UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE APOIO AO APRENDIZADO DO JOGO DE XADREZ

MARCELO AUGUSTO BAMBINO

BLUMENAU
2015

2015/1-22

MARCELO AUGUSTO BAMBINO

FERRAMENTA DE APOIO AO APRENDIZADO DO JOGO DE XADREZ

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Roberto Heinzle, Doutor - Orientador

BLUMENAU
2015

2015/1-22

FERRAMENTA DE APOIO AO APRENDIZADO DO JOGO DE XADREZ

Por

MARCELO AUGUSTO BAMBINO

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Roberto Heinzle, Doutor – Orientador, FURB
Membro:	Prof. Maurício Capobianco Lopes, Doutor – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 30 de junho de 2015

Dedico este trabalho a todos que contribuíram para que ele fosse realizado e para todos que contribuirão numa possível continuação.

AGRADECIMENTOS

À minha família, em especial ao meu pai, pelo apoio em toda minha educação.

Aos meus amigos, por todo o apoio durante o decorrer do trabalho.

Ao meu orientador, professor Roberto Heinzle, pela confiança.

Se você encontrar um caminho sem obstáculos, ele provavelmente não leva a lugar nenhum.

Frank Clark

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta *desktop* na linguagem de programação Java para o ensino e aprendizado do jogo de xadrez, baseado na metodologia de Goulart e Frei (2004), juntamente com uma *engine* implementada com base nos algoritmos Minimax e Poda Alfa-Beta. Para as telas da aplicação foram utilizados `JFrame` e `JPanel`, com as informações do sistema sendo armazenadas no banco de dados relacional *Oracle*. A ferramenta obteve êxito na implementação da metodologia, porém encontrou limitações de *hardware* na implementação dos algoritmos devido à profundidade das árvores de movimentos geradas. Os resultados demonstram que a utilização dos algoritmos Minimax e Poda Alfa-Beta para o jogo de xadrez é válida, porém necessita de um hardware potente para suportar a performance e o consumo que os algoritmos exigem.

Palavras-chave: Xadrez. Minimax. Poda Alfa-Beta. Metodologia de ensino.

ABSTRACT

This paper presents the development of a desktop tool in the Java programming language for teaching and learning chess game, based on the methodology Goulart and Frei (2004), along with a engine implemented on the basis of Minimax algorithms and Pruning Alfa-Beta . For application screens was used `JFrame` and `JPanel`, and system information is stored in relational Oracle database. The tool was successful in implementing the methodology, but found hardware limitations in the implementation of algorithms due to the depth of the generated movements trees. The results demonstrate that the use of the pruning algorithms and Minimax Alpha-Beta for the game of chess is valid, but requires a powerful hardware to support the performance and consumption algorithms that require.

Keywords: Chess. Minimax. Pruning Alfa-Beta. Teaching methodology.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

DOC – *Document*

FIDE – *World Chess Federation*

GPS – *General Problem Solver*

IA – Inteligência Artificial

PDF – *Portable Document Format*

PNG – *Portable Network Graphics*

PSP – *PlayStation Portable*

SUMÁRIO

1 INTRODUÇÃO

Para Antonio Villar Marques de Sá, professor da faculdade de educação da Universidade de Brasília e um dos maiores especialistas em Pedagogia Enxadrística no Brasil, o xadrez é uma das melhores atividades para desenvolver a capacidade intelectual dos jovens (CANAL XADREZ, 2009). França (2012) destaca que em vários países como a Rússia, Inglaterra, Argentina, Cuba e Espanha o jogo de xadrez é uma ferramenta utilizada na forma de projetos ou de disciplinas extracurriculares que são incorporados nas escolas buscando explorar os benefícios, vantagens e virtudes do seu aprendizado.

Para acompanhar esta disseminação do xadrez no ambiente pedagógico, várias metodologias vêm surgindo e sendo exploradas, dentre as quais se podem destacar as de Gomes (2010), Assim Se Faz (2012) e Goulart e Frei (2004). Estas têm em comum o objetivo de facilitar o aprendizado do jogo e estimular a sua prática.

Igualmente a indústria de software tem manifestado interesse no desenvolvimento de ferramentas capazes de permitir aos jogadores e aprendizes praticarem o esporte. A IBM foi uma das pioneiras, iniciando seus trabalhos nesta área no início da década de 1950 com o surgimento da Inteligência Artificial (IA) (IBM, 2014). Segundo Barbosa, Veiga e Carvalho (2012), a IA tem forte presença na maioria dos jogos, permitindo que a interação seja mais dinâmica e passando ao jogador a sensação de que a máquina realmente está pensando e se comportando como um ser humano, objetivando vencê-lo. Com o desenvolvimento de algoritmos e heurísticas, esta ascensão da IA, não apenas no cenário dos jogos, torna-se evidente. Esta característica encaixa-se perfeitamente no cenário do xadrez, pois possibilita que o jogador possa praticar por conta própria, facilitando a acessibilidade deste esporte.

Dentre o vasto leque de técnicas e algoritmos que a IA oferece, os algoritmos Minimax e Poda Alfa-Beta destacam-se no cenário dos jogos de dois jogadores como xadrez e o jogo da velha, por exemplo, e serão explorados no desenvolvimento deste trabalho. Diante deste cenário, percebe-se o potencial em explorar a união entre o xadrez e a IA. Sendo assim, este trabalho propõe-se a desenvolver uma ferramenta baseada na metodologia de Goulart e Frei (2014) para auxiliar pessoas a aprenderem a jogar xadrez.

1.1 OBJETIVOS DO TRABALHO

O objetivo geral deste trabalho é disponibilizar uma ferramenta didática voltada ao ensino/aprendizado do jogo de xadrez.

Os objetivos específicos são:

a) usar os algoritmos Minimax e Poda Alfa-Beta, para que este aumente a eficiência do primeiro;
b) disponibilizar uma interface gráfica para aprender a jogar xadrez compatível com a metodologia de Goulart e Frei (2004);
c) desenvolver uma *engine* capaz de jogar xadrez.

1.2 ESTRUTURA

Este trabalho divide-se em fundamentação teórica, desenvolvimento do software e conclusões.

No capítulo referente à fundamentação teórica está descrito o jogo de xadrez. Em seguida são abordadas metodologias de ensino deste jogo, com destaque para a de Goulart e Frei (2004), que será utilizada como base neste trabalho. Depois é abordada Inteligência Artificial no xadrez, focando nos algoritmos Minimax e Poda Alfa-Beta que são utilizados no trabalho. No fim do capítulo, apresentam-se os trabalhos correlatos.

No capítulo que aborda o desenvolvimento da ferramenta é apresentada a metodologia de desenvolvimento e os algoritmos utilizados, através de diagramas.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo divide-se em quatro seções. A seção 2.1 trata do jogo de xadrez. A seção 2.2 trata de metodologias de ensino de xadrez. A seção 2.3 aborda sobre a aplicação da Inteligência Artificial no xadrez. Por fim, a seção 2.4 trata dos trabalhos correlatos.

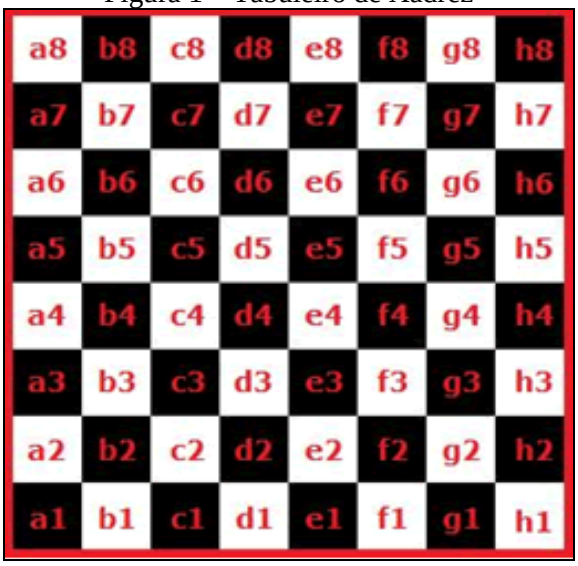
2.1 O JOGO DE XADREZ

Xadrez é um jogo de mesa que é praticado em um tabuleiro de 64 casas (divididas em 32 claras e 32 escuras) onde dois jogadores se enfrentam. Eles efetuam lances alternadamente, operando seis tipos de peças diferentes, também de cores opostas, com o objetivo de finalizar o Rei adversário, jogada popularmente conhecida como Xeque-mate (CLUBE DE XADREZ ONLINE, 2001). Segundo Ferreira (2012, p.13), "o xadrez é considerado um jogo intelectual e ao mesmo tempo emocionante, de tática, estratégia, raciocínio lógico e cálculos. Também é considerado arte e cultura, servindo de inspiração para músicas, filmes, pinturas, esculturas, propagandas, entre outros".

2.1.1 Tabuleiro

O tabuleiro de xadrez é dividido em 8 colunas verticais e 8 fileiras horizontais. As casas são alternadamente escuras e claras. A primeira casa no extremo esquerdo do tabuleiro deve ser uma casa preta e a última casa no extremo direito, uma casa branca. Cada coluna é designada por uma letra (a...h), enquanto as fileiras são designadas por um número (1..8). Dessa forma cada casa é referenciada pela letra e número correspondentes a sua coluna e fileira (a1, b6, f5, etc.), sendo esse o sistema padrão de notação em competições oficiais (SÓ XADREZ, 2013b). Na Figura 1 tem-se o tabuleiro de xadrez.

Figura 1 – Tabuleiro de Xadrez



Fonte: Só Xadrez (2013b).

2.1.2 Peças

Cada jogador possui um conjunto de 16 peças, composto por Rei, Rainha, Torre, Bispo, Cavalo e Peão. Inicialmente, as peças brancas iniciam nas fileiras 1 e 2, enquanto que as pretas iniciam nas fileiras 7 e 8 (SÓ XADREZ, 2013a). Na Figura 2 tem-se a posição inicial das peças no tabuleiro.

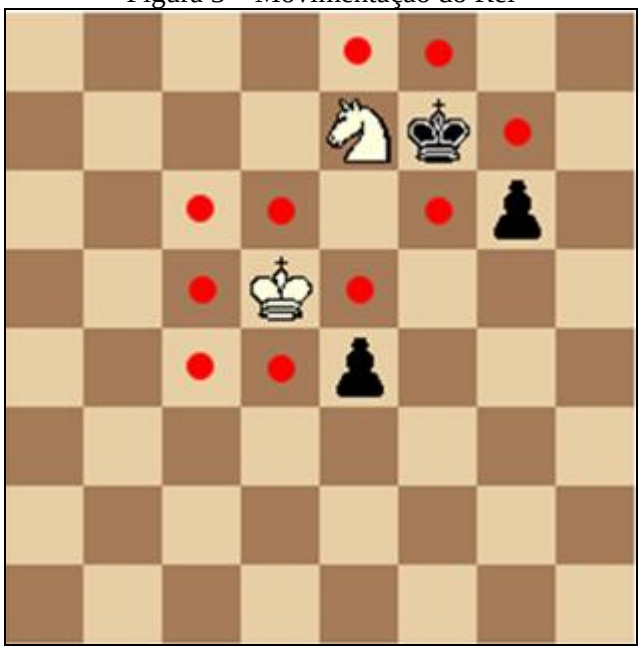
Figura 2 – Posição Inicial do Xadrez

Fonte: Só Xadrez (2013a).

2.1.3 Movimentação das Peças

O Rei move-se em qualquer sentido, podendo deslocar-se uma única casa por vez, com as restrições de o Rei não ficar em xeque ao final do movimento e de que o Rei branco e o Rei preto fiquem pelo menos a uma casa de distância (SÓ XADREZ, 2013a). Na Figura 3 tem-se um exemplo de movimentação dos Reis.

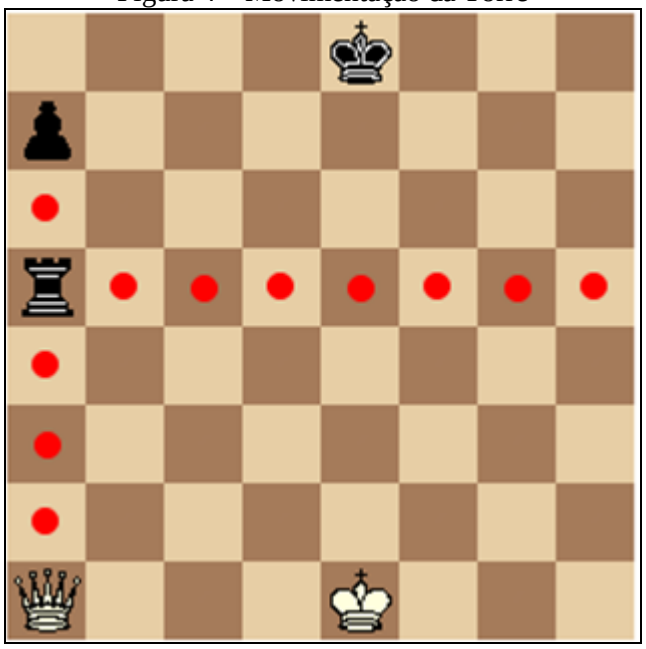
Figura 3 – Movimentação do Rei



Fonte: Xadrez Regional (2012).

A Torre move-se na horizontal e na vertical, quantas casas quiser, porém sem poder pular por cima de nenhuma peça (SÓ XADREZ, 2013a). Na Figura 4 tem-se um exemplo de movimentação da Torre.

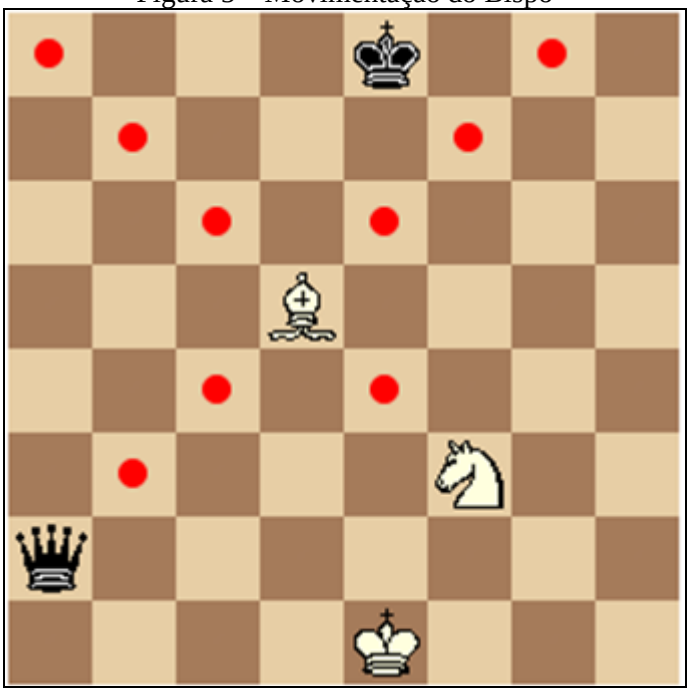
Figura 4 – Movimentação da Torre



Fonte: Xadrez Regional (2012).

O Bispo move-se na diagonal para qualquer direção, quantas casas quiser, porém sem poder pular por cima de nenhuma peça (SÓ XADREZ, 2013a). Na Figura 5 tem-se um exemplo de movimentação do Bispo.

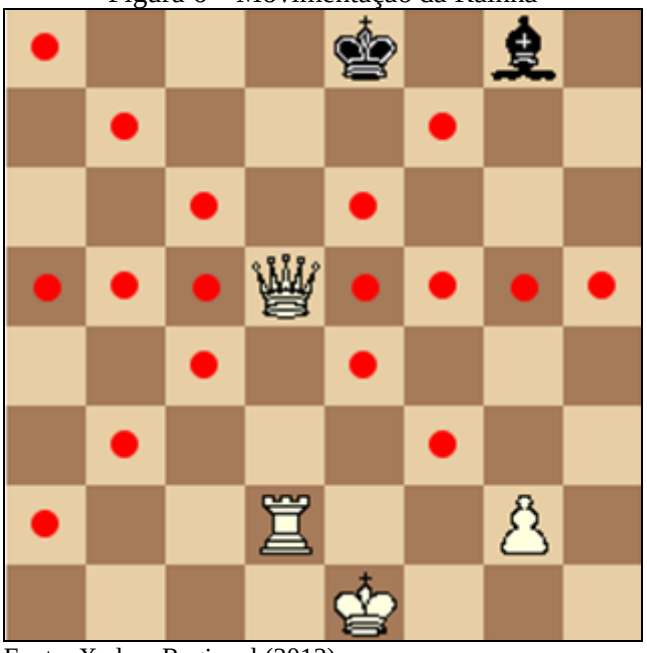
Figura 5 – Movimentação do Bispo



Fonte: Xadrez Regional (2012).

A Rainha possui a movimentação da Torre e do Bispo, ou seja, pode mover-se em linha reta e/ou diagonal para qualquer direção, porém sem poder pular por cima de nenhuma peça (SÓ XADREZ, 2013a). Na Figura 6 tem-se um exemplo de movimentação da Rainha.

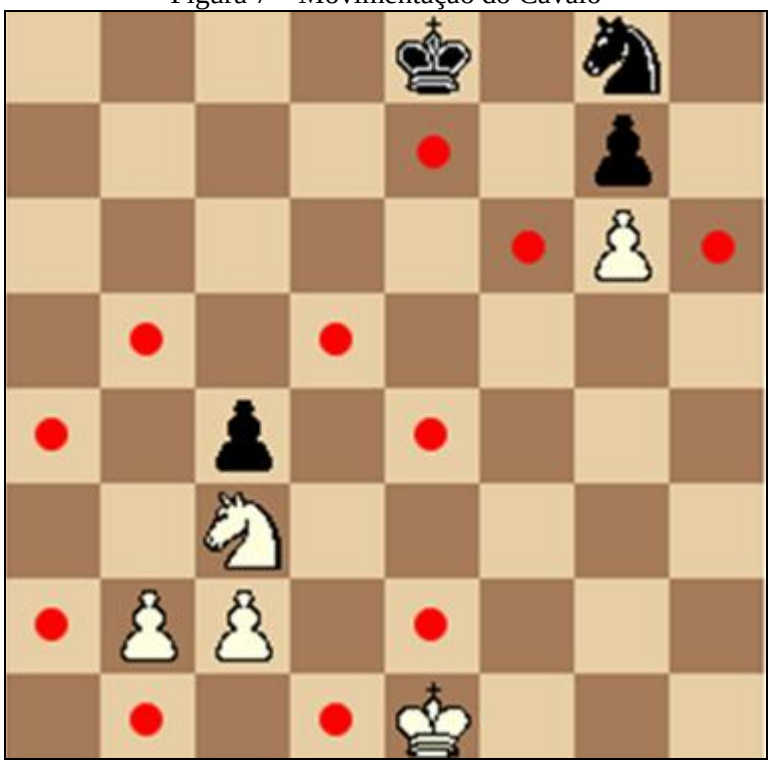
Figura 6 – Movimentação da Rainha



Fonte: Xadrez Regional (2012).

O Cavalo é a única peça que pode pular sobre outra peça, independente da cor da peça. Seu movimento é no formato de “L” e pode ser para qualquer direção (SÓXADREZ, 2013a). Na Figura 7 tem-se um exemplo das casas que o Cavalo pode movimentar-se.

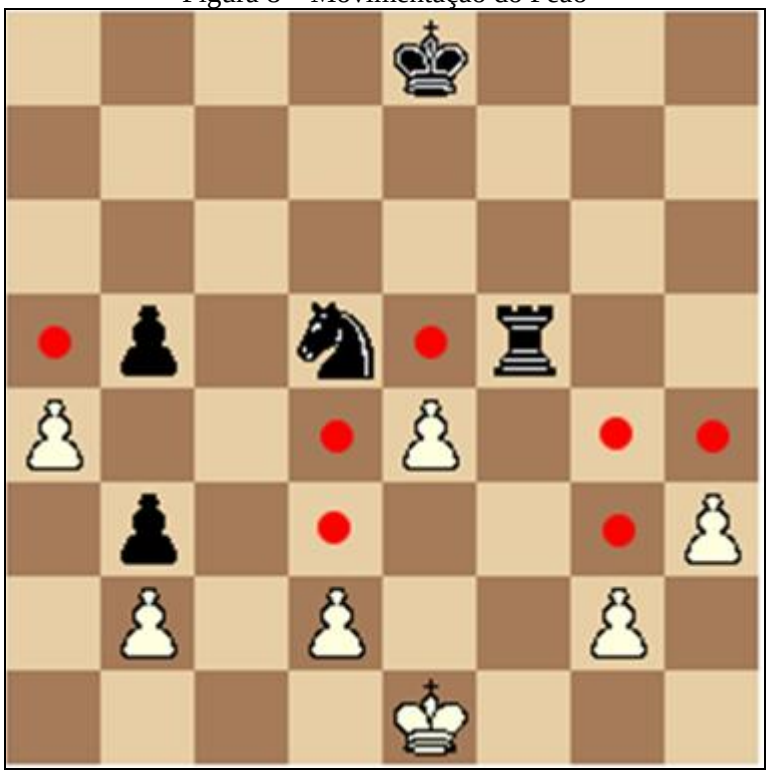
Figura 7 – Movimentação do Cavalo



Fonte: Xadrez Regional (2012).

O Peão move-se para frente, desde que a casa esteja desocupada. No primeiro movimento de cada Peão, ele pode andar uma ou duas casas. Para capturar outra peça, o Peão o faz na diagonal, esquerda ou direita. É importante ressaltar que o Peão também é a única peça que não pode andar nem capturar para trás (SÓ XADREZ, 2013a). Na Figura 8 tem-se um exemplo de movimentação do Peão.

Figura 8 – Movimentação do Peão



Fonte: Xadrez Regional (2012).

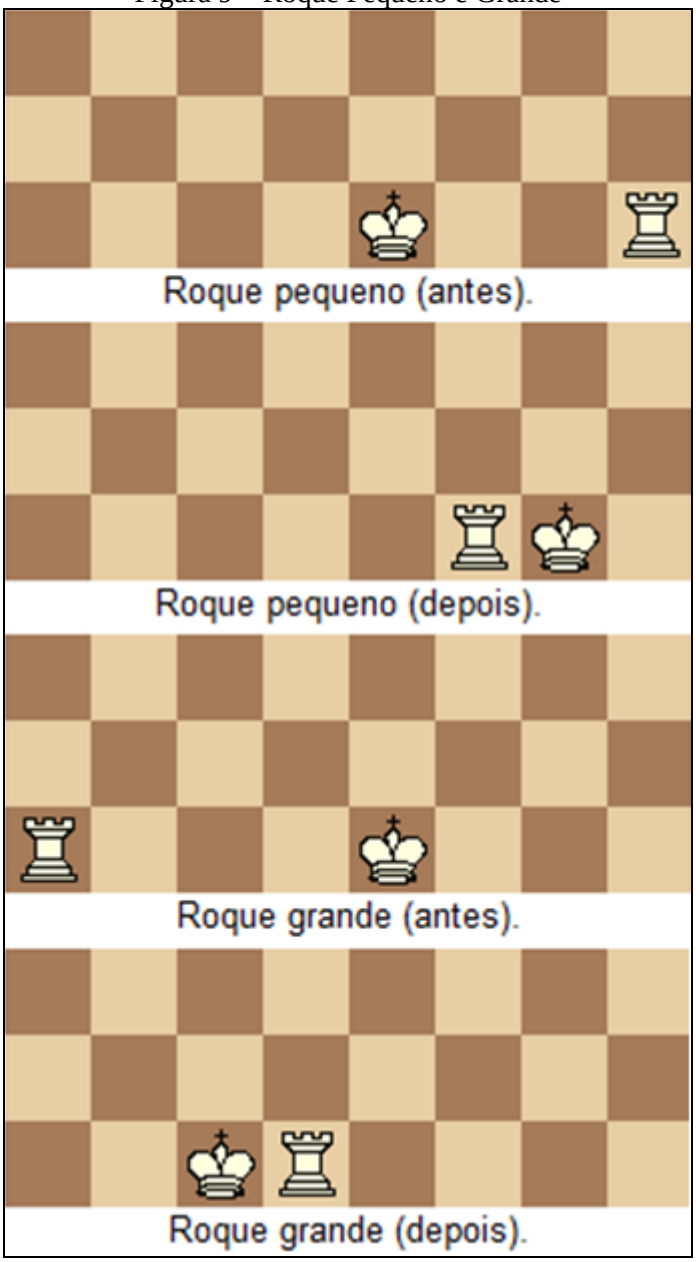
2.1.3.1 Roque

Roque é o único lance que envolve o movimento de duas peças ao mesmo tempo, o Rei e a Torre. Seu objetivo é deixar o Rei em uma posição mais segura no canto do tabuleiro. Há o Roque grande e o Roque pequeno (XADREZ REGIONAL, 2012). Independente do tipo de Roque, o Rei anda duas casas em direção da Torre e esta passa por cima do Rei. Para fazer o Roque, as seguintes condições devem ser verdadeiras (XADREZ REGIONAL, 2012):

a) o Rei e a Torre que participarão do movimento não podem ter se movido até então;
b) o Rei não pode estar em xeque no início do movimento;
c) o Rei não pode ficar em xeque no fim do movimento;
d) o Rei não pode passar por uma casa atacada pelo adversário durante o movimento;
e) não pode haver peça alguma entre o Rei e a Torre.

Na Figura 9 tem-se um exemplo de Roque pequeno e Roque grande, respectivamente.

Figura 9 – Roque Pequeno e Grande



Fonte: Xadrez Regional (2012).

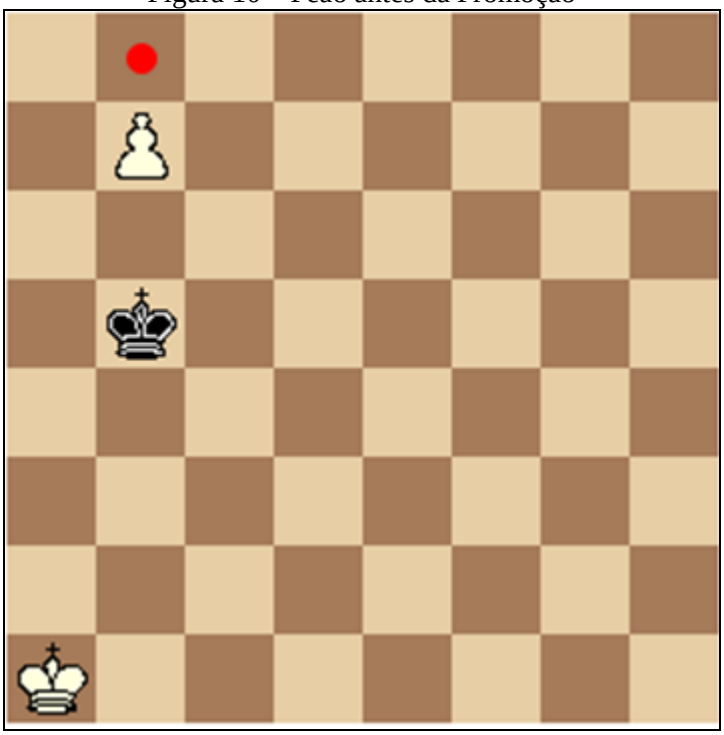
2.1.3.2 Captura de Peças

A captura das peças ocorre sempre no sentido do movimento da peça, com exceção do peão, que se move para frente e captura em diagonal. (XADREZ REGIONAL, 2012). Portanto, sendo de cor oposta e estando no raio de ação da peça, esta pode capturar qualquer outra.

2.1.3.3 Promoção

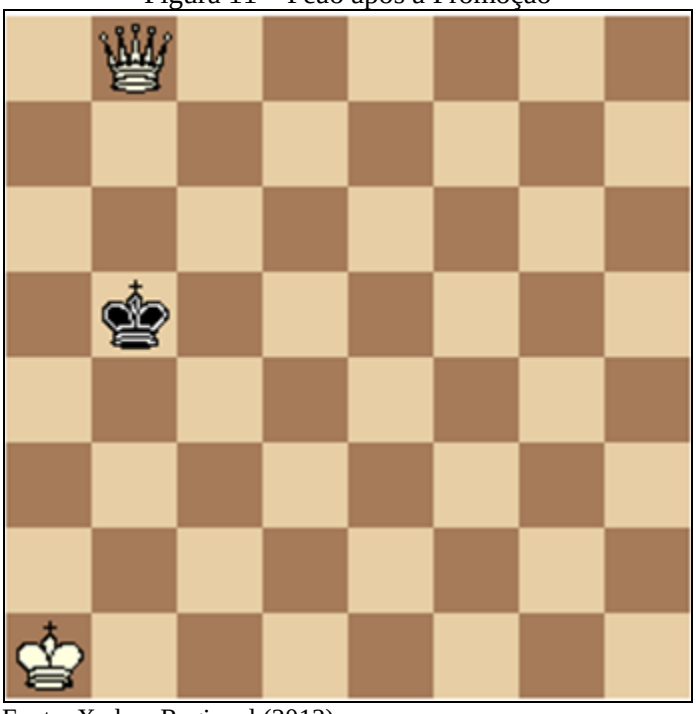
A promoção ocorre com o Peão quando este chega à última fileira, podendo transformar-se em uma Torre, Cavalo, Bispo ou Rainha (XADREZ REGIONAL, 2012). Nas Figuras 10 e 11 têm-se um exemplo de Promoção.

Figura 10 – Peão antes da Promoção



Fonte: Xadrez Regional (2012).

Figura 11 – Peão após a Promoção



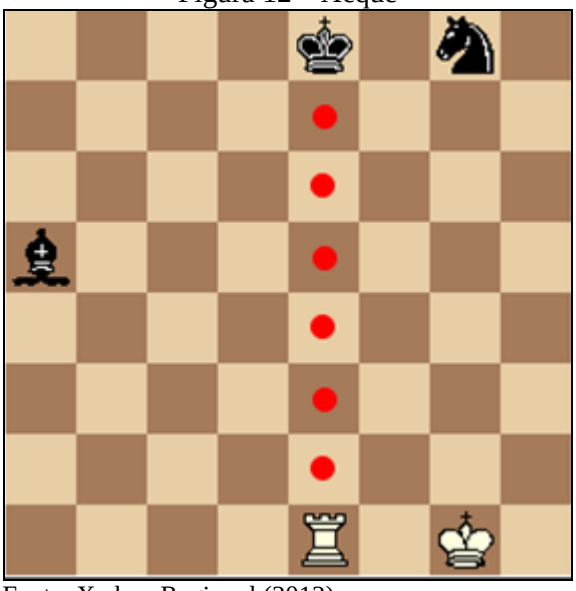
Fonte: Xadrez Regional (2012).

.

2.1.3.4 Xeque e Xeque-mate

Xeque é quando o Rei está sob ataque de alguma peça adversária. Na Figura 12 tem-se um exemplo de Xeque, onde a Torre branca está atacando o Rei preto (XADREZ REGIONAL, 2012).

Figura 12 – Xeque

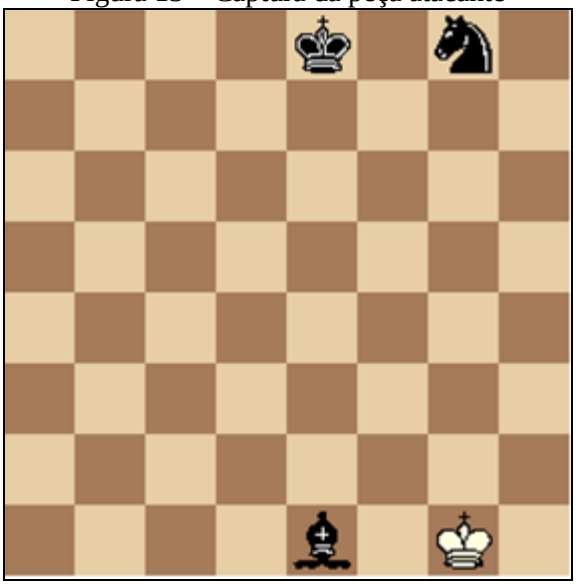


Fonte: Xadrez Regional (2012).

Para sair do xeque, há três possibilidades:

a) capturar a peça que está atacando o Rei conforme tem-se na Figura 13;

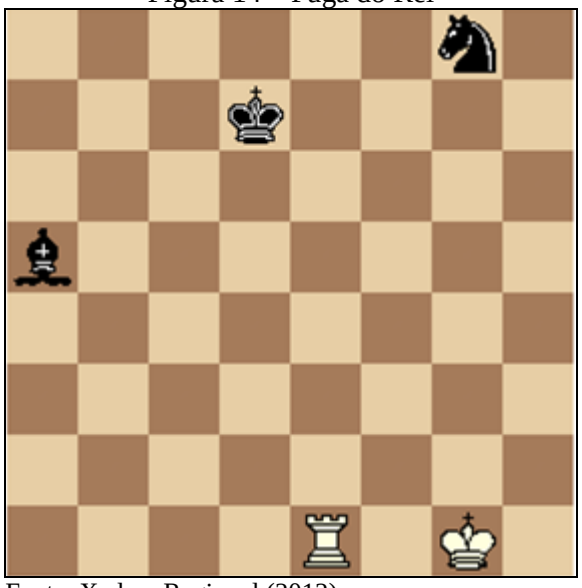
Figura 13 – Captura da peça atacante



Fonte: Xadrez Regional (2012).

b) mover o Rei para alguma casa que não esteja atacada conforme tem-se na Figura 14;

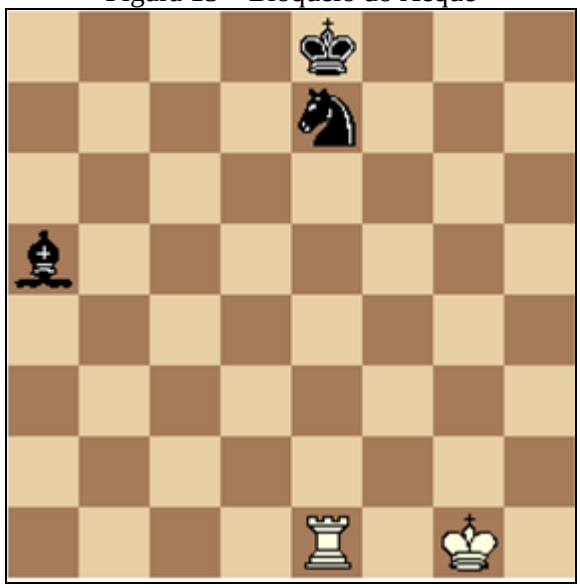
Figura 14 – Fuga do Rei



Fonte: Xadrez Regional (2012).

c) bloquear o xeque, colocando alguma peça entre a peça atacante e o Rei conforme tem-se na Figura 15.

Figura 15 – Bloqueio do Xeque



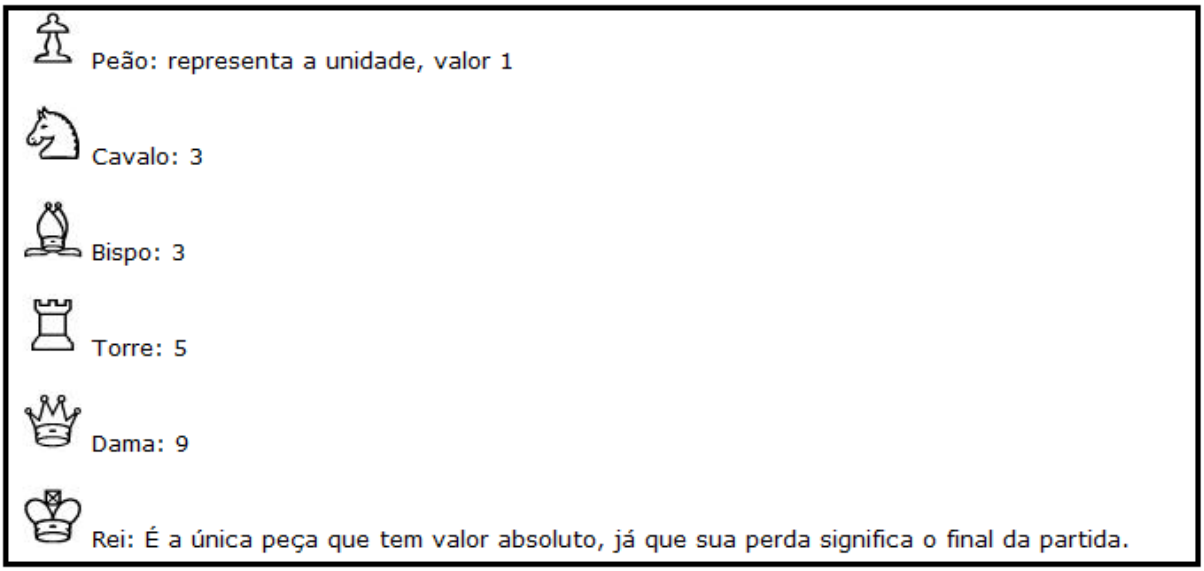
Fonte: Xadrez Regional (2012).

Caso nenhuma dessas três possibilidades seja possível, então o Rei está em Xeque-mate, decretando assim sua derrota (XADREZ REGIONAL, 2012).

2.1.4 Pontuação das Peças

O valor das peças no xadrez é relativo, dependendo da sua posição no tabuleiro, sua mobilidade, entre outros (CLUBE DE XADREZ, 2012). Porém, para servir como base para jogadores iniciantes, foram estipulados valores para cada peça, conforme mostrado na Figura 16.

Figura 16 – Valores das Peças



Fonte: Clube de Xadrez (2012).

2.1.5 Anotação da Partida

As peças são referenciadas pela sua inicial maiúscula, com exceção dos peões, que não recebem sua inicial na anotação do lance. O formato de cada lance é a inicial da peça seguida da casa destino desta, por exemplo, Ce4 indica que o Cavalo foi para a casa e4. O roque pequeno é O-O e o roque grande é O-O-O.

Na Figura 17 tem-se um exemplo de uma partida anotada.

Figura 17 – Anotação de uma partida de Xadrez

Karpov,Anatoly 2705 – **Kasparov,Garry** 2740

1.d4 ♘f6 2.c4 g6 3.♘c3 d5 4.♗f4 ♗g7 5.e3 c5 6.dxc5 ♕a5 7.♖c1 ♘e4 8.cxd5 ♘xc3 9.♕d2 ♕xa2 10.bxc3 ♕xd2+ 11.♔xd2 ♘d7 12.♗b5 O-O 13.♗xd7 ♗xd7 14.e4 f5 15.e5 e6 16.c4 ♖fc8 17.c6 bxc6 18.d6 c5 19.h4 h6 20.♘h3 a5 21.f3 a4 22.♖he1 a3 23.♘f2 a2 24.♘d3 ♖a3 25.♖a1 g5 26.hxg5 hxg5 27.♗xg5 ♔f7 28.♗f4 ♖b8 29.♖ec1 ♗c6 30.♖c3 ♖a5 31.♖c2 ♖ba8 32.♘c1 1-0

Fonte: Chessbase (2014).

2.1.6 Fide

A entidade reguladora máxima do xadrez mundial é a FIDE, fundada em 1924 em Paris. Ela é responsável pela confecção de regras e regulamentos com o objetivo de padronizar o formato das competições ao redor do mundo. Para determinar o nível dos jogadores, a FIDE criou uma pontuação chamada de *rating* FIDE, e esta pontuação varia conforme o jogador vence ou perde as partidas em competições oficiais. Atualmente, o melhor jogador do mundo é o norueguês Magnus Carlsen, com 2853 pontos de *rating*

FIDE, seguido pelo indiano Viswanathan Anand com 2816 pontos. A classificação de julho de 2015 é mostrada na Figura 18 (FIDE, 2015).

Figura 18 – Top 20 jogadores de xadrez

Rank	Name	Title	Country	Rating	Games	B-Year
1	Carlsen, Magnus	g	NOR	2853	9	1990
2	Anand, Viswanathan	g	IND	2816	9	1969
3	Topalov, Veselin	g	BUL	2816	9	1975
4	Nakamura, Hikaru	g	USA	2814	9	1987
5	Caruana, Fabiano	g	USA	2797	9	1992
6	Giri, Anish	g	NED	2791	19	1994
7	Kramnik, Vladimir	g	RUS	2783	0	1975
8	So, Wesley	g	USA	2780	15	1993
9	Grischuk, Alexander	g	RUS	2771	9	1983
10	Aronian, Levon	g	ARM	2765	9	1982
11	Jakovenko, Dmitry	g	RUS	2757	4	1983
12	Karjakin, Sergey	g	RUS	2753	0	1990
13	Gelfand, Boris	g	ISR	2751	0	1968
14	Ding, Liren	g	CHN	2749	3	1992
15	Li, Chao b	g	CHN	2748	0	1989
16	Tomashevsky, Evgeny	g	RUS	2745	0	1987
17	Adams, Michael	g	ENG	2740	0	1971
18	Svidler, Peter	g	RUS	2740	0	1976
19	Radjabov, Teimour	g	AZE	2738	0	1987
20	Yu, Yangyi	g	CHN	2736	13	1994

Fonte: FIDE (2015).

2.2 ENSINO E APRENDIZADO DO JOGO

Há diversas metodologias voltadas ao ensino do jogo de xadrez. Nesta seção serão comentadas três metodologias, com destaque para a de Goulart e Frei (2004), a qual será utilizada na implementação da ferramenta e possuirá uma seção própria.

Gomes (2010) propôs uma metodologia onde afirma que o conhecimento do xadrez é um elemento complementar no contexto educacional, pois atua como suporte no aprendizado de disciplinas curriculares. Ainda segundo o autor, isto ajuda a melhorar a capacidade de concentração e a velocidade de raciocínio dos aprendizes. Sua metodologia divide-se nos seguintes tópicos:

a) aulas expositivas e dialogadas;
b) exercícios escritos;
c) torneios de treinamento entre as turmas;
d) torneios *online* entre instituições;
e) tirar dúvidas em cada mesa e máquina;
f) pesquisas e trabalhos individuais em grupos;
g) filmes (sessões cinematográficas);
h) oficinas de capacitação para a comunidade escolar.

Avaliações também estão presentes nessa metodologia. Com base na participação e desempenho dos alunos, incluindo uma prova de múltipla escolha, os professores podem avaliar a evolução dos alunos no decorrer do ano letivo (GOMES, 2010).

Outra metodologia proposta consiste em introduzir inicialmente ao aluno o nome das peças e o tabuleiro, não sendo necessário abordar a história do jogo. Em seguida, ensinar como as peças se movimentam, podendo, para isso, organizar os aprendizes em grupos para que estes possam se ajudar e assimilar o conhecimento. Posteriormente, o aluno deve aprender o xeque-mate e outras jogadas especiais do xadrez. Neste ponto, o aprendiz encontra-se apto para conhecer o esporte na prática e jogar uma partida (ASSIM SE FAZ, 2012).

2.2.1 Metodologia de Goulart e Frei

Goulart e Frei (2014) destacam o quão importante é uma primeira narrativa da história do xadrez e sua importância no mundo, aguçando a curiosidade dos alunos e motivando-os a pesquisarem sobre o esporte. Técnicas de várias fontes de mídia contribuem, como filmes, figuras e vídeos sobre xadrez. Afirmam também que o xadrez é um alvo rico de estudo, pois oferece diversos tipos de jogadas que poderiam apenas ser exploradas por meio da prática. A arte do jogo está na sua originalidade imaginativa.

Para Goulart e Frei (2004), o aprendizado deve iniciar com a introdução de um pouco da história do jogo e os conceitos que este jogo trabalha e desenvolve. Em seguida, apresentar ao aluno o tabuleiro, a nomenclatura, a movimentação e a pontuação das peças, questionando-as frequentemente para que seja bem assimilado pelo aprendiz. É importante também mostrar os movimentos especiais do xadrez, como o roque e o xeque-mate, para só então permitir aos alunos que joguem e aprendam na prática.

A partir da metodologia apresentada, o ensino de xadrez é facilitado, pois oferece diversas formas de abordar o jogo. Estes instrumentos pedagógicos possibilitam ensinar o xadrez de forma prazerosa, tornando-o uma atividade cotidiana, trazendo benefícios para o desenvolvimento intelectual e psicológico dos alunos (Goulart e Frei, 2004).

2.3 INTELIGÊNCIA ARTIFICIAL

John McCarthy criou a nomenclatura IA durante o *Workshop* do Dartmouth College em 1956, sendo este o primeiro encontro de cientistas oficialmente organizado para discutir aspectos de inteligência e sua implementação em máquinas. Até então havia um grande entusiasmo devido a experimentos que estavam tendo bons resultados, com

destaque para o *General Problem Solver* (GPS), criado por Allen Newell e Herbert Simon para simular os métodos humanos de resolução de problemas (NAVEGA, 2000).

O significado de IA abrange muitos aspectos. Estas podem ser vistas como sistemas que pensam como humanos, sistemas que agem como humanos, sistemas que pensam racionalmente e sistemas que agem racionalmente (RUSSEL; NORVIG, 1995). Apesar de ser uma área do ramo computacional, encontra-se presente em diversas áreas de atuação, tais como medicinal, diagnósticos, jogos, entre outros.

2.3.1 Aplicação da IA no Xadrez

Os primeiros passos na criação de máquinas que jogassem xadrez começaram no ano de 1890, mas apenas a partir dos anos de 1950, com o surgimento da IA, foi que as primeiras *engines* mais competitivas começaram a existir (SOCIEDADE DOS MESTRES DE XADREZ, 2012). Acompanhando a ascensão da IA, novas *engines* foram criadas e constantemente melhoradas desde então a partir de algoritmos, técnicas e heurísticas criadas, sendo várias dessas *engines* presentes nos dias atuais, como o Chessmaster e o Fritz. Dentro os algoritmos citados utilizados para fortalecer as *engines*, dois bastante utilizados são o Minimax e Poda Alfa-Beta. As técnicas utilizadas são variadas, como implementação de conceitos de xadrez, árvores de abertura e base de dados de partidas para a *engine* basear-se.

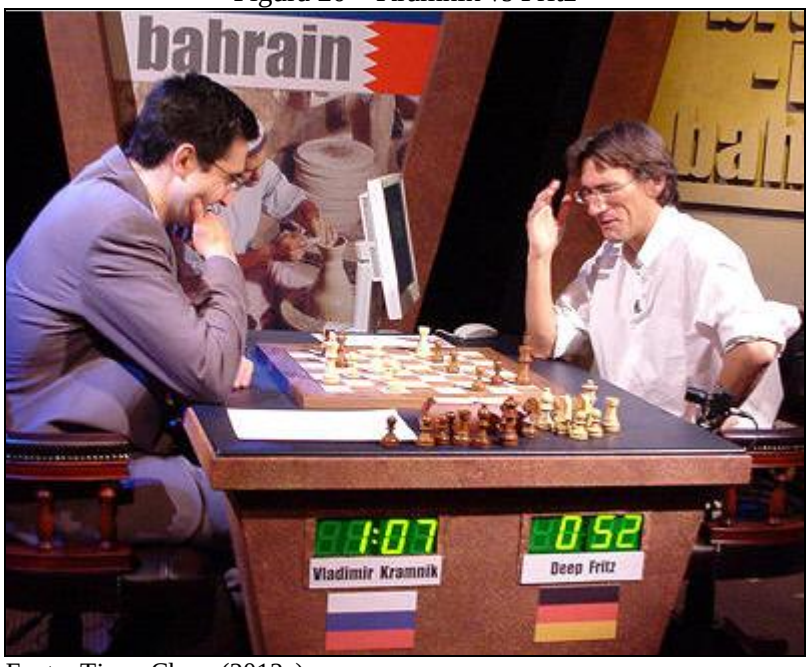
Porém, o estopim das máquinas no mundo de xadrez foi em 1997, onde a máquina desenvolvida pela IBM denominada Deep Blue derrotou o campeão mundial, Garry Kasparov (IBM, 2014). Este confronto marcou a primeira derrota imposta por uma máquina a um campeão mundial. A Figura 19 mostra o campeão Kasparov jogando contra a máquina, que possuía um assistente para reproduzir o lance feito pela máquina no tabuleiro.

Figura 19 – Kasparov vs Deep Blue

Fonte: FIDE (2015).

Influenciados com o feito do Deep Blue, outras *engines* começaram a ser desenvolvidas pelos pesquisadores. Em 2002, ocorreu outra disputa entre o campeão mundial Vladimir Kramnik contra a *engine* Fritz. Esse desafio acabou empatado. Em 2006 houve um novo confronto, mas desta vez Fritz venceu por 4-2 (TIPPY CHESS, 2012c).

Figura 20 – Kramnik vs Fritz



Fonte: Tippy Chess (2012c).

Em 2007, a *engine* Rybka se mostrou superior a qualquer jogador humano, inclusive rodando em computadores pessoais (SOCIEDADE DOS MESTRE DE

XADREZ, 2012). Eventos de xadrez homem *versus* máquina não atrai mais investidores, visto que o nível de jogo das máquinas já ultrapassou o dos humanos. Portanto, o foco atual passa a ser utilizar a força da máquina para evoluir com ela e melhorar o nível dos jogadores, permitindo realizar análises profundas de posições específicas e descobrir jogadas e combinações que não são percebidas com facilidade aos olhos humanos (MADEIROS; QUARANTA, 2013).

2.3.2 Algoritmo Minimax

O algoritmo Minimax é um procedimento recursivo direto que se baseia em dois procedimentos auxiliares, específicos do jogo, que são o gerador de movimentos (implementação da movimentação das peças) e a avaliação estática (pontuação das peças) (Tippy Chess, 2012b). Através deste algoritmo, o computador analisa a melhor ação a tomar, construindo uma árvore com as ações possíveis para cada jogador e todas as ações seguintes. Assim, um dos jogadores deverá obter o valor máximo e outro o valor mínimo, e a todo nó da árvore gerada será atribuído o valor do nó filho com o valor MIN ou MAX obtido, fazendo isso na árvore até chegar à raiz (FERREIRA; OLIVEIRA, 2007). Visto que o xadrez é um jogo de dois jogadores que jogam alternadamente, a implementação deste algoritmo torna-se bastante eficiente e propício dentro do cenário.

O algoritmo necessita de dois parâmetros: a posição do tabuleiro e a atual profundidade de busca. No Quadro 1 tem-se o pseudocódigo do algoritmo Minimax.

Quadro 1 - Pseudocódigo algoritmo MiniMax

```
ROTINA minimax(nó, profundidade)
    SE nó é um nó terminal OU profundidade = 0 ENTÃO
        RETORNE o valor da heurística do nó
    SENÃO SE o nó representa a jogada de algum adversário ENTÃO
        α ← +∞
        PARA CADA filho DE nó
            α ← min(α, minimax(filho, profundidade-1))
        FIM PARA
        RETORNE α
    SENÃO
        α ← -∞
        PARA CADA filho DE nó
            α ← max(α, minimax(filho, profundidade-1))
        FIM PARA
        RETORNE α
    FIM SE
FIM ROTINA
```

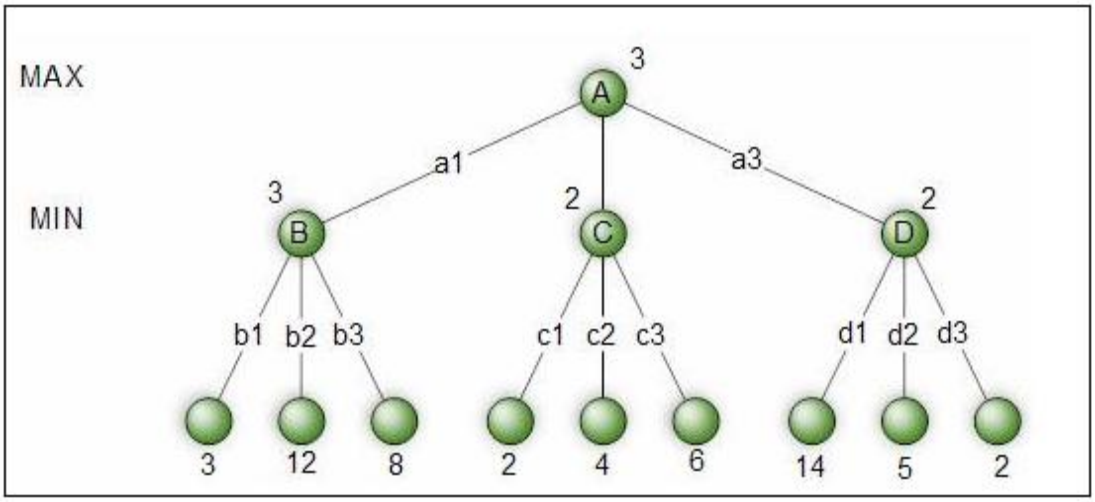
Fonte: Wikipedia (2014).

Para limitar o tempo de processamento, podem ser utilizadas diversas técnicas, como limitar o número por nós gerado, por tempo ou até que se encontre um lance razoável (que pode não ser o melhor); dado o limite de tempo do jogo, a técnica de limitar o processamento por tempo é o mais indicado (Tippy Chess, 2012b). Portanto, dependendo

da profundidade da árvore e do tempo estipulado para esta análise, torna-se inviável percorrê-la por completo a fim de encontrar a melhor ação a tomar.

Na Figura 21, é mostrada uma ilustração exemplificando o algoritmo de MiniMax, onde tem-se a1, a2 e a3 como sendo os movimentos possíveis para MAX no nó raiz. As respostas possíveis para a1 correspondentes a MIN são b1, b2 e b3 e assim sucessivamente. O primeiro nó de MIN, identificado por B, possui três sucessores com valores 3, 12 e 8. Logo, seu valor minimax é 3. De modo semelhante, os outros dois nós de MIN possuem valor minimax 2. O nó raiz é um nó de MAX e seus sucessores possuem valores minimax 3, 2 e 2. Logo, seu valor minimax é 3. Assim, pode-se identificar que a decisão ótima para MAX na raiz é a ação a1 porque leva ao sucessor com o mais alto valor minimax (TippyChess, 2012b).

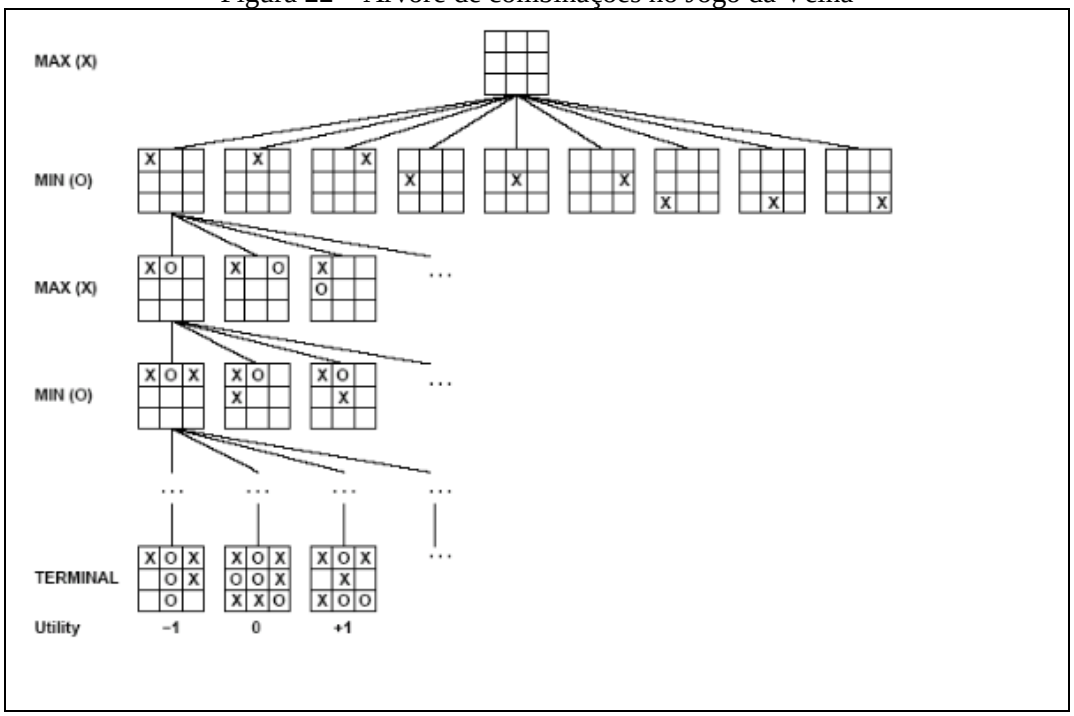
Figura 21 – Exemplo algoritmo Minimax



Fonte: TippyChess (2012b).

Outra ilustração do algoritmo pode ser vista no jogo Tic-Tac-Toe, popular Jogo da Velha. Por ser um jogo de combinações finitas, pode-se encontrar a melhor jogada sem dificuldades. Na Figura 22 tem-se um exemplo das árvores de combinações possíveis no Jogo da Velha.

Figura 22 – Árvore de combinações no Jogo da Velha



Fonte: Jeiks (2011).

2.3.3 Algoritmo Poda Alfa-Beta

O algoritmo Poda Alfa-Beta é uma alternativa para otimizar o algoritmo Minimax. Este algoritmo chega à mesma resposta que o Minimax, com a diferença de que não precisa examinar todos os nós na árvore de jogo. Isto ocorre porque o algoritmo armazena o valor das situações anteriores e não analisa uma ramificação da árvore que seja pior do que uma já previamente analisada, cortando todo este ramo e otimizando a consulta (Tippy Chess, 2012a). Em outras palavras, o algoritmo Poda Alfa-Beta, ao ser aplicado em uma árvore Minimax, retorna o mesmo movimento que o Minimax retornaria, mas poda as ramificações que não terão influência possível sobre a decisão final (Quinet, 2011). No Quadro 2 tem-se sua implementação.

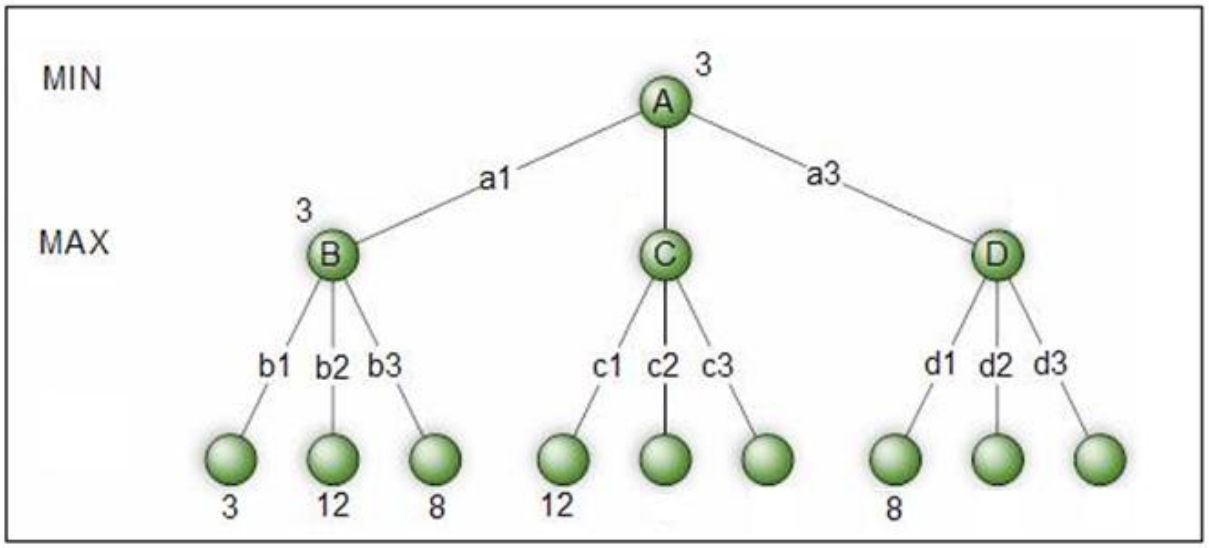
Quadro 2 - Pseudocódigo Poda Alfa-Beta

```
Se:
É no MAX: Se valor_alfa_de(nó) >= valor_beta_ancestral; ou
É no MIN: Se valor_beta_de(nó) <= valor_alfa_de_ancestral; ENTÃO
corta_busca_abaixo(nó)
```

Fonte: Jeiks (2011).

Em outras palavras, caso o nó atual do algoritmo for um nó de Max e algum filho seu tiver um valor maior que algum nó do mesmo nível previamente analisado, todo este ramo pode ser desconsiderado. Na Figura 23 tem-se um exemplo desse conceito.

Figura 23 – Corte Poda Alfa-Beta nó de Max

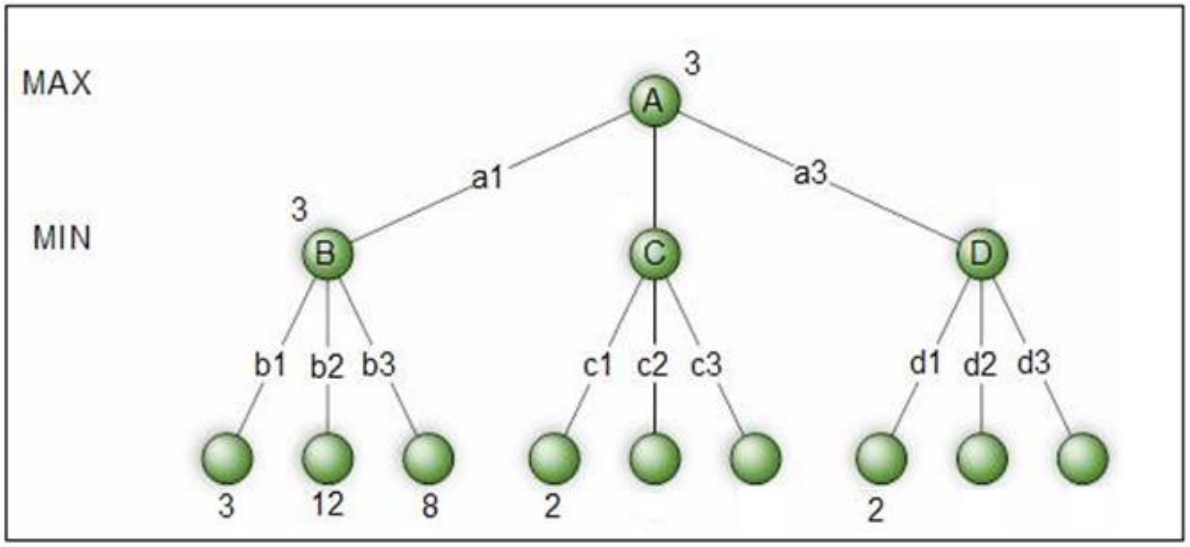


Fonte: adaptado de TippyChess (2012a).

O nó c1 possui valor de 12. Logo, considerando que o nó C é de Max, o valor de C será no mínimo 12, independente do valor de c2 e c3. Pode-se concluir que o nó C nunca será acessado por A, pois este é um nó de Min e há ao menos um nó menor que 12 encontrado (no caso, B). O mesmo ocorre para o nó D. Logo, o algoritmo Poda Alfa-Beta podou dois ramos da árvore, otimizando a execução do MiniMax.

De forma inversa, se o nó atual for um nó de Min e algum filho seu tiver um valor menor que algum nó do mesmo nível previamente analisado, todo este ramo pode ser desconsiderado. Na Figura 24 tem-se um exemplo prático.

Figura 24 – Corte Poda Alfa-Beta nó de Min



Fonte: adaptado de TippyChess (2012a).

O nó c1 possui valor de 2. Logo, considerando que o nó C é de Min, o valor de C será no máximo 2, independente do valor de c2 e c3. Pode-se concluir que o nó C nunca será acessado por A, pois este é um nó de Max e há ao menos um nó maior que 3

encontrado (no caso, B). O mesmo ocorre para o nó D. Logo, o algoritmo Poda Alfa-Beta podou dois ramos da árvore, otimizando a execução do MiniMax.

2.4 TRABALHOS CORRELATOS

Existem vários softwares voltados ao ensino de xadrez. Alguns deles são o Chessmaster (UBISOFT, 2014), Chessimo (CHESSIMO, 2007) e Fritz 2014 (CHESSBASE, 2014).

2.4.1 Chessmaster

O software desenvolvido pela Ubisoft (2014) permite que o usuário jogue contra vários tipos de oponentes (*engines*), que são classificados com uma pontuação numérica dependendo do seu nível. Quanto maior a pontuação atribuída a *engine*, mais profunda é a árvore de movimentos gerada. As *engines* deste software utilizam os algoritmos Minimax e Poda Alfa-Beta, assim como heurísticas para auxiliar na procura do melhor movimento possível. Os usuários também possuem esta pontuação, que é atribuída ou decrementada dependendo do resultado final da partida. É possível também assistir a várias aulas lecionadas pelo anfitrião ChessMaster (a *engine* mais forte do jogo), permitindo que usuários aprendam vários conceitos importantes do xadrez, tais como estratégia, tática, finais, aberturas, etc.

O sistema possui várias versões e está disponível nas mais variadas plataformas, que incluem Nintendo DS, Playstation, Computador, Xbox Live, PlayStation Portable (PSP). Na Figura 25, vê-se uma partida prestes a iniciar.

Figura 25 – Tabuleiro inicial no software Chessmaster

Fonte: UBISOFT (2014).

Outras qualidades do software incluem disponibilizar livros acessíveis para todos os usuários, podendo também ser organizado torneios entre jogadores e *engines*. Por ser um software não gratuito, é constantemente aprimorado e adaptado, possuindo poucos pontos fracos. Porém, algo que poderia ser melhorado é a compatibilidade entre o software e *engines* externas, como por exemplo, Fritz (CHESSBASE, 2014), Houdini (HOUDINI, 2013) e StockFish (STOCKFISH, 2014).

2.4.2 Chessimo

Em parceria com o jogador brasileiro de xadrez Gilberto Milos, a empresa Chessimo (2007) desenvolveu um software que auxilia o jogador na sua evolução enxadrística, independente do seu nível. O foco do sistema é fornecer milhares de diagramas e exercícios, separados por assunto e tema, repetindo-os de forma que obrigue o aluno a resolvê-los diversas vezes, fixando a solução para aplicar em situações semelhantes.

O sistema está disponível para as plataformas iOS, Android e PC, sendo um software que deve ser paga (com exceção de aparelhos da Apple) uma licença para poder utilizá-lo. Um aspecto ausente no programa é a possibilidade de jogar contra a máquina (não há *engine* implementada neste software) ou contra outro jogador. Na Figura 26 tem-se a interface gráfica utilizada no sistema.

Figura 26 – Tela de exercícios do Chessimo

Fonte: Chessimo (2007).

2.4.3 Fritz 2014

O software Fritz, autoria da empresa ChessBase (2014), teve sua primeira versão lançada no ano de 1995, e sua versão mais recente foi lançada no ano de 2014, não sendo um software gratuito. Sendo uma das *engines* mais fortes já criadas, é utilizada por jogadores de todos os níveis com o intuito de melhorar seu jogo com base nas análises da *engine*. A *engine* deste software utiliza diversos algoritmos, tais como Minimax e Poda Alfa-Beta, assim como heurísticas para auxiliar na procura do melhor movimento possível, justificando o seu posto de uma das *engines* mais fortes existentes.

A Figura 27 mostra a tela inicial do software, onde se pode observar vários detalhes. Alguns deles são o tabuleiro com as peças na posição inicial, o relógio que marca o tempo que as brancas e as pretas têm para finalizar a partida e o espaço destinado a anotar os lances efetuados na partida (que não possui nenhum lance anotado pois a partida ainda não iniciou).

Figura 27 – Tela inicial do Fritz

Fonte: ChessBase (2014).

Ao contrário do Chessimo, o Fritz não possui opções para resolução de exercícios e aprendizado de temas específicos de xadrez. Seu foco se baseia no poder de processamento e de cálculo, permitindo aos jogadores analisarem partidas antigas e ver quais lances a máquina faria, aprendendo assim novas ideias. É importante ressaltar que o software permite que se jogue contra ele regulando-o em diversos níveis, possibilitando inclusive que jogadores iniciantes possam enfrentá-lo de igual para igual.

3 DESENVOLVIMENTO

Neste capítulo são detalhadas as etapas do desenvolvimento do trabalho. Na seção 3.1 são apresentados os principais requisitos. A seção 3.2 apresenta a especificação. A seção 3.3 apresenta a operacionalidade da ferramenta por parte do usuário. A seção 3.4 mostra de forma detalhada os trechos de código mais relevantes do trabalho e as ferramentas utilizadas. A seção 3.5 apresenta os resultados obtidos, discussões e sugestões de melhorias.

3.1 REQUISITOS PRINCIPAIS DA FERRAMENTA A SER TRABALHADO

Durante a análise foram especificados os seguintes requisitos para o desenvolvimento da ferramenta proposta:

a) permitir o cadastro de usuários (Requisito Funcional – RF);
b) permitir que cada usuário possua um *rating* próprio (RF);
c) disponibilizar uma seção para que os usuários possam ver o cronograma de estudo (RF);
d) disponibilizar uma seção para que os usuários possam aprender a história e conceitos do xadrez (RF);
e) disponibilizar uma seção para apresentar o tabuleiro e as peças com seus respectivos valores (RF);
f) disponibilizar uma seção que mostre aos usuários os movimentos especiais do xadrez (RF);
g) disponibilizar questionários para o usuário testar seu conhecimento (RF);
h) permitir que os usuários joguem contra uma *engine* (RF);
i) atribuir ou decrementar pontos de *rating* de um usuário após o término da partida dependendo de seu resultado (RF);
j) ser implementada na linguagem de programação Java (Requisito Não Funcional – RNF);
k) utilizar o banco de dados Oracle (RNF);
l) ser desenvolvida no ambiente Eclipse (RNF);
m) possuir o Adobe Reader (RNF).

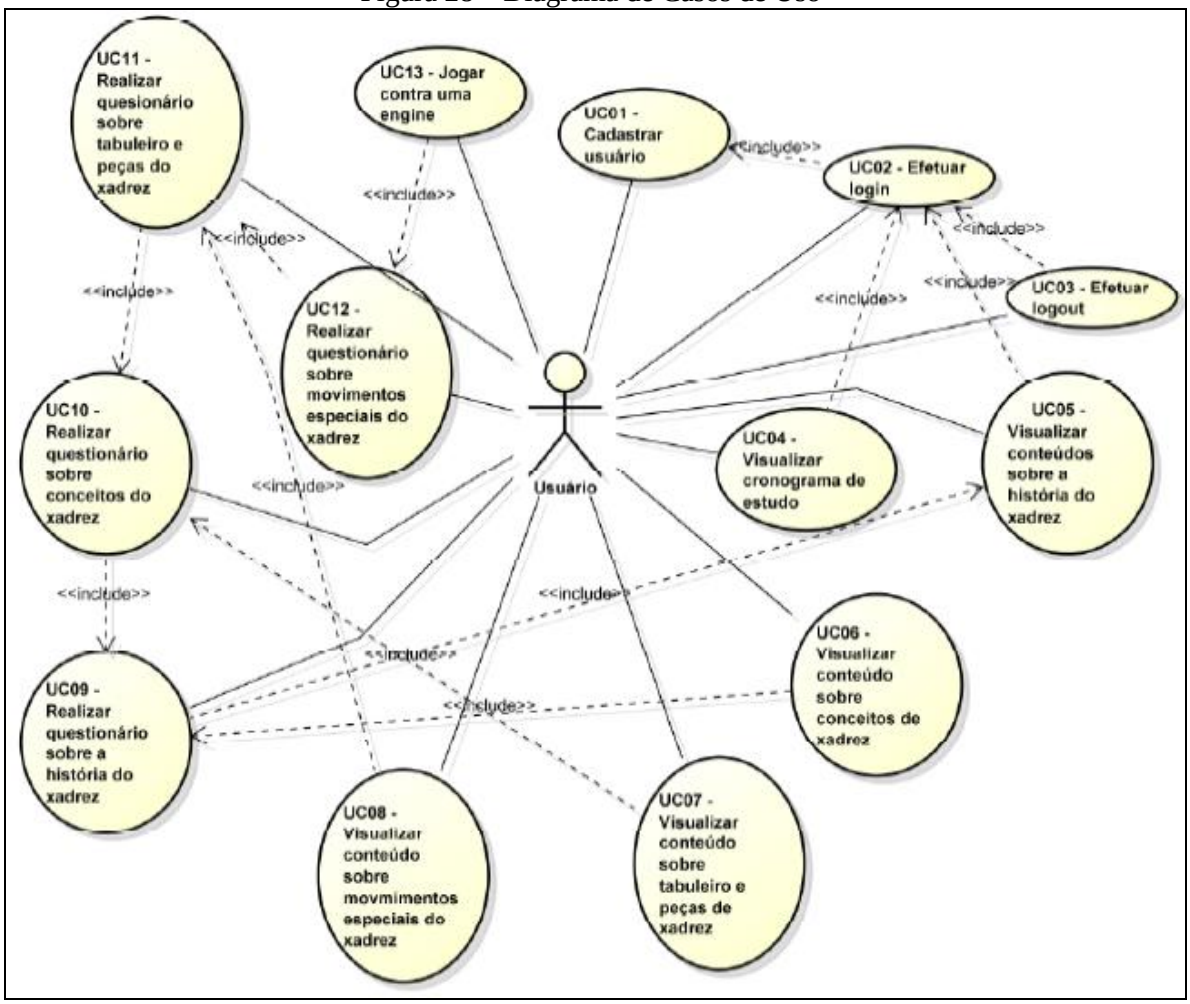
3.2 ESPECIFICAÇÃO

A especificação da ferramenta foi realizada com base em alguns diagramas da UML, modelada através das ferramentas Enterprise Architect e Astah UML. Neste trabalho foram elaborados os diagramas de casos de uso, de pacotes e de classes.

3.2.1 Diagrama de Casos de Uso

Nesta seção são apresentados, através da Figura 28, os casos de uso da ferramenta. Identificou-se somente um ator, denominado usuário, o qual utiliza todas as funcionalidades da ferramenta.

Figura 28 – Diagrama de Casos de Uso



O caso de uso `UC01 - Cadastrar usuário` descreve como o ator Usuário efetua o cadastro na ferramenta. Detalhes deste caso de uso no Quadro 3.

Quadro 3 – Caso de uso UC01

Caso de uso	UC01 - Cadastrar usuário
Ator	Usuário.
Objetivo	Efetuar o cadastro de um usuário na ferramenta.
Pré-condições	Possuir máquina virtual Java instalada e possuir a ferramenta instalada.
Pós-condições	Usuário cadastrado no sistema.
Cenário principal	1. Usuário informa ID, senha, nome, sobrenome, nascimento e sexo; 2. Ferramenta faz o cadastro no banco de dados.
Cenário alternativo	Caso o usuário não informe os campos do item 1, a ferramenta emitirá uma mensagem alertando que há campos que devem ser preenchidos.
Exceção	1. Caso o formato da data digitada não siga o padrão dd/mm/aaaa a ferramenta mostrará uma mensagem alertando que o formato da data está inválido. 2. Caso haja outro usuário com a mesma ID cadastrada, a ferramenta emitirá uma mensagem alertando usuário existente.
Requisitos Funcionais relacionados	1. Permitir o cadastro de usuários (Requisito Funcional – RF).

O caso de uso `UC02 - Efetuar login` descreve como o ator Usuário efetua o *login* na ferramenta. Detalhes deste caso de uso no Quadro 4.

Quadro 4 – Caso de uso UC02

Caso de uso	UC02 - Efetuar *login*
Ator	Usuário.
Objetivo	Efetuar o cadastro de um usuário na ferramenta.
Pré-condições	Possuir cadastro.
Pós-condições	Acesso a funções do sistema.
Cenário principal	1. Usuário informa *login* e senha; 2. Ferramenta verifica no banco de dados se o *login* é válido.
Cenário alternativo	1. Caso o usuário não informe os campos do item 1, a ferramenta emitirá uma mensagem alertando que há campos que devem ser preenchidos.
Exceção	1. Caso o *login* e senha não constem no banco de dados, a ferramenta emitirá uma mensagem informando *login* inválido.
Requisitos Funcionais relacionados	1. Permitir o cadastro de usuários (Requisito Funcional – RF).

O caso de uso `UC03 - Efetuar logout` descreve como o ator Usuário efetua o *logout* na ferramenta. Detalhes deste caso de uso no Quadro 5.

Quadro 5 – Caso de uso UC03

Caso de uso	UC03 - Efetuar *logout*
Ator	Usuário.
Objetivo	Efetuar *logout* na ferramenta.
Pré-condições	Estar logado na ferramenta.
Pós-condições	Fechamento da ferramenta.
Cenário principal	1. Usuário acessa o menu Usuário e clica no submenu Deslogar.
Requisitos Funcionais relacionados	1. Permitir o cadastro de usuários (Requisito Funcional – RF).

O caso de uso `UC04 - Visualizar cronograma de estudo` descreve como o ator Usuário visualiza o cronograma de estudo na ferramenta. Detalhes deste caso de uso no Quadro 6.

Quadro 6 – Caso de uso UC04

Caso de uso	UC04 - Visualizar cronograma de estudo
Ator	Usuário.
Objetivo	Visualizar cronograma que servirá de base de estudo para o usuário
Pré-condições	Estar logado na ferramenta.
Pós-condições	Usuário ciente do cronograma de estudo.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Cronograma.
Requisitos Funcionais relacionados	1. Disponibilizar uma seção para que os usuários possam ver o cronograma de estudo (RF).

O caso de uso `UC05 - Visualizar conteúdo sobre a história do xadrez` descreve como o ator Usuário visualiza o conteúdo de estudo que aborda a história do xadrez. Detalhes deste caso de uso no Quadro 7.

Quadro 7 – Caso de uso UC05

Caso de uso	UC05 - Visualizar conteúdo sobre a história do xadrez
Ator	Usuário.
Objetivo	Aprender o conteúdo proposto.
Pré-condições	Estar logado na ferramenta.
Pós-condições	Usuário apto para realizar o questionário sobre história do xadrez.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário clica no botão Conteúdo.
Requisitos Funcionais relacionados	1. Disponibilizar uma seção para que os usuários possam aprender a história e conceitos do xadrez (RF).

O caso de uso `UC06 - Visualizar conteúdo sobre conceitos do xadrez` descreve como o ator Usuário visualiza o conteúdo de estudo que aborda os conceitos trabalhados e desenvolvidos no xadrez. Detalhes deste caso de uso no Quadro 8.

Quadro 8 – Caso de uso UC06

Caso de uso	UC06 - Visualizar conteúdo sobre conceitos do xadrez
Ator	Usuário.
Objetivo	Aprender o conteúdo proposto.
Pré-condições	Estar logado na ferramenta e ter concluído o questionário sobre a história do xadrez.
Pós-condições	Usuário apto para realizar o questionário sobre conceitos do xadrez.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário clica na aba Conceitos; 3. Usuário clica no botão Conteúdo.
Requisitos Funcionais relacionados	1. Disponibilizar uma seção para que os usuários possam aprender a história e conceitos do xadrez (RF).

O caso de uso `UC07 - Visualizar conteúdo sobre tabuleiro e peças do xadrez` descreve como o ator Usuário visualiza o conteúdo de estudo que aborda as peças e o tabuleiro do xadrez. Detalhes deste caso de uso no Quadro 9.

Quadro 9 – Caso de uso UC07

Caso de uso	UC07 - Visualizar conteúdo sobre tabuleiro e peças do xadrez
Ator	Usuário.
Objetivo	Aprender o conteúdo proposto.
Pré-condições	Estar logado na ferramenta e ter concluído o questionário sobre conceitos do xadrez.
Pós-condições	Usuário apto para realizar o questionário sobre tabuleiro e peças do xadrez.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário clica na aba Tabuleiro e Peças; 3. Usuário clica no botão Conteúdo.
Requisitos Funcionais relacionados	1. Disponibilizar uma seção para apresentar o tabuleiro e as peças com seus respectivos valores (RF).

O caso de uso `UC08 - Visualizar conteúdo sobre movimentos especiais do xadrez` descreve como o ator Usuário visualiza o conteúdo de estudo que aborda os movimentos especiais do xadrez. Detalhes deste caso de uso no Quadro 10.

Quadro 10 – Caso de uso UC08

Caso de uso	UC08 - Visualizar conteúdo sobre movimentos especiais do xadrez
Ator	Usuário.
Objetivo	Aprender o conteúdo proposto.
Pré-condições	Estar logado na ferramenta e ter concluído o questionário sobre tabuleiro e peças do xadrez.
Pós-condições	Usuário apto para realizar o questionário sobre movimentos especiais do xadrez.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário clica na aba Movimentos Especiais; 3. Usuário clica no botão Conteúdo.
Requisitos Funcionais relacionados	1. Disponibilizar uma seção que mostre aos usuários os movimentos especiais do xadrez (RF).

O caso de uso `UC09 - Realizar questionário sobre a história do xadrez` descreve como o ator Usuário efetua o questionário sobre a história do xadrez. Detalhes deste caso de uso no Quadro 11.

Quadro 11 – Caso de uso UC09

Caso de uso	UC09 - Realizar questionário sobre a história do xadrez
Ator	Usuário.
Objetivo	Liberar a aba sobre conceitos do xadrez.
Pré-condições	Estar logado na ferramenta e usuário ter lido o conteúdo sobre a história do xadrez.
Pós-condições	Aba Conceitos liberada para o usuário caso este acerte três das seis perguntas.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário clica no botão Questionário.
Cenário alternativo	1. Caso o usuário não acerte três das seis perguntas, o usuário poderá realizar o questionário novamente.
Requisitos Funcionais relacionados	1. Disponibilizar questionários para o usuário testar seu conhecimento (RF).

O caso de uso `UC10 - Realizar questionário sobre conceitos do xadrez` descreve como o ator Usuário efetua o questionário sobre conceitos trabalhados e desenvolvidos no xadrez. Detalhes deste caso de uso no Quadro 12.

Quadro 12 – Caso de uso UC10

Caso de uso	UC10 - Realizar questionário sobre conceitos do xadrez
Ator	Usuário.
Objetivo	Liberar a aba sobre tabuleiro e peças do xadrez.
Pré-condições	Estar logado na ferramenta, usuário ter completado o questionário sobre História do xadrez e usuário ter lido o conteúdo sobre conceitos do xadrez.
Pós-condições	Aba Tabuleiro e Peças liberada para o usuário caso este acerte três das seis perguntas.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário acessa a aba Conceitos; 3. Usuário clica no botão Questionário.
Cenário alternativo	1. Caso o usuário não acerte três das seis perguntas, o usuário poderá realizar o questionário novamente.
Requisitos Funcionais relacionados	1. Disponibilizar questionários para o usuário testar seu conhecimento (RF).

O caso de uso `UC11 - Realizar questionário sobre tabuleiro e peças do xadrez` descreve como o ator Usuário efetua o questionário sobre tabuleiro e peças do xadrez. Detalhes deste caso de uso no Quadro 13.

Quadro 13 – Caso de uso UC11

Caso de uso	UC11 - Realizar questionário sobre tabuleiro e peças do xadrez
Ator	Usuário.
Objetivo	Liberar a aba sobre movimentos especiais do xadrez.
Pré-condições	Estar logado na ferramenta, usuário ter completado o questionário sobre história do xadrez, usuário ter completado o questionário sobre conceitos do xadrez e usuário ter lido o conteúdo sobre tabuleiro e peças do xadrez.
Pós-condições	Aba Movimentos Especiais liberada para o usuário caso este acerte três das seis perguntas.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário acessa a aba Tabuleiro e Peças; 3. Usuário clica no botão Questionário.
Cenário alternativo	1. Caso o usuário não acerte três das seis perguntas, o usuário poderá realizar o questionário novamente.
Requisitos Funcionais relacionados	1. Disponibilizar questionários para o usuário testar seu conhecimento (RF).

O caso de uso `UC12 - Realizar questionário sobre movimentos especiais do xadrez` descreve como o ator Usuário efetua o questionário sobre os movimentos especiais do xadrez. Detalhes deste caso de uso no Quadro 14.

Quadro 14 – Caso de uso UC12

Caso de uso	UC12 - Realizar questionário sobre movimentos especiais do xadrez
Ator	Usuário.
Objetivo	Liberar a aba sobre tabuleiro e peças do xadrez.
Pré-condições	Estar logado na ferramenta, usuário ter completado o questionário sobre história do xadrez, usuário ter completado o questionário sobre conceitos do xadrez, usuário ter completado o questionário sobre Tabuleiro e Peças e usuário ter lido o conteúdo sobre movimentos especiais do xadrez.
Pós-condições	Aba Movimentos Especiais liberada para o usuário caso este acerte três das seis perguntas.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Tutorial; 2. Usuário acessa a aba Movimentos Especiais; 3. Usuário clica no botão Questionário.
Cenário alternativo	1. Caso o usuário não acerte três das seis perguntas, o usuário poderá realizar o questionário novamente.
Requisitos Funcionais relacionados	1. Disponibilizar questionários para o usuário testar seu conhecimento (RF).

O caso de uso `UC13 - Jogar contra uma` *`engine`* descreve como o ator Usuário joga contra a *engine* da ferramenta. Detalhes deste caso de uso no Quadro 15.

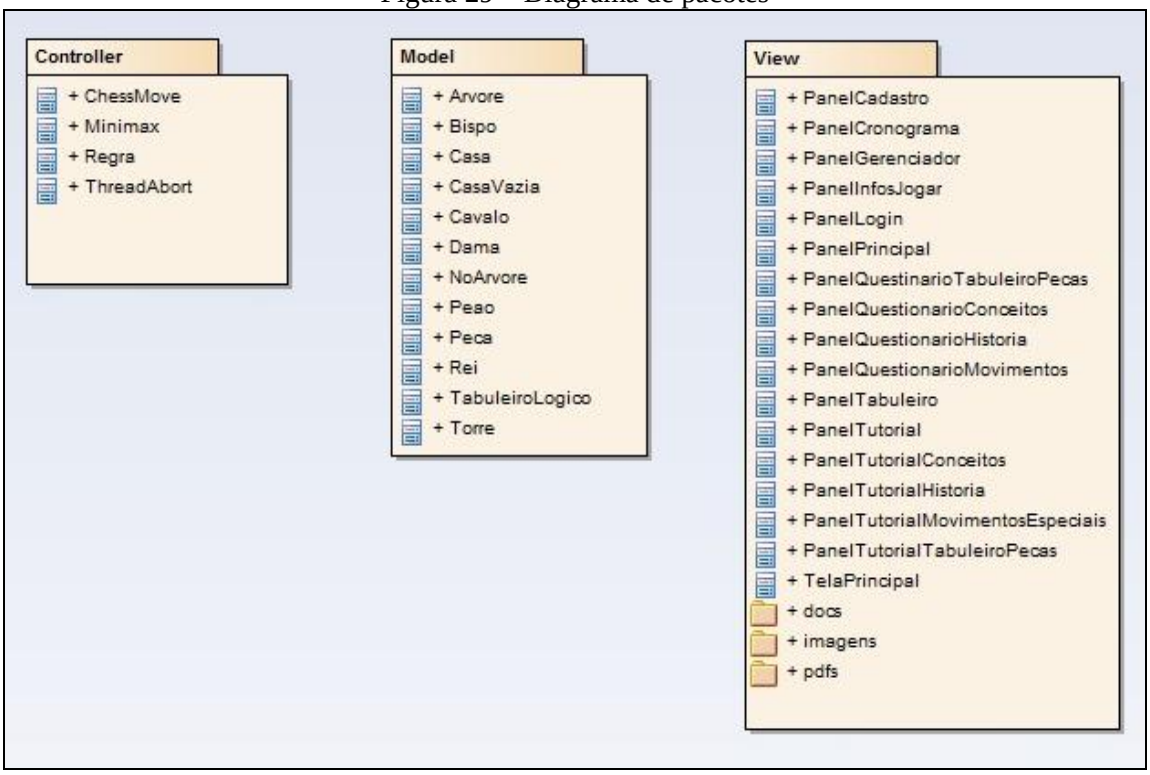
Quadro 15 – Caso de uso UC13

Caso de uso	UC13 - Jogar contra uma *engine*
Ator	Usuário.
Objetivo	Pôr em prática o conhecimento do usuário.
Pré-condições	Estar logado na ferramenta , usuário ter completado o questionário sobre história do xadrez, usuário ter completado o questionário sobre conceitos do xadrez, usuário ter completado o questionário sobre Tabuleiro e Peças e usuário ter completado o questionário sobre movimentos especiais do xadrez.
Cenário principal	1. Usuário acessa o menu Xadrez e clica no submenu Jogar; 2. Usuário clica na casa origem e na casa destino para efetuar um lance; 3. Sistema valida a jogada; 4. *Engine* faz o movimento e o ciclo se repete até o fim do jogo.
Requisitos Funcionais relacionados	1. Permitir que os usuários joguem contra uma engine (RF); 2. Atribuir ou decrementar pontos de rating de um usuário após o termino da partida dependendo de seu resultado (RF); 3. permitir que cada usuário possua um rating próprio (RF).

3.2.2 Diagrama de pacotes

Nesta seção está descrita a estrutura de classes da ferramenta desenvolvida. Para maior organização, a ferramenta foi dividida em três pacotes, sendo eles o `Model`, `View` e `Controller` conforme Figura 29.

Figura 29 – Diagrama de pacotes



3.2.2.1 Pacote `Model`

O pacote `Model` tem por objetivo armazenar as classes responsáveis por descrever os objetos e estrutura de dados utilizadas pela camada de controle. É onde há todos os modelos de objetos utilizados para desenvolvimento do xadrez.

A classe `Árvore` é responsável por criar a estrutura de dados que será utilizada para armazenar os lances calculados pela *engine* através do algoritmo Minimax. Os níveis dela são lista que possuirão os nós criados para o algoritmo Minimax executar, ou seja, esta árvore trabalha com o conceito de que cada nó possui uma lista de filhos. A classe `NoArvore` é utilizada como modelo para os nós que vão compor a árvore. A classe `Peca` é uma classe abstrata utilizada como base para a criação de todas as peças do jogo de xadrez, possuindo os atributos `nome` e `cor`, do tipo `String`, e pontuação do tipo `int`. As classes `Bispo`, `CasaVazia`, `Cavalo`, `Dama`, `Peao`, `Rei` e `Torre` estendem da classe `Peca` e possuem os atributos `pontuação` (`int`), imagem (`ImageIcon`) e o construtor recebe uma cor (`String`), com exceção da classe `CasaVazia` que serve apenas para montar o tabuleiro lógico e possui apenas um atributo `cor` (`String`) e da classe `Casa` que possui os atributos `nome` e `cor` (`String`) e serve para montar o tabuleiro gráfico do pacote `View`. Por fim, a classe `TabuleiroLogico` armazena o tabuleiro de xadrez que é utilizado para validações de movimentação e regras do xadrez.

3.2.2.2 Pacote `View`

O pacote `View` tem por objetivo armazenar as classes que são visuais para o usuário, como telas e painéis. Dentro do pacote `View` há três pacotes: pacote `docs`, que armazena os conteúdos que serão disponibilizados para o usuário no formato DOC; pacote `pdf`, que armazena os conteúdos que serão disponibilizados para o usuário em formato PDF; pacote `imagens`, que armazena as imagens em formato PNG que serão carregadas pelas peças para montar o tabuleiro gráfico.

Em relação às classes, tem-se: a classe `PanelCadastro` é responsável por receber os dados cadastrais do usuário a fim de criar uma conta e posteriormente salvá-la no banco de dados. A classe `PanelCronograma` mostra para o usuário quais são os assuntos que serão abordados no aprendizado do xadrez. A classe `PanelInfosJogar` é responsável por carregar visualmente as informações de nome, *rating*, coordenadas do tabuleiro, entre outras. A classe `PanelLogin` mostra as informações que o usuário precisa digitar para efetuar *login* na ferramenta. A classe `PanelPrincipal` tem por finalidade exclusivamente

carregar a imagem na tela inicial da ferramenta. As classes `PanelQuestionarioHistoria`, `PanelQuestionarioConceitos`, `PanelQuestionarioMovimentos` e `PanelQUestionarioTabuleiroPecas` possuem as perguntas (carregadas do banco de dados) relativas aos seus respectivos assuntos abordados. A classe `PanelTabuleiro` é responsável pela interface gráfica do tabuleiro de xadrez, onde o usuário vai interagir para jogar contra a *engine*. A classe `PanelTutorial` possui um `JTabbedPane` que armazena as abas dos assuntos abordados no tutorial da ferramenta. As classes `PanelTutorialConceitos`, `PanelTutorialHistoria`, `PanelTutorialMovimentosEspeciais` e `PanelTutorialTabuleiroPecas` possuem uma introdução dos seus respectivos assuntos na tela e trazem os botões para acesso ao conteúdo completo e ao questionário do assunto em questão. Todas as classes `Panel` citadas estão inseridas na classe `PanelGerenciador`, que tem por finalidade gerenciar qual destas aparecerá para o usuário conforme desejado por ele. Por fim, a classe `TelaPrincipal` é a classe `main` do projeto, onde a ferramenta é iniciada.

3.2.2.3 Pacote `Controller`

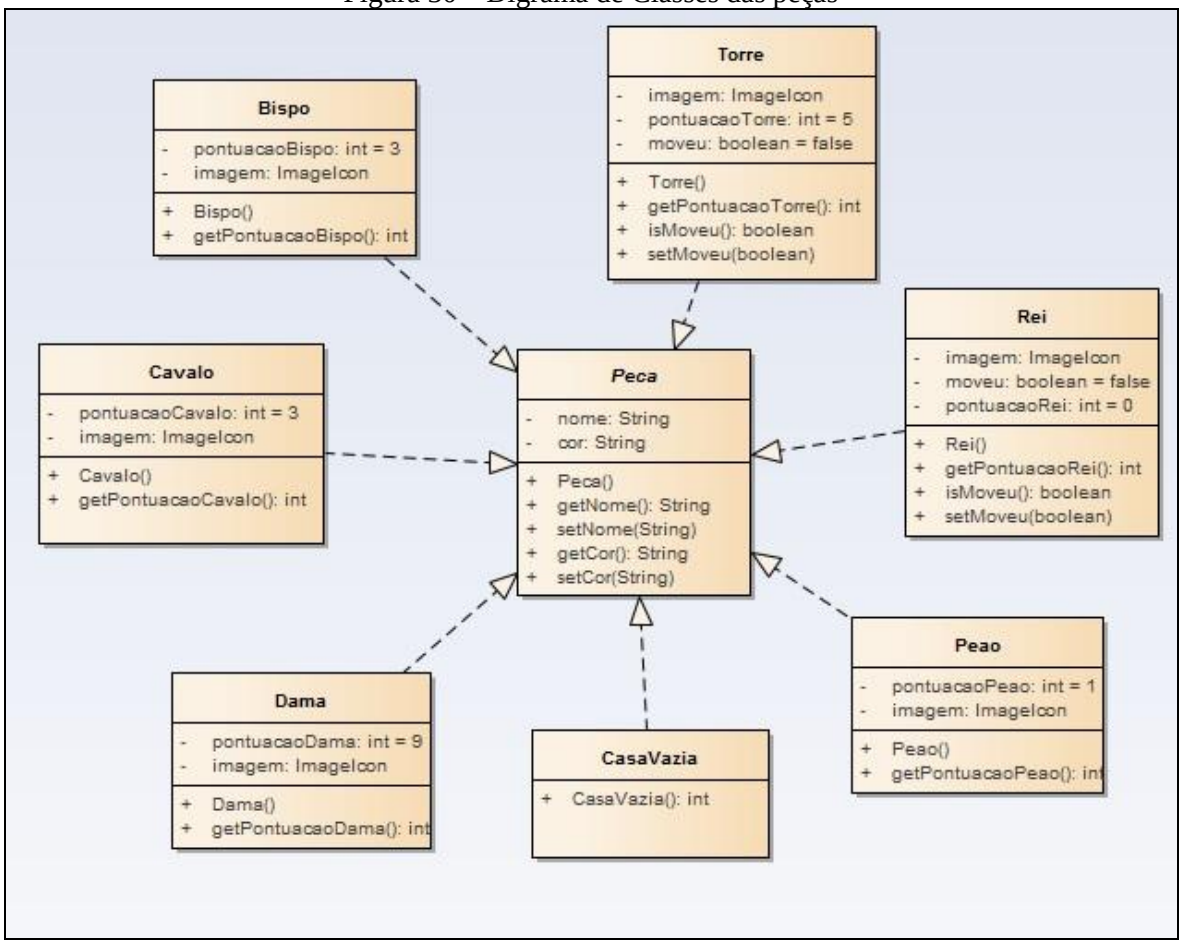
O pacote `Controller` tem por objetivo armazenar as classes responsáveis pelo controle entre classes do pacote `View` e do pacote `Model`.

A classe `ChessMove` é responsável pela geração de movimentos da *engine* quando o usuário jogar uma partida contra ela. A classe `ThreadAbort` é utilizada para ajudar a gerenciar os métodos da classe `ChessMove` e diminuir a perda de performance de execução dos métodos desta classe. A classe `Regra` possui as regras de movimentação das peças do xadrez. Por fim, a classe `Minimax` possui a implementação dos algoritmos Minimax e Poda Alfa-Beta que são responsáveis pela movimentação das peças pela *engine*.

3.2.3 Diagrama de Classes

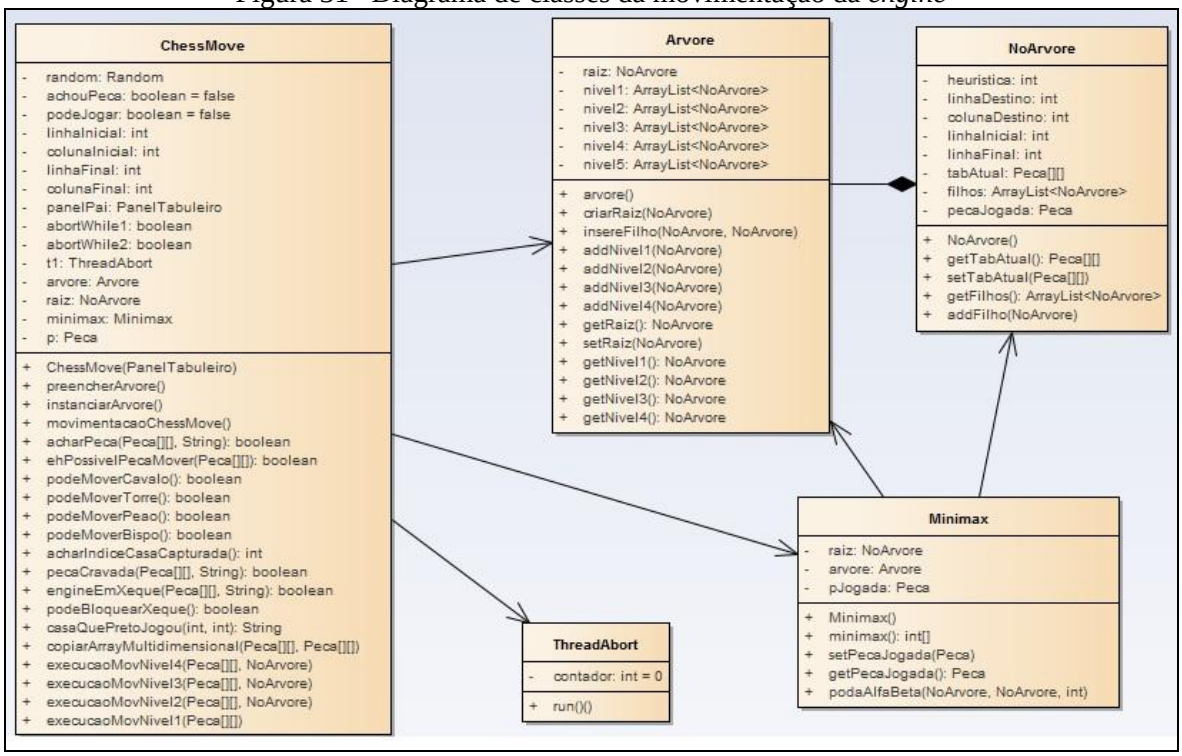
Os diagramas apresentados nesta seção apresentam as principais classes da ferramenta. Na Figura 30 podem-se observar as classes relacionadas às peças do jogo de xadrez, onde tem-se a classe `Peca` abstrata, que serve de base para todas as outras classes referentes às peças do xadrez, que por sua vez possuem suas respectivas pontuações e imagens. A classe `CasaVazia` extende de `Peca` também pois o tabuleiro lógico da ferramenta é um array bidiomensional de `Peca`.

Figura 30 – Digrama de Classes das peças



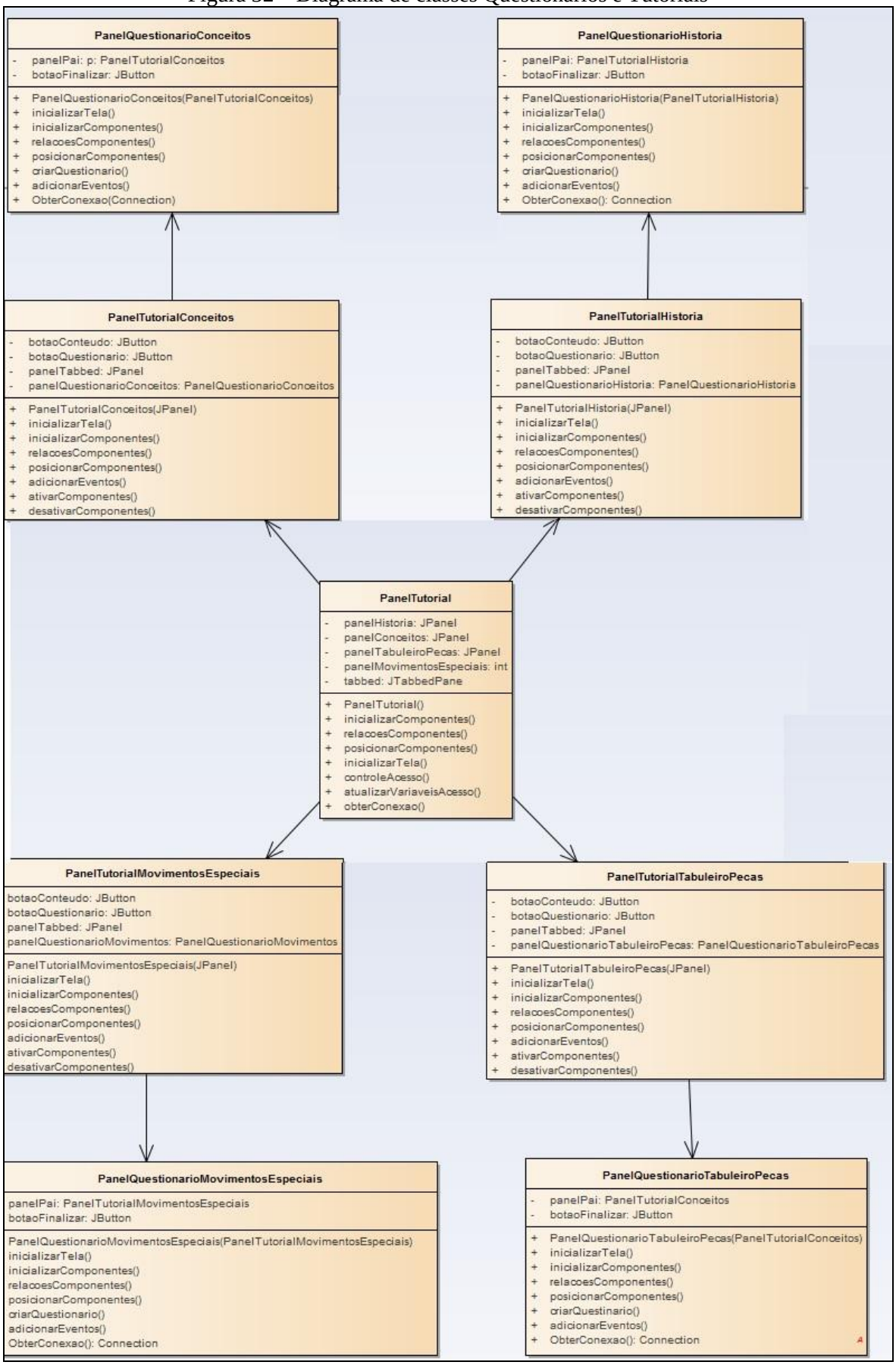
Na Figura 31 tem-se as classes que são utilizadas para gerar e gerenciar a movimentação da *engine*, onde pode-se observar que a classe `ChessMove` é a classe central responsável pela movimentação da *engine*.

Figura 31– Diagrama de classes da movimentação da *engine*



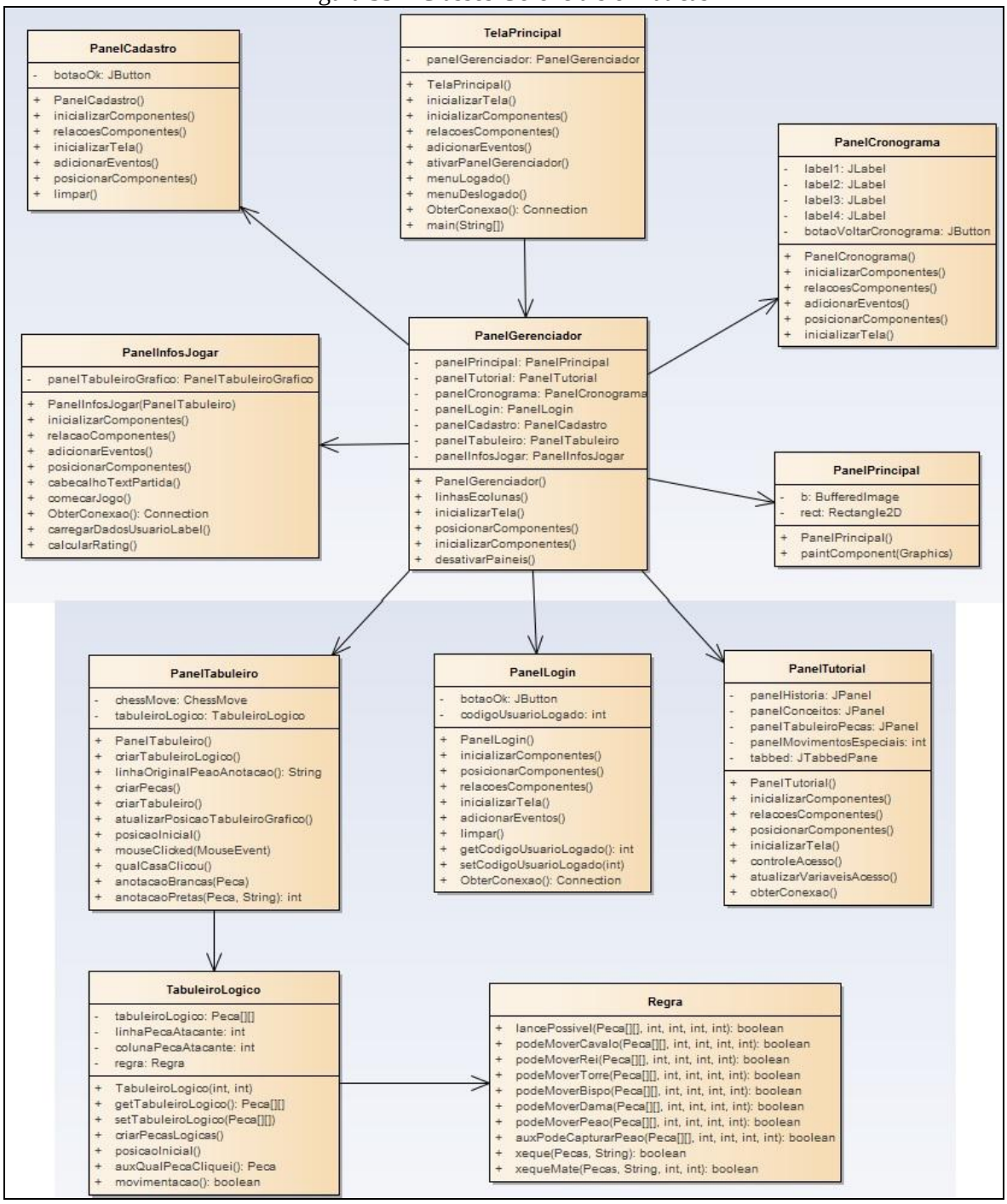
Na Figura 32 tem-se o diagrama destinado às classes responsáveis por gerenciar a área de questionário e tutorial. Cada questionário possui uma classe própria e todas elas estão centralizadas na classe `PanelTutorial`, que é responsável por gerenciar e exibir para o usuário as informações dos respectivos questionários.

Figura 32 – Diagrama de classes Questionários e Tutoriais



Por fim, na Figura 33 tem-se o diagrama das demais telas do sistema, que inclui classes como `Login`, `Cadastro` e `PanelTabuleiro`.

Figura 33 – Classes Gerenciais e Práticas

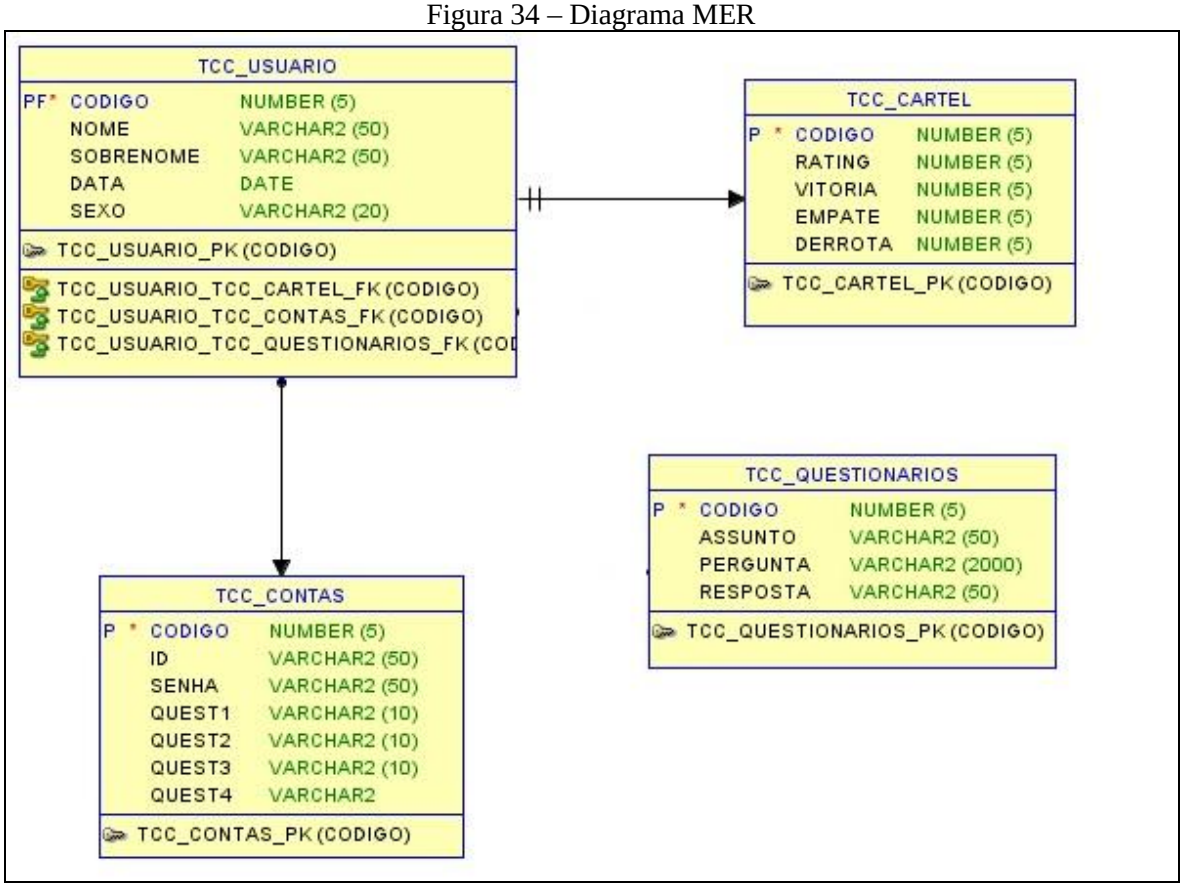


3.2.4 MER

Na Figura 34 tem-se a ilustração do MER das tabelas armazenadas no banco de dados *Oracle*. A tabela `TCC_USUARIO` é responsável por armazenar as informações de cadastro dos usuários. A tabela `TCC_CARTEL` armazena as informações de *rating*, vitória, derrota e empates de cada usuário. A tabela `TCC_CONTAS` possui todas as contas cadastradas

(para controle) e possui também uma coluna para cada questionário efetuado pelo usuário, para controle de acesso entre os tutoriais, para garantir que o usuário veja os tutoriais sequencialmente. Por fim, a tabela `TCC_QUESTIONARIOS` é responsável por armazenar as perguntas e respostas dos questionários da ferramenta. A informação do rating é calculada na própria ferramenta e atualizado em seguida na tabela `CARTEL`.

Figura 34 – Diagrama MER



3.3 CÓDIGOS RELEVANTES

A seguir serão detalhados os trechos mais relevantes da ferramenta desenvolvida.

3.3.1 Classe `ChessMove`

Esta classe tem papel semelhante a da classe `Regra` descrita anteriormente, mas esta classe trata da movimentação das peças coordenadas pela *engine*. O trecho mais relevante desta classe pode ser visto no Quadro 16.

Quadro 16 – Movimentação classe ChessMove

```
public void movimentacaoChessMove() {

            preencherArvore();

            int[] coordenadas = new int[4];

            minimax = new Minimax(raiz, arvore);

            coordenadas = minimax.minimax();
            TabuleiroLogico.movimentacao(coordenadas[0],
coordenadas[1], coordenadas[2], coordenadas[3], true);
            panelPai.atualizarPosicaoTabuleiroGrafico();

            String casa = casaQuePretoJogou(coordenadas[2],
coordenadas[3]);

            PanelTabuleiro.anotacaoPretas(minimax.getpJogada(), casa);

            PanelTabuleiro.setBrancoJoga(true);
            PanelTabuleiro.setPretoJoga(false);

            raiz.getFilhos().clear();
            arvore.getNivel1().clear();
            arvore.getNivel2().clear();
            arvore.getNivel3().clear();
            arvore.getNivel4().clear();

      }
```

Primeiramente é criada uma árvore com as possibilidades de lance no tabuleiro atual. Em seguida é chamada a classe `Minimax` para analisar esta árvore e retornar o melhor lance a ser efetuado. Por fim, é chamado o método movimentação da classe `TabuleiroLogico` para efetuar o lance.

3.3.2 Classe `TabuleiroLogico`

A classe `TabuleiroLogico` tem por objetivo armazenar o tabuleiro (`array` bidimensional de `Peca`) com a posição atual. Tanto o tabuleiro gráfico quanto as validações de movimentação e execução dos algoritmos de IA vão ter como base este tabuleiro lógico. O trecho mais relevante desta classe pode ser visto no Quadro 17.

Quadro 17 – Movimentação classe TabuleiroLogico

```
// método de movimentação
      public static boolean movimentacao(int xOrigem, int yOrigem, int
xDestino,
                  int yDestino, boolean ehPraMover) {
            Peca pOrigem;
            Peca pDestino;
            Peca casaVazia;
            boolean moveu = false;

        String corAdversaria = "";

            pOrigem = tabuleiroLogico[xOrigem][yOrigem];
            pDestino = tabuleiroLogico[xDestino][yDestino];

        if (pOrigem.getCor().contains("branco")) {
            corAdversaria = "negro";
        }

        if (pOrigem.getCor().contains("negro")) {
            corAdversaria = "branco";
        }


            if (Regra.lancePossivel(tabuleiroLogico, xOrigem, yOrigem,
xDestino,
                        yDestino)) { // Se o lance for possível, ele
entra aqui

                  if(pDestino.getNome().contains("ranco") ||
pDestino.getNome().contains("egro")) { // entra aqui se houve captura

                        casaVazia = new CasaVazia("b"); // cria uma casa
vazia porque quando há captura a casa destino recebe a peça de origem e
a casa origem deve receber uma casa vazia, pois a peça que estava
anteriormente na destino foi capturada e não existe mais
                        casaVazia.setNome("Casa");
                        tabuleiroLogico[xDestino][yDestino] = pOrigem;
// peça origem chega na casa destino normalmente
                        tabuleiroLogico[xOrigem][yOrigem] = casaVazia;//
peça destino desaparece pois foi capturada

                        if (!Regra.xeque(tabuleiroLogico,
pOrigem.getCor())) {// se fizer o movimento e não estiver em xeque,
está ok;

                              if (ehPraMover == false) { // se cair aqui
é a validação do xeque-mate ou análises.. não é pra mover
                                    tabuleiroLogico[xOrigem][yOrigem] =
pOrigem;
                                    tabuleiroLogico[xDestino][yDestino]
= pDestino;
                                    return true;
                              }

                        } else { // se o movimento não for válido volta
as peças
                              tabuleiroLogico[xOrigem][yOrigem] =
pOrigem;
                              tabuleiroLogico[xDestino][yDestino] =
pDestino;
                              return false;
                        }

                        if(Regra.xeque(tabuleiroLogico, corAdversaria))
{ // se estiver em xeque, deve setar para true para ter o '+' na
anotação
                              xequeAnotacao = true;
                        }
```

```
                        PanelTabuleiro.setHouveCaptura(true); // aqui
também é para controle de anotação e botar o 'x' na anotação

                        return true;


                  } else {
                        // se não houve captura (peça moveu para uma
casa vazia), entra aqui neste else
                        tabuleiroLogico[xOrigem][yOrigem] = pDestino;
                        tabuleiroLogico[xDestino][yDestino] = pOrigem;

                        if (!Regra.xeque(tabuleiroLogico,
pOrigem.getCor())) {// se fizer o movimento e não estiver em xeque,
está ok;

                              if (ehPraMover == false) {
                                    tabuleiroLogico[xOrigem][yOrigem] =
pOrigem;
                                    tabuleiroLogico[xDestino][yDestino]
= pDestino; // se cair aqui é a validação do xeque-mate.. não é pra
mover
                                    return true;
                              }

                        } else { // senão volta as peças
                              tabuleiroLogico[xOrigem][yOrigem] =
pOrigem;
                              tabuleiroLogico[xDestino][yDestino] =
pDestino;
                              return false;
                        }


                  PanelTabuleiro.setHouveCaptura(false);
                  }

                  if(Regra.xeque(tabuleiroLogico, corAdversaria)) {
                        xequeAnotacao = true;
                  }

                  moveu = true;

            }
            return moveu;
      }
```

Este é o método principal da movimentação da ferramenta, que recebe como parâmetro a casa inicial e casa final que o usuário deseja mover a peça. Com estas informações em mãos, o método sabe qual peça que foi movimentada pelo usuário. Primeiramente ele chama o método `lancePossivel` da classe Regra para saber se o lance é possível e o efetua. Após isso, valida se o Rei ficou em xeque ao final do movimento. Caso tenha ficado em xeque, ele volta as peças para as posicionais originais anteriores ao movimento e emite uma mensagem afirmando que o lance é inválido. Caso não tenha ficado em xeque, ele mantém as peças na movimentação após o movimento e finaliza o método.

3.3.3 Classe `Regra`

A classe `Regra` determina se um lance efetuado pelo usuário é válido. Cada peça possui uma movimentação específica, que será validada no segundo clique do usuário no tabuleiro. Para ser validado, após o lance o Rei não pode estar em xeque. Para conferir isso, foi criada a rotina vista no Quadro 18, que recebe como parâmetro o tabuleiro atual e a cor do jogador que efetuou o lance.

Quadro 18 – Rotina xeque

```
public static boolean xeque(Peca[][] pecas, String cor) {
        int linhaDoRei = 0;
        int colunaDoRei = 0;
        String corAdversaria = "";

        if (cor.contains("branco")) {
            corAdversaria = "negro";
        }

        if (cor.contains("negro")) {
            corAdversaria = "branco";
        }

        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                Peca p = pecas[i][j];
                if (p.getNome().contains("ei") && p.getCor() == cor) {
// pega a posição do seu Rei
                    linhaDoRei = i;
                    colunaDoRei = j;
                }
            }
        }

        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                if (pecas[i][j] != null) {
                    Peca p = pecas[i][j];
                    if (p.getCor() == corAdversaria) { // veriifcar se
alguma peça adversária ataca seu rei
                        if (p.getNome().contains("avalo") &&
podeMoverCavalo(pecas, i, j, linhaDoRei, colunaDoRei)
                                || p.getNome().contains("ispo") &&
podeMoverBispo(pecas, i, j, linhaDoRei, colunaDoRei)
                                || p.getNome().contains("ama") &&
podeMoverDama(pecas, i, j, linhaDoRei, colunaDoRei)
                                || p.getNome().contains("orre") &&
podeMoverTorre(pecas, i, j, linhaDoRei, colunaDoRei)
                                || p.getNome().contains("eao") &&
auxPodeCapturarPeao(pecas, i, j, linhaDoRei, colunaDoRei)
                                ){
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }
```

Primeiramente, localiza-se o Rei no tabuleiro e armazena-se as coordenadas deste nas variáveis `linhaDoRei` e `colunaDoRei`. Em seguida, faz-se uma nova busca no tabuleiro, chamando o método de movimentação das peças adversárias para validar se

alguma dessas podem chegar à casa onde está o Rei. Caso consiga, quer dizer que está em xeque e, consequentemente, retorna *true*. Caso não consiga, sai do método retornando *false*.

O cavalo move-se em formato da letra “L” para qualquer direção, podendo passar por cima de qualquer peça. O método que valida a movimentação do cavalo pode ser visto no Quadro 19.

Quadro 19 – Método de movimentação do Cavalo

```
      // Método de movimento do Cavalo
      public static boolean podeMoverCavalo(Peca[][] pecas, int
linhaInicial,
                  int colunaInicial, int linhaFinal, int colunaFinal)
{
            if (pecas[linhaInicial][colunaInicial].getCor() ==
pecas[linhaFinal][colunaFinal]
                        .getCor()) {
            return false;
            }

            if ((linhaFinal == linhaInicial + 1 || linhaFinal ==
linhaInicial - 1)
                        && (colunaFinal == colunaInicial + 2 ||
colunaFinal == colunaInicial - 2)) {
                  return true;
            }

            if ((linhaFinal == linhaInicial + 2 || linhaFinal ==
linhaInicial - 2)
                        && (colunaFinal == colunaInicial + 1 ||
colunaFinal == colunaInicial - 1)) {
                  return true;
            }
            return false;
      }
```

O método de movimentação do Rei valida, além do roque, se ele pode mover-se para alguma casa adjacente. No Quadro 20 tem-se a rotina de movimentação do Rei.

Quadro 20 – Método de movimentação do Rei

```
      // Método de movimento do Rei
      public static boolean podeMoverRei(Peca[][] pecas, int
linhaInicial,
                  int colunaInicial, int linhaFinal, int colunaFinal) {
            Peca pOrigem;
            Peca pDestino;
            Peca pDestino2;

            if (pecas[linhaInicial][colunaInicial].getCor() ==
pecas[linhaFinal][colunaFinal]
                        .getCor()) { // se clicar em uma casa que
contenha uma peça da mesma cor, não pode
                  return false;
            }

            if ((linhaFinal == linhaInicial && colunaFinal ==
colunaInicial - 1) // esquerda
                        || (linhaFinal == linhaInicial - 1 &&
colunaFinal == colunaInicial - 1) // diagonal esquerda cima
                        || (linhaFinal == linhaInicial - 1 &&
colunaFinal == colunaInicial) // para cima
                        || (linhaFinal == linhaInicial - 1 &&
colunaFinal == colunaInicial + 1) // diagonal direita cima
                        || (linhaFinal == linhaInicial && colunaFinal ==
colunaInicial + 1) // direita
                        || (linhaFinal == linhaInicial + 1 &&
colunaFinal == colunaInicial + 1) // diagonal direita baixo
                        || (linhaFinal == linhaInicial + 1 &&
colunaFinal == colunaInicial) // baixo
                        || (linhaFinal == linhaInicial + 1 &&
colunaFinal == colunaInicial - 1)) { // diagonal esquerda baixo
                  if
(pecas[linhaInicial][colunaInicial].getCor().equalsIgnoreCase("branco"))
{
                        TabuleiroLogico.setReiBrancoMoveu(true); // se a
peça que se moveu for branca, então o branco jogou.
                  } else {
                        TabuleiroLogico.setReiNegroMoveu(true); // se a
peça que se moveu for preta, então o preto jogou.
                  }
                  return true;
            }
            return false;
      }
```

O método de movimentação da Torre valida se a `linhaFinal` é igual a `linhaInicial` ou se a `colunaInicial` é igual a `colunaFinal`, pois a Torre anda somente em linha reta, juntamente com a validação se não há peças na trajetória da Torre. No Quadro 21 tem-se o trecho de código da movimentação da Torre.

Quadro 21 – Método de movimentação da Torre

```
      // Método para movimento da Torre
            public static boolean podeMoverTorre(Peca[][] pecas, int
linhaInicial,
                  int colunaInicial, int linhaFinal, int colunaFinal)
{
              if (pecas[linhaInicial][colunaInicial].getCor() ==
pecas[linhaFinal][colunaFinal].getCor()) {
                  return false;
              }

              if (linhaInicial != linhaFinal && colunaInicial !=
colunaFinal) { // valida se a torre está andando em linha reta
                  return false;
              }

              int x;
              if (colunaFinal == colunaInicial && linhaFinal <
linhaInicial) { // baixo para cima
                  x = linhaInicial - 1;
                  while (x != linhaFinal) {
                      if (pecas[x][colunaInicial].getCor() != "b" &&
pecas[x][colunaInicial].getCor() != "n") {
                        // se entrar aqui, é porque tem peça no
caminho.. pq 'b' e 'n' são as cores da casa vazia.. se for diferente
disso
                          // é porque a cor é duma peça
                            return false;
                      }
                      x--;
                  }

              if (colunaFinal == colunaInicial && linhaFinal >
linhaInicial) { // cima para baixo
                  x = linhaInicial + 1;
                  while (x != linhaFinal) {
                      if (pecas[x][colunaInicial].getCor() != "b" &&
pecas[x][colunaInicial].getCor() != "n"){
                          return false;
                      }
                      x++;
                  }

              if (colunaFinal < colunaInicial && linhaFinal ==
linhaInicial) { // direita para esquerda
                  x = colunaInicial - 1;
                  while (x != colunaFinal) {
                      if (pecas[linhaInicial][x].getCor() != "b" &&
pecas[linhaInicial][x].getCor() != "n") {
                        return false;
                      }
                      x--;
                  }

              if (colunaFinal > colunaInicial && linhaFinal ==
linhaInicial) { // esquerda para direita
                  x = colunaInicial + 1;
                  while (x != colunaFinal) {
                      if (pecas[linhaInicial][x].getCor() != "b" &&
pecas[linhaInicial][x].getCor() != "n") {
                          return false;
                      }
                      x++;
                  }

              return true;
          }
```

O método de movimentação do Bispo valida primeiramente se este está movimentando-se em diagonal, que é o correto. Caso sim, valida se há alguma peça no caminho do Bispo. No Quadro 22 tem-se o método de movimentação do Bispo.

Quadro 22 – Método de movimentação do Bispo

```
// Método de movimento do Bispo
      public static boolean podeMoverBispo(Peca[][] pecas, int
linhaInicial,
            int colunaInicial, int linhaFinal, int colunaFinal) {
        if (pecas[linhaInicial][colunaInicial].getCor() ==
pecas[linhaFinal][colunaFinal].getCor()) {
            return false;
        }

        if (Math.abs(linhaInicial - linhaFinal) !=
Math.abs(colunaInicial - colunaFinal)) {
            // valor absoluto da operação.. ou seja, 1-2 = 2-1
            return false;
        }

        int x;
        int y;
        if (linhaFinal < linhaInicial && colunaFinal < colunaInicial)
{ //  diagonal para cima e esquerda
            x = linhaInicial - 1;
            y = colunaInicial - 1;
            while (x != linhaFinal || y != colunaFinal) {
                if (pecas[x][y].getCor() != "b" &&
pecas[x][y].getCor() != "n") {
                    return false;
                }
                x--;
                y--;
            }
        }

        if (linhaFinal < linhaInicial && colunaFinal > colunaInicial)
{ //   diagonal para cima e direita
            x = linhaInicial - 1;
            y = colunaInicial + 1;
            while (x != linhaFinal || y != colunaFinal) {
                if (pecas[x][y].getCor() != "b" &&
pecas[x][y].getCor() != "n") {
                    return false;
                }
                x--;
                y++;
            }
        }

        if (linhaFinal > linhaInicial && colunaFinal < colunaInicial)
{ // diagonal para baixo e esquerda
            x = linhaInicial + 1;
            y = colunaInicial - 1;
            while (x != linhaFinal || y != colunaFinal) {
                if (pecas[x][y].getCor() != "b" &&
pecas[x][y].getCor() != "n") {
                    return false;
                }
                x++;
                y--;
            }
        }

        if (linhaFinal > linhaInicial && colunaFinal > colunaInicial)
{ // diagonal para baixo e direita
            x = linhaInicial + 1;
            y = colunaInicial + 1;
            while (x != linhaFinal || y != colunaFinal) {
                if (pecas[x][y].getCor() != "b" &&
pecas[x][y].getCor() != "n") {
                    return false;
                }
```

```
                x++;
                y++;
            }
        }

        return true;
    }
```

O método de movimentação da Dama é a soma dos métodos de movimentação do Bispo e da Torre, pois a dama move-se em diagonal ou em linha reta. No Quadro 23 tem-se o método de movimentação da Dama.

Quadro 23 – Movimentação da Dama

```
    // Método de movimento da Dama
     public static boolean podeMoverDama(Peca[][] pecas, int
linhaInicial,
            int colunaInicial, int linhaFinal, int colunaFinal) {
      // A dama se move como um Bispo e uma Torre.. então basta
reutiliza os dois métodos de movimentação da Torre e do Bispo
        if (podeMoverTorre(pecas, linhaInicial, colunaInicial,
linhaFinal, colunaFinal)
                || podeMoverBispo(pecas, linhaInicial, colunaInicial,
linhaFinal, colunaFinal)) {
            return true;
        }
        return false;
    }
```

O método de movimentação do Peão valida se este está movendo-se para frente, com exceção do primeiro movimento, que pode ser duas casas. No Quadro 24 tem-se o método de movimentação do Peão.

Quadro 24 – Método de movimentação do Peão

```
// Método de movimento do Peão
      public static boolean podeMoverPeao(Peca[][] pecas, int
linhaInicial,
            int colunaInicial, int linhaFinal, int colunaFinal) {

        Peca p = pecas[linhaInicial][colunaInicial];

        if (p.getCor().contains("branco") && linhaFinal >
linhaInicial) { // peão branco não pode ir para trás
            return false;
        }

        if (p.getCor().contains("negro") && linhaFinal < linhaInicial)
{ // peão negro não pode ir para trás
            return false;
        }

        if (p.getCor().contains("branco") && linhaInicial == 6 &&
linhaFinal == 4 && colunaInicial == colunaFinal) { // peão branco só
pode andar duas casas no primeiro lance
           if(pecas[linhaInicial - 1][colunaInicial].getCor().length()
== 1 && pecas[linhaInicial - 2][colunaInicial].getCor().length() == 1)
// se for length = 1, então é casa vazia (b ou n)
               return true;
            }

        if (p.getCor().contains("negro") && linhaInicial == 1 &&
linhaFinal == 3 && colunaInicial == colunaFinal) { // peão preto só
pode andar duas casas no primeiro lance
            if(pecas[linhaInicial +
1][colunaInicial].getCor().length() == 1 && pecas[linhaInicial +
2][colunaInicial].getCor().length() == 1) // se for length = 1, então
é casa vazia (b ou n)
                  return true;
         }

        if (p.getCor().contains("branco") && linhaInicial != 6 &&
linhaFinal < linhaInicial - 1) { // não pode andar duas casas se não
for o primeiro movimento do peão
            return false;
        }

        if (p.getCor().contains("negro") && linhaInicial != 1 &&
linhaFinal > linhaInicial + 1) { // não pode andar duas casas se não
for o primeiro movimento do peão
            return false;
        }

        if (p.getCor().contains("branco") && linhaFinal ==
linhaInicial - 1 && pecas[linhaInicial -
1][colunaInicial].getCor().length() == 1 && colunaInicial ==
colunaFinal) {
        // pode andar uma casa se não tiver uma peça na frente, com
exceção se for pra linha 0 (coroação), que terá um if específico
            return true;
        }

        if (p.getCor().contains("negro") && linhaFinal == linhaInicial
+ 1 && pecas[linhaInicial + 1][colunaInicial].getCor().length() == 1
&& colunaInicial == colunaFinal) {
            // pode andar uma casa se não tiver uma peça na frente,
com exceção se for pra linha 7 (coroação), que terá um if específico
            return true;
        }

        return false;
     }
```

Por fim, o método `lancePossivel` é responsável por receber as coordenadas iniciais e finais clicadas do lance efetuado e, dependendo da peça que foi jogada, chamar o respectivo método de movimentação para saber se o lance é válido. No Quadro 25 tem-se a ilustração do método descrito.

Quadro 25 – Método `lancePossivel`

```
// método que, a partir do nome da peça, chama respectivo método de
movimento
      public static boolean lancePossivel(Peca[][] pecas,
                  int linhaInicial, int colunaInicial, int linhaFinal,
int colunaFinal) {

             if (linhaInicial == linhaFinal && colunaInicial ==
colunaFinal) { // clicar duas vezes na mesma casa;
                  return false;
               }

               Peca p = pecas[linhaInicial][colunaInicial];

               if (p.getNome().contains("avalo")) {
                  return podeMoverCavalo(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               if (p.getNome().contains("ispo")) {
                  return podeMoverBispo(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               if (p.getNome().contains("ama")) {
                  return podeMoverDama(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               if (p.getNome().contains("orre")) {
                  return podeMoverTorre(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               if (p.getNome().contains("ei")) {
                  return podeMoverRei(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               if (p.getNome().contains("eao")) {
                  return podeMoverPeao(pecas, linhaInicial,
colunaInicial, linhaFinal, colunaFinal);
               }

               return false;
      }
```

3.3.4 Classe `Minimax`

Esta classe é responsável por analisar com os algoritmos Minimax e Poda Alfa-Beta a árvore de movimentos que é gerada em toda jogada da *engine*. Há dois métodos que são importantes destacar. O primeiro deles é o método `minimax`, que pode ser visto no Quadro 26.

Quadro 26 – Método `minimax`

```
public int[] minimax() {
            // retorna um array de 4 posições inteiras : 1- linha
inicial, 2- linha final, 3 - colunaInicial , 4 - colunaFinal
            int[] coordenadas = new int[4];
            int alpha;
            int linhaInicial = 0, linhaFinal = 0, colunaInicial = 0,
colunaFinal = 0; // receberá ao fim do método as coordenadas para que
seja feito o lance escolhido pela engine

            // 1° - devo calcular a heurística dos nós folha, ou seja,
do arvore.getNivel4();
            for(NoArvore no : arvore.getNivel4()) {

      no.setHeuristica(calcularHeuristica(no.getTabAtual()));
            }

            //for(NoArvore no : arvore.getNivel4()) {
                  //System.out.println(no.getHeuristica());
            //}

            // jogada do preto
            alpha = Integer.MAX_VALUE; // Nó de MIN
            for(NoArvore no : arvore.getNivel3()) { // para cada nó do
n3
                  for(NoArvore filho : arvore.getNivel4()) {
                     if(podaAlfaBeta(no, filho, 3)) {
                           no.getFilhos().clear(); // limpa os filhos
do nó pois este pode ser desconsiderado
                           break;
                     }
                  if (filho.getHeuristica() < alpha) {   // analisar
os filhos em n4
                        alpha = filho.getHeuristica();
                  }
               }
                  no.setHeuristica(alpha); // seta o valor do pai
(nivel3) com o menor encontrado nos filhos (nivel4)
            }

            // jogada do branco
            alpha = Integer.MIN_VALUE; // Nó de MAX
            for(NoArvore no : arvore.getNivel2()) { // para cada nó do
n2
                  for(NoArvore filho : arvore.getNivel3()) {
                           if(podaAlfaBeta(no, filho, 2)) {
                                 no.getFilhos().clear(); // limpa os
filhos do nó pois este pode ser desconsiderado
                                 break;
                           }
                  if (filho.getHeuristica() > alpha) {   // analisar
os filhos em n3
                        alpha = filho.getHeuristica();
                  }
               }
                  no.setHeuristica(alpha); // seta o valor do pai
(nivel3) com o menor encontrado nos filhos (nivel4)
            }

            // jogada do preto
            alpha = Integer.MAX_VALUE; // Nó de MIN
            for(NoArvore no : arvore.getNivel1()) { // para cada nó do
n1
                  for(NoArvore filho : arvore.getNivel2()) {
                           if(podaAlfaBeta(no, filho, 1)) {
                                 no.getFilhos().clear(); // limpa os
filhos do nó pois este pode ser desconsiderado
                                 break;
                           }
```

```
                    if (filho.getHeuristica() < alpha) {   // analisar
os filhos em n2
                          alpha = filho.getHeuristica();
                    }
               }
                    no.setHeuristica(alpha); // seta o valor do pai
(nivel3) com o menor encontrado nos filhos (nivel4)
             }

             alpha = Integer.MIN_VALUE; // Nó de MAx
             for(NoArvore filho : raiz.getFilhos()) {
                    //for(NoArvore filho : arvore.getNivel1()) {
                    if (filho.getHeuristica() > alpha) {   // analisar
os filhos em n2
                          alpha = filho.getHeuristica();
                          linhaInicial = filho.getLinhaInicial();
                          linhaFinal = filho.getLinhaDestino();
                          colunaInicial = filho.getColunaInicial();
                          colunaFinal = filho.getColunaDestino();
                          pJogada = filho.getPecaJogada();
                    }
               }
                    raiz.setHeuristica(alpha); // seta o valor do pai
(nivel3) com o menor encontrado nos filhos (nivel4)

             coordenadas[0] = linhaInicial;
             coordenadas[1] = colunaInicial;
             coordenadas[2] = linhaFinal;
             coordenadas[3] = colunaFinal;

             return coordenadas;
       }
```

Inicialmente o método calcula a heurística dos nós folhas do nível 4. Em seguida, obtém o valor Min dos nós do nível 3 com base nos seus filhos de nível 4. Depois, obtém-se o valor Max dos nós de nível 2. Quando chega no nível 1, ele procura o valor Min dos nós baseados nos filhos de nível 2. Por fim, procura entre os filhos da raiz o valor máximo obtido e vê qual o lance que foi efetuado para chegar até ele, sendo esse lance o lance que a *engine* vai fazer.

O segundo método é o `podaAlfaBeta` que, conforme diz seu nome, é responsável por podar os ramos da árvore que nunca serão acessados. No Quadro 27 tem-se a implementação deste método.

Quadro 27 – Método `podaAlfaBeta`

```
public boolean podaAlfaBeta(NoArvore pai , NoArvore filho, int nivel)
{
            if (nivel == 3) { // MIN
                  if(arvore.getNivel3().isEmpty()) { // se o array de
'primos' do pai estiver vazios, quer dizer que ele é o 1° primo que
está sendo analisado, então é false
                        return false;
                  }
                  for(NoArvore primos : arvore.getNivel3()) {
                        if (filho.getHeuristica() <
primos.getHeuristica()) {
                              return true; // poderá podar todo o nó
pai se cair aqui
                        }
                  }
            }

            if (nivel == 2) { // MAX
                  if(arvore.getNivel2().isEmpty()) { // se o array de
'primos' do pai estiver vazios, quer dizer que ele é o 1° primo que
está sendo analisado, então é false
                        return false;
                  }
                  for(NoArvore primos : arvore.getNivel2()) {
                        if (filho.getHeuristica() >
primos.getHeuristica()) {
                              return true; // poderá podar todo o nó
pai se cair aqui
                        }
                  }
            }

            if (nivel == 1) { // MIN
                  if(arvore.getNivel1().isEmpty()) { // se o array de
'primos' do pai estiver vazios, quer dizer que ele é o 1° primo que
está sendo analisado, então é false
                        return false;
                  }
                  for(NoArvore primos : arvore.getNivel1()) {
                        if (filho.getHeuristica() <
primos.getHeuristica()) {
                              return true; // poderá podar todo o nó
pai se cair aqui
                        }
                  }
            }

            return false;
      }
```

Visto que a árvore criada contém 4 níveis, desconsiderando o nível da raiz, inicialmente o algoritmo roda para os nós do nível 3, depois nível 2 e por fim nível 1. O objetivo do algoritmo é podar ramos que nunca serão acessados pelo algoritmo Minimax a fim de ganhar performance. A análise de corte efetuada pelo algoritmo está descrita na fundamentação teórica deste trabalho.

3.4 TÉCNICAS E FERRAMENTAS UTILIZADAS

A ferramenta foi desenvolvida no ambiente de desenvolvimento Eclipse utilizando a linguagem de programação Java. Não foi utilizado nenhum *framework* e não é necessária a

preparação de ambiente para a execução da ferramenta. O banco de dados relacional utilizado foi o *Oracle XE*. As telas da ferramenta foram feitas utilizando `JFrame` e `JPanel`, bibliotecas nativas do Java. Por fim, a *engine* foi desenvolvida utilizando os algoritmos de IA Minimax e Poda Alfa-Beta.

3.5 OPERACIONALIDADE

A seguir está descrita a operacionalidade da ferramenta por parte do usuário.

3.5.1 Execução da Ferramenta

A tela inicial da ferramenta pode ser vista na Figura 35, onde pode-se observar três menus na parte superior.

Figura 35 – Tela inicial da ferramenta

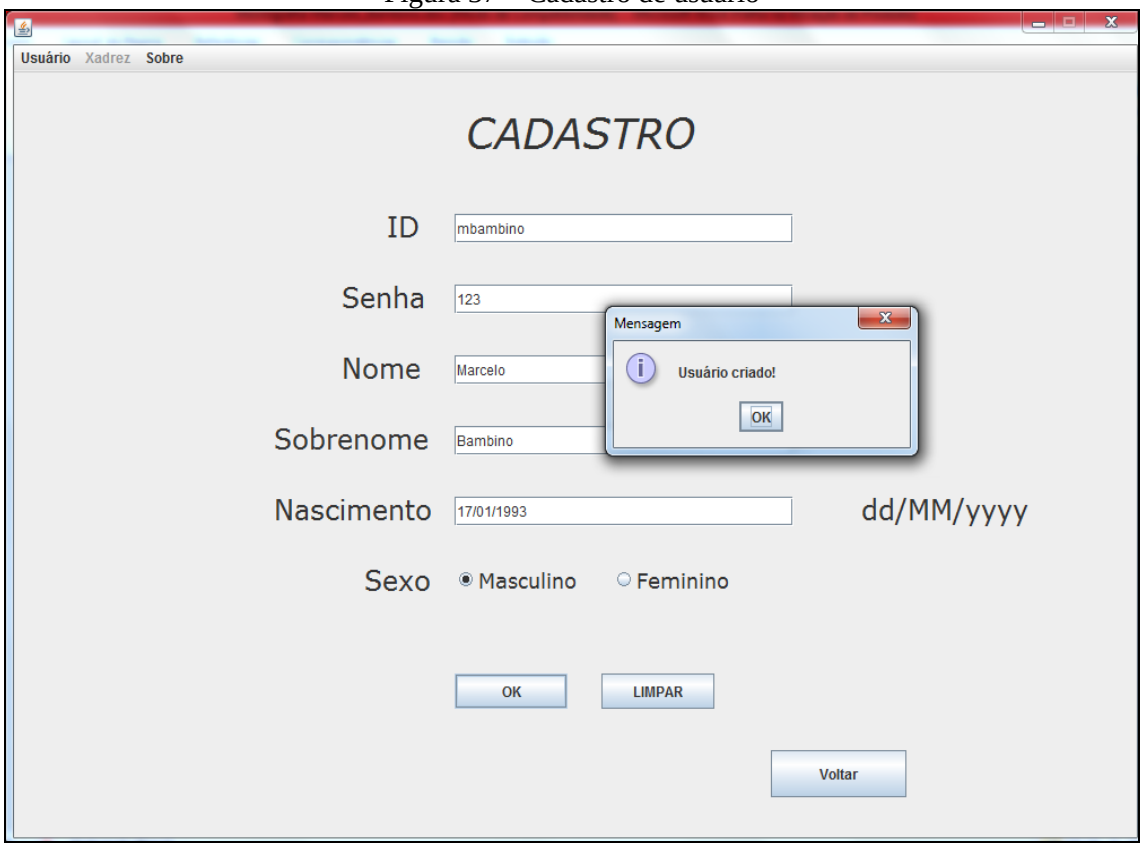
O menu `Sobre` possui um subitem Autor, que mostra uma informação do criador da ferramenta (Figura 36).

Figura 36 – Menu sobre da ferramenta

O menu `Usuário` possui três submenus:

a) `cadastro`: este menu abre um `JPanel` (Figura 37) com as informações necessárias para o usuário efetuar o cadastro na ferramenta;

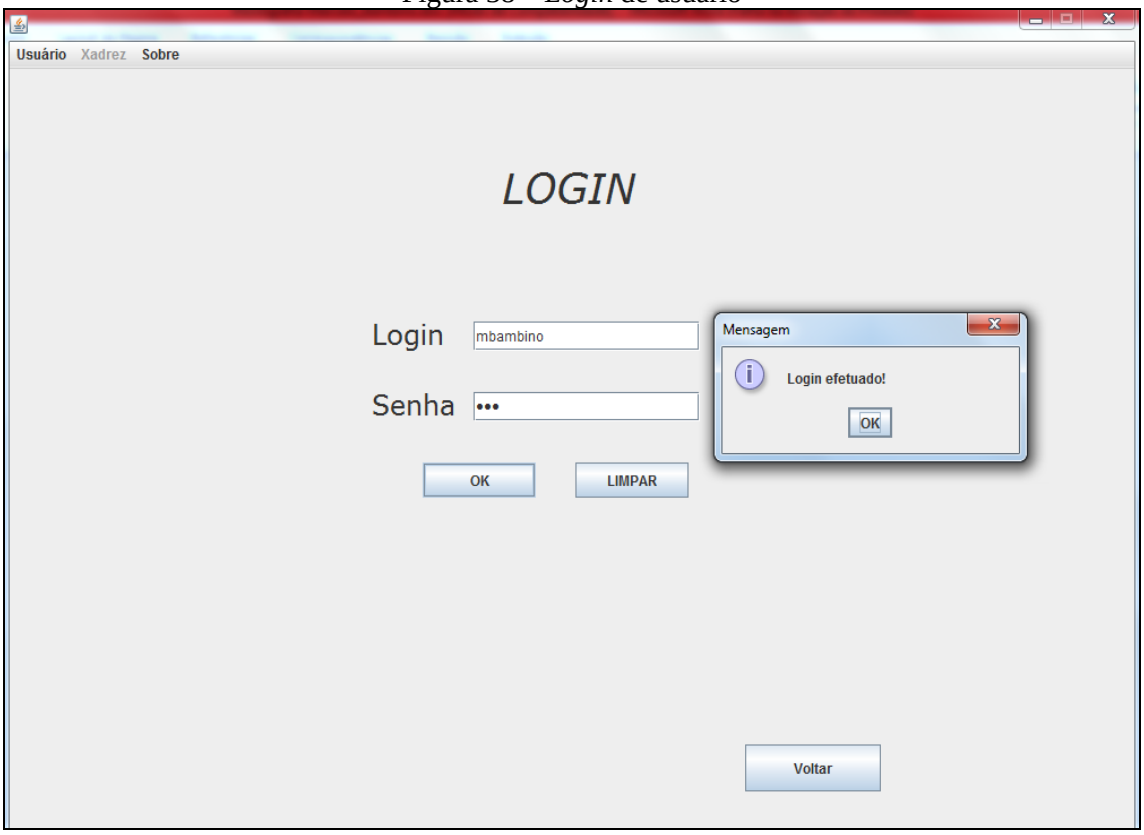
Figura 37 – Cadastro de usuário



Após preencher os campos e clicar no botão *OK*, caso as informações tenham sido devidamente preenchidas, os dados são enviados para o banco de dados *Oracle*, onde as informações são inseridas nas tabelas persistidas `TCC_CONTAS`, `TCC_CARTEL` e `TCC_USUARIO`.

b) `login`: este menu abre um `JPanel` (Figura 38) com as informações necessárias para o usuário efetuar o *login* na ferramenta;

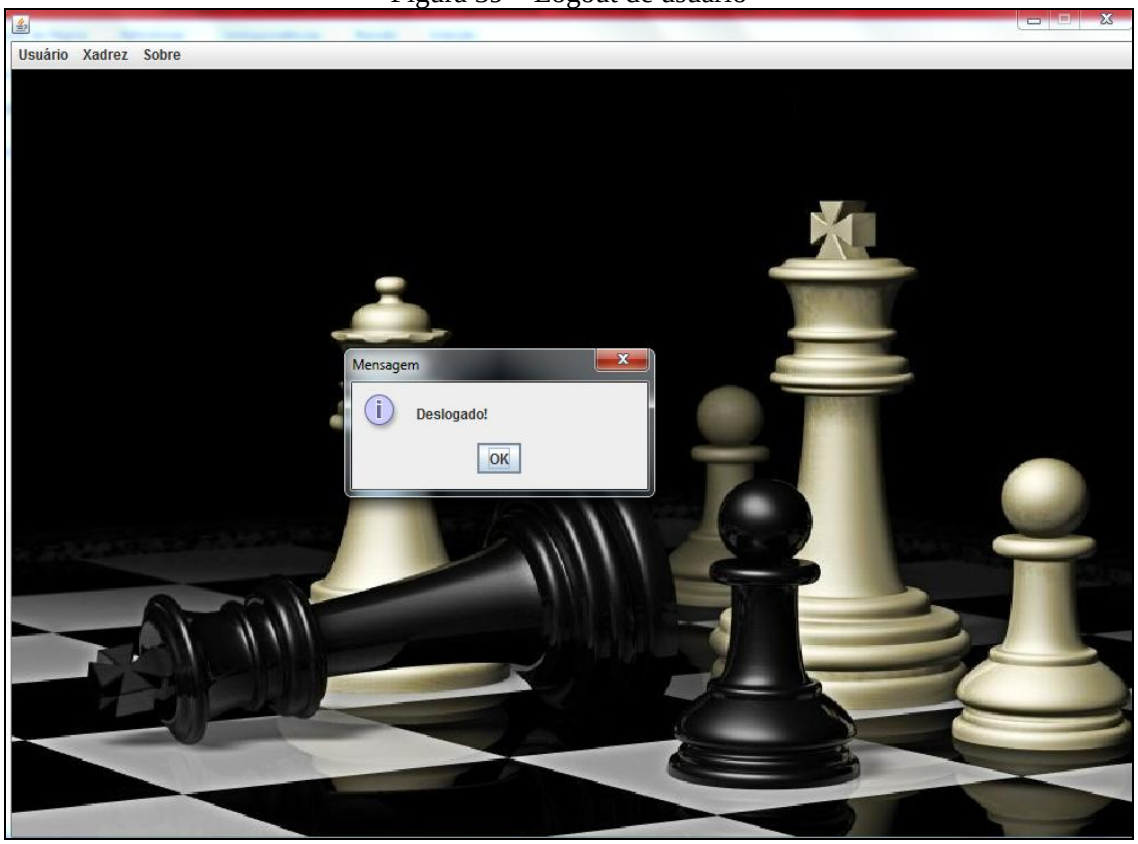
Figura 38 – *Login* de usuário



Após preencher os campos e clicar no botão *OK*, a ferramenta valida se as informações são válidas no banco de dados. Caso seja, efetua-se o *login* e, consequentemente, o submenu `Xadrez` fica disponível. Caso contrário, emitirá uma mensagem afirmando "*Login* inválido!".

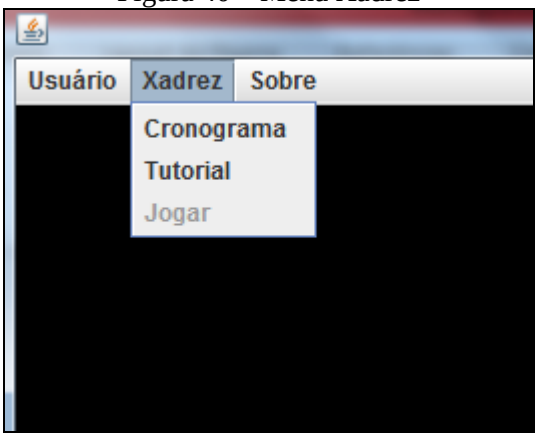
c) `logout`: este menu simplesmente emite uma mensagem afirmando que o usuário efetuou *logout* e encerra a execução da ferramenta.

Figura 39 – Logout de usuário



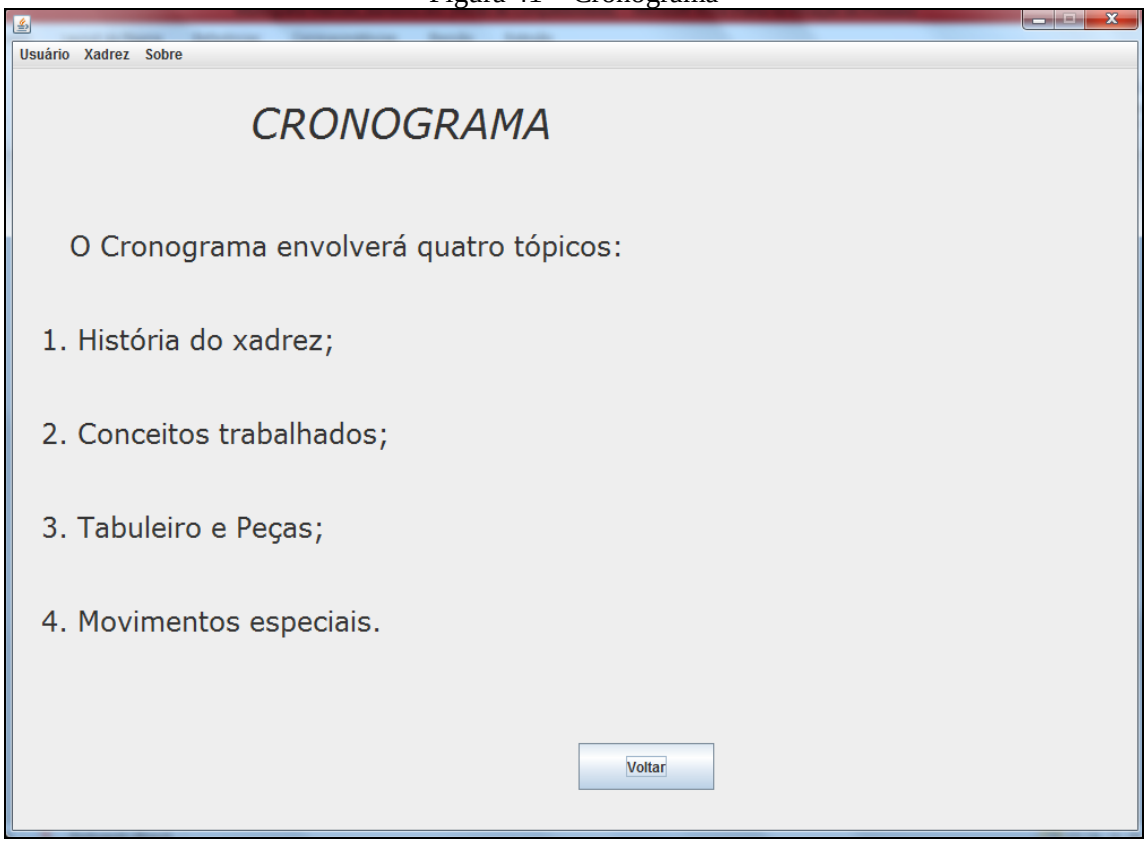
O menu `Xadrez` está disponível apenas para usuários que estejam logados. Nele, há três submenus (Figura 40):

Figura 40 – Menu Xadrez



`Cronograma`: este submenu mostra ao usuário os conteúdos que este terá à disposição para seu aprendizado do jogo de xadrez. Na Figura 41 tem-se a interface gráfica respectiva.

Figura 41 – Cronograma



`Tutorial`: este submenu possui quatro abas, uma sobre cada assunto, e em cada uma delas tem um botão para acessar o conteúdo/questionário respectivo. Nas Figura 42, 43, 44 e 45 têm-se as interfaces gráficas descritas;

Figura 42 – Tutorial História

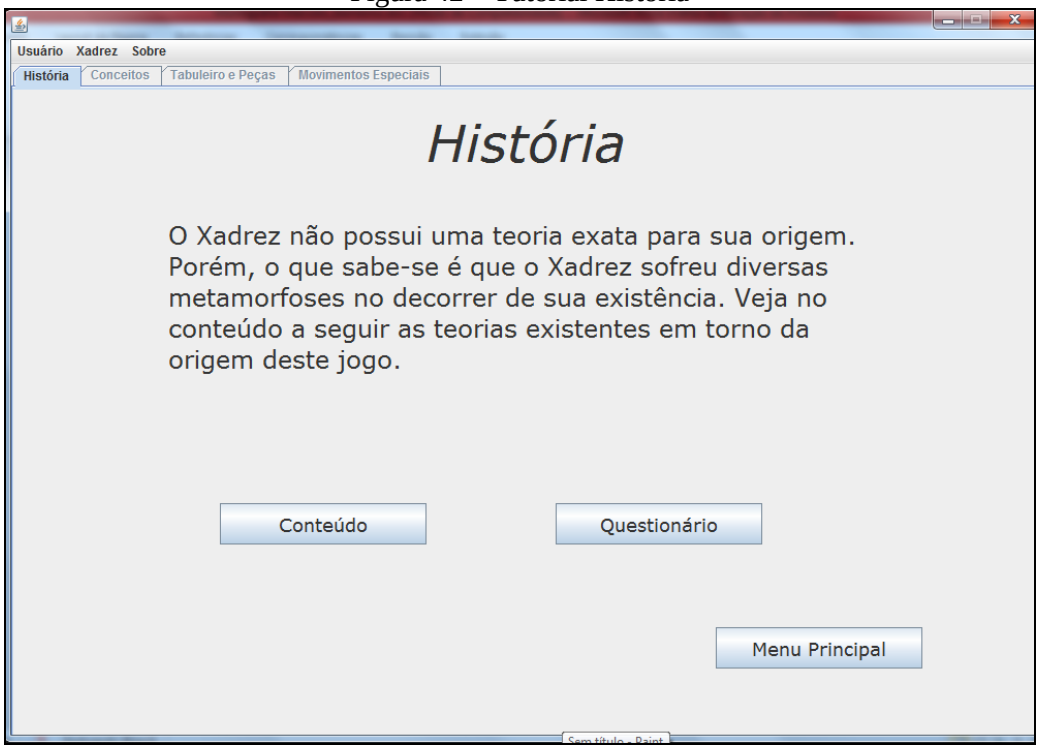


Figura 43 – Tutorial Conceitos

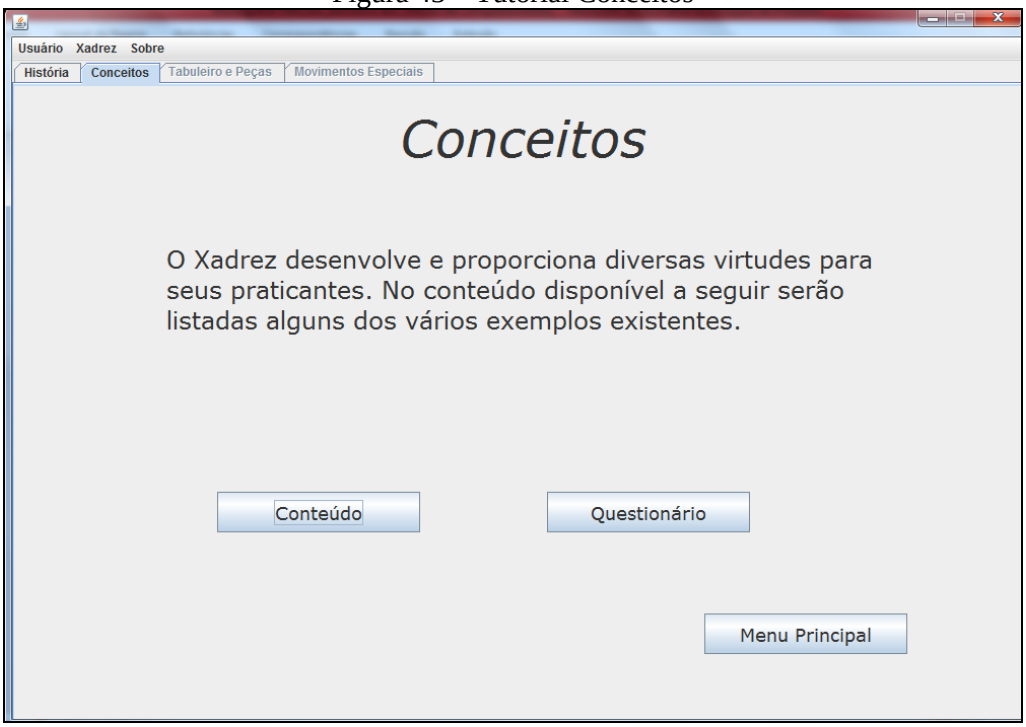


Figura 44 – Tutorial Tabuleiro e Peças

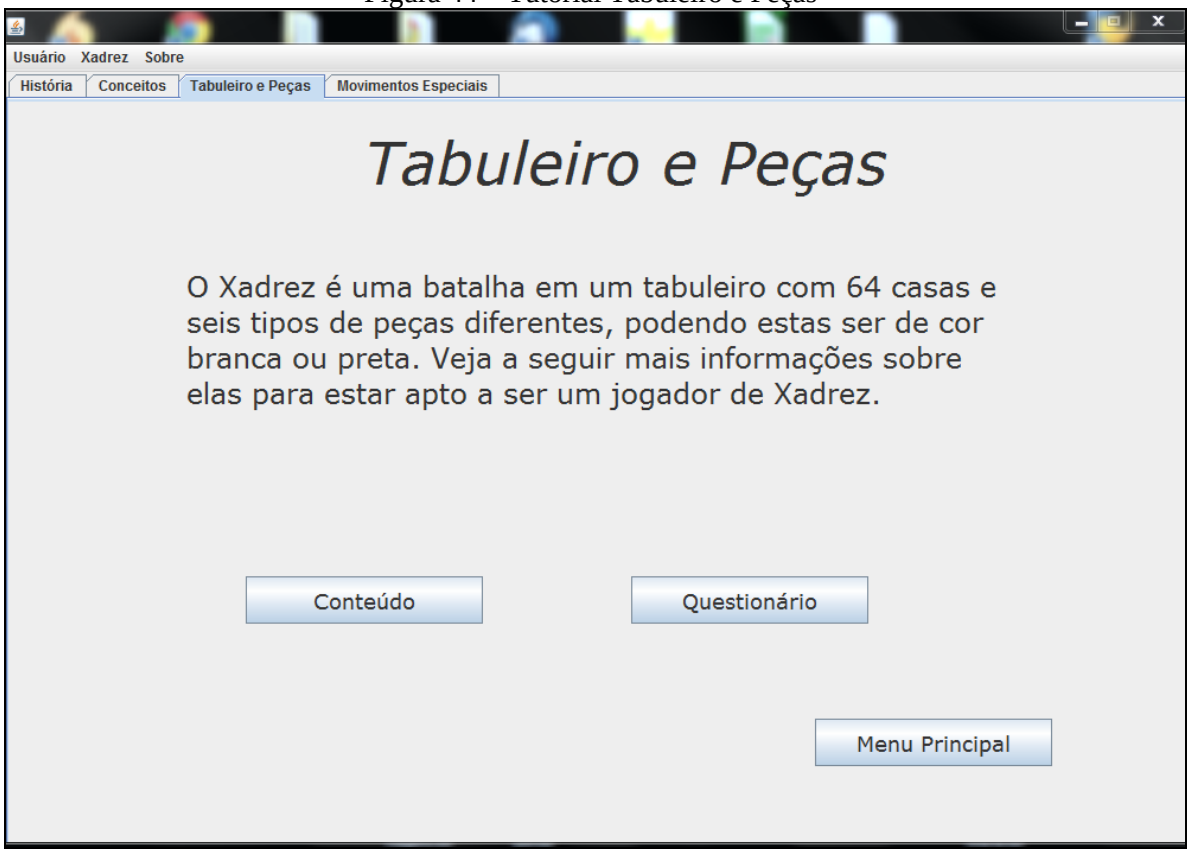
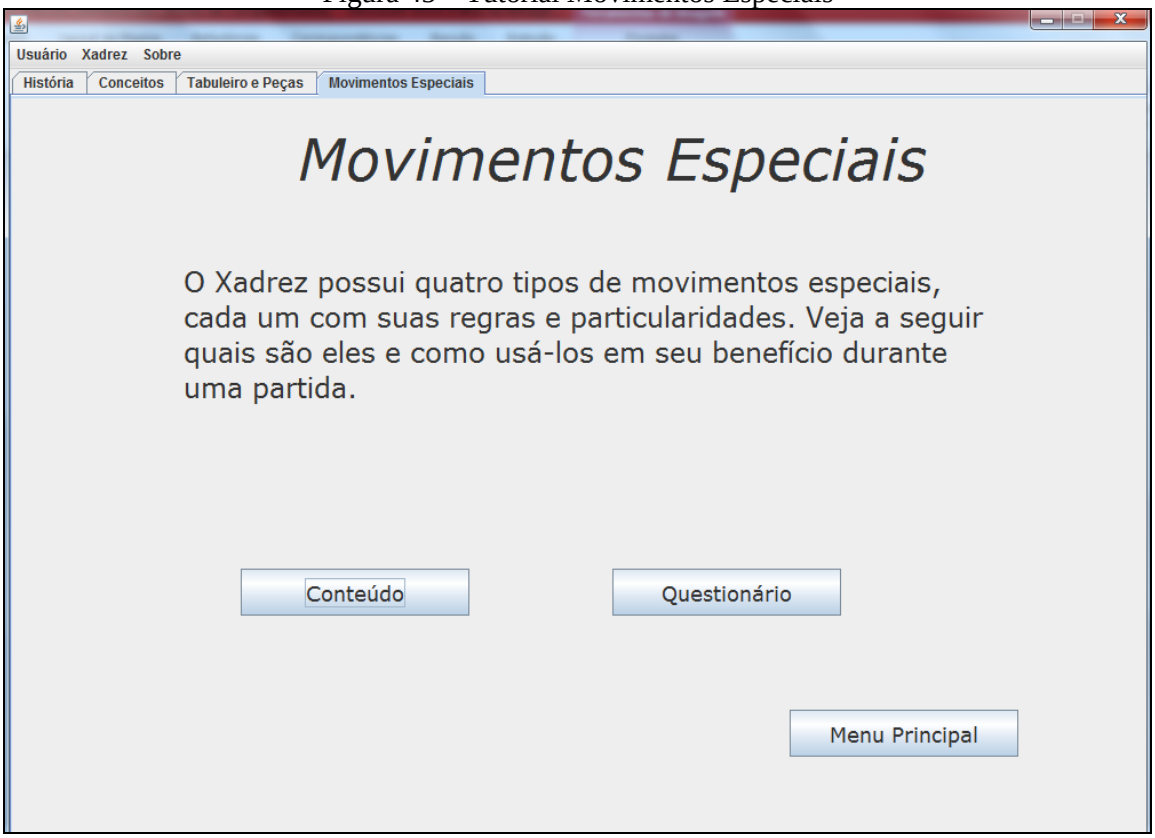


Figura 45 – Tutorial Movimentos Especiais



Observando a Figura 42, pode-se observar que as próximas abas só estarão disponíveis após a realização e aprovação do questionário em ordem sequencial. Todas as telas têm o mesmo comportamento: o botão `Conteúdo` abrirá o respectivo conteúdo em um arquivo externo PDF, enquanto que o questionário abrirá na própria aplicação uma tela com seis perguntas, sendo necessário o usuário acertar no mínimo três respostas para prosseguir para a próxima unidade. Na Figura 46 tem-se uma imagem do PDF que mostra o conteúdo de história e na Figura 47 tem-se o questionário de história respondido.

Figura 46 – Conteúdo História

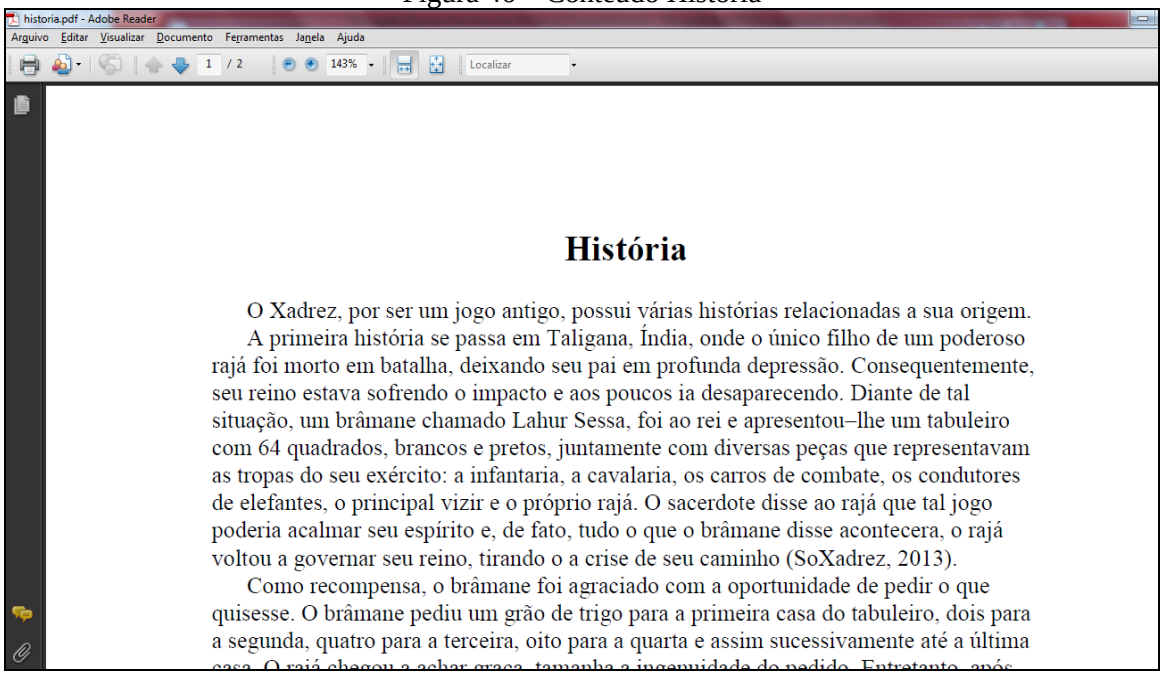
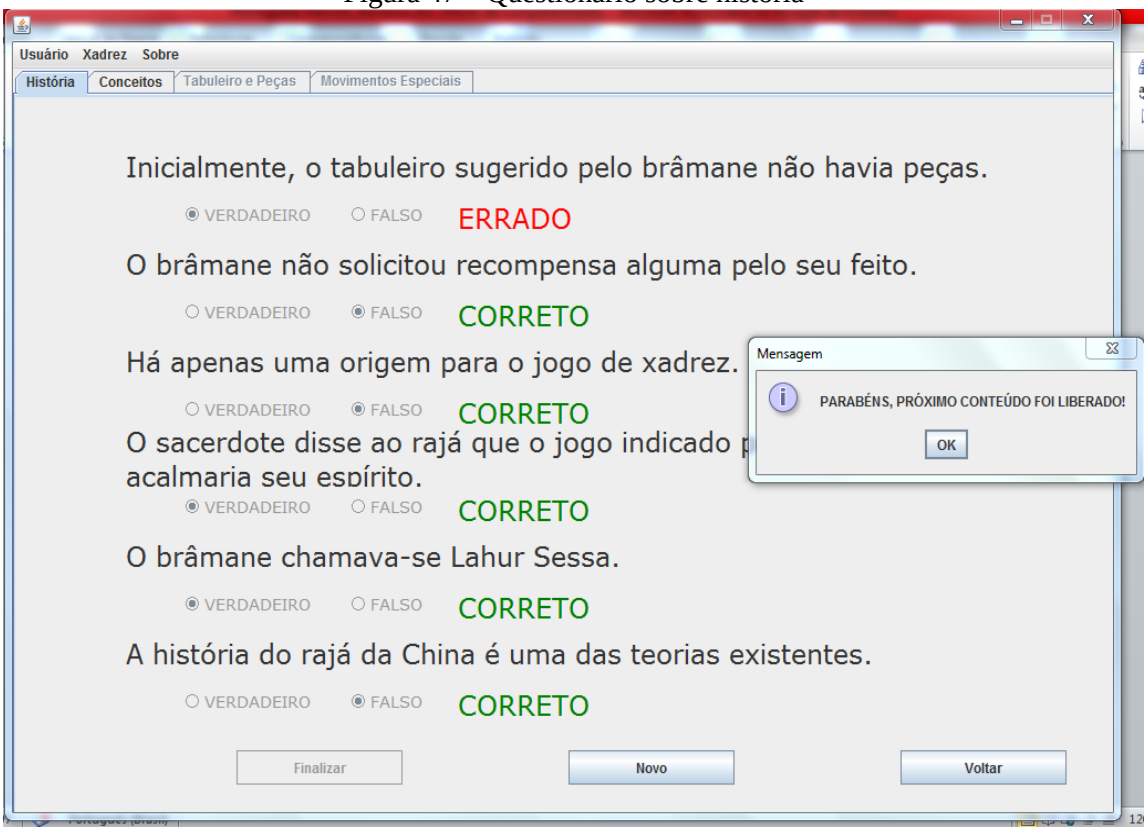


Figura 47 – Questionário sobre história



Na Figura 48 tem-se uma imagem do PDF que mostra o conteúdo de conceitos e na Figura 49 tem-se o questionário de conceitos respondido;

Figura 48 – Conteúdo conceitos

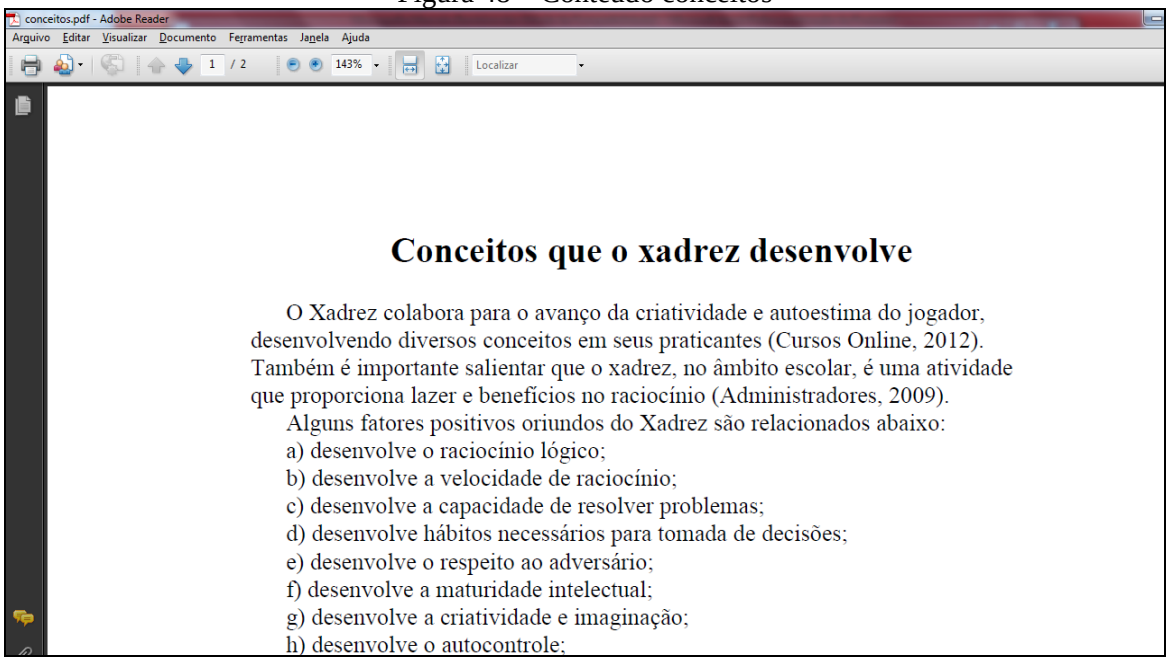
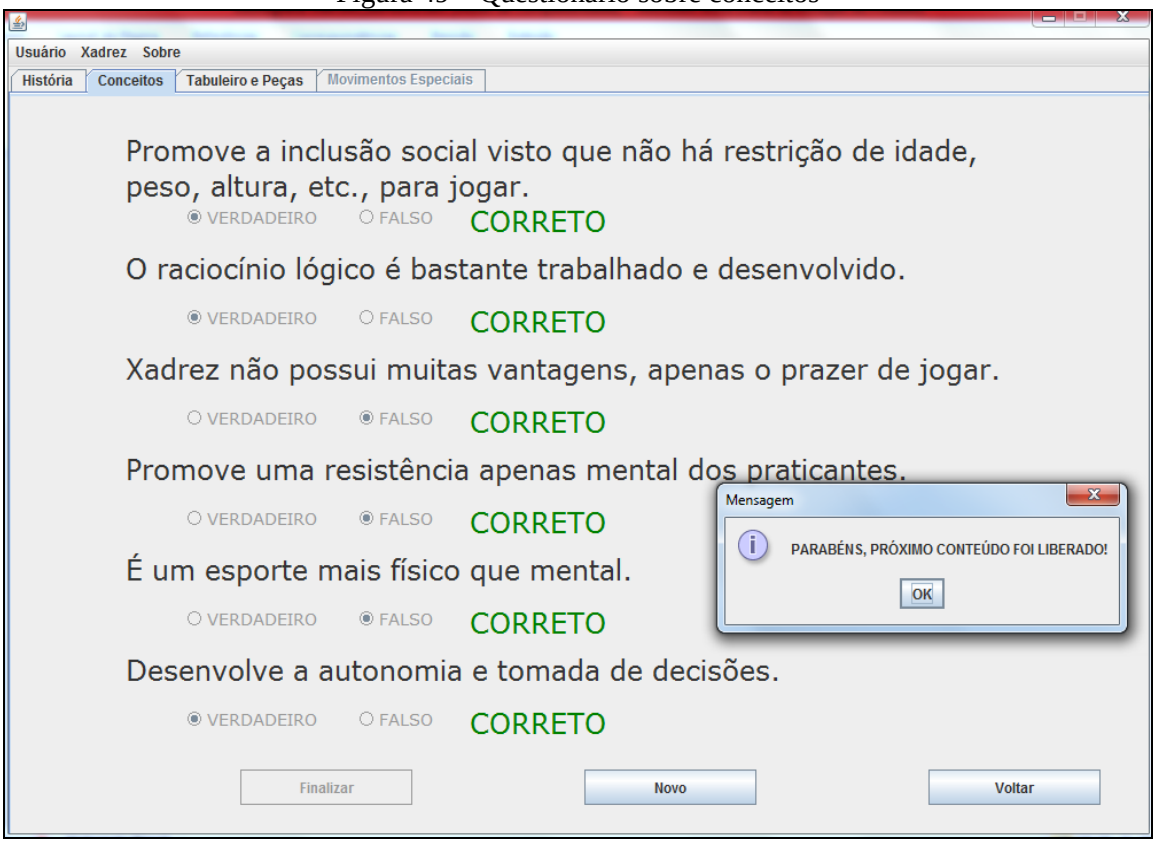


Figura 49 – Questionário sobre conceitos



Na Figura 50 tem-se uma imagem do PDF que mostra o conteúdo de tabuleiro e peças e na Figura 51 tem-se o questionário de tabuleiro e peças respondido;

Figura 50 – Conteúdo tabuleiro e peças

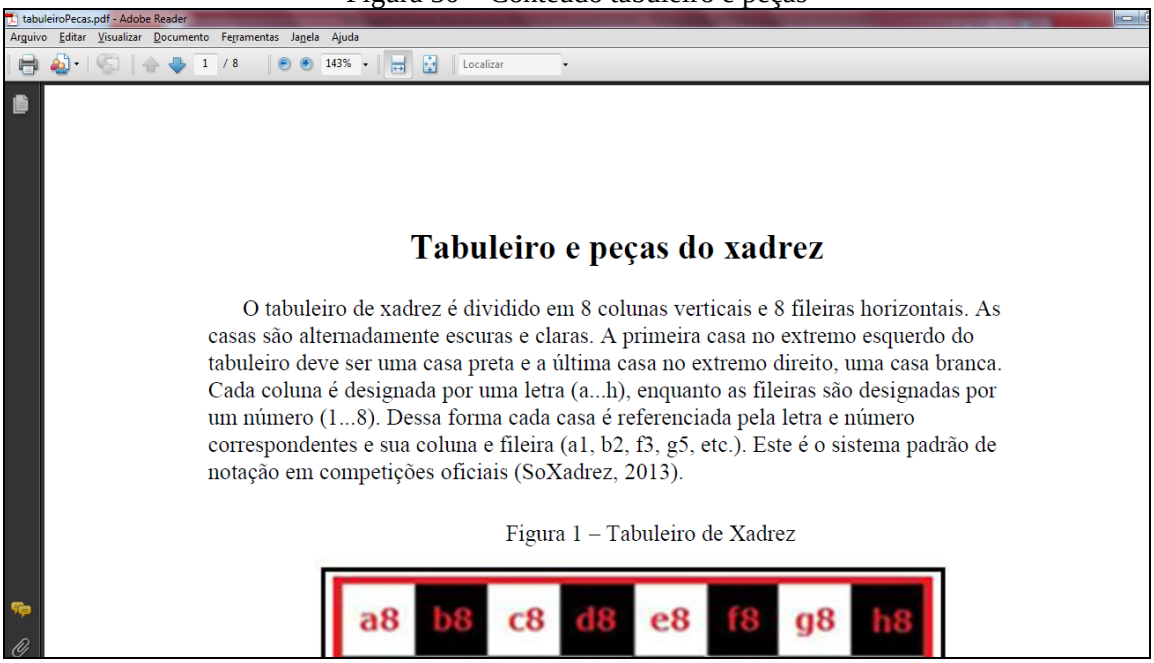
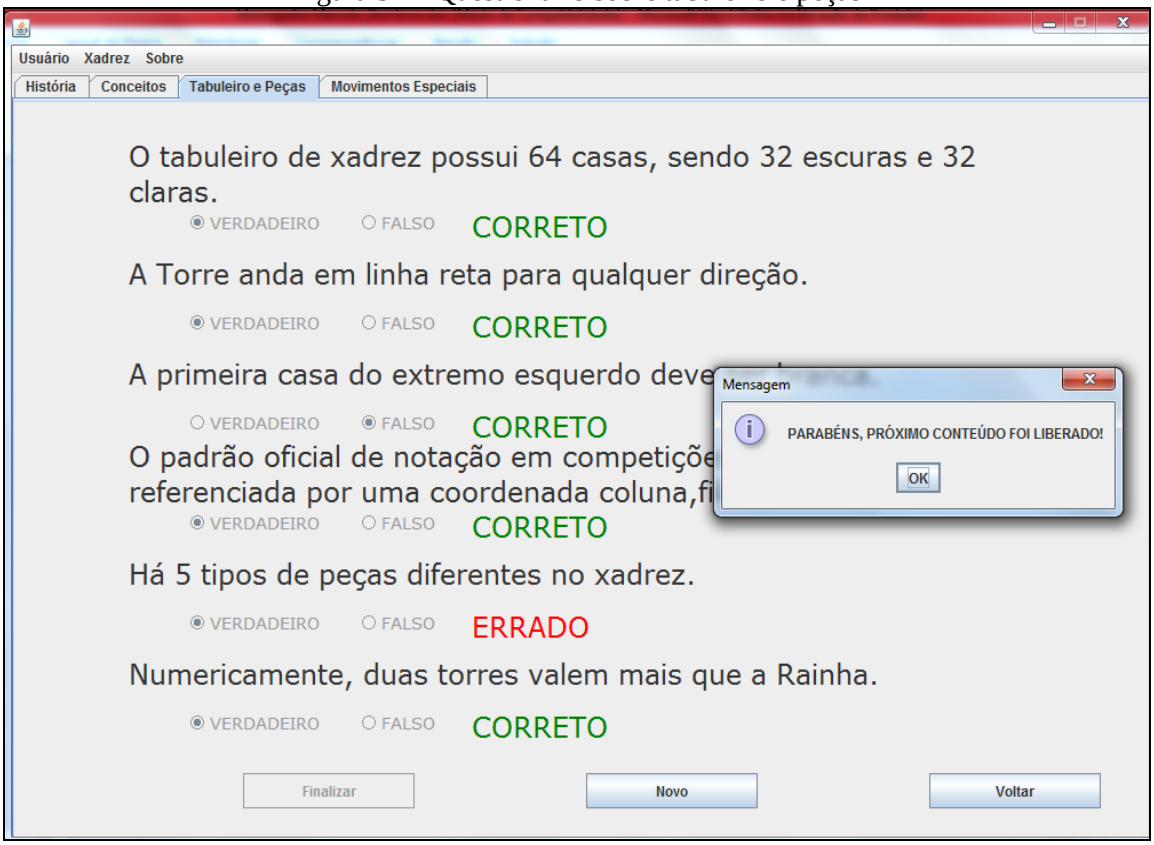


Figura 51 – Questionário sobre tabuleiro e peças



Por fim, na Figura 52 tem-se uma imagem do PDF que mostra o conteúdo de movimentos especiais e na Figura 53 tem-se o questionário de movimentos especiais respondido;

Figura 52 – Conteúdo movimentos especiais

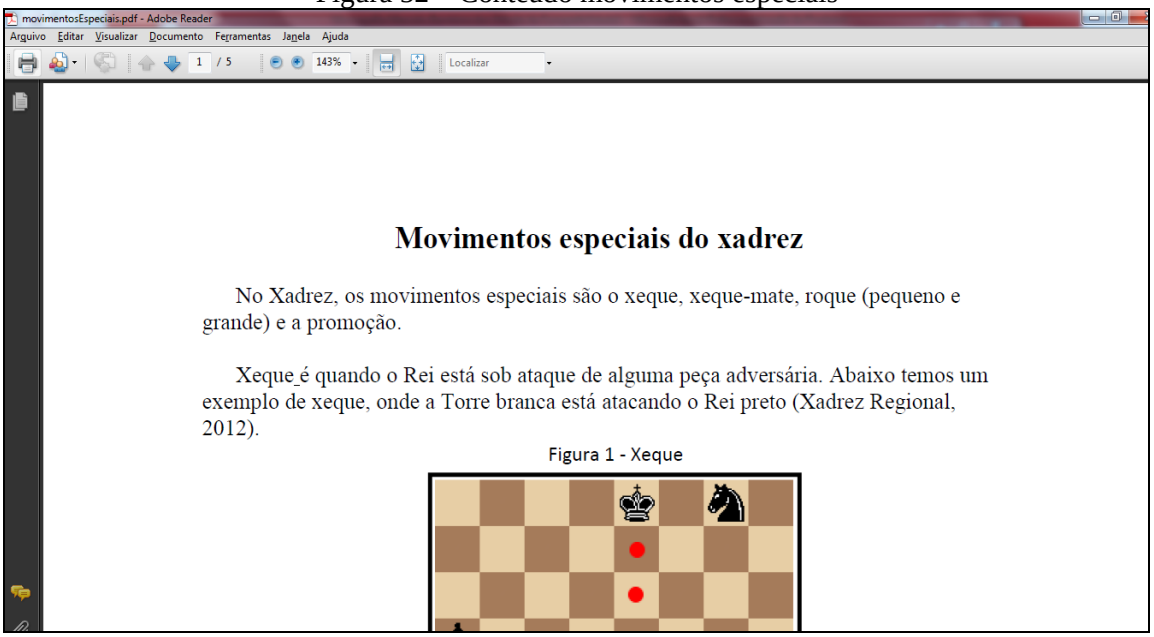
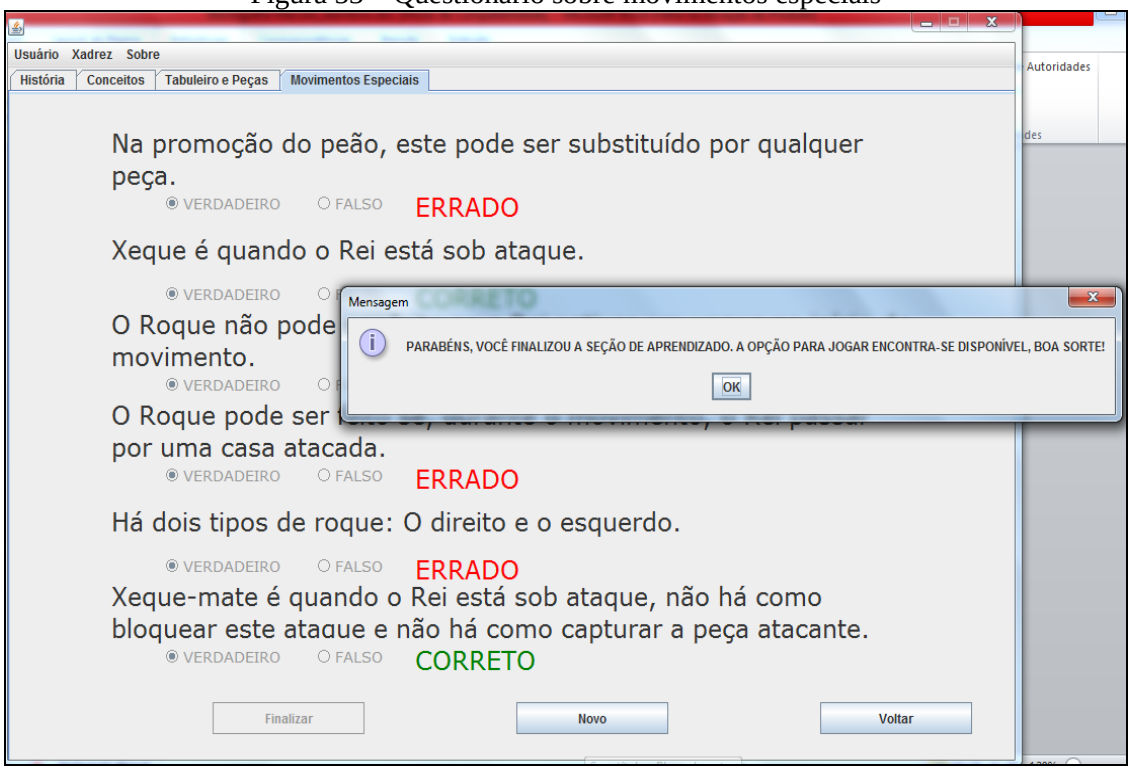


Figura 53 – Questionário sobre movimentos especiais



`Jogar`: possibilita para o usuário jogar contra uma *engine*, sendo habilitado após o usuário concluir todos os questionários do subitem anterior. Na Figura 54 tem-se a interface gráfica correspondente. Os *ratings* podem ser vistos abaixo dos nomes dos jogadores. Os algoritmos estão todos implementados no tabuleiro, e são executados a cada movimento efetuado. Por fim, a área do retângulo preto é responsável por marcar os lances efetuados na partida.

Figura 54 – Submenu Jogar

3.6 RESULTADOS E DISCUSSÕES

No Quadro 28 encontra-se uma comparação entre os trabalhos correlatos e o trabalho desenvolvido.

Quadro 28 – Características dos trabalhos relacionados

características / trabalhos relacionados	Chessmaster (2014)	Chessimo (2007)	Fritz (2014)	Ferramenta proposta
foco didático	Sim	Sim	Não	Sim
permite jogar contra a máquina	Sim	Não	Sim	Sim
gratuito	Não	Não	Não	Sim

Conforme pode ser observado no Quadro 26, o diferencial do trabalho proposto em relação aos correlatos é a aplicação de uma metodologia de ensino de xadrez.

A metodologia de Goulart e Frei (2004) utilizada como base mostrou-se compatível com o cenário proposto, facilitando a divisão dos conteúdos na seção de tutorial e na confecção dos conteúdos e questionários da ferramenta para o aprendizado do xadrez e, consequentemente, ajudando a alcançar um dos objetivos principais deste trabalho. Em futuras versões da ferramenta esta metodologia será utilizada de forma mais abrangente, explorando outras técnicas de ensino, como interações com os alunos com base em

revisões conceituais durante os jogos. A linguagem de programação utilizada forneceu todos os recursos necessários para a integração desta metodologia na ferramenta.

Em relação ao jogo de xadrez, as regras foram testadas com usuários que tinham pouco ou nenhum conhecimento em xadrez e mostraram-se funcionais. A interface gráfica ficou adequada à proposta do trabalho, bem como os outros recursos visuais da ferramenta.

No que diz respeito à implementação da *engine*, o algoritmo Poda Alfa-Beta mostrou-se eficiente e alcançou os objetivos propostos, conseguindo diminuir a árvore de movimentos, otimizando assim a busca que o algoritmo Minimax efetua sobre ela. Porém, a baixa performance do algoritmo Minimax ficou evidente. Por se tratar de um jogo com milhares de possibilidades de combinações de jogadas, a árvore com os movimentos gerados é gigantesca, tornando-se necessária a implementação de uma biblioteca de aberturas de xadrez, juntamente com conceitos e estratégias de meio-jogo e final de jogo, a fim de poupar a máquina de calcular jogadas óbvias.

No decorrer da implementação foram efetuados diversos testes de desempenho na ferramenta. Sem a implementação do algoritmo Poda Alfa-Beta, verificou-se que o tempo de execução do algoritmo Minimax tornava-se mais custoso, demorando cerca de 2 segundos para executar em árvores com nível 4. Ao aplicar o algoritmo Poda Alfa-Beta o tempo de execução levou fração de segundos para uma árvore de mesmo nível.

4 CONCLUSÕES

Este trabalho propôs o desenvolvimento de uma ferramenta voltada ao jogo de xadrez, onde os principais objetivos eram proporcionar o ensino e aprendizado deste jogo e relacionar o xadrez com a IA através dos algoritmos Minimax e Poda Alfa-Beta.

Para confecção da *engine*, o uso do algoritmo Poda Alfa-Beta mostrou-se bastante funcional, ajudando na melhoria de performance do algoritmo Minimax e agilizando o processamento de procura de lance por parte da *engine*.

A maior limitação da ferramenta foi a implementação do algoritmo Minimax pois, devido ao tamanho das árvores de movimentos geradas, a performance foi mais baixa do que o esperado. Para melhorar o desempenho do algoritmo Minimax seria interessante técnicas como inserir uma biblioteca de aberturas de xadrez, implementar conceitos óbvios de jogo para poupar a *engine* de perder tempo em tais situações e utilizar *threads* para agilizar a construção da árvore de movimentos, para que o Minimax possa começar a executar antes, ganhando assim fração de segundos importantes, que elevado a quantidade de lances proporciona um ganho de performance considerável.

As interfaces gráficas ficaram visualmente atrativas e obtiveram um retorno positivo por parte dos usuários que testaram a ferramenta graças às bibliotecas nativas do Java, `JFrame` e `JPanel`, que ofereceram todo o recurso necessário. É importante destacar que a linguagem mostrou-se compatível com a implementação dos algoritmos de IA, bem como com a comunicação da ferramenta com o banco de dados relacional *ORACLE XE*.

Por fim, este trabalho apresentou funcionalidades que podem servir de base para futuros trabalhos em áreas como inteligência artificial e setores pedagógicos.

4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho, têm-se:

a) aplicar outros algoritmos de IA para melhorar a performance da *engine*;
b) implementar seções mais focadas no aprendizado do jogo, como resoluções de diagramas ou estudos de estratégias de jogo;
c) implementar uma árvore de aberturas e estratégias de final de jogo a fim de otimizar os algoritmos nos lances iniciais e finais, respectivamente;
d) estudar uma forma de criar os ramos das árvores com *threads*;
e) abranger novas técnicas de ensino na ferramenta que sejam mais interativas com os usuários.

REFERÊNCIAS

ASSIM SE FAZ. **Como ensinar xadrez a crianças.** [S.l.], [2012]. Disponível em: <http://www.assimsefaz.com.br/sabercomo/como-ensinar-xadrez-a-criancas>. Acesso em: 13 mar. 2015.

BARBOSA, Soraia T.; VEIGA, Janaína; CARVALHO, Carlos V. A. Estudo do uso de técnicas de Inteligência Artificial em jogos 2D**. Revista Eletrônica TECCEN**, Vassouras, v. 5, n. 1, p. 5-20, jan./abr., 2012.

CANAL XADREZ. **O ensino de xadrez como ferramenta pedagógica.** [S.l.], [2009]. Disponível em: <http://xadrez.altervista.org/xadrez/ensinoxadrez.htm>. Acesso em: 18 mar. 2015.

CHESSBASE. **Deep Fritz 14 - english version.** [S.l.], [2014]. Disponível em: <http://shop.chessbase.com/en/products/deep_fritz_14_english>. Acesso em: 12 abr. 2015.

CHESSIMO. **Improve your chess.** [S.l.], [2007]. Disponível em: <http://www.chessimo.com/home>. Acesso em: 11 abr. 2015.

CLUBE DE XADREZ. **Valor relativo das peças.** [S.I], [2012]. Disponível em: <http://www.clubedexadrez.com.br/portal/torre21/iniciantes/valor.htm> Acesso em: 07 mar. 2015.

CLUBE DE XADREZ ONLINE. **O que é xadrez?.** [S.l.], [2001]. Disponível em: <http://www.clubedexadrezonline.com.br/artigo.asp?doc=878>. Acesso em: 12 mar. 2015.

FERREIRA, Lincon O. **A utilização de softwares para o auxílio do jogo de xadrez.** 2012. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Centro Universitário Barão de Mauá, Ribeirão Preto.

FERREIRA, Paulo A.; OLIVEIRA, Rilson M. Trabalho de implementação - **Jogo Reversi.** Lavras, [2007]. Disponível em: <http://www2.dc.ufscar.br/~paulo_junior/reversi/relatorio.pdf>. Acesso em: 17 mar. 2015

FIDE. **World Chess Federation.** [S.l.], [2015]. Disponível em: < http://www.fide.com/>. Acesso em: 01 jul. 2015.

FRANÇA, Cristiano S. **O xadrez como ferramenta pedagógica para as aulas de educação física escolar.** 2012. 31 f. Trabalho de Conclusão de Curso (Especialização em Educação Física Escolar) – Centro de Ciências Biológicas e Saúde, Universidade Estadual da Paraíba, Campina Grande.

GOMES, Antônio S. **Projeto xadrez escolar on line.** 2010. 11 f. Trabalho de Metodologia Científica (Curso de Licenciatura em Computação) – Centro de Ciências Exatas e Naturais, Universidade Federal Rural da Amazônia, Amazônia.

GOULART, Edson; FREI, Fernando. **O ensino de xadrez para crianças de 3º e 4º séries do ensino fundamental.** São Paulo, [2004]. Disponível em: <http://www.unesp.br/prograd/PDFNE2004/artigos/eixo10/oensinodexadrex.pdf>. Acesso em: 03 mar. 2015.

HOUDINI. **Houdini Chess Engine.** [S.l.], [2013]. Disponível em: <http://www.cruxis.com/chess/houdini.htm>. Acesso em: 11 abr. 2015.

IBM. **Deep Blue. [**S.l.], [2014]. Disponível em: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/transform/>. Acesso em: 11 abr. 2015.

JEIKS. **Minimax.** [S.l.], [2011]. Disponível em: <http://jeiks.net/wp-content/uploads/2013/05/MiniMax.pdf>. Acesso em: 16 mar. 2015.

MEDEIROS, Israel C. S.; QUARANTA, Francisco N. Introdução dos computadores para o xadrez contemporâneo. In: CONGRESSO DE INICIAÇÃO CIENTÍFICA, 9., 2013, Rio Grande do Norte. **Anais...** Rio Grande do Norte: UFRN, 2013. p. 1-10.

NAVEGA, Sérgio. Inteligência Artificial, **Educação de crianças e o cérebro humano.** Niterói, [2000]. Disponível em: <http://www.intelliwise.com/reports/p4port.htm>. Acesso em: 17 mar. 2015

QUINET, Marcos. **Inteligência Artificial.** Niterói, [2011]. Disponível em: <http://www.professores.uff.br/mquinet/07_IA.pdf>. Acesso em: 14 abr. 2015

RUSSEL, Stuart. NORVIG, Peter. Artificial Intelligence, a modern approach.(negrito) 1. ed. Englewood: New Jersey, 1995. 5 p.

SOCIEDADE DOS MESTRES DE XADREZ. **Xadrez, Computação e Internet.** [S.l.], [2012]. Disponível em: <http://sociedadedosmestresdexadrez.blogspot.com.br/p/xadrez-computacao-e-internet.html>. Acesso em: 16 mar. 2015.

STOCKFISH. **Strong open source chess engine.** [S.l.], [2014]. Disponível em: <https://stockfishchess.org/>. Acesso em: 10 abr. 2015.

SÓ XADREZ. **Tabuleiro.** [S.I], [2013a]. Disponível em: <http://www.soxadrez.com.br/conteudos/tabuleiro_pecas/> Acesso em: 07 mar. 2015.

SÓ XADREZ. **Notação enxadrística.** [S.I], [2013b]. Disponível em: <http://www.soxadrez.com.br/conteudos/notacao/> Acesso em: 18 mar. 2015.

TIPPY CHESS. **Poda Alfa-Beta.** [S.l.], [2012a]. Disponível em: <http://www.tippychess.com/portugues/conteudo.asp?titulo=poda_alfa_beta>. Acesso em: 18 out. 2014.

TIPPY CHESS. **Minimax.** [S.l.], [2012b]. Disponível em: <http://www.tippychess.com/portugues/conteudo.asp?titulo=minimax>. Acesso em: 18 out. 2014.

TIPPY CHESS. **Panorama atual.** [S.l.], [2012c]. Disponível em: <http://www.tippychess.com/conteudo.asp?titulo=panorama_atual>. Acesso em: 18 out. 2014.

UBISOFT. **Chessmaster, the art of learning.** [S.l.], [2014]. Disponível em: <http://chessmaster.uk.ubi.com/xi/index.php>. Acesso em: 10 abr. 2015.

WIKIPEDIA. **Minimax.** [S.l.], [2014]. Disponível em: <http://pt.wikipedia.org/wiki/Minimax>. Acesso em: 02 abr. 2015.

XADREZ REGIONAL. **Como jogar xadrez.** [S.I], [2012]. Disponível em: <http://www.xadrezregional.com.br/e04.html> Acesso em: 04 mar. 2015.