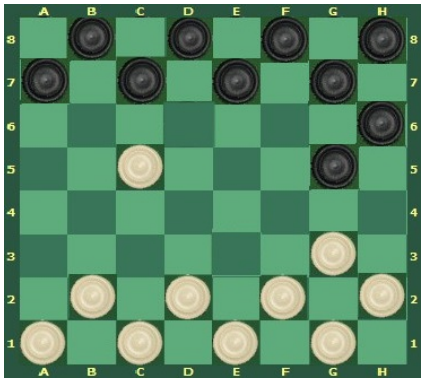


(e) Abertura alma inversa ou ivanow

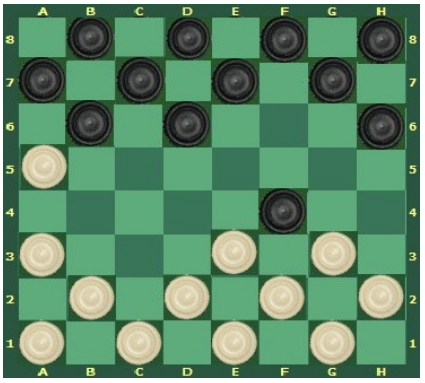


(f) Abertura alma inversa recusada

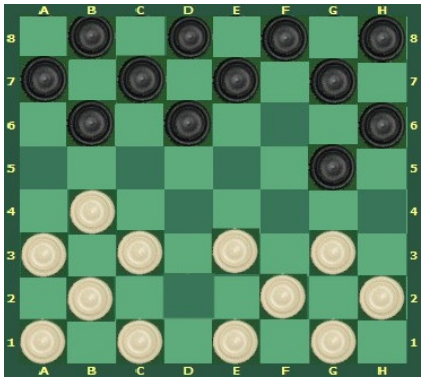
Figura 2.1: Disposição das peças ilustrando as respectivas aberturas



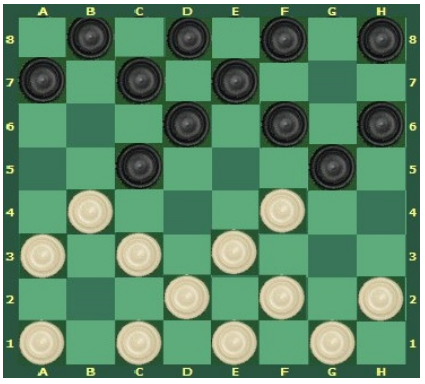
(a) Abertura pioneiro



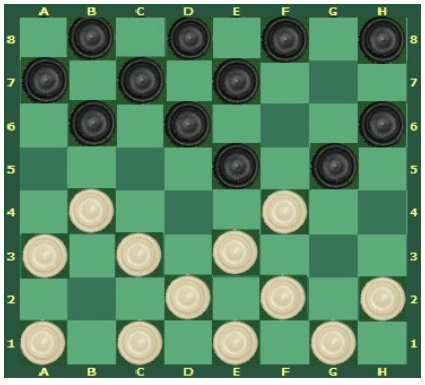
(b) Abertura pioneiro preto



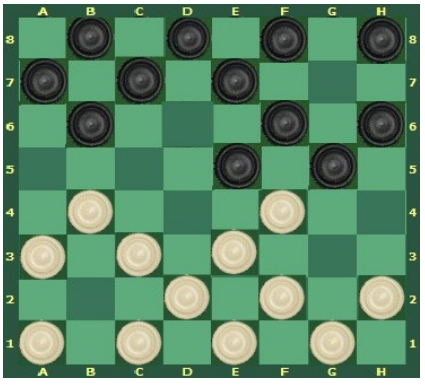
(c) Abertura Blinder



(d) Abertura Napolitana

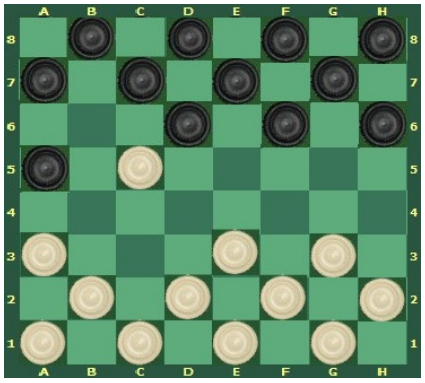


(e) Abertura napolitana inversa

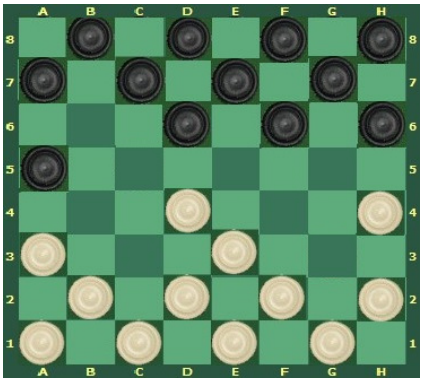


(f) Abertura napolitana recusada

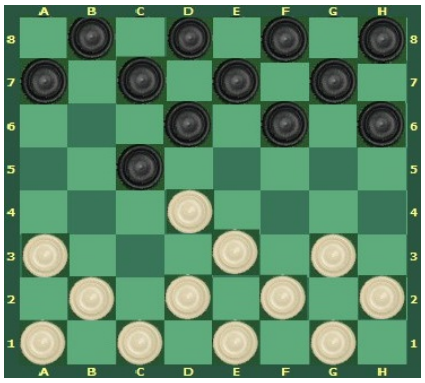
Figura 2.2: Disposição das peças ilustrando as respectivas aberturas



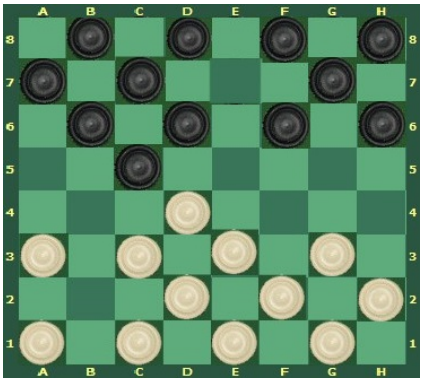
(a) Defesa russa



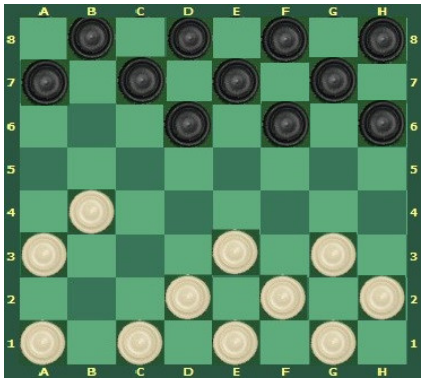
(b) Defesa russa recusada



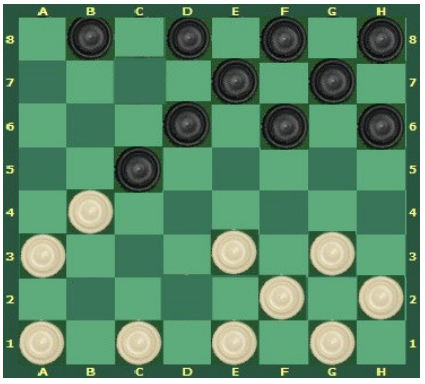
(c) Abertura americana



(d) Abertura antiga ou Naylor

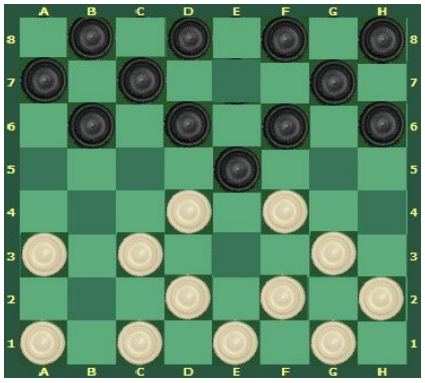


(e) Abertura central

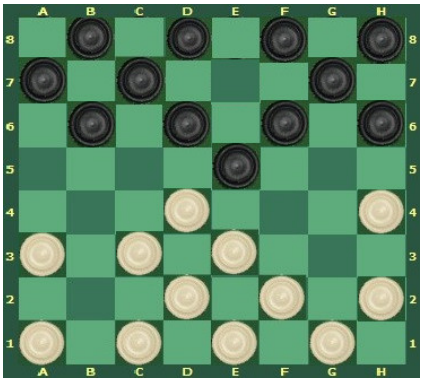


(f) Abertura central para frente

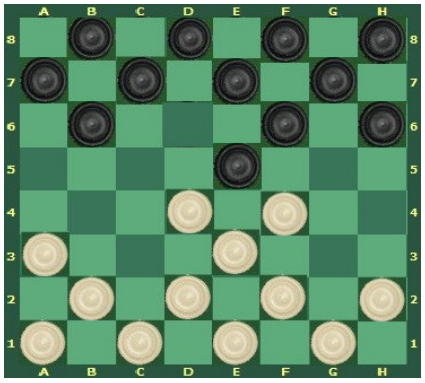
Figura 2.3: Disposição das peças ilustrando as respectivas aberturas



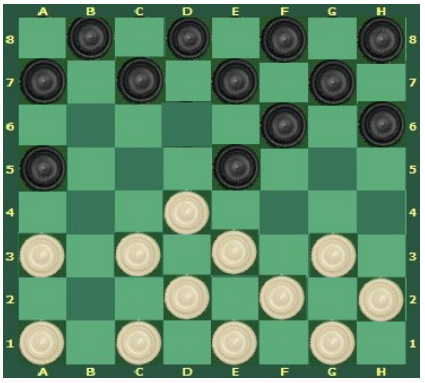
(a) Abertura cruz



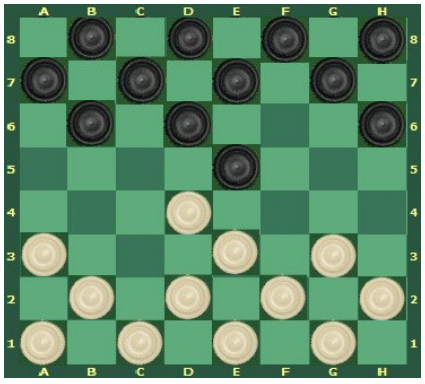
(b) Abertura cruz preta



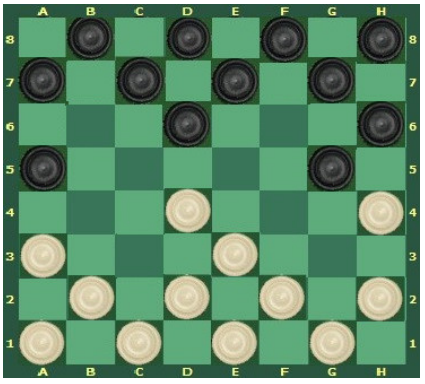
(c) Abertura Garfo



(d) Defesa holandesa

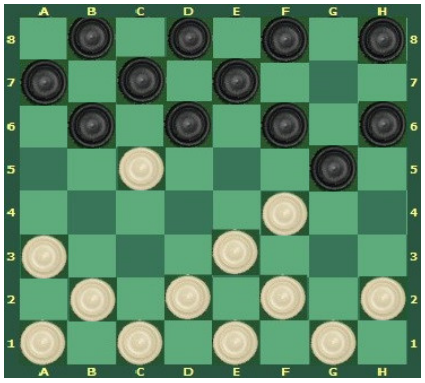


(e) Abertura Kelso

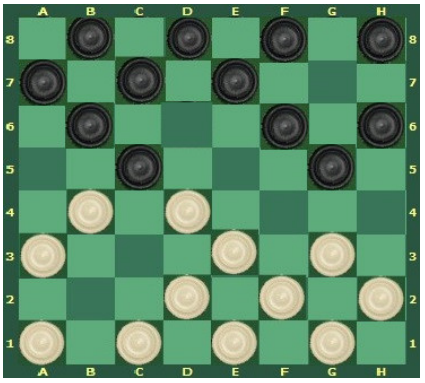


(f) Abertura ataque à pedra g5

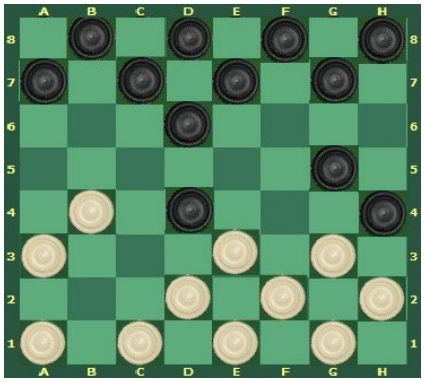
Figura 2.4: Disposição das peças ilustrando as respectivas aberturas



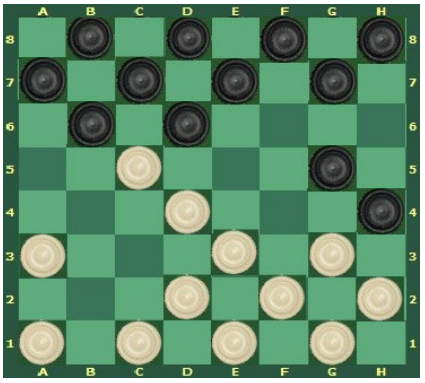
(a) Abertura Belga



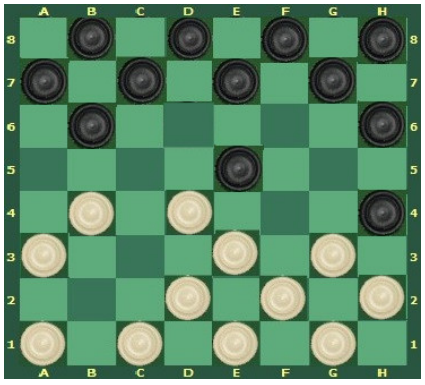
(b) Defesa Diatchkov



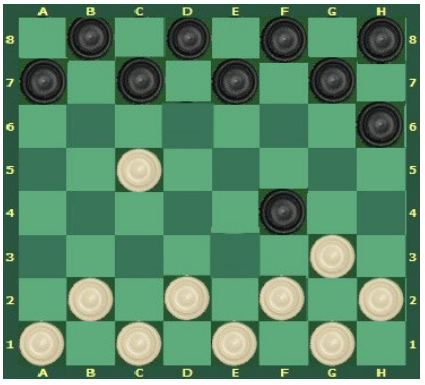
(c) Defesa de Kiew



(d) Defesa nova de kiew

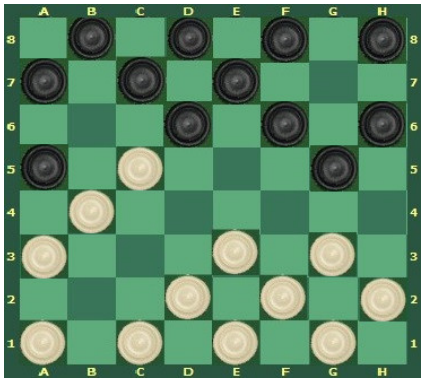


(e) Defesa Kogan

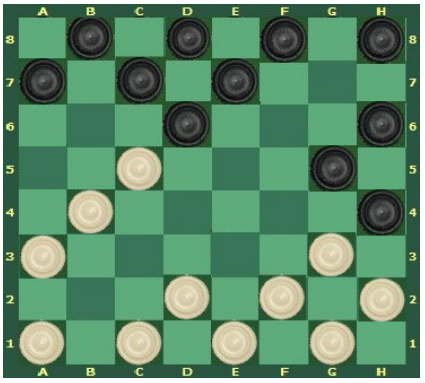


(f) Gambito Kukuiew

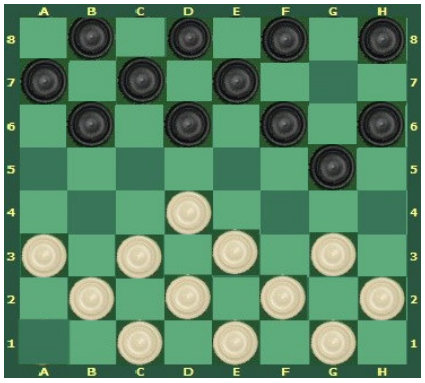
Figura 2.5: Disposição das peças ilustrando as respectivas aberturas



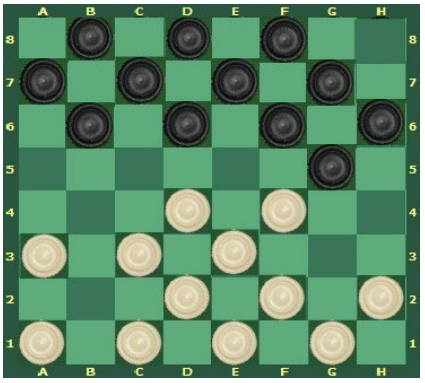
(a) Defesa de Leningrado



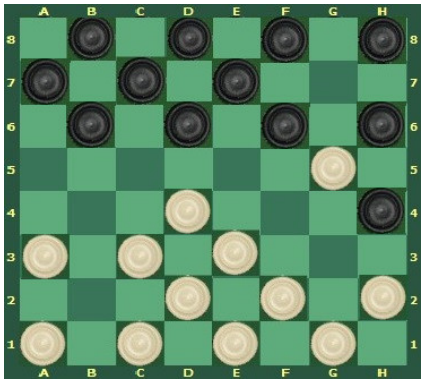
(b) Defesa nova de Leningrado



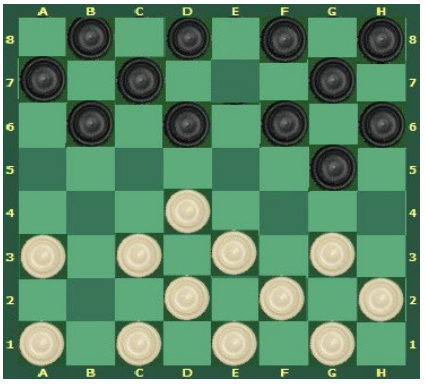
(c) Jogo de Medkov



(d) Abertura Polonesa

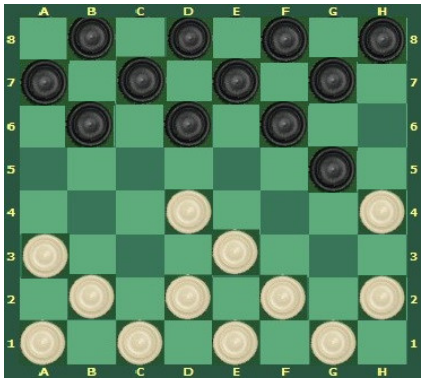


(e) Abertura Ramm

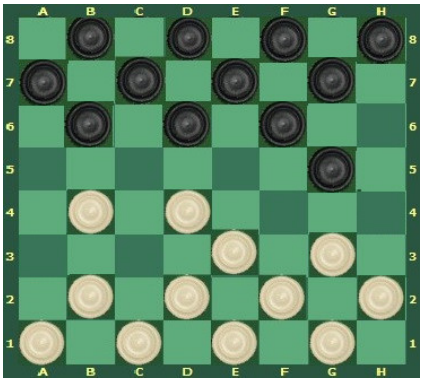


(f) Abertura Sokov

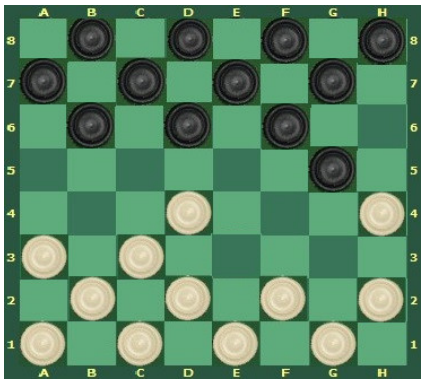
Figura 2.6: Disposição das peças ilustrando as respectivas aberturas



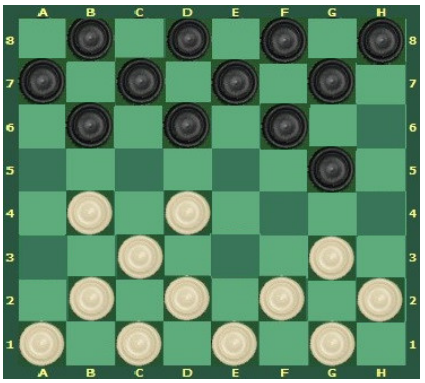
(a) Abertura Asa Preta



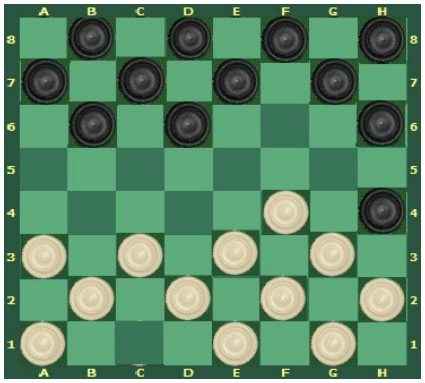
(b) Abertura Asa Preta recusada



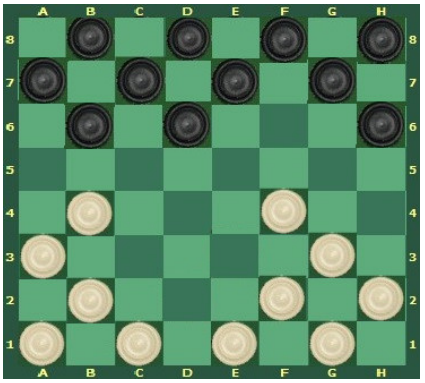
(c) Abertura Filipov



(d) Abertura Filipov recusada

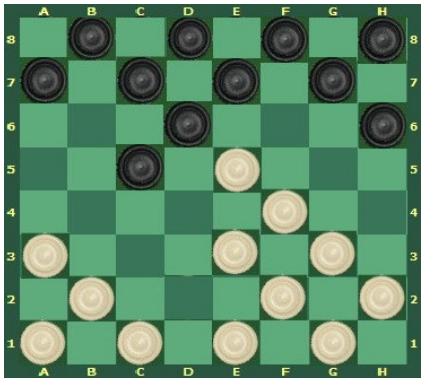


(e) Abertura Bobrov

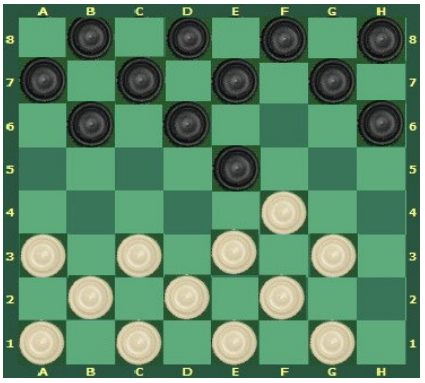


(f) Abertura Schmulian

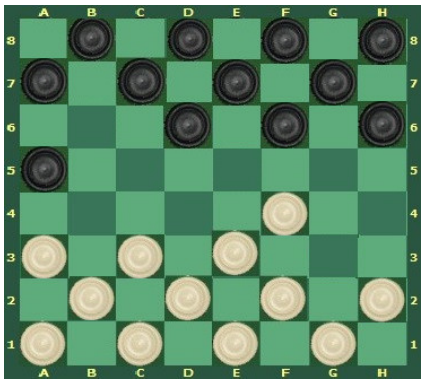
Figura 2.7: Disposição das peças ilustrando as respectivas aberturas



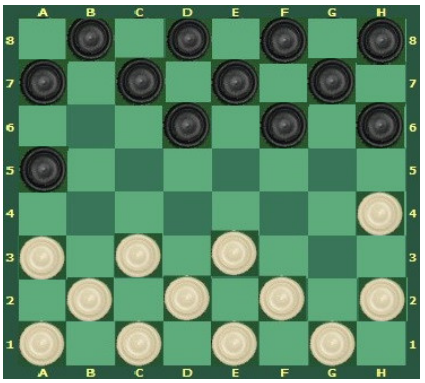
(a) Gambito Lélio Sarcedo



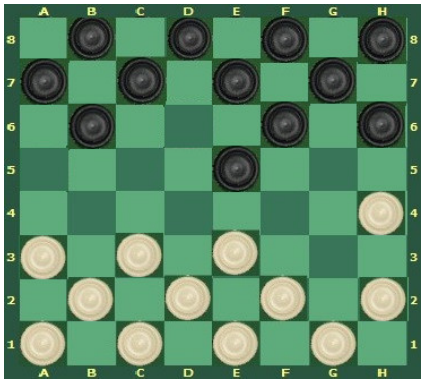
(b) Abertura Kaulen



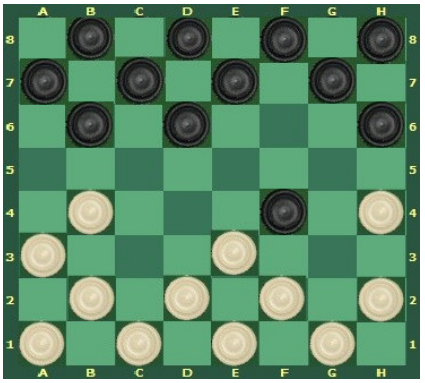
(c) Abertura Kaulen recusada



(d) Abertura Corner Duplo (Petrov)



(e) Abertura Corner (ou Petrov Recusada)



(f) Abertura Australiana

Figura 2.8: Disposição das peças ilustrando as respectivas aberturas

Algumas dessas aberturas podem ser encontradas também no livro de Silva (1987), onde o mestre Genaldo da Silva aborda técnicas para jogadores nos mais variados níveis de entendimento do jogo, e dentre essas técnicas ele apresenta algumas aberturas.

Com exceção da abertura Kelso (2.4(e)) com tomada para e5, que é muito inferior para as pretas, não há abertura melhor ou pior, mas sim aberturas que são teoricamente boas para as brancas e outras que são teoricamente boas para as pretas, nesse caso são:

**boas para as brancas:** 2.1(d) – 2.2(d) – 2.3(a) – 2.3(c) – 2.3(f) – 2.4(e) – 2.6(f) – 2.7(c)

**boas para as pretas:** 2.1(c) – 2.2(c) – 2.4(f) – 2.5(b) – 2.6(a) – 2.6(c) – 2.6(d) – 2.7(a) – 2.7(e) – 2.7(f)

As demais aberturas acabam tendo a mesma possibilidade para qualquer um dos lados, teoricamente (SOUSA, 2012b).

As estratégias do meio de jogo ficam por conta do estudo de posições típicas, porém um agrupamento de posições típicas pode se dar por diversos fatores, mas sabe-se que essas posições estão estritamente ligadas a uma ou outra variante de abertura, logo trabalhar com essas posições é trabalhar com as aberturas (SOUSA, 2012b).

Os jogadores costumam estudar essa parte baseando-se em jogos onde ambos os jogadores ou um deles foi impecável, e procuram analisar os planos de jogo, procedimentos táticos e suas ideias (SOUSA, 2012b).

Também deve-se desenvolver as peças para que haja um domínio do tabuleiro, nesse sentido a aplicação das combinações (golpes) ajudam a obter essa vantagem. Os golpes consistem de uma série de movimentos onde o adversário é forçado a capturar várias peças mas sofre o revés de perder um número de peças maior do que capturou. Existem vários golpes famosos que jogadores profissionais estudam e fazem uso durante as partidas (SOUSA, 2012b).

As finais de jogos são muito importantes e costumam ser muito difíceis pois é o momento onde o jogador tentará converter a vantagem que obteve durante a partida em vitória. Em partidas onde o tempo é mais limitado é comum que ocorram erros que acabem fazendo com que o jogo termine antes de alcançar esse estágio, porém em partidas com um tempo maior há a tendência do jogo se manter equilibrado e ele costuma alcançar esse estágio (SOUSA, 2012b).

Assim, o estudo dessa etapa se faz necessário pois converter a vantagem adquirida durante o jogo em vitória pode ser complicado e não é incomum que partidas que dão uma vantagem a um jogador acabem empatadas ou o mesmo acabe perdendo (SOUSA, 2012b).

Esse estudo é um dos aspectos mais complexos pois cada tipo de final apresenta sua particularidade, assim, os jogadores que buscam se aperfeiçoar nessa etapa costumam despender muito

tempo estudando jogadores que se destacaram como finalistas e também treinam a capacidade de cálculo das jogadas (SOUSA, 2012b).

# Capítulo 3

# Árvores de Decisão

São métodos de aprendizagem supervisionado não-paramétricos, usados em regressão e classificação. É uma estrutura de dados formada por um conjunto de elementos chamados de nós que armazenam informações. Ela toma como entrada um objeto ou situação descritos por um conjunto de atributos e retorna uma decisão - valor de saída previsto, de acordo com a entrada. Os valores de entrada podem ser discretos ou contínuos (RUSSEL; NORVIG, 2004).

A árvore de decisão contém as seguintes características para o problema do jogo de damas (RUSSEL; NORVIG, 2004):

- O estado inicial, que inclui a posição do tabuleiro e identifica o jogador que fará o movimento.
- Uma função sucessor, que retorna uma lista de pares (movimento, estado), cada qual indicando um movimento válido e o estado resultante.
- Um teste de término, que determina quando o jogo termina. Os estados em que o jogo é encerrado são chamados estados terminais.
- Uma função utilidade (também chamada função objetivo ou função compensação), que dá um valor numérico aos estados terminais.

Na dama, assim como no xadrez, o resultado é uma vitória, uma derrota ou um empate, com valores +1, -1 ou 0. Alguns jogos têm uma variedade mais ampla de resultados possíveis: a compensação no gamão varia de 192 até -192.

O estado inicial e os movimentos válidos para cada lado definem a árvore de jogo. Dessa forma, espera-se um oponente no jogo de dama que melhore suas jogadas e consequentemente

sua performance durante as partidas com o jogador, conseguindo assim aumentar seu desempenho conforme o jogador humano se porta durante as partidas.

A figura 3.1 apresenta um exemplo de árvore de decisão que ilustra a tomada de decisão para uma abertura bodianski (Figura 2.1(a)).

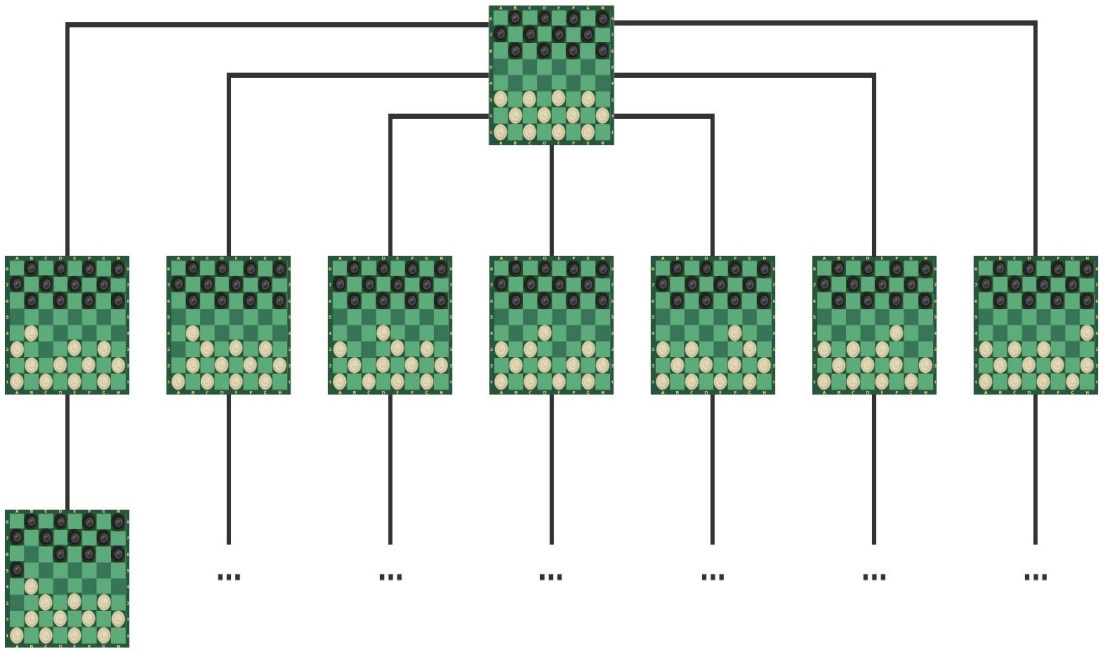


Figura 3.1: Imagem da árvore de decisão ilustrando a abertura bodianski

## 3.1 Algoritmo Minimax

Dada uma árvore de jogo, o algoritmo minimax procura determinar a estratégia ótima para uma jogada com base no valor minimax de um determinado nó. Esse valor se refere a utilidade para o jogador que se encontra no estado corrente, levando em consideração que ambos os jogadores terão um desempenho ótimo desse estado até o fim do jogo. Assim temos que um jogador procurará um estado de valor máximo enquanto o outro procurará um estado de valor mínimo. Dessa forma, temos a seguinte definição (RUSSEL; NORVIG, 2004):

$$\text{VALOR-MINIMAX(n)} = \begin{cases} UTILIDADE(n) & \text{se } n \text{ é um estado terminal} \\ max_{s \in Sucessores(n)} VALOR-MINIMAX(s) & \text{se } n \text{ é um nó de MAX} \\ min_{s \in Sucessores(n)} VALOR-MINIMAX(s) & \text{se } n \text{ é um nó de MIN} \end{cases}$$

O $s$ na definição se refere aos possíveis estados sucessores do estado $n$. O algoritmo calcula

a decisão minimax a partir de um estado corrente, utilizando uma computação recursiva simples dos valores mínimos ou máximos de cada estado sucessor, dependendo se o jogador atual é o MAX ou o MIN (RUSSEL; NORVIG, 2004).

Este algoritmo executa uma exploração completa em profundidade da árvore de jogo. Se a profundidade da árvore é $\mathtt{m}$ e existem $\mathtt{b}$ possibilidades de jogadas válidas em cada ponto, a complexidade de tempo do algoritmo é $O(b^m)$. A complexidade de espaço é $O(bm)$ para um algoritmo que gera todos os sucessores de uma vez ou $O(m)$ para um algoritmo que gera um sucessor de cada vez. Em jogos reais, o custo de tempo é totalmente impraticável, mas esse algoritmo serve como base para a análise matemática de jogos e para algoritmos mais práticos (RUSSEL; NORVIG, 2004).

## 3.2 Poda alfa-beta

O algoritmo minimax apresenta o problema onde o número de estados de jogo que ele precisa examinar é exponencial em relação ao número de movimentos. Não é possível eliminar o expoente, mas podemos reduzi-lo pela metade. Buscaremos a possibilidade de calcular a decisão minimax correta sem examinar todos os nós na árvore de jogo, usando a técnica chamada poda alfa-beta. Quando aplicada a uma árvore minimax padrão, ela retorna o mesmo movimento que minimax retornaria, mas poda as ramificações que não terão influência possível sobre a decisão final (RUSSEL; NORVIG, 2004).

A poda alfa-beta pode ser aplicada à árvores de qualquer profundidade, e frequentemente, é possível podar subárvores inteiras em lugar de podar apenas folhas.

A busca minimax é do tipo em profundidade, então, em qualquer instante só é preciso considerar os nós ao longo de um caminho na árvore (RUSSEL; NORVIG, 2004). A poda alfa-beta obtém seu nome a partir dos dois parâmetros que descrevem limites sobre os valores propagados de volta que aparecem em qualquer lugar ao longo do caminho:

- $\alpha$ é o caminho da melhor escolha, do valor mais alto, que encontramos até o momento, em qualquer ponto de escolha ao longo do caminho para MAX.
- $\beta$ é o valor da melhor escolha, a de mais baixo valor que encontramos até agora em qualquer ponto de escolha ao longo do caminho para MIN.

A busca alfa-beta atualiza os valores de $\alpha$ e $\beta$ à medida que prossegue e poda as ramificações restantes em um nó (isto é, encerra a chamada recursiva) tão logo se sabe que o valor do nó corrente é pior que o valor corrente de $\alpha$ ou $\beta$ para MAX ou MIN, respectivamente (RUSSEL; NORVIG, 2004).

# Capítulo 4

# Materiais e Métodos

## 4.1 Heurística

Uma heurística pode ser definida como qualquer abordagem para a solução de um determinado problema que não garante ser ótima mas leva a uma solução. O uso da informação heurística na resolução de problemas apareceu em um ensaio inicial por Newell, Shaw e Simon (1958), mas a expressão "busca heurística" e o uso de funções heurísticas que estimam a distância até o objetivo vieram um pouco mais tarde (NEWELL; ERNST, 1965); (LIN, 1965); (RUSSEL; NORVIG, 2004).

A busca heurística procura estimar qual o melhor nó da fronteira a ser expandido com base em funções heurísticas. Assim, ela parte do princípio que, dado um objetivo e partindo de um estado inicial, tenta-se atingir a solução para aquele problema, apoiando-se em funções heurísticas já estabelecidas. Exemplos de heurísticas nesse modelo são as aplicadas ao jogo de quebra cabeças de 8 peças (RUSSEL; NORVIG, 2004):

**h1:** número de peças fora de lugar. `h1` é uma heurística admissível porque é claro que qualquer peça que esteja fora de lugar deverá ser movida pelo menos uma vez.

**h2:** soma das distâncias das peças de suas posições-objetivo. Devido às peças não poderem ser movidas ao longo de diagonais, a distância que vai contar é a soma das distâncias horizontal e vertical. Isso, às vezes, é chamado de distância de quarteirão da cidade ou distância de Manhattan. `h2` também é admissível porque todo movimento que pode ser feito move uma peça um passo mais perto do objetivo.

### 4.1.1 Função de avaliação

Uma função de avaliação retorna uma estimativa de utilidade esperada do jogo a partir de uma determinada posição, da mesma forma que as funções de heurística retornam uma estimativa de distância até a meta. Uma função de avaliação não exata levará a direção e posições que serão de derrota. Uma forma para projetar a função de avaliação é (RUSSEL; NORVIG, 2004):

- A função de avaliação deve ordenar os estados terminais assim como a verdadeira função de utilidade.
- A computação não deve demorar tempo demais.
- Em caso de estados não-terminais, a função de avaliação deve estar fortemente relacionada com as chances reais de vitória.

### 4.1.2 Heurística Posicional

A maioria das funções de avaliação calcula contribuições numéricas separadas de cada característica e depois as combina para encontrar o valor total (JUSTUS, 2006).

A heurística posicional calcula a força de cada tabuleiro conforme a disposição das peças. Dada a forma como essas peças se apresentam no tabuleiro, suas posições, é possível verificar se ela está a ponto de ser promovida ou não, dada as regras do jogo. Assim, atribui-se um valor material aproximado para as peças, de forma que (JUSTUS, 2006):

- O peão vale 5, caso esteja a ser promovido vale 7.
- A dama vale 10.

As casas do tabuleiro também têm peso, como mostra a figura 4.1.

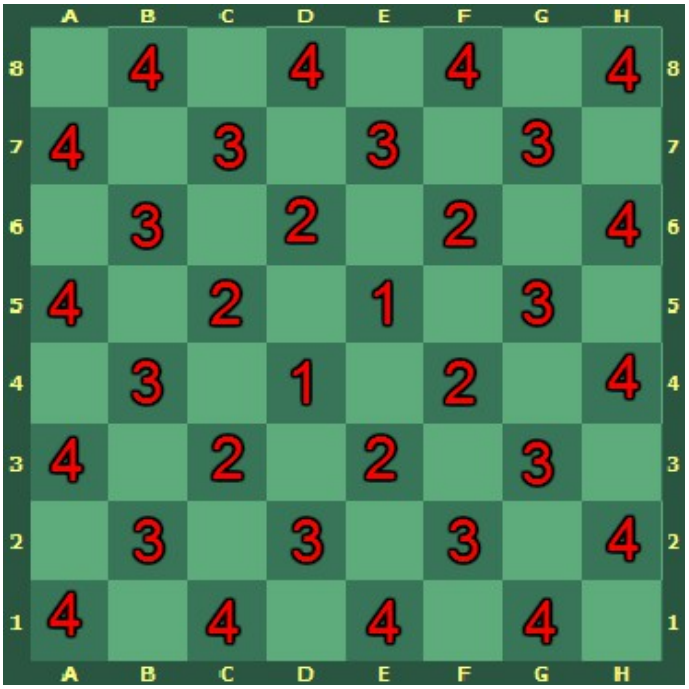



Figura 4.1: Pesos de cada casa na heurística posicional

O cálculo da força de cada jogador é feito por meio do somatório do peso de cada peça (w) multiplicado pelo peso da respectiva casa do tabuleiro (f), e a força do tabuleiro é calculada pela subtração da força das peças brancas (jogador) pela força das peças pretas (computador) (JUSTUS, 2006).

### 4.1.3 Algoritmo Posicional

```
int calcularPontuacaoJogador(int peca, int posicao){
    int valorDaPeca;

    if(peca == BRANCO)//Peca simples
    {
        if(posicao >= 4 && posicao <= 7)
            valorDaPeca = 7;
        else
            valorDaPeca = 5;
    }
    else if(peca == PRETO)//Peca simples
    {
        if(posicao >= 24 && posicao <= 27)
            valorDaPeca = 7;
        else
            valorDaPeca = 5;
    }
```

```
18      else //dama
19          valor = 10;
20
21      return valor*tabelaPesos[pos];
22  }
```

Algoritmo 4.1: Algoritmo de heurística posicional

### 4.1.4 Heurística do triângulo defensivo

A heurística do triângulo defensivo é calculada por meio da fórmula $(p - b)/(p + b)$ sendo `p` a pontuação das pretas e `b` a pontuação das brancas. Cada pontuação é calculada através de caracteres defensivos (casas privilegiadas da defesa) e caracteres materiais (1 ponto para o peão e 3 pontos para a dama), ou seja, além de levar em consideração a quantidade e o tipo da peça (peão ou dama) de cada jogador, a heurística do triângulo defensivo também privilegia o fato da peça estar em sua base ou pertencer ao triângulo defensivo, como mostra a figura 4.2 (JUSTUS, 2006).

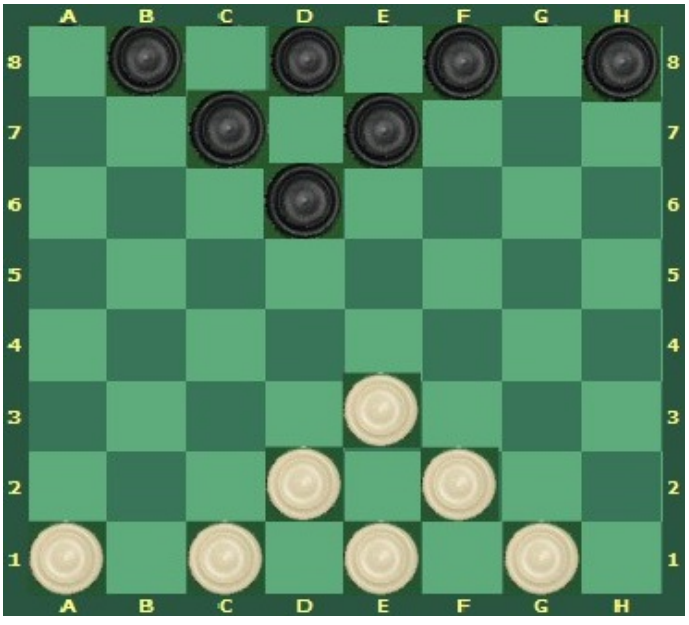



Figura 4.2: Disposição das peças exibindo um triângulo defensivo

### 4.1.5 Algoritmo triângulo

```
float calcularPontuacaoTrianguloDefensivo(Tabuleiro tabuleiro){
    int preta, branca;

    for(int i=0; i<=32; i++){
        if(tabuleiro->obterPeca(i) == VAZIO)
            continuar;

        /*casas de defesa*/
        if((i<4)&&((tabuleiro->obterPeca(i) & ~REI)==cor))
            preta+=1;

        if((i>27)&&((tabuleiro->obterPeca(i) & ~REI)==inimigo))
            branca+=1;

        if((i==5 || i==6 || i==9 )&&((tabuleiro->obterPeca(i) & ~REI)
           ==cor))
            preta+=1;

        if((i==25 || i==26 || i==22)&&((tabuleiro->obterPeca(i) & ~
           REI)==inimigo))
            branca+=1;

        /* pecas */
        if(tabuleiro->obterPeca(i) == cor)
            preta+=1;

        if(tabuleiro->obterPeca(i) == corDama)
            preta+=3;

        if(tabuleiro->obterPeca(i) == inimigo)
            branca+=1;

        if(tabuleiro->obterPeca(i) == damaInimiga)
            branca+=3;
    }
    return (float)(p-b)/(p+b);
}
```

Algoritmo 4.2: Algoritmo da heurística do triângulo defensivo

## 4.2 Softwares utilizados

A implementação do programa foi feita em JAVA (ORACLE, 2019), com o paradigma orientado a objetos, utilizando a IDE Netbeans (NETBEANS, 2019). Utilizou-se também a Java

Native Interface, interface Java que faz integração com o código em C, desenvolvido para utilizar as bibliotecas que fazem acesso a base de dados.

O software foi desenvolvido no sistema operacional Windows, e utiliza bibliotecas no formato `.dll` que só funcionam nessa plataforma.

### 4.2.1 Base de Dados

É utilizada a base de dados do projeto Chinook (University of Alberta, 2019) para finais de jogo, como parâmetros para testes. Ela contém informações perfeitas para todas as posições de damas com 8 peças ou menos, o que leva a um total de 443.748.401.247 posições (University of Alberta, 2019). A base de dados é disponibilizada para uso no site do projeto Chinook da Universidade de Alberta no Canadá.

Ela armazena para cada estado do tabuleiro um valor associado indicando se ele apresenta derrota, empate ou vitória, assim, é chamada de WLD (*win, loss or draw*). Essa base de dados não retorna resultados válidos quando há um movimento de captura, exigindo tratamento do oponente para essa situação.

Dada a propriedade de oferecer informações perfeitas para 8 peças ou menos, ela é um excelente parâmetro para mensurar o desempenho do oponente nessas situações. O oponente fará os cálculos tentando estimar as melhores jogadas nessa situação, e terá como comparação a melhor resposta possível, fornecida pela base Chinook.

O projeto fornece um código de consulta das bases desenvolvido em C e vários arquivos com bases diferentes, com disposições e número de peças diferentes. O código fornecido recebe um estado do tabuleiro e retorna qual resultado esse estado levaria, se é uma vitória, empate ou derrota. Esse programa foi utilizado para comparar o resultado que o oponente atinge acessando a base de dados. Cada base vem com 2 arquivos, um binário e outro `.idx`; o segundo contém os números das bases a serem escolhidas em cada linha. Ambos os arquivos são lidos pelo programa e usados para encontrar a melhor jogada.

O arquivo `.idx` contém toda a sub-database, que consiste de todo o cabeçalho da base de dados. O formato de uma entrada é dado pela descrição:

```
BASEabcd.ef simb
S       k       x/y
E       j       m/n
```

Onde:

**a** indica o número de damas pretas;

**b** indica o número de damas brancas;

**c** indica o número de peões pretos;

**d** indica o número de peões brancos;

**e** indica a posição da peça preta;

**f** indica a posição da peça branca;

**simb** indica quais os resultados possíveis para essa configuração de final de jogo, fornecidos pela base de dados. Pode assumir um dos seguintes códigos:

**=** indica que essa configuração retorna principalmente empates.

**+** indica que há possíveis vitórias para qualquer um dos lados.

**==** indica que a configuração atual só gera empates.

**++** significa que toda base de dados correspondente àquele cabeçalho gera vitória para a configuração atual.

**- -** significa que toda base de dados correspondente a aquele cabeçalho gera derrota para a configuração atual.

**S** significa *Start* e refere-se a posição inicial no arquivo binário;

**k** mostra o valor da posição inicial da jogada no arquivo binário;

**x** indica o bloco no arquivo binário onde começam as jogadas;

**y** indica o valor do byte no arquivo binário onde começam as jogadas;

**E** significa *End* e refere-se a posição final no arquivo binário;

**j** mostra o valor da posição final da jogada no arquivo binário;

**m** indica o bloco no arquivo binário onde terminam as jogadas;

**n** indica o valor do byte no arquivo binário onde terminam as jogadas.

Segue um exemplo de entrada da base de dados ilustrando as informações acima:

```
BASE1100.00 =
S       0        0/0
E     995        0/189
```

A BASE1100.00 indica que há uma dama preta, uma dama branca, nenhum peão preto, nenhum peão branco, gerando principalmente empates.

`S` (*Start*), e se refere a posição inicial, nesse caso a posição 0 e 0/0 se refere ao bloco 0, byte 0, no arquivo binário.

`E` (*End*), na posição 995 e 0/189 se refere ao bloco 0, byte 189 no arquivo binário, o que mostra que a base de dados de 995 posições foi comprimida em 189 bytes.

# Capítulo 5

# Implementação e testes

O jogo implementado é um jogo completo, podendo transicionar entre todos os estágios, início, meio e fim de jogo, e dependendo da habilidade do jogador, pode ser encerrado antes do estágio final.

O programa é dividido em várias classes, cada uma gerenciando um aspecto diferente do jogo.

A classe `Damas` é onde se define os detalhes da interface como cor de fundo, que é verde, os valores para a janela, nesse caso suas dimensões são 350 pixels de largura e 250 pixels de altura.

Declara-se um tabuleiro com os valores apropriados para os botões de 'novo jogo', 'desistência' e 'mensagem', que exibirá mensagens na tela para o usuário.

Na figura 5.1 pode-se visualizar a disposição das peças no início do jogo junto com os botões de desistência e novo jogo.

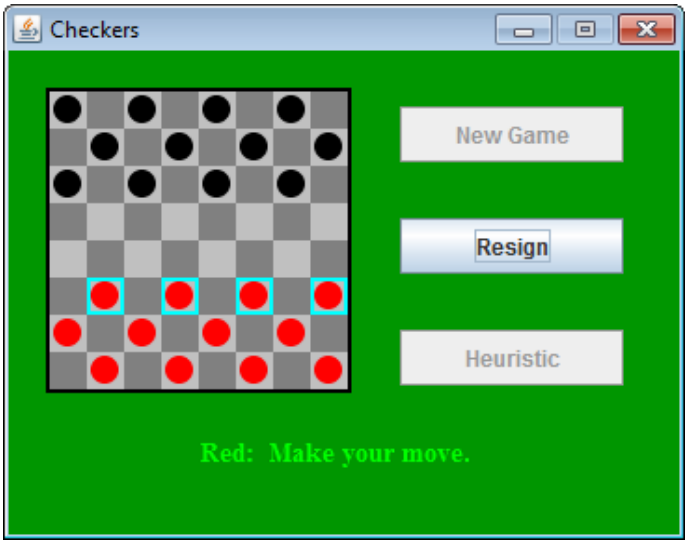



Figura 5.1: Imagem da tela inicial do programa

Há ainda o botão de 'heurística' onde o jogador poderá escolher se prefere jogar contra um oponente que utiliza a heurística posicional ou a heurística do triângulo defensivo. A figura 5.2 mostra a janela com a possibilidade de escolhas.

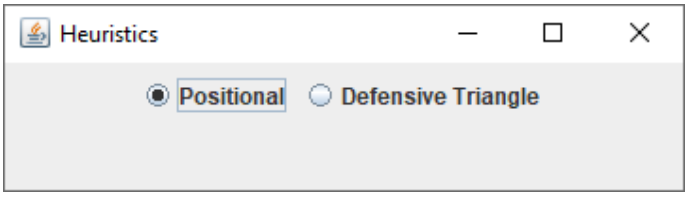



Figura 5.2: Imagem da tela de seleção da heurística

A classe `Tabuleiro` recebe como parâmetro os valores dos botões que aparecerão para o usuário; ela também é responsável por gerenciar as ações do jogo, exibir mensagens na tela para o usuário.

Na classe `DamasDados` temos as informações referentes ao jogo, a cor de peça dos jogadores, as posições onde as peças estão no decorrer do jogo, e quando um movimento é feito na classe `Tabuleiro` pelo usuário, ela invoca o método da classe `DamasDados` que fará essa mudança da disposição das peças, o decremento do número de peças caso haja captura.

A classe `Heuristicas` contém a implementação da heurística posicional e a heurística do triângulo defensivo, utilizadas como estratégia para o meio do jogo. Ela contém o método chamado `HeuristicaPosicional` que recebe dois parâmetros, a peça e a posição

dela no tabuleiro, assim, é feita a verificação se a peça é branca ou preta e com base na sua posição estabelecemos qual o valor que será associado a mesma, caso esteja próxima a ser promovida ela recebe o valor 7, senão recebe o valor 5. Se a peça já for uma Dama, ela recebe o valor 10. Há também a implementação da heurística defensiva, em um método chamado `HeuristicaDefensiva` que recebe como parâmetro o tabuleiro, onde serão percorridas todas as casas calculando a força de cada jogador. O oponente irá escolher a jogada que maximizará sua força e minimizará suas fraquezas.

Na classe `ArvoreDecisao` há a implementação da estrutura da árvore de decisão onde cada nó armazena os estados possíveis do jogo, porém, graças ao uso da técnica de corte alfa-beta, é reduzido consideravelmente o número de estados possíveis na memória. Essa redução também ocorre devido ao uso das técnicas de aberturas e heurísticas utilizadas no início e meio de jogo, que fazem com que o armazenamento desses estados seja desnecessário.

A imagem A.1, apresentada no apêndice A, ilustra a interação entre as classes do programa.

As entradas no software correspondem aos movimentos realizados pelo jogador, esses movimentos vão interferir na resposta do oponente, que terá uma nova disposição do tabuleiro e irá efetuar novos cálculos posicionais, buscando maximizar a eficiência da sua jogada, que é a resposta do software. Essa dinâmica se repete até o fim da partida. A figura 5.3 mostra esse comportamento durante toda a partida.

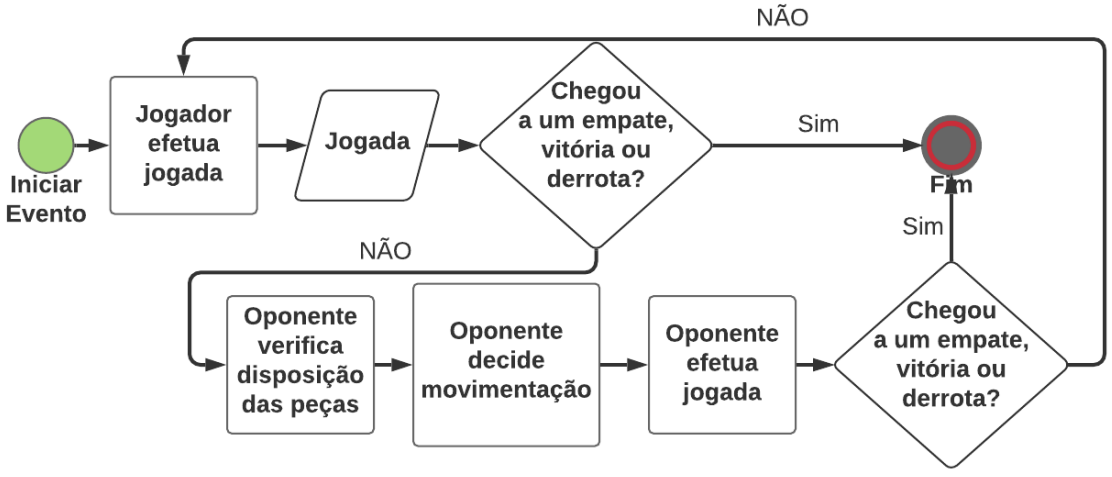



Figura 5.3: Fluxograma ilustrando a dinâmica de entrada e saída do software

## 5.1 Biblioteca para acesso da base de dados

Na implementação foram utilizadas as bibliotecas desenvolvidas por Gilbert (2004) e utilizadas no Kingsrow, em formato `.dll`, que fornecem acesso às bases de dados Chinook. O Kingsrow é um jogador de damas desenvolvido por Gilbert (2004) e que devido as bibliotecas desenvolvidas e disponibilizadas pelo autor, pode ter acesso a base de dados com formatos de BitBoards diferentes, como a base Chinook, a própria base Kingsrow desenvolvida por ele, dentre outras. As figuras 5.4 e 5.5 ilustram a diferença entre as representações do tabuleiro em BitBoard.

```
/*
 * Table to map between DB board representation and Chinook's
 * Note: the colours are on opposite sides
 *
 *       Database Board:         Chinook's Board:
 *             WHITE                   BLACK
 *         28  29  30  31          7  15  23  31
 *       24  25  26  27          3  11  19  27
 *         20  21  22  23          6  14  22  30
 *       16  17  18  19          2  10  18  26
 *         12  13  14  15          5  13  21  29
 *        8   9  10  11          1   9  17  25
 *          4   5   6   7          4  12  20  28
 *        0   1   2   3          0   8  16  24
 *            BLACK                   WHITE
 */
/* Note that if your board representation is already the left one, you  */
/* do not need the code to convert it to the right one.                 */
```

Figura 5.4: Esquerda: Representação do tabuleiro para uso com as bases de dados das finais de jogo. Direita: Representação antes das finais

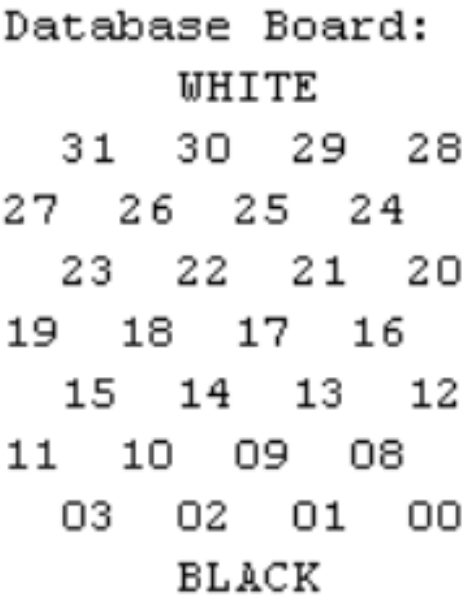



Figura 5.5: Representação do tabuleiro de entrada para a base Kingsrow

Essas bibliotecas foram desenvolvidas em 2004 com o intuito de auxiliar programadores a terem acesso as bases de dados de finais de jogos de uma forma mais fácil e foram utilizadas por Caexêta (2008) para consultas de finais de jogo do *Vision Draughts*.

Essas bibliotecas apresentam algumas funções para acessos na base de dados:

$egdb\_identify()$ é responsável por encontrar a base de dados no caminho especificado. Possui os parâmetros:

- $DB\_PATH$: string com o caminho absoluto para o diretório onde estão as bases de dados;
- $egdb\_type$: informa qual a base de dados encontrada ao fazer a verificação, dentre as bases que podem ser acessadas, seja a base Chinook, a base Cake ou a base Kingsrow. Outras bases de finais de jogos que podem ser consultadas;
- $max\_pieces$: parâmetro que indica o número máximo de peças encontradas na base de dados.

$egdb\_open()$ realiza a abertura da base de dados. Possui os parâmetros:

- $EGDB\_NORMAL$: indica o tipo de representação do tabuleiro que será utilizada para consultas, pois as bibliotecas fornecidas por Gilbert (2004) aceitam diferentes representações para BitBoards, como mostram as figuras 5.4 e 5.5, assim o argumento $EGDB\_NORMAL$ indica que o tipo de tabuleiro é o mesmo utilizado pelo Chinook;
- $max\_pieces$: indica o número máximo de peças permitidas no tabuleiro para que a base seja consultada. Como utiliza-se a base Chinook, o valor do número máximo de peças é 8. Assim, se o número de peças for menor que 8, menos memória RAM é necessária e a abertura da base de dados é mais rápida.
- $quantidadeMemoria$: representa a quantidade de memória RAM em MB que será utilizada para as bases de dados. No exemplo fornecido por Gilbert (2004) é utilizado $2000$ MB, em Caexêta (2008) a quantidade utilizada é 1 GB de RAM;
- $DB\_PATH$: representa a localização das bases de dados;
- $print\_msgs$: é a mensagem que será retornada durante o procedimento.

$lookup()$ realiza as buscas de um determinado estado no tabuleiro, retorna se o estado leva a vitória, empate ou derrota. Possui os parâmetros:

- $handle$: base de dados aberta pela função $egdb\_open()$;
- $pos$: estado do tabuleiro que se quer consultar;
- $EGDB\_BLACK$: cor do jogador que fará o próximo movimento;
- $somenteMemoria$: indica se a consulta acontecerá somente em memória ou também em disco.

## 5.2 Representação do tabuleiro em BitBoards

A representação do tabuleiro no programa é uma matriz de valores inteiros de dimensões 8x8, porém, para se trabalhar com a base de dados Chinook é preciso realizar a conversão do tabuleiro atual para a representação em BitBoards.

Como as peças do jogo de damas se resumem a peões brancos, peões pretos e damas, é feita uma conversão da disposição atual do tabuleiro armazenado em forma de matriz para uma

representação utilizando três inteiros. Como cada inteiro possui 32 bits de tamanho, cada bit do inteiro indica uma posição no tabuleiro, se a casa possui uma peça de determinada cor em uma posição, então aquela posição correspondente no bit do inteiro, recebe 1, senão recebe 0. Assim, se uma casa do tabuleiro possui um peão branco, o inteiro responsável por armazenar a posição dos brancos recebe o valor 1 no seu bit correspondente, o mesmo ocorre para o inteiro que armazena as peças pretas e o que armazena as damas.

Por exemplo, dada a disposição das peças no tabuleiro mostrada na figura 5.6.

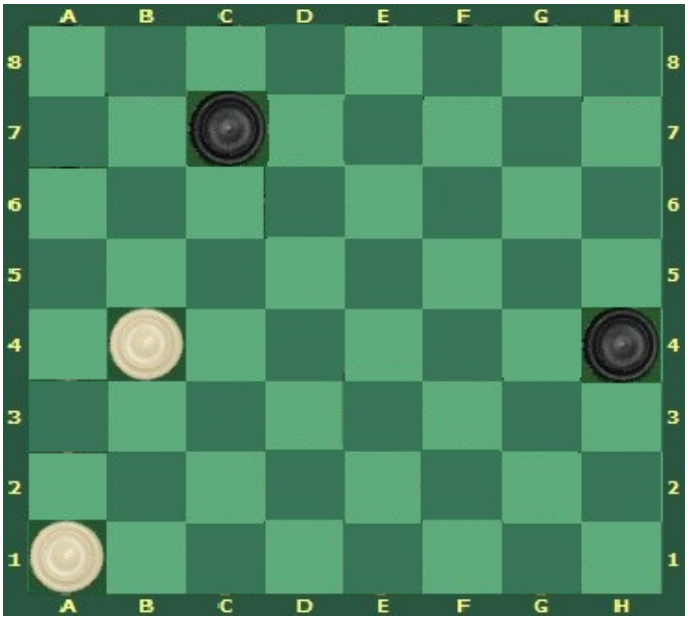



Figura 5.6: Disposição das peças para a primeira final

A representação dos valores em binário a seguir estão no formato `little endian`, cuja leitura dos bits é feita do bit mais a direita para o bit mais a esquerda.

O inteiro responsável por armazenar as peças pretas, receberá o valor 1 no bit 15 e no bit 25, sua representação em binário será: 00000001000000000100000000000000.

Já o inteiro que representará as peças pretas receberá o valor 1 no bit 0 e no bit 12, sendo representado em binário da seguinte forma: 00000000000000000001000000000001.

Como não há damas no tabuleiro, o inteiro que armazena as posições das damas receberá apenas o valor 0.

## 5.3 Testes

Os testes foram realizados em um computador com a seguinte configuração:

- Sistema operacional Windows 10, 64 bits
- Processador Intel Core i3-7100U 2.40GHz
- Memória RAM 4GB

O objetivo dos testes foi avaliar o desempenho do oponente durante os estágios finais das partidas, realizando-se jogadas contra o ele, procurando utilizar a propriedade da base de dados que fornece informações perfeitas, para obter um desempenho satisfatório dele nesses estágios finais.

A busca na base de dados, que pode ser vista no fluxograma da figura 5.7, ocorre da seguinte maneira:

- É obtido o estado atual do tabuleiro em formato BitBoard;
- Obtém-se as jogadas possíveis de serem realizadas pelo jogador;
- Faz-se a consulta na base de dados para avaliar qual resultado aquela disposição possível de peças pode levar, se a um empate, uma derrota ou uma vitória. Esse valor é associado a cada estado da árvore de jogo que foi consultado na base de dados;
- Após isso, utiliza-se a árvore de decisão para buscar a melhor jogada naquele momento.

Toda vez que é preciso realizar uma jogada, há essa consulta na base de dados para avaliar qual o melhor movimento a ser realizado.

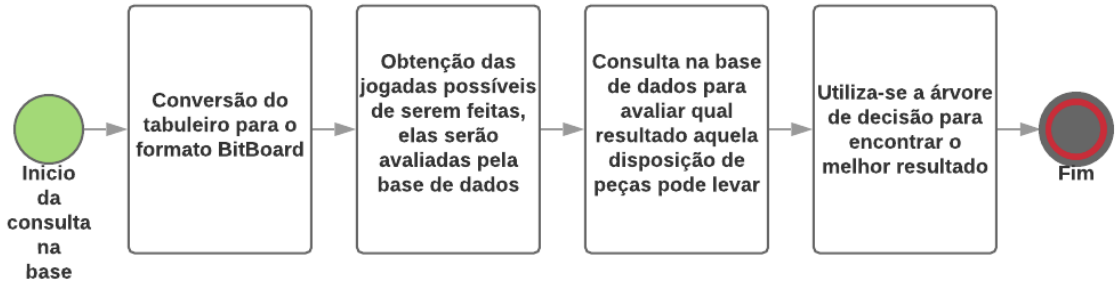
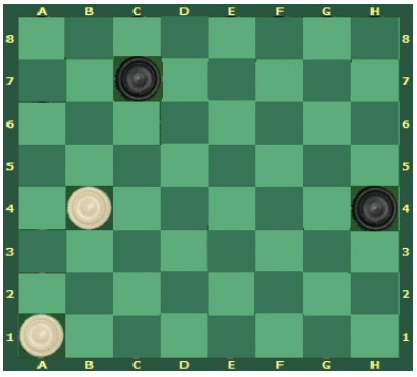



Figura 5.7: Fluxograma que ilustra o acesso da base de dados

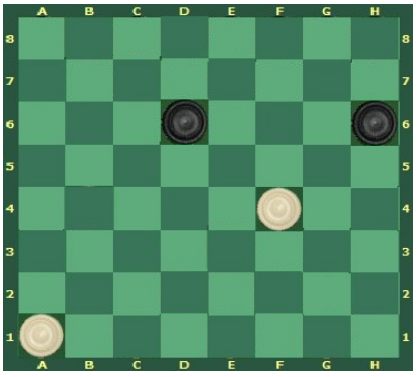
Ao fazer a consulta na base de dados, a avaliação do estado de jogo atual pode ser classificada de três formas:

- O valor de retorno igual a $1$ indica que o estado que está sendo avaliado no momento é um estado de vitória para quem fará o movimento;
- O valor de retorno igual a $2$ indica que o estado que está sendo avaliado no momento é um estado de derrota para quem fará o movimento;
- O valor de retorno igual a $3$ indica que o estado que está sendo avaliado no momento é um estado de empate para quem fará o movimento.

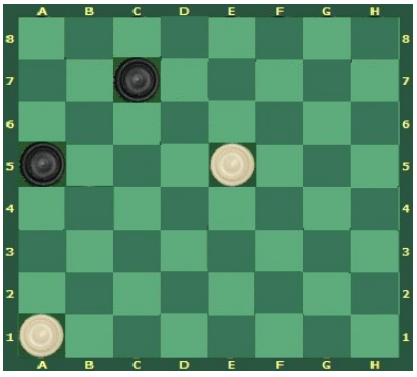
Os testes foram realizados comparando o oponente implementado neste trabalho com a base de dados Chinook, usando-a como parâmetro para medir a resposta do mesmo, se ele retorna resultados tão positivos quanto o da base Chinook. Para efetuar esses testes foram utilizadas finais de jogo como as apresentadas por Malamed (SOUSA, 2012a). As figuras 5.8 e 5.9 apresentam algumas dessas finais. As finais 5.9(f) e 5.9(g) podem ser encontradas em Reinfield (2011).



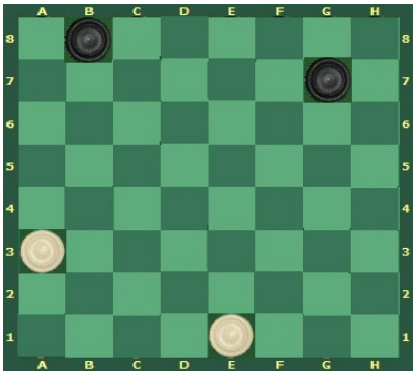
(a) Primeira final



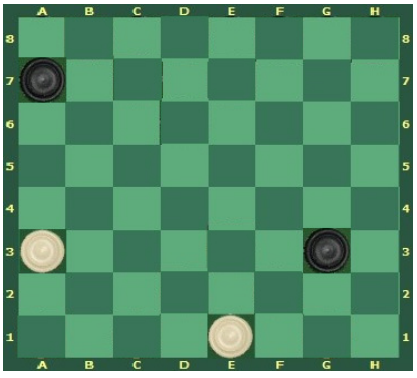
(b) Segunda final



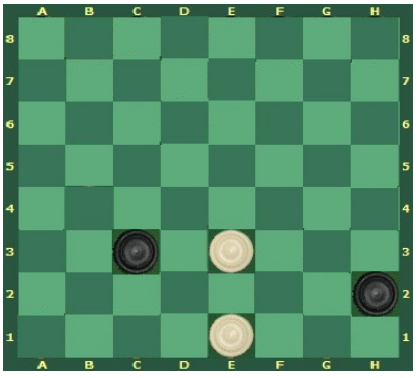
(c) Terceira final



(d) Quarta final

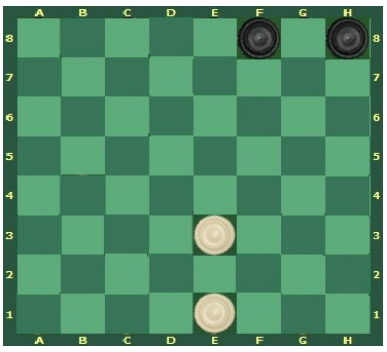


(e) Quinta final

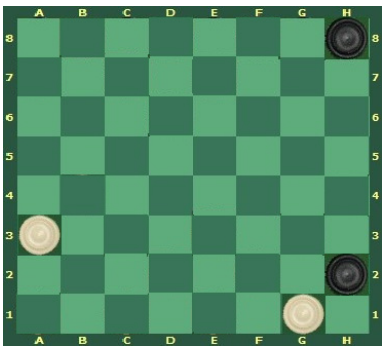


(f) Sexta final

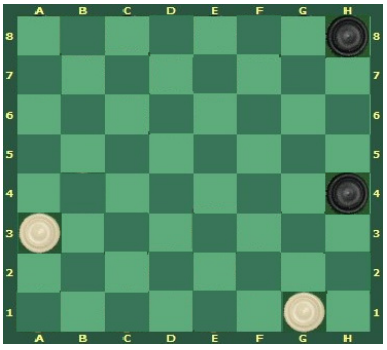
Figura 5.8: Disposição das peças ilustrando as respectivas finais



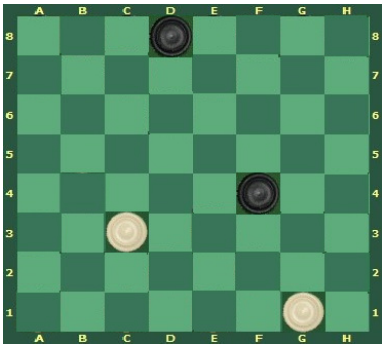
(a) Sétima final



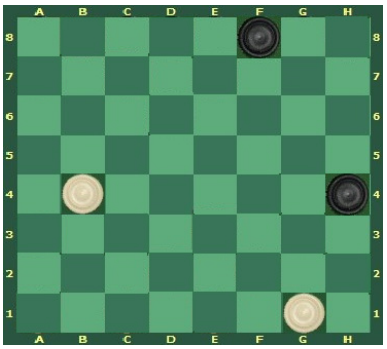
(b) Oitava final



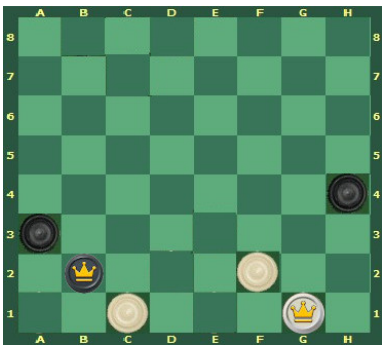
(c) Nona final



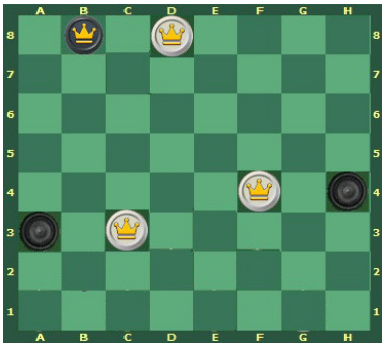
(d) Décima final



(e) Décima primeira final



(f) Décima segunda final



(g) Décima terceira final

Figura 5.9: Disposição das peças ilustrando as respectivas finais

Os testes foram realizados sobre as finais apresentadas nas figuras 5.8 e 5.9, onde cada possível jogada é armazenada na árvore de decisão e as jogadas que são mais vantajosas, que levam a vitória ou empate, são preteridas. Assim, para cada jogada faz-se um acesso na base de dados e verifica-se se aquela posição é uma posição de vitória, empate ou derrota, faz-se isso até atingir o final do jogo. A árvore irá armazenar o estado do jogo que representa uma movimentação e o valor associado àquele estado, atribuído pela base de dados. Assim, durante a iteração na árvore para procura da melhor jogada, o valor associado ao estado é avaliado na busca, e estados que não apresentam melhora não precisam ser visitados, dada a propriedade da poda alfa-beta de não visitar tais nós. Caso o estado não seja encontrado na base de dados, busca-se visitar os filhos desse estado até encontrar um que esteja, respeitando a profundidade máxima da árvore de 3 níveis.

Nesse sentido, após a realização de testes com profundidades fixas para árvore de nível 3, 5 e 7, foi estabelecida a escolha de nível 3 para a árvore, dada a robustez da base de dados, onde um aumento no nível da árvore não apresentou melhoras nos resultados encontrados, porém, tornou a busca mais custosa. Tal forma de se realizar os testes aumentando a profundidade da árvore é feita por Caexêta (2008).

Assim, dadas essas posições de finais de jogo, os testes ocorreram da seguinte forma: as disposições das peças nas imagens 5.8 e 5.9 são inseridas manualmente no programa e é feita uma busca na base de dados, conforme figura 5.7, utilizando as bibliotecas fornecidas por Gilbert (2004), as possíveis jogadas são armazenadas na árvore de busca e o algoritmo minimax com a poda alfa-beta entra em ação, procurando avaliar quais as jogadas são as melhores, ou seja, as que levam a uma vitória ou um empate pelo menos.

A forma que os testes foram realizados são ilustrados pelo fluxograma da figura 5.10, esses passos são repetidos para cada teste realizado.

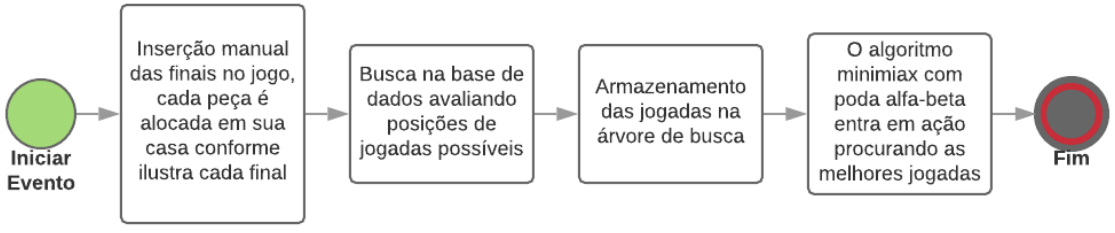



Figura 5.10: Fluxograma que ilustra os testes com a ação do algoritmo minimax

## 5.4 Resultados e Discussões

Os resultados em relação ao desempenho do oponente enquanto estratégia para uma melhor ação se mostraram bons contra um oponente humano, pois a característica da base de dados Chinook de fornecer informações perfeitas para estados de 8 peças ou menos acaba favorecendo o oponente a sempre encontrar uma solução ótima ou satisfatória, nesse caso vitória ou empate respectivamente.

Dada a comparação com o arquivo em C fornecido pelo projeto Chinook com o oponente e a consulta realizada por ele na base de dados, temos os resultados apresentados na tabela 5.1.

Tabela 5.1: Resultados dos testes em comparação com o resultado da base Chinook

| Finais | Resultados dos testes | Resultados do projeto Chinook |
|---|---|---|
| 5.8(a) | Empate | Empate |
| 5.8(b) | Vitória | Empate |
| 5.8(c) | Empate | Empate |
| 5.8(d) | Empate | Empate |
| 5.8(e) | Vitória | Empate |
| 5.8(f) | Empate | Empate |
| 5.9(a) | Derrota | Derrota |
| 5.9(b) | Empate | Empate |
| 5.9(c) | Empate | Empate |
| 5.9(d) | Empate | Empate |
| 5.9(e) | Empate | Empate |
| 5.9(f) | Empate | Empate |
| 5.9(g) | Derrota | Derrota |

Devido as configurações das finais serem mais limitadas, dado o baixo número de peças, a probabilidade de empate é maior e se apresentou majoritariamente nos testes, pois mesmo um jogador com pouco conhecimento, nessas configurações, pode levar o jogo a um empate apenas escapando de capturas acidentais. Para as duas finais com um número maior de peças, o desempenho do oponente atingiu um resultado esperado pela base. Apesar disso, o oponente mostra ter um bom desempenho, pois exibe um resultado parecido com os resultados gerados da avaliação do código em C da base Chinook, porém, nas posições 5.8(b) e 5.8(e) conseguiu ser melhor do que o resultado calculado pelo algoritmo do projeto Chinook, pois conseguiu levar a vitória.

Em relação ao gasto de memória, para cada acesso dentro da base de dados Chinook, é

alocado um espaço de 2000 MB da memória RAM, que por si só torna o desempenho do jogo menos eficiente. Como são avaliados mais de um estado por busca, a média de acesso é de 11 consultas, e apresenta o seguinte cenário:

- 1124 KBytes alocados para indexação;
- 47003 kBytes alocados para buffer de cache da base de dados, responsável por armazenar os dados da base de dados;
- 390870,909 KBytes de espaço livre na memória em média.

Os dois primeiros valores são fixos pois todos os cenários de fim de jogo acessam a base de dados com 6 peças ou menos. Nesse sentido, uma estratégia de busca na árvore com profundidade muito grande geraria um gasto de tempo mais elevado e um estouro na pilha de recursão, logo, a adoção de um valor fixo para a profundidade acaba por evitar esse problema. Outra abordagem é limitar a busca na árvore por um determinado tempo, técnica utilizada por Caexêta (2008).

# Capítulo 6

# Considerações Finais

A implementação do jogo e do oponente se deu da seguinte forma: primeiro estabeleceu-se a forma como o tabuleiro seria representado e quais os métodos para verificação de alternância de jogadores e critérios de parada do jogo.

Para as jogadas do oponente definiu-se que as três primeiras jogadas seriam feitas de forma aleatória. Após isso, ocorreu a implementação das heurísticas para meio de jogo, a do triângulo defensivo e a heurística posicional. O jogador escolhe qual das duas será utilizada no jogo.

Por fim, para finais de jogo com 8 peças ou menos, é feito o acesso a base Chinook por meio das bibliotecas de Gilbert (2004), avaliando a posição atual e as possibilidades de jogadas, buscando a melhor através do algoritmo minimax com poda alfa-beta.

Os testes ocorreram com as finais mostradas nas figuras 5.8 e 5.9, onde a posição atual e as possibilidades de jogadas são avaliadas pelo algoritmo minimax com poda alfa-beta, procurando a melhor jogada dentre as possíveis, que leve a uma vitória ou empate pelo menos.

Em finais de jogos o oponente apresenta um bom desempenho dada a base de dados robusta onde realiza os acessos, assim, a menos que durante a partida as estratégias de início e meio de jogo o coloque em uma situação onde a derrota é irreversível, em finais de jogos ele conseguirá manter a vantagem obtida nos estágios anteriores do jogo.

## 6.1 Trabalhos Futuros

Como sugestão de trabalhos futuros, podemos considerar as opções a seguir:

- Uso de tabelas de transposição aliadas a poda alfa-beta, utilizada por Caexêta (2008), que visa reduzir ainda mais o número de nós visitados durante a poda.

- A utilização de configurações maiores para finais de jogos, com um número maior de peças, procurando evitar empates.

- Uma abordagem para meio de jogo utilizando algoritmos genéticos, podendo-se utilizar a representação em Bitboard do tabuleiro como cromossomos.

- Utilização do método *Random Forest*[1] aplicado a finais de jogo, e realizar uma comparação do seu desempenho com a base de dados Chinook.

- Comparação entre as bases de dados Chinook com outras bases, como a Kingsrow e a Cake, avaliando por exemplo, desempenho e tempo de resposta.

[1]É um classificador que contém várias árvores de decisão. Sua proposta é criar uma estrutura que contém pequenas árvores de decisão cujo desempenho individual é inferior a uma árvore de decisão comum, porém, a união delas resultaria em uma estrutura mais poderosa, com uma maior gama de predições. Mais detalhes podem ser vistos em Breiman (2001).

# Apêndice A

# Diagrama de classes

A seguir temos o diagrama de classes ilustrando o comportamento do programa durante o jogo e a comunicação entre as classes, a relação entre a classe `tabuleiro`, que é a classe onde estão os métodos principais de movimentação do jogador, com a classe do oponente, que contém os principais métodos de movimentação dele, a classe `BitBoard`, que converte o tabuleiro para a representação BitBoard que é utilizado para acessar a base de dados, dentre outras classes.

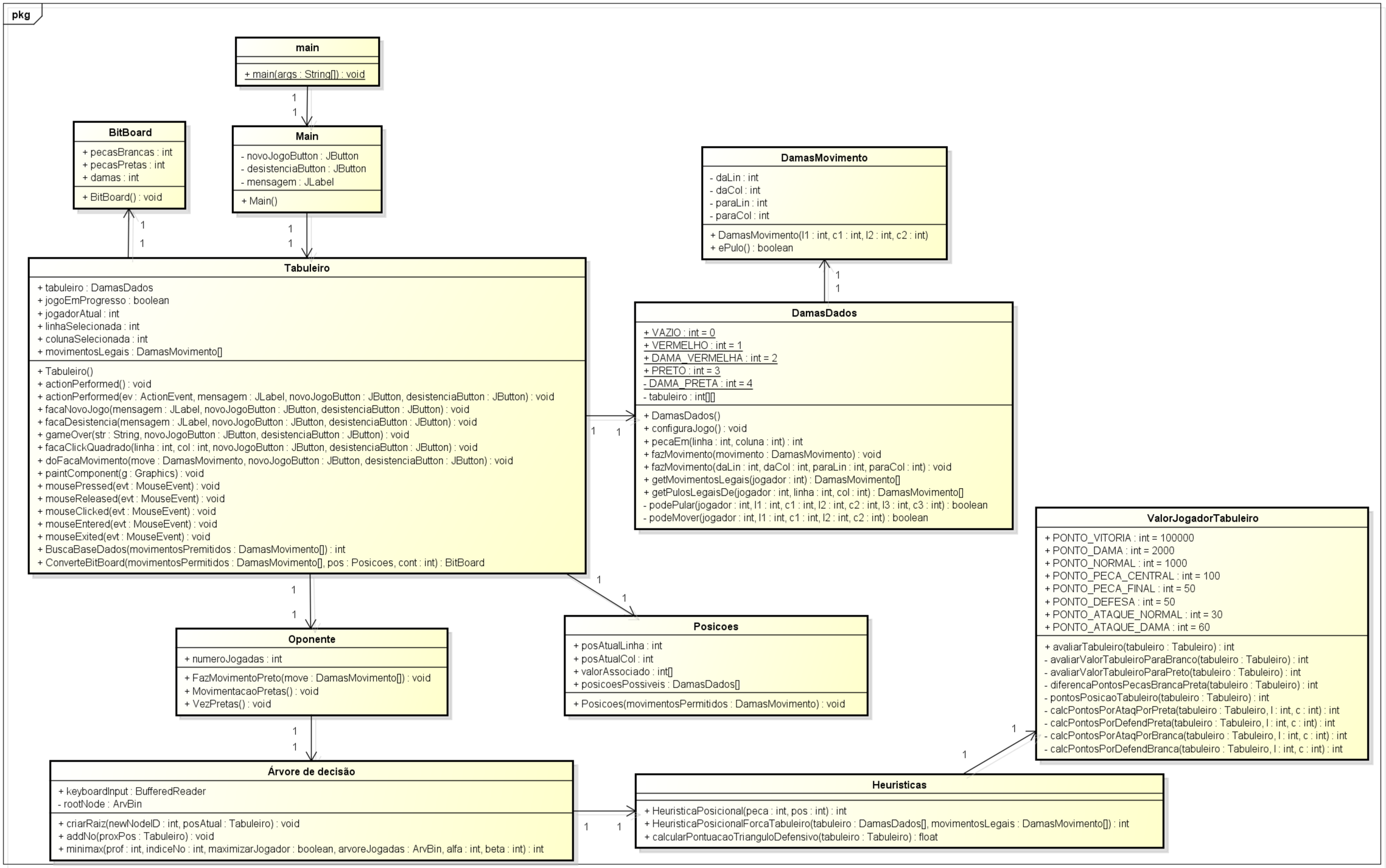



Figura A.1: Imagem do diagrama de classes do programa

# Apêndice B

# Principais algoritmos do programa

```
fun miniMax(depth:int, nodeIndex:int,
                  maximizingPlayer:boolean,
                  values[]:int, alpha:int,
                  beta:int):int

    if depth == 3
        return values[nodeIndex]

    if maximizingPlayer == true
        best:int
        i:int
        best := MIN

        for i = 0,...,2
            val:int
            val := minimax(depth + 1, nodeIndex * 2 + i,
                           false, values, alpha, beta)
            best := Math.max(best, val)
            alpha := Math.max(alpha, best)

            if beta <= alpha then
                break
        end for
        return best

    else
        best:int
        i:int
        best := MAX

        for i = 0,...,2
            val:int
            val := minimax(depth + 1, nodeIndex * 2 + i,
                           true, values, alpha, beta)
```

```
            best := Math.min(best, val)
            beta := Math.min(beta, best)


            if beta <= alpha then
                break
        end for
        return best
```

Algoritmo B.1: Pseudocódigo do algoritmo minimax com poda alfa-beta

```
Java_dama_Main_NativeBuscaChinook
  (env:JNIEnv, obj:jobject, pecasBrancas:jint, pecasPretas:jint,
     damas:jint):JNIEXPORT jchar JNICALL

    i:int
    status:int
    max_pieces:int
    value:int
    nerrors:int
    resultado:char

    pos:EGDB_BITBOARD
    egdb_type:EGDB_TYPE
    handle:EGDB_DRIVER

    pos.normal.white := pecasBrancas
    pos.normal.black := pecasPretas
    pos.normal.king := damas

    status := egdb_identify(DB_PATH, egdb_type, max_pieces)

    if status == true then
            print "base de dados nao encontrada em" + DB_PATH
            return 1
        endif

    print "Base de dados do tipo" + egdb_type "encontrada com maximo
       de pecas" + max_pieces

    // Abrindo base de dados.
    handle := egdb_open(EGDB_NORMAL, max_pieces, 2000, DB_PATH,
       print_msgs)
    if !handle == true then
            print "Error returned from egdb_open()"
            return 1
    endif
```

```
34
35      print "Testando posicao."
36      value := handle->lookup(handle, &pos, EGDB_BLACK, 0)
37
38      resultado := char(value)
39
40      return resultado
```

Algoritmo B.2: Pseudocódigo do algoritmo de acesso a base de dados

# Referências Bibliográficas


BARCELOS, A. R. A. *D-VISIONDRAUGHTS: Uma Rede Neural Jogadora de Damas que Aprende por Reforço em um Ambiente de Computação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia - MG, Janeiro 2011.

BREIMAN, L. Random forests. *Machine Learning*, California, v. 45, n. 1, p. 5–32, October 2001.

CAEXÊTA, G. S. *VisionDraughts – Um Sistema de Aprendizagem de Jogos de Damas Baseado em Redes Neurais, Diferenças Temporais, Algoritmos Eficientes de Busca em Árvores e Informações Perfeitas Contidas em Bases de Dados*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia - MG, Julho 2008.

CARVALHO, L. F. B. S. de; NETO, H. C. S.; LOPES, R. V. V. *Um Jogo de Damas Evolutivo*. Dissertação (Mestrado) — Universidade Federal de Alagoas, Maceió - AL, Outubro 2009.

CHELLAPILLA, K.; FOGEL, D. B. Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, New Jersey, v. 5, n. 4, p. 422–428, August 2001.

DUARTE, V. A. R. *MP-DRAUGHTS - Um Sistema Multiagente de Aprendizagem Automática para Damas Baseado em Redes Neurais de Kohonen e Perceptron Multicamadas*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia - MG, Julho 2009.

FILHO, P. B. *Nomes das Aberturas Técnicas do Jogo de Damas*. 2015. Disponível em: <https://www.facebook.com/FederacaoBrasiliense/posts/1042714082422475/>. Acesso em: 20 nov 2019.

GILBERT, E. *Kingsrow*. 2004. Disponível em: <http://edgilbert.org/Checkers/KingsRow.htm>. Acesso em: 20 nov 2019.

JUSTUS, C. G. *Jogo de Damas Embarcado Multinível*. Dissertação (Monografia) — Centro Universitário Positivo, Curitiba, fevereiro 2006.

KENDALL, G. *Checkers Research Page*. 2001. Disponível em: <http://www.cs.nott.ac.uk/~pszgxk/games/checkers/research.html>. Acesso em: 20 nov 2019.

LIN, S. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, New York, v. 44, n. 10, p. 2245–2269, December 1965.

NETBEANS, A. *NetBeans*. 2019. Disponível em: <https://netbeans.org/>. Acesso em: 20 nov 2019.

NETO, H. de C. *LS-Draughts – Um Sistema de Aprendizagem de Jogos de Damas baseado em Algoritmos Genéticos, Redes Neurais e Diferenças Temporais*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia - MG, Janeiro 2007.

NEWELL, A.; ERNST, G. The search for generality. In: KALENICH, W. A. (Ed.). *Information Processing 1965: Proceedings of IFIP Congress 65*. New York City: Spartan Books, 1965. v. 1, p. 17 – 24.

NEWELL, A.; SHAW, J. C.; SIMON, H. A. Elements of a theory of human problem solving. *American Psychological Association*, v. 65, n. 3, p. 151 – 166, 1958.

ORACLE. *JAVA*. 2019. Disponível em: <https://www.java.com/pt_BR/>. Acesso em: 20 nov 2019.

RAYMENT, W. J. *History of Checkers or Draughts*. 2019. Disponível em: <http://www.indepthinfo.com/checkers/history.shtml>. Acesso em: 20 nov 2019.

REINFIELD, F. *How to win at checkers*. 2011. Disponível em: <http://www.bobnewell.net/filez/reinfeld2ndedition.pdf>. Acesso em: 20 nov 2019.

RUSSEL, S.; NORVIG, P. *Inteligência Artificial*. 2. ed. Reading: Campus, 2004.

SARCEDO, L. M. L. *História - Jogo de damas*. 2019. Disponível em: <http://www.damasciencias.com.br/historia_do_jogo_damas.html>. Acesso em: 20 nov 2019.

SCHAEFFER, J. et al. *Solving Checkers*. 2019. Disponível em: <https://www.ijcai.org/Proceedings/05/Papers/0515.pdf>. Acesso em: 20 nov 2019.

SCHAEFFER, J. et al. Chinook the world man-machine checkers champion. *AI Magazine*, New York, v. 17, n. 1, p. 21–29, February 1996.

SCHAEFFER, J. et al. Checkers is solved. *Science*, Edmonton, v. 317, n. 5844, p. 1518–1522, September 2007.

SILVA, G. G. da. *Segredos do Jogo de Damas*. 1. ed. Rio de Janeiro - RJ: Editora Gráfica Pitombeira, 1987.

SOUSA, F. G. *Curso de Finais*. 2012. Disponível em: <http://www.damasciencias.com.br/base_teorica/finais_malamede/jogos.htm>. Acesso em: 20 nov 2019.

SOUSA, F. G. de. *A FORÇADA*. 2012. Disponível em: <http://www.damasciencias.com.br/base_teorica/a_forcada.pdf>. Acesso em: 20 nov 2019.

SOUSA, F. G. de. *Jogo de Damas – Regras Oficiais*. 2019. Disponível em: <http://www.damasciencias.com.br/regras/regras_do_jogo.html>. Acesso em: 20 nov 2019.

TOMAZ, L. B. P. *D-MA-Draughts: Um sistema multiagente jogador de damas automático que atua em um ambiente de alto desempenho*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia - MG, Maio 2013.

University of Alberta. *Chinook*. 2019. Disponível em: <https://webdocs.cs.ualberta.ca/~chinook/project/>. Acesso em: 20 nov 2019.