FAESA CENTRO UNIVERSITÁRIO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLEBER JUNIO SOUZA ABREU
WILIAM TRANCOSO DAL CIM JUNIOR

INTELIGÊNCIA ARTIFICIAL PARA O JOGO OTHELLO

VITÓRIA
2017

CLEBER JUNIO SOUZA ABREU

WILIAM TRANCOSO DAL CIM JUNIOR

INTELIGÊNCIA ARTIFICIAL PARA O JOGO OTHELLO

Trabalho de Conclusão do Curso de Graduação em Ciência da Computação apresentando FAESA Centro Universitário, sob orientação do profª. Cinthia Cristina Lucio Caliari.

VITÓRIA

2017

CLEBER JUNIO SOUZA ABREU
WILIAM TRANCOSO DAL CIM JUNIOR

Trabalho de Conclusão do Curso de Graduação em Ciência da Computação apresentando FAESA Centro Universitário, sob orientação do profª. Cinthia Cristina Lucio Caliari.

INTELIGÊNCIA ARTIFICIAL PARA O JOGO OTHELLO

BANCA EXAMINADORA

Cinthia Cristina Lucio Caliari
Orientadora

Elvira Padua Lovatte
Membro da Banca

Howard Cruz Roatti
Membro da Banca

VITÓRIA
2017

AGRADECIMENTOS

Agradecemos primeiramente a nossa orientadora Cinthia, por sua atenção, pelas suas correções e incentivos dedicado à elaboração deste trabalho. A FAESA, pelo seu corpo docente, direção e administração que contribuíram para nossa graduação. Agradecemos também ao nosso colega de turma mais próximo, Keoma Klaver Klain, pelos momentos de distração e por estar sempre acompanhando o desenvolvimento desse trabalho, com sugestões e incentivos.

Cleber Junio Souza Abreu e Wiliam Trancoso Dal Cim Junior

Primeiramente, agradeço a Deus por ter me dado saúde e força para superar as dificuldades. Aos meus pais, Cleber Geraldo de Abreu e Maria Odete Souza Abreu, pelo amor, incentivo, apoio incondicional e pela forma que me educaram. A minha irmã Kelen Dayane Souza Abreu, pelo apoio e incentivo nos meus estudos. A minha esposa, Fernanda de Paiva Picoli Tavares, pelo amor, companheirismo, incentivo, os momentos de carinho e descontração. Ao meu amigo, Wiliam Trancoso Dal Cim Junior, pela parceria e dedicação nesse trabalho, pelos momentos de distração e apoio ao longo do curso. A todos, colegas de turma, trabalho e amigos, que me incentivaram e participaram desta etapa da minha vida.

Cleber Junio Souza Abreu

Agradeço minha mãe, Eny Gonçalves Lemos e minha tia, Edir Gonçalves Lemos, por estarem sempre ao meu lado, me educando com muito amor e responsabilidade. A minha esposa, Leticia De Morais Rocha, e ao meu filho, Pedro Rocha Dal Cim, pelo amor e incentivo. Ao meu sogro, Olival Rocha Filho, por ter me incentivado e orientado em minha jornada acadêmica. Ao Cleber Junio Souza Abreu, parceiro do TCC e da vida, pelo empenho nesse trabalho, pelas conversas e pela sua amizade. Agradeço a todos os meus familiares, amigos, professores, colegas de classe e trabalho que me ajudaram, incentivaram ou torceram para o meu sucesso

Wiliam Trancoso Dal Cim Junior

RESUMO

Este trabalho apresenta o jogo Othello e o desenvolvimento de um ambiente virtual em Java para o jogo, possibilitando a realização de partidas entre jogadores humanos e jogadores artificiais. Com o objetivo de cria um jogador, que apresente jogadas inteligentes de acordo com o nível do usuário, de forma que as partidas sejam desafiantes paras jogadores de níveis experiência diferentes no jogo. Foram implementados dois jogadores artificiais, um jogador que faz jogadas aleatórias e outro que toma decisões mais inteligentes utilizando o algoritmo MINIMAX.

Palavra-chave: Othello; MINIMAX; Inteligência Artificial.

ABSTRACT

This work presents the game Othello and the development of a virtual environment in Java for the game, allowing the realization of matches between human players and artificial players. With the goal of creating a player who presents intelligent moves according to the user level, so matches are challenging for players of different experience levels in the game. Two artificial players were implemented, one player making random plays and one making smarter decisions using the MINIMAX algorithm.

Keyword: Othello; MINIMAX; Artificial intelligence.

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Uma crítica que é feita entre os jogadores de jogos digitais de tabuleiros é o fato de que as jogadas da máquina, embora inteligentes, passam a ser previsíveis com o passar do tempo e a experiência do jogador.

Dessa forma, um desafio para os criadores é propor um jogo mais desafiante e atrativo, que não tenha um nível de dificuldade estática. Um jogo que possibilite ao jogador escolher o nível de dificuldade que deseja jogar.

A ideia desse projeto - “Inteligência artificial para o jogo Othello”, é desenvolver um agente inteligente para o jogo de tabuleiro Othello, e que o jogador é desafiado com as jogadas inteligentes da máquina.

Os jogos digitas vêm evoluindo a cada ano, mas mesmo contendo diversos atrativos, um deles é vencer as jogadas inteligentes da máquina, observa-se que, no modo de um único jogador humano, dependendo da habilidade do jogador, a dificuldade do jogo pode ser superada rapidamente e assim ele não será mais desafiante. O oposto também ocorre, ou seja, o jogo pode ter uma dificuldade exagerada inicialmente e não consegue cativar as pessoas por isso. Assim, o ideal seria que a dificuldade do jogo acompanhasse o nível de evolução do jogador e possuísse um nível máximo realmente desafiante.

O mercado dos jogos está em constante crescimento, e com ele a cobrança por parte dos jogadores que buscam um maior realismo, não só no visual, mas também na interação com o ambiente do jogo. Personagens controlados pelas máquinas são previsíveis e isso não condiz com o mundo real. Para aumentar o realismo seria interessante o ajuste de dificuldade de acordo com o usuário.

As técnicas de inteligência artificial estão sendo cada vez mais aprimoradas, com sua utilização expandindo para vários campos. Em muitos casos, com poder de resolução maior que a de humanos, a aplicação com maior frequência de inteligência artificial em várias áreas, além dos jogos, proporcionará grandes benefícios à humanidade.

No capítulo 2 será apresentado todo o referencial teórico utilizado, conceituando o jogo de tabuleiro Othello, a Inteligência Artificial, os algoritmos MINIMAX e Q-

LEARNING. No capítulo seguinte, será descrita a metodologia utilizada, por meio dos diagramas de casos de uso e de classe, utilizados para o desenvolvimento dos protótipos. No capítulo 4, serão apresentados os protótipos desenvolvidos. No quinto capitulo, serão relatados os resultados do protótipo final. Finalmente, no último capítulo, serão apresentadas as conclusões e perspectivas para trabalhos futuros.

2. REFERENCIAL TEÓRICO

Neste capítulo é apresentado o jogo de tabuleiro Othello e suas regras, o conceito de IA e sua aplicação em jogos. Em seguida é abordada a técnica de inteligência artificial, que será utilizada nesse trabalho.

2.1 O JOGO DE TABULEIRO OTHELLO

Othello é um jogo de tabuleiro, criado por Goro Hasegawa em 1972 no Japão. Apesar de pouco conhecido no Brasil, existe campeonato brasileiro, realizado pela Federação Brasileira de Othello (FBO), e mundial realizado pela Federação Mundial de Othello (*World Othello Federation* - WOF) (FEDEREÇÃO BRASILEIRA DE OTHELLO, [201-?]).

Jogado em um tabuleiro 8 x 8 (oito por oito), possui 64 (sessenta e quatro) peças circulares, que são brancas de um lado e pretas do outro, cada cor representa um jogador (QUAGLIO, 2013, p. 32).

Com o tabuleiro tendo 64 (sessenta e quatro) casas, podendo ter três estados diferentes, a complexidade de espaço de estados do Othello é aproximadamente $3^{64} \approx 10^{64}$ estados possíveis. (CHUDASAMA; TRIPATHI; PRAJAPATI, 2014, p. 10, tradução nossa)

Na Figura 1, pode-se observar o tabuleiro do jogo comercializado pela fabricante de brinquedos Mattel (SILVA, 2011).

Figura 1 - Tabuleiro do jogo Othello

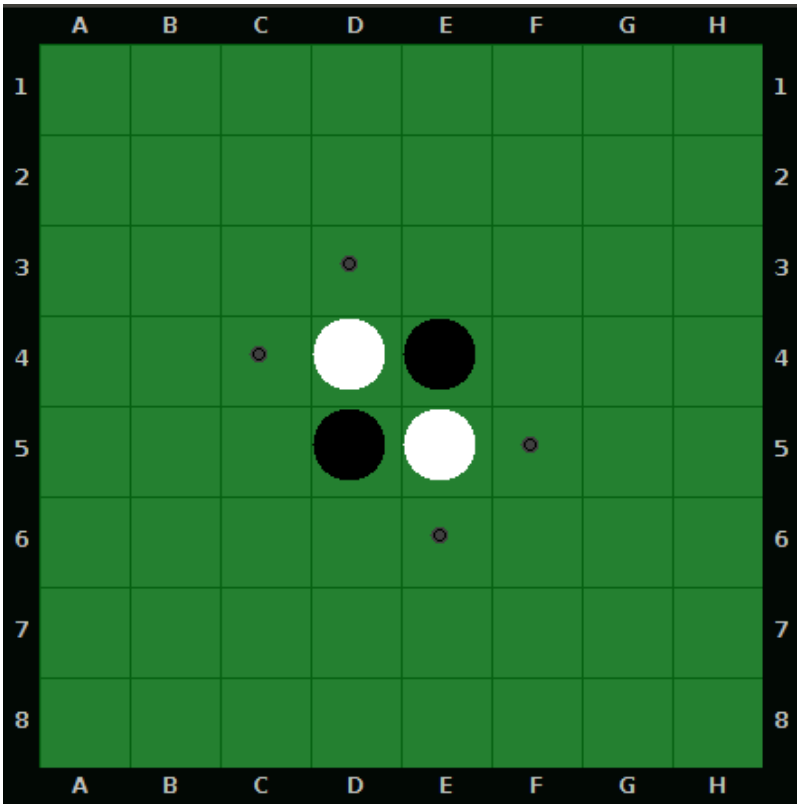
Fonte: SILVA, 2011.

É um jogo determinístico soma-zero, ou seja, a vitoria é pontuada como um, derrota como menos um e empate zero, no final necessariamente a soma resultará em zero. Ou seja, um jogador terá uma vitória e seu oponente uma derrota, não sendo possível a atribuição de derrota ou vitoria para os dois jogadores em uma partida. (QUAGLIO, 2013, p. 31).

Como a maioria dos jogos clássicos de tabuleiros, este é de informação perfeita, devido ao fato de que as informações da partida podem ser vistas por ambos jogadores a todo momento, contrastando por exemplo, com jogos de cartas, isso não acontece, já que um jogador não tem conhecimento das cartas que estão na mão do outro (QUAGLIO, 2013 p. 32).

A partida inicia com duas peças de cada jogador na região central do tabuleiro, como mostra Figura 2.

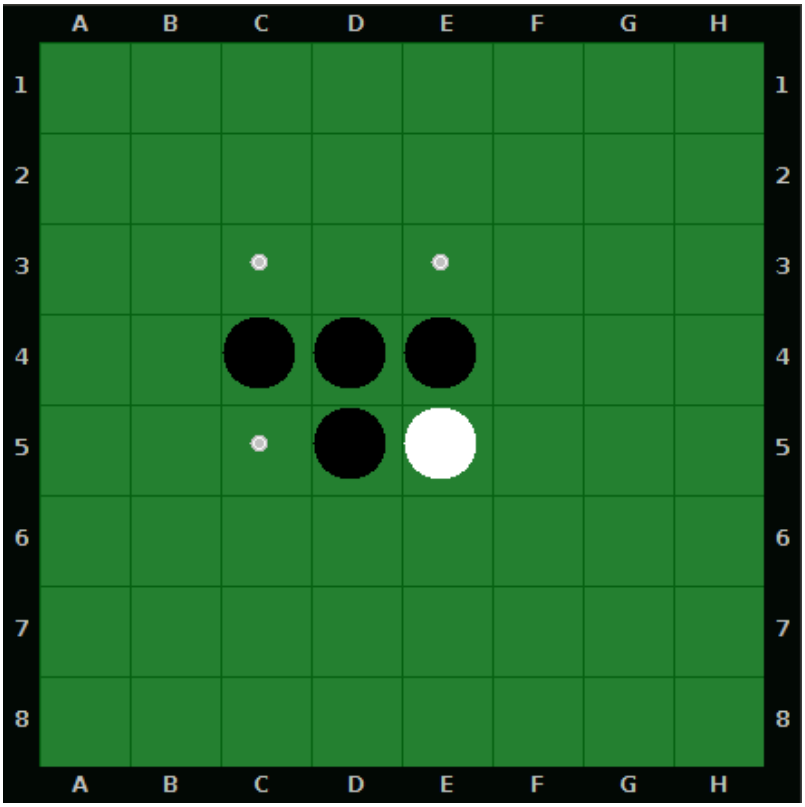
Figura 2 - Tabuleiro no início de partida.



As jogadas são intercaladas entre os jogadores A (discos pretos) e B (discos brancos), sendo necessário que a peça do jogador A deve ser inserida em uma posição vazia, cercando na horizontal, vertical ou diagonal pelo menos uma peça do jogador B. As peças brancas que ficaram entre as pretas são viradas, transformando-as em pretas. Caso não haja possibilidade de jogada que cerque peças do adversário, o jogador A cede a vez.

Os campos destacados na Figura 2, 3D, 4C, 5F e 6E, são as jogadas possíveis para o jogador dos discos pretos, ao inserir disco na posição 4C, a peça do adversário no campo 4D será convertido em preto, como mostra Figura 3.

Figura 3 - Tabuleiro com captura de peças



O jogo termina quando todo tabuleiro é preenchido ou quando não há jogadas possíveis para nenhum jogador. Vence quem terminar com maior número de peças, mas caso ambos os jogadores terminem o jogo com a mesma quantidade de peças, ocorre o empate.

2.2 INTELIGÊNCIA ARTIFICIAL

De acordo com Teixeira (1998, p. 11) o termo *Artificial Intelligence* (A.I. – I.A. em português) foi usado pela primeira vez em 1956 por John McCarthy, em uma conferência que reuniu os maiores especialistas em Ciência da Computação da época.

Para Nikolopoulos (1997), citado por Sellitto (2002, p. 364):

> “A Inteligência Artificial é um campo de estudos multidisciplinar, originado da computação, da engenharia, da psicologia, da matemática e da cibernética, cujo principal objetivo é construir sistemas que apresentem comportamento inteligente e desempenhem tarefas com um grau de competência equivalente ou superior ao grau com que um especialista humano as desempenharia”.

"Não existe uma definição para inteligência artificial (IA), mas várias. Basicamente, IA é fazer com que os computadores pensem como os seres humanos ou que sejam tão inteligentes quanto o homem" (MÓDOLO, [201-?], citado por SATO, [201-?]).

Pensando dessa forma, LUGER (2014, p.321) define IA como a capacidade de aprender, a qual deve fazer parte de qualquer sistema que reivindique possuir inteligência em um sentido geral. Mas, o campo da inteligência artificial tenta não apenas compreender, mas também construir entidades inteligentes. (RUSSELL; NORVIG, 2013, p. 29).

2.3 INTELIGÊNCIA ARTIFICIAL PARA JOGOS

“Para os desenvolvedores de jogos eletrônicos, as aplicações computacionais de IA e o significado do termo IA são diferentes dos encontrados no meio acadêmico” (FUNGE, 2004, citado por KISHIMOTO, 2004, p. 1).

Segundo Schwab (2004), citado por Kishimoto (2004, p. 4):

> “No começo do desenvolvimento de jogos eletrônicos, a programação de IA era mais usualmente conhecida por “programação de jogabilidade”, pois não havia nada de inteligente sobre os comportamentos exibidos pelos personagens controlados pelo computador”.

De acordo com Schwab (2009, p. 2) IA para jogos é especificamente o código que controla elementos, fazendo-os parecer tomar decisões inteligentes, a partir de várias opções em determinadas situações. Uma evolução do uso de IA em jogos eletrônicos podem ser vistos no Quadro 1.

Quadro 1 - Exemplos de IA utilizadas em jogos

Ano	Descrição	IA utilizada
1962	Primeiro jogo de computador, *Spacewar*, para 2 jogadores.	Nenhuma
1972	Lançamento do jogo *Pong*, para 2 jogadores.	Nenhuma
1974	Jogadores tinham que atirar em alvos móveis em *Pursuit* e *Qwak*.	Padrões de movimento
1975	*Gun Fight* lançado, personagens com movimentos aleatórios.	Padrões de movimento
1978	*Space Invaders* contém inimigos com movimentos padronizados, mas também atiram contra o jogador.	Padrões de movimento
1980	O jogo *Pac-man* conta com movimentos padronizados dos inimigos, porém cada fantasma (inimigo) tem uma "personalidade" sobre o modo em que caça o jogador.	Padrões de movimento
1990	O primeiro jogo de estratégia em tempo real, *Herzog Wei*, é lançado. Junto, os jogadores puderam noticiar uma péssima busca de caminho.	Máquina de estados
1993	*Doom* é lançado como primeiro jogo de tiro em primeira pessoa.	Máquina de estados
1996	*BattleCruiser: 3000AD* é publicado como o primeiro jogo a utilizar redes neurais em um jogo comercial.	Redes neurais
1998	*Half-Life* é lançado e analisado como a melhor IA em jogos até a época, porém, o jogo utiliza IA baseada em *scripts*.	Máquina de estados/ Script
2001	O jogo *Black & White* é alvo da mídia a respeito de como as criaturas do jogo aprendem com as decisões feitas pelo jogador. Utiliza redes neurais, *reinforcement* e *observational learning*.	Diversos

Fonte: KISHIMOTO, 2004, p. 4.

2.4 IA CONTRA HUMANOS EM JOGOS DE TABULEIRO

> Jonathan Schaeffer e seus colegas desenvolveram um programa de computador que joga damas, chamado CHINOOK. O Chinook derrotou o campeão humano de longa data em uma partida em 1990, e desde 2007 tem sido capaz de jogar perfeitamente utilizando busca alfa-beta combinada com uma base de dados de 39 trilhões de posições finais (RUSSELL; NORVIG, 2013, p.397).

Em 1997, o programa Logistello derrotou o campeão mundial do jogo Othello, Takeshi Murakami, por seis jogos a zero (RUSSELL; NORVIG, 2013, p.397).

Um outro exemplo é o programa de xadrez da IBM, *Deep Blue*, que foi conhecido por derrotar o campeão mundial Garry Kasparov (RUSSELL; NORVIG, 2013, p. 396).

Depois do xadrez, para alguns autores o jogo mais complicado seria o *Go*, considerando quase impossível fazer uma IA para vencer um humano. Segundo Coppin (2004, p. 146) ele é jogado em tabuleiro 19 x 19 (dezenove por dezenove), mais opções de movimentos do que o xadrez, resultando em um enorme fator de ramificação (na média, cerca de 360, comparando com cerca de 38 para o xadrez). Devido a esse número alto de possibilidades, torna-se impossível desenvolver um sistema que realize busca na árvore de jogo para *GO* da mesma forma que para xadrez ou damas.

Por isso, de acordo com o site da UOL, Hut citado por JOHNSON (2016) afirmava, nos idos de 1997, que ainda seriam necessários mais cem anos para que o computador ganhasse uma partida de *Go*. Se uma pessoa razoavelmente inteligente aprendesse a jogar, em poucos meses iria ganhar de qualquer computador do mundo.

Até mesmo os designers do *Deep Blue* admitiram que não poderiam escrever um programa para vencer um jogador, mesmo que iniciante, nem tão cedo (LAI, 2004, p. 8, tradução nossa).

Mas, em 2016, o programa *AlphaGo*, criado pela empresa DeepMind, adquirida pelo Google em 2014, venceu o sul-coreano Lee Sedol, um dos maiores campeões desse milenar jogo de tabuleiro chinês (PINTO, 2016). Após esse feito, Ribeiro citado por Pinto (2016) afirma, referenciando aos registros pessimistas de vários autores sobre uma IA para o *Go*: "Os livros terão que ser reescritos."

2.5 ALGORITMO MINIMAX

O teorema **MINIMAX**, que foi provado por John von Neumann em 1928, estabelece uma estratégia mista ótima para todo jogo finito de soma zero de dois jogadores (WEISSTEIN, 2003, p. 1915, tradução nossa).

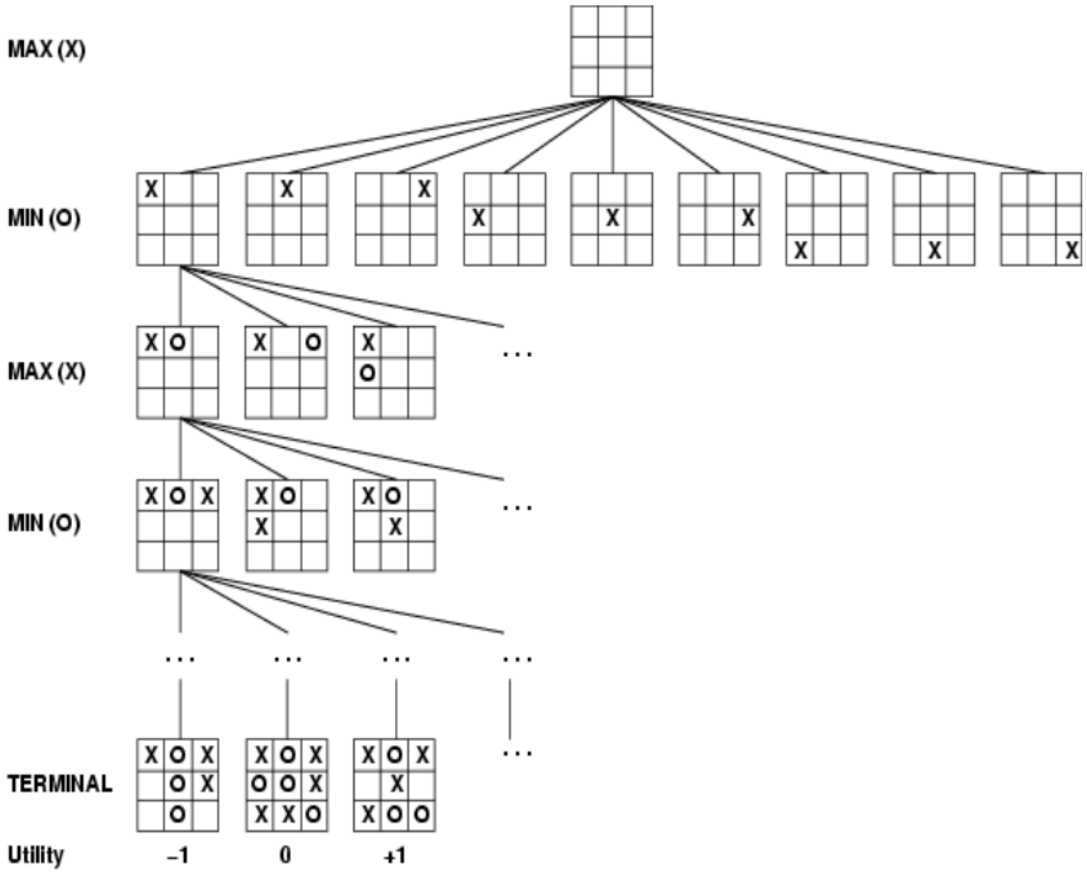
O algoritmo **MINIMAX** é utilizado para escolher bons movimentos, através de uma função estática adequada de avaliação, que seja capaz de fornecer uma pontuação geral para uma dada posição (COPPIN, 2004, p. 130).

> “Utiliza uma computação recursiva simples dos valores **MINIMAX** de cada estado sucessor, implementando diretamente as equações da

> definição. A recursão percorre todo o caminho descendente até as folhas da árvore e, depois, os valores **MINIMAX** são propagados de volta pela árvore, à medida que a recursão retorna" (RUSSELL; NORVIG, 2013, p. 359).
>
> Os oponentes em um jogo são chamados de MIN e MAX. MAX representa o jogador tentado vencer ou maximizar sua vantagem, MIN é o adversário que tenta minimizar o placar de MAX. Supondo que MIN use as mesmas informações e sempre tente se mover para um estado que é pior para MAX (LUGER, 2014, p. 321).

A Figura 4 demonstra um exemplo da aplicação reduzida do **MINIMAX** para o jogo da velha:

Figura 4 - MINIMAX para jogo da velha



Fonte: http://www.ime.usp.br/~slago/slago-jogos.pdf

Nessa implementação para o jogo da velha, a árvore inicia com o estado atual do tabuleiro como raiz, os nós filhos representam as jogadas possíveis e nó folha quando não há mais jogadas disponíveis ou algum jogador venceu a partida. A escolha da melhor jogada é dada pela maior soma do caminho até o nó folha da árvore, que

retorna um caso haja vitória do jogador, menos um no caso de derrota e zero para empate.

O Quadro 2 mostra um exemplo do pseudocódigo do **MINIMAX**:

Quadro 2 - Pseudocódigo do MINIMAX

```
begin miniMax (no_corrente)
    if ehFolha(no_corrente) then
        return pontuacao(no_corrente);
    end
    if ehNoMin(no_corrente) then
        return min(miniMax(filhosDe(no_corrente)));
    end
    if ehNoMax(no_corrente) then
        return max(miniMax(filhosDe(no_corrente)));
    end
end
```

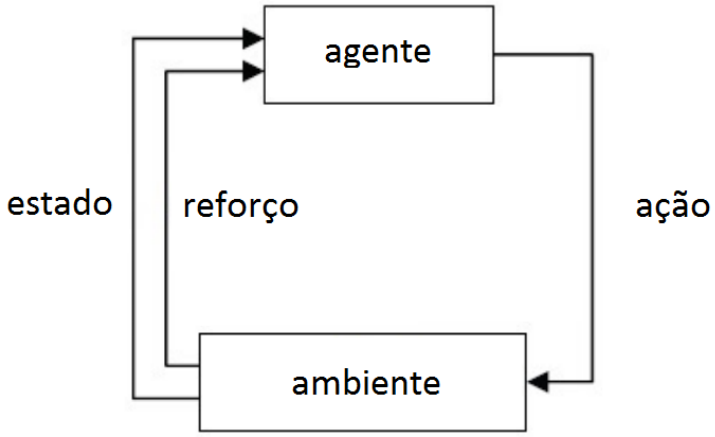
Fonte: QUAGLIO, 2013, p. 27.

2.6 APRENDIZADO DE MAQUINA POR REFORÇO

“Aprendizagem por reforço consiste em usar recompensas observadas para aprender uma política ótima (ou quase ótima) para o ambiente” (RUSSELL; NORVIG, 2013, p. 1646).

No aprendizado por reforço (AR), como mostra Figura 5, a aprendizagem é realizada através da tentativa e erro, em interações do agente com o meio ambiente (QUAGLIO, 2013, p. 29). O agente observa, a cada passo de iteração, o estado corrente do ambiente e escolhe a ação. Ao executar esta ação – que altera o estado do ambiente – o agente recebe um sinal escalar de reforço (penalização ou recompensa), que indica quão desejável é o estado resultante. O AR permite que o agente possa determinar, após várias iterações, qual a melhor ação a ser executada em cada estado, isto é, a melhor política de atuação (BIANCHI, 2004, p. 8).

Figura 5 - Aprendizado por reforço



Fonte: QUAGLIO, 2013, p. 30.

O reforço é atribuído positivo quando o agente operar corretamente e negativo quando operar incorretamente. Por exemplo, um agente robótico aprenderia a pegar um objeto. Quando ele pegar o objeto corretamente, receberá um reforço positivo (COPPIN, 2004, p.).

Segundo Luger (2014, p. 366) no aprendizado por reforço, o agente não sabe diretamente qual ação tomar, ele descobre por meio de exploração quais ações oferecem a maior recompensa. As ações escolhidas não afetam apenas a recompensa imediata, mas também tem impacto sobre ações e eventuais recompensas subsequentes. Ou seja, o próprio agente de aprendizado, por tentativa erro e realimentação, aprende uma política ótima para alcançar objetivos no seu ambiente.

2.6.1 Q-LEARNING

O algoritmo de aprendizado por reforço Q-LEARNING (aprendizagem Q), proposto por Watkins em 1989, aprende iterativamente a política ótima quando o modelo do sistema não é conhecido (BIANCHI, 2004, p. 15).

> Um agente de aprendizagem Q aprende uma função ação-valor, ou função Q, que fornece a utilidade esperada de se adotar uma dada ação em um estado especifico. Portanto, pode comparar utilidades esperadas de suas escolhas disponíveis sem precisar conhecer seus resultados e, assim, não precisa de um modelo do ambiente (RUSSELL; NORVIG, 2013, p. 1648).

O método do Q-LEARNING é baseado na medida de custo de ações, Q(s,a), o qual representa o custo descontado esperado por executar uma ação *a* no estado *s*, seguindo-se uma política ótima. Em cada interação, o ambiente Q(s,a) é atualizado pela regra da equação 1 (FARIA; ROMERO, 2002, p. 222):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

Onde:

- *r* - é o reforço dado pelo ambiente;
- *γ* - é o fator de desconto, utilizado para manter os valores de Q finitos.
- *α* - é o fator de aprendizado, que pode ser calculado através da equação 2:

$$\alpha = \frac{1}{1+visitas(x,a)} \quad (2)$$

Em que visitas (x, a) é o número de vezes que a ação a foi escolhida e executada no estado x (QUAGLIO, 2013, p. 31).

Para treinamento e atualização dos pares estado/ação é realizado de acordo com o pseudocódigo do Quadro 3:

Quadro 3 - Pseudocódigo do Q-LEARNING

```
foreach x, a do
    Inicialize a tabela Q(x,a);
end
Observe o estado atual x;
repeat
    Selecione uma ação a e execute;
    Receba um reforço imediato;
    Observe o novo estado x';
    Atualize a tabela Q(x,a) de acordo com a equação 1;
    x ← x';
until Critério de parada satisfeito;
```

Fonte: QUAGLIO, 2013, p. 31.

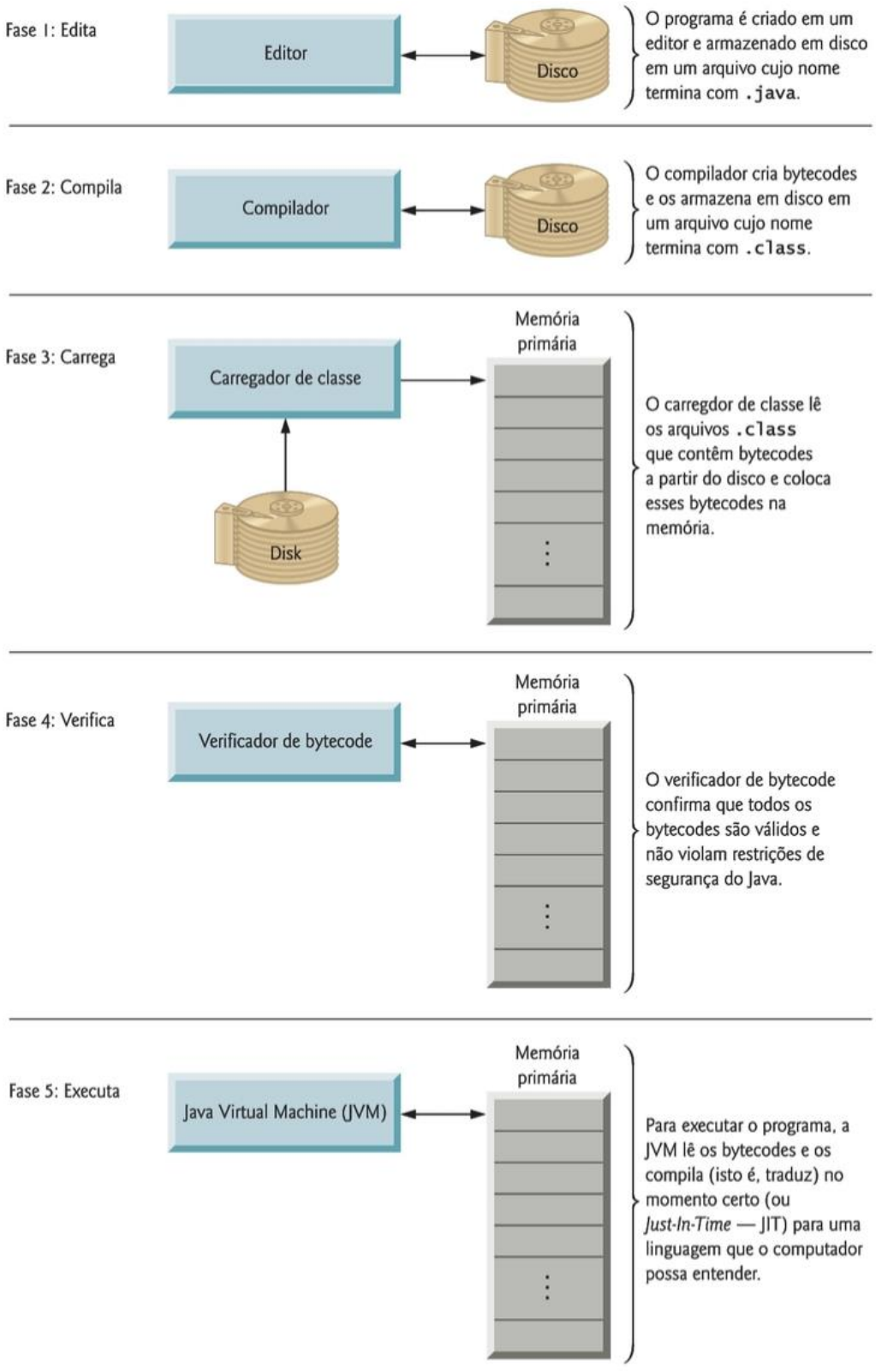
2.7 PLATAFORMA JAVA

A linguagem Java foi escolhida para o desenvolvimento do projeto, principalmente pelo fato da portabilidade, obtendo independência de sistema operacional específico.

O compilador do Java gera um código de máquina conhecido como *“bytecode”, que para execução é* interpretado *pela JVM (Java Virtual Machine, em português: Máquina Virtual do Java).* (RICARTE, 2001, p. 34)

Segundo DEITEL (2009, p. 8), a criação de uma aplicação Java é geralmente composta por cinco fases, como mostra a Figura 6:

Figura 6 - Fase da criação de uma aplicação Java



Fonte: DEITEL, 2009, p. 8.

2.8 GIT E GITHUB

O Git é um sistema de controle de versões distribuído com ênfase em velocidade, foi inicialmente projetado e desenvolvido por Linus Torvalds para desenvolvimento do kernel Linux. (EIS, 2012)

Todos que estiverem trabalhando em um projeto no Git terão uma cópia de todo histórico do projeto, e não somente do estado atual dos arquivos. (BELL; BEER, 2015, p. 13)

O GitHub é um serviço web que oferece diversas funcionalidades extras aplicadas ao Git, facilita a colaboração com outras pessoas em um projeto, feito por meio da disponibilização de um local centralizado para compartilhar o repositório, que permitem especificar, discutir e revisar alterações junto à equipe de maneira eficiente. (SCHMITZ, 2015)

2.9 SQLITE

Para armazenamento dos dados do algoritmo Q-LEARNING, foi escolhido banco de dados SQLite.

O SQLite é um mecanismo de banco de dados incorporado do SQL, não possui processo de servidor separado, lê grava diretamente em arquivos de disco comuns. (SQLITE, 2000)

Pode ser definido como um banco de dados amplamente distribuído no mundo, ideal para quando precisa de um local em que a inserção de dados seja rápida e fácil sem toda complexidade de uma instalação cliente-servidor. (NIELD, 2016, p. 25)

2.10 TRABALHO CORRELATO

Quaglio (2003) em seu trabalho de conclusão de curso, implementou o MINIMAX e Q-LEARNING para o jogo Othello. Entretanto, ambos podem ter implementações diferentes para problemas distintos ou até mesmo para um problema idêntico.

No MINIMAX, Quaglio relata que foi necessário limitar o nível de busca, pois inicialmente o algoritmo se apresentou inviável. Havia casos que a profundidade de busca chegava a 30 (trinta), depois foi limitada obtendo bons resultados a três.

O Q-LEARNING foi implementado de forma que o reforço seja aplicado a cada jogada. Inicialmente é realizada uma busca no banco de dados para obter o estado do tabuleiro. Caso o estado não exista no banco, é adicionado e inicializado com valor máximo para todas jogadas possíveis naquele momento. Baseado no retorno da jogada, o reforço é aplicado, tomando como referência a quantidade de peças capturadas e o peso da posição escolhida.

O treinamento do Q-LEARNING foi realizado com 2.000 (duas mil) partidas e teste de aprendizado com 200 (duzentas) partidas, jogadas contra um jogador aleatório. Essa quantidade é baixa, levando em consideração o espaço amostral do Othello, visto que o tabuleiro possui cerca 10^{18} possíveis combinações. De acordo com o que é mostrado na Tabela 1, não houve muita eficiência em quantidade de vitórias, chegando a ter pouco acima de 50% (cinquenta por cento) após o treinamento.

Tabela 1 - Comparação de vitórias, antes e depois do treinamento

	Quantidade de vitórias
Antes do treinamento	62
Após o treinamento	113

Fonte: QUAGLIO, 2003, p. 42

Também é mostrado um comparativo desempenho do Q-LEARNING jogando contra o MINIMAX, destacando o tempo de resposta menor. Porém, esse tempo se deve ao fato de que o treinamento realizado foi de 2.000 partidas apenas, gerando um número pequeno de registro para o banco de dados. Além disso, o resultado final da partida é de vitória para o MINIMAX com uma grande diferença de número de discos, como pode ser visto na Tabela 2.

Tabela 2 - Q-LEARNING contra o MINIMAX

Algoritmo	Maior Tempo	Menor Tempo	Tempo Médio	Nº de discos
Q-LEARNING	2872ms	0ms	1471ms	14
MINIMAX	818162ms	0ms	389216ms	50

Fonte: QUAGLIO, 2003, p. 42

3. METODOLOGIA

Inicialmente, foi feita uma revisão sobre o jogo Othello, na qual foram levantados os elementos e as regras do jogo. A etapa seguinte compôs em construir um protótipo, utilizando o paradigma de programação orientado a objetos, através da linguagem Java, que constituiu em um tabuleiro virtual do jogo, permitindo a realização e validação das jogadas, a visualização do placar e identificação do ganhador no final da partida.

Todo jogador implementado, deverá herdar de uma classe abstrata Jogador, sendo necessário a implementação do método que realiza as jogadas. Toda vez que o jogador receber a vez de jogar, o método será chamado, passando as jogadas disponíveis naquele momento, e ao terminar seu procedimento de escolha, deve retornar a jogada escolhida.

O protótipo permitiu a realização de partidas do jogador humano contra humano, que facilitou na validação de diferentes situações.

Logo após, foi criado um jogador que faz jogadas aleatórios, para servir de comparação entre partidas com os outros jogadores.

A seguir foi realizado com a implementação do algoritmo MINIMAX, de forma que ao receber as jogadas possíveis, o jogador percorre uma simulação em cada uma dessas posições, fazendo uma árvore de busca. Cada posição é avaliada com um valor, de acordo com a Tabela 3, decisão é tomada pelo caminho que possuir a maior soma.

Tabela 3 - Classificação das regiões

	A	B	C	D	E	F	G	H
1	100	-20	10	5	5	10	-20	100
2	-20	-50	-2	-2	-2	-2	-50	-20
3	10	-2	-1	-1	-1	-1	-2	10
4	5	-2	-1	-1	-1	-1	-2	5
5	5	-2	-1	-1	-1	-1	-2	5
6	10	-2	-1	-1	-1	-1	-2	10
7	-20	-50	-2	-2	-2	-2	-50	-20
8	100	-20	10	5	5	10	-20	100

Fonte: http://play-othello.appspot.com/files/Othello.pdf

Os valores são definidos valorizando os cantos e arestas com maior peso. Os cantos são as posições que após ser conquistadas, não podem ser tomadas pelo adversário, além de facilitar a captura de mais peças. As arestas também possibilitam ao jogador a captura de maior número de peças em uma jogada, porem podem ser retomadas. Algumas áreas são ruins, assim como outras regiões, por dar a oportunidade de o adversário conquistar o canto.

Além dessa classificação, foram implementadas outras versões de classificação das jogadas, para comparação de eficiência, com outros critérios de avaliação, como a contagem de peças capturadas e uma junção desta com a técnica anterior.

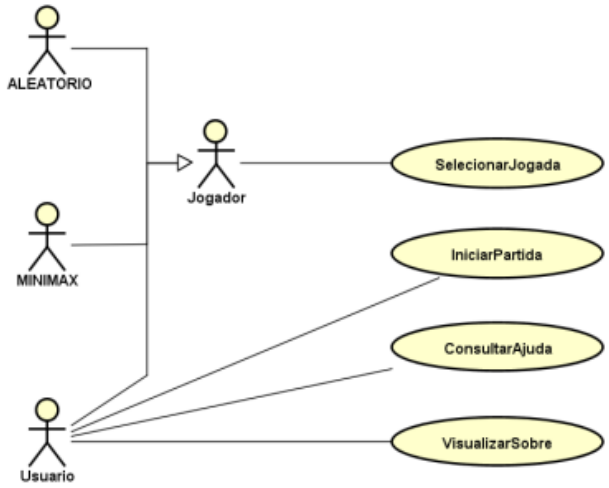
Foi observado que trabalhar com o MINIMAX, sem limitar a busca, pode causar lentidão, inviabilizando sua implementação, a limitação da busca foi realizada de acordo com nível de dificuldade escolhido pelo usuário, podendo ser de um a 10 (dez). Quanto maior o nível da busca, mais precisas serão as jogadas do jogador MINIMAX.

Também foi realizado a implementação de protótipo de testes, que demonstrou a inviabilidade do uso do Q-LEARNING para este jogo.

3.1 DIAGRAMAS DE CASOS DE USO

Esta seção apresenta o diagrama de casos de uso do tabuleiro virtual desenvolvido. A Figura 7 apresenta o do protótipo, no qual o usuário pode iniciar jogo, selecionar jogadas, consultar a ajuda e visualizar sobre.

Figura 7 - Diagrama de Casos de Uso



3.1.1 Caso de uso iniciar partida

O ator dá início a uma nova partida.

- **Ator**: Usuário
- **Pré-condições**:
- **Pós-condições**: o tabuleiro é preenchido com as peças iniciais e são destacados os campos das possíveis jogadas.
- **Fluxo normal:**
  1. O Ator seleciona o jogador 1, que pode ser um jogador humano, o algoritmo MINIMAX ou o Aleatório.
  2. O Ator seleciona o jogador 2, que pode ser um jogador humano, o algoritmo MINIMAX ou o Aleatório.
  3. O Ator seleciona a opção de iniciar partida.
  4. O sistema preenche o tabuleiro com as quatro peças iniciais.
  5. O sistema atualiza os dados de quantidades de discos no painel lateral.
- **Fluxo alternativo:**
  1. Já existe uma partida em andamento não finalizada.
     1.1 O sistema remove todas peças do tabuleiro.
     1.2 O sistema descarta os dados da partida em andamento.
     1.3 Retorna ao fluxo normal no passo 2.
  2. Tabuleiro já está preenchido de uma partida finalizada.
     2.1. O sistema remove todas peças do tabuleiro.

2.2. Retorna ao fluxo normal no passo 2.

3.1.2. Caso de uso iniciar partida

Ao decorrer da partida, é destacado no tabuleiro os campos de possíveis jogadas, na cor referente aos discos do jogador que deverá selecionar a jogada naquele momento.

- **Ator**: Jogador
- **Pré-condições**: já possua uma partida iniciada.
- **Pós-condições**: o sistema passa a vez de jogada para o adversário.
- **Fluxo normal:**
  1. O Ator recebe as jogadas disponíveis
  2. O Ator seleciona uma jogada disponível.
  3. O sistema insere um disco da cor de referência do jogador no campo selecionado.
  4. O sistema modifica a cor dos discos capturados para a cor de referência do jogador realizou a jogada.
  5. O sistema atualiza os dados de quantidade de discos.
  6. O sistema remove os destaques das jogadas disponíveis.
  7. O sistema gera para o adversário os destaques das jogadas disponíveis.
  8. O sistema volta ao passo 1.
- **Fluxo alternativo:**
  1. No passo 1, o campo selecionado não está disponível.
     1.1. Retorna ao fluxo normal no passo 1.
  2. No passo 6, não há jogadas disponíveis ou o tabuleiro já foi preenchido por completo.
     2.1. O sistema atualiza o placar.
     2.2. O sistema encerra a partida e informa o vencedor.
     2.3. O sistema apresenta opção para iniciar nova partida.
  3. No passo 6, não há jogadas disponíveis para o adversário.
     3.1. Retorna ao passo 1, mantendo a vez do mesmo jogador.

3.1.3. Caso de uso consultar ajuda

O ator obtém descrição sobre as regras do jogo.

- **Ator**: Usuário
- **Pré-condições**:
- **Pós-condições**: as regras do jogo são exibidas.
- **Fluxo normal:**
  1. O Ator seleciona a opção de consultar ajuda.
  2. O sistema exibe as regras do jogo.
- **Fluxo alternativo:**

3.1.4. CASO DE USO VISUALIZAR SOBRE

O ator obtém informações sobre o sistema e seus desenvolvedores.

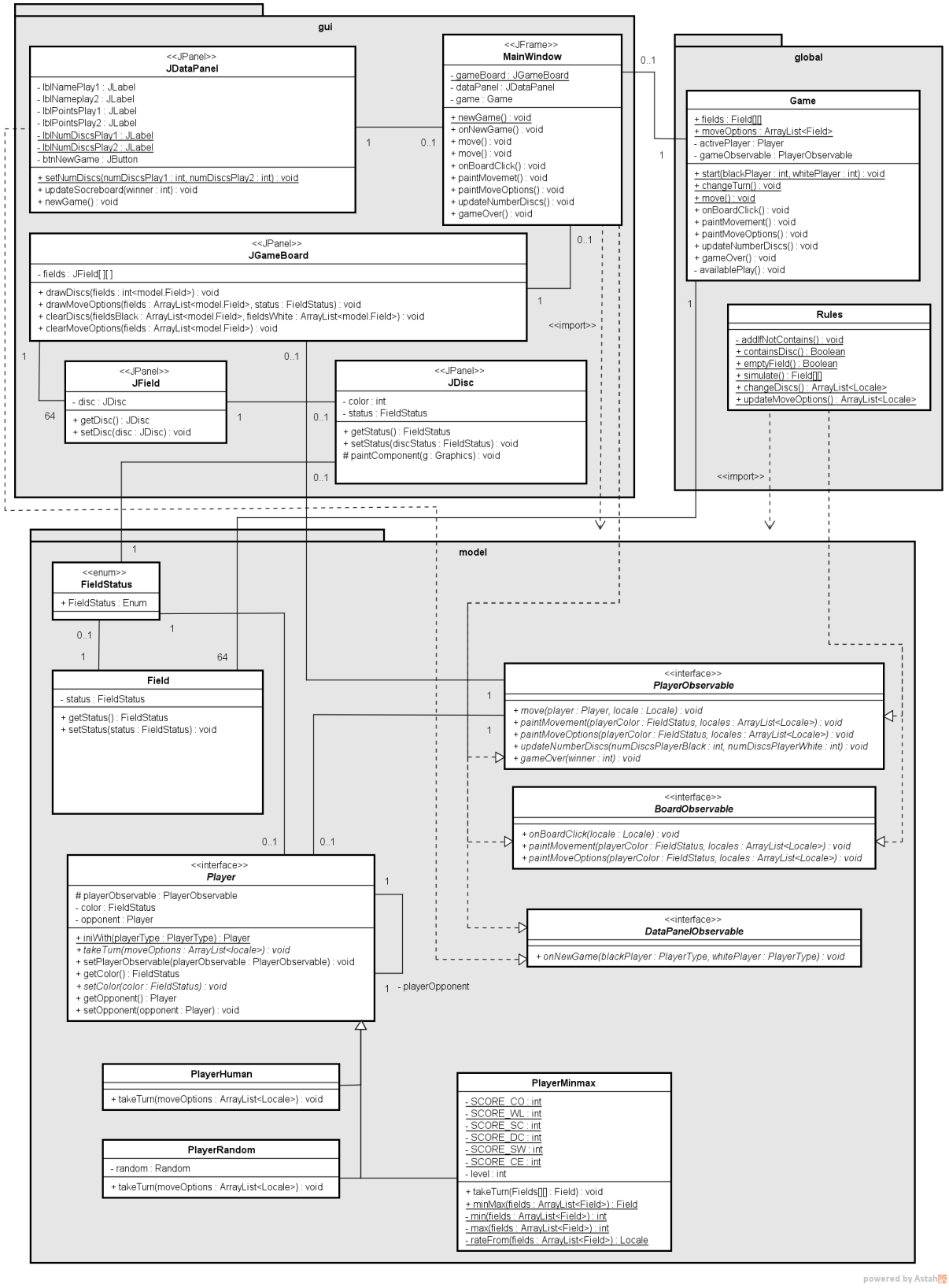
- **Ator**: Usuário
- **Pré-condições**:
- **Pós-condições**: os detalhes sobre o sistema e os desenvolvedores são exibidos.
- **Fluxo normal:**
  1. O Ator seleciona a opção de visualizar sobre.
  2. O sistema exibe as informações sobre o projeto e os desenvolvedores.

Fluxo alternativo:

3.2 DIAGRAMA DE CLASSE

O diagrama de classe utilizado para o desenvolvimento do protótipo é formado de classes, separadas em três pacotes: *GUI* para as classes de interface visual, *MODEL* para classes de modelos e *GLOBAL* para demais classes que realizam o controle do jogo, como pode ser visualizado na Figura 8.

Figura 8 - Diagrama de Classe



4. PROTÓTIPOS

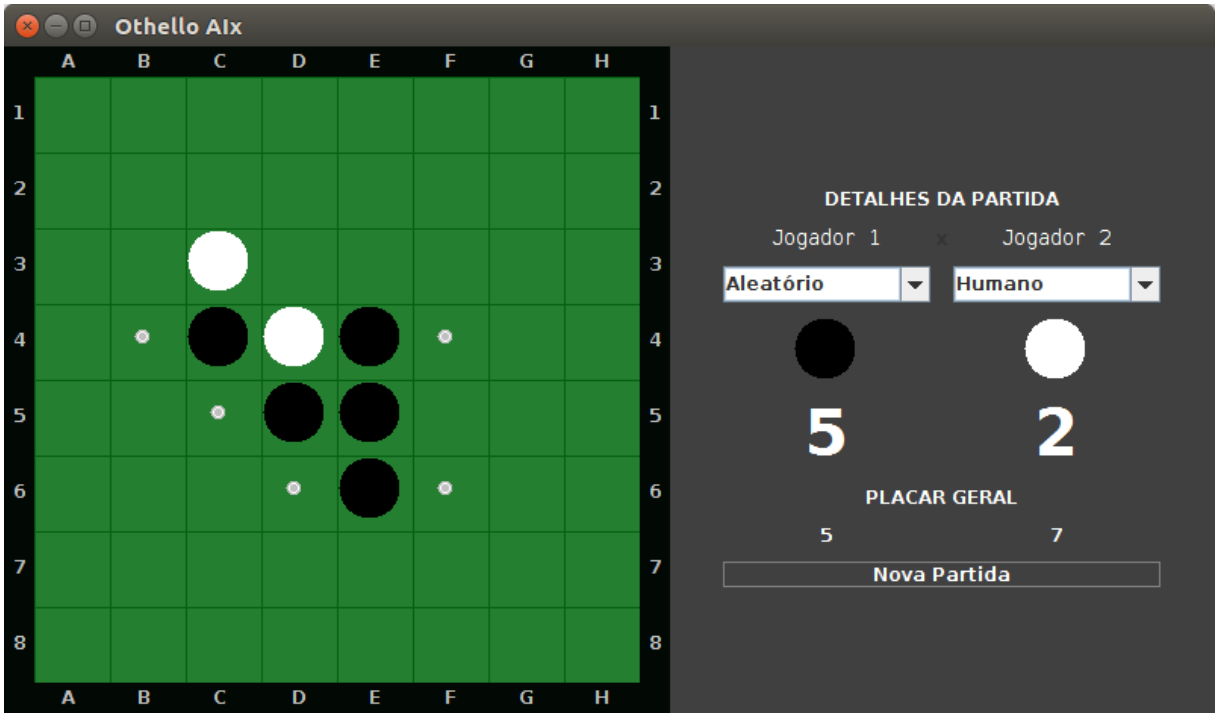
O desenvolvimento dos protótipos, foram realizados utilizando Java, a IDE Eclipse e o *GitHub* para versionamento. O projeto foi estruturado em três pacotes: GUI (do inglês *Graphical User Interface*, em português Interface Gráfica do Utilizador), MODEL e GLOBAL.

4.1 PRIMEIRO PROTÓTIPO

O desenvolvimento do primeiro protótipo teve como objetivo o funcionamento do pacote GUI, com todos elementos visuais necessários funcionando para a realização de partidas entre jogadores humanos.

Ao iniciar a partida, é acrescentado no tabuleiro as peças inicias e destaques nos campos de jogadas possíveis (sendo os únicos campos ativos como clicáveis no tabuleiro) com a cor do jogador da vez. Também foi incluso um painel lateral com informações da partida, com quantidade de discos atual de cada jogador e placar geral de vitorias. Na Figura 9 pode ser visualizado a tela do protótipo durante uma partida.

Figura 9 - Tela do protótipo



Após cada jogada, é realizado a atualização no tabuleiro das peças capturadas, em seguida, verificado as jogadas possíveis do adversário, e assim, é realizado o destaque desses campos no tabuleiro e passado a vez para o outro jogador.

Na verificação de jogadas possíveis, caso não possua nenhuma jogada, é verificado se outro jogador possui, se sim, a vez de jogar é passada para ele. Caso contrário, é o indicativo que a partida acabou, então é realizado a identificação do vencedor, de acordo com o jogador que possuir a maior quantidade de discos, ou empate caso seja a mesma para ambos.

4.2 SEGUNDO PROTÓTIPO

Neste protótipo foi acrescentado um jogador aleatório, possibilitando a realização de partidas contra o jogador humano, ou até mesmo entre jogadores aleatórios.

Na primeira implementação ocorreu um problema nas partidas entre jogador humano e aleatório. Quando um jogador recebe a vez de jogar, lhe é passado as jogadas possíveis e é aguardado o retorno de qual jogada escolhida. O jogador aleatório, ao receber as jogadas possíveis, seleciona sua jogada aleatoriamente e retorna imediatamente. Em relação ao jogador humano, o sistema necessitava aguardar a escolha feita com o clique do jogador, o que não ocorria, o fluxo do jogo seguia antes de o jogador concluir sua jogado.

Foram feitas várias tentativas para que o fluxo principal da partida aguardasse o retorno do jogador humano para prosseguir, como por exemplo usando *Thread*, porém sem sucesso.

Este problema foi resolvido com a utilização de uma classe abstrata, com método que realiza jogada, sendo implementada em cada jogador e na classe da janela principal. Na implementação de cada jogador é realizado a chamada do método na classe da janela principal, que por sua vez, faz as atualizações necessárias.

4.3 TERCEIRO PROTÓTIPO

O desenvolvimento deste protótipo foi a implementação mais complexa do projeto, trata da inclusão do jogador MINIMAX. Para o algoritmo MINIMAX tomar a decisão de qual jogada executar, faz se uma simulação de jogadas adiantes, do jogador e do adversário.

Para realizar a simulação de jogadas, foi necessário alterar o projeto, retirando da classe responsável pelo controle lógico da partida, todos os métodos encarregados por alterações e validações de jogadas no tabuleiro. Com essa retirada, foi criada uma nova classe com esses métodos de forma estática, proporcionando que todo objeto do projeto tivesse acesso a essas funcionalidades, o que facilitou para que algoritmo MINIMAX pudesse realizar suas simulações sem alterar os dados do tabuleiro real da partida.

4.4 QUARTO PROTÓTIPO

Neste protótipo, foram realizados testes de inserção de dados no banco, afim de atestar a viabilidade da máquina de aprendizado, a ser desenvolvida utilizando o algoritmo Q-LEARNING.

Utilizando nesse momento o SQLite, o protótipo teve como propósito inserir sequencialmente no banco os dados, todos estados de tabuleiro e os valores de cada jogada. Porém não foi possível concluir a inserção dos registros necessários, correspondentes a todos estados possíveis. Foi notado, que o tempo gasto para o banco ser devidamente carregado, era inviável e esta implementação foi descartada.

O tempo médio de inserção de 1.000 (mil) registros foram de 223 milissegundos, para um ciclo de todas as combinações possíveis, precisaria de cerca de 3^{64} registros, que de acordo com tempo médio de inserção seria necessários $2{,}22^{19}$ anos.

5. RESULTADOS

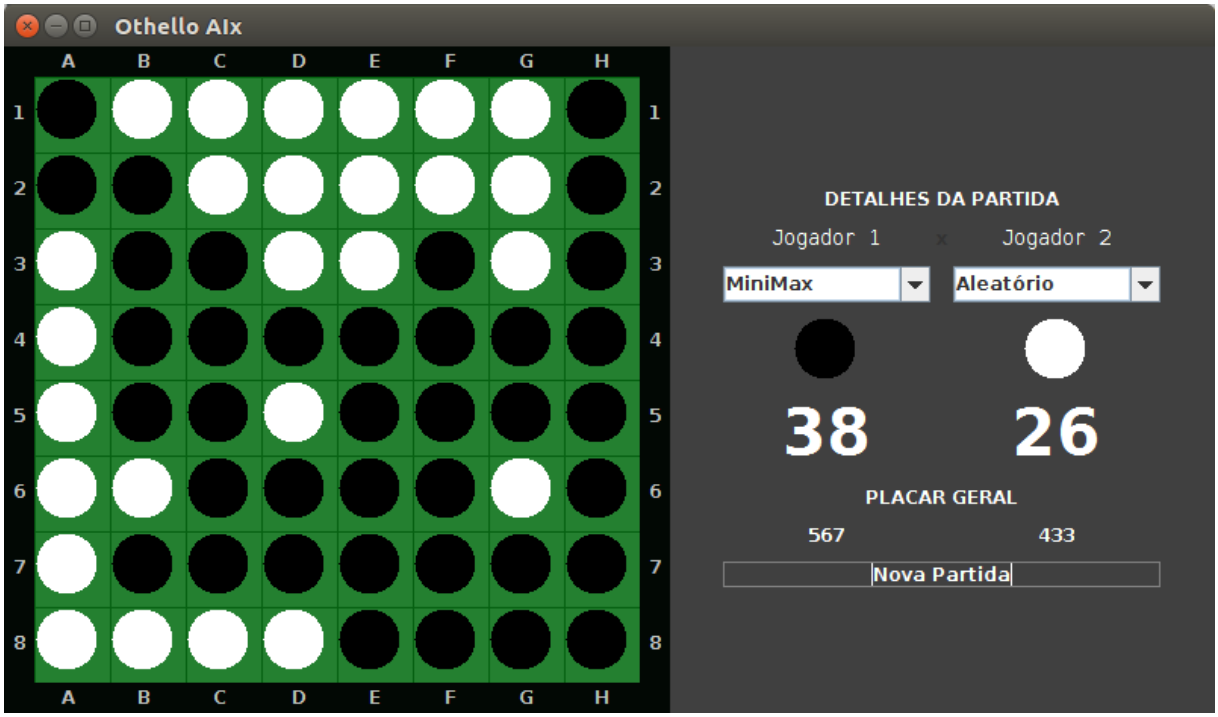
Os resultados dos testes descritos nesse capítulo se restringem ao protótipo final, com foco na eficiência do algoritmo MINIMAX.

Para a execução dos testes, foi utilizado um único computador, um notebook Dell modelo 5470, com processador Intel i5 de 2,7 GHz, 4Gb (quatro giga bytes) de memória RAM, com sistema operacional Ubuntu versão 16,04 LTS 64 bits.

Para medir a empenho do algoritmo foram realizadas partidas contra o jogador aleatório, que deve apresentar um nível bem baixo de dificuldade, pela falta de critério na escolha de suas jogadas. Com a finalidade de verificar o aumento do nível de profundidade de busca do MINIMAX, realmente refletisse na sua eficácia. Para cada nível analisado, foram realizados 1.000 (mil) partidas, considerando somente as partidas com vencedores, ou seja, descartando os empates, que ocorrem com pouco frequência.

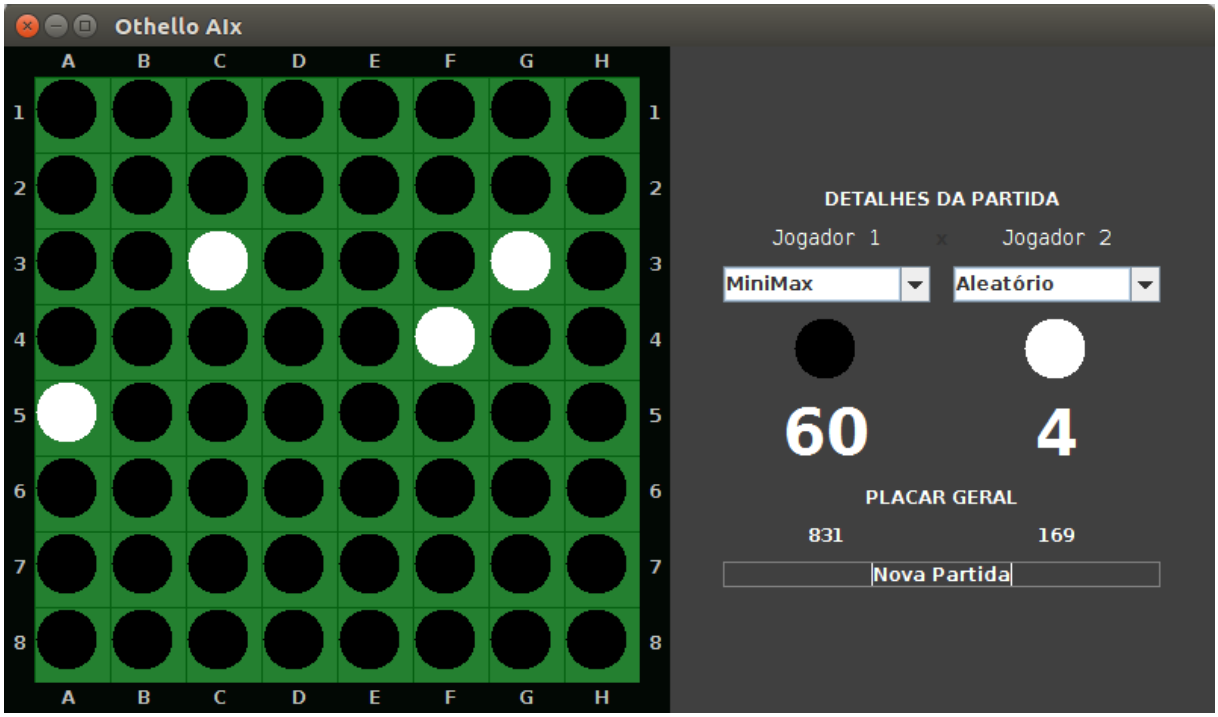
De início, foi utilizado o nível de profundidade zero, como esperado, o número de vitorias foram próximos ao do jogador aleatório, vencendo apenas cerca de 56,7% (cinquenta e seis virgula sete por cento) das partidas, como mostra Figura 10:

Figura 10 - MINIMAX nível zero



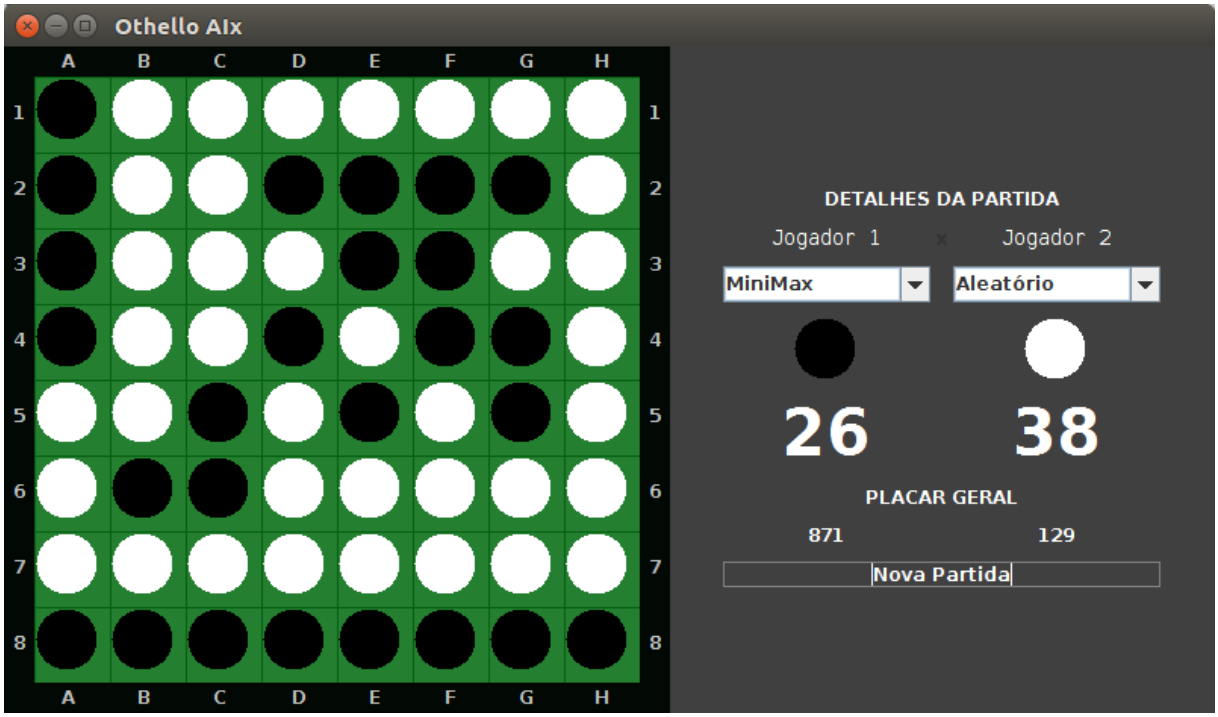
Ao elevar o nível do MINIMAX para um, já se obtém uma melhora considerável, aumentando o número de vitórias para 83,1% (oitenta e três virgula um por cento), como apresenta a Figura 11:

Figura 11 - MINIMAX nível um



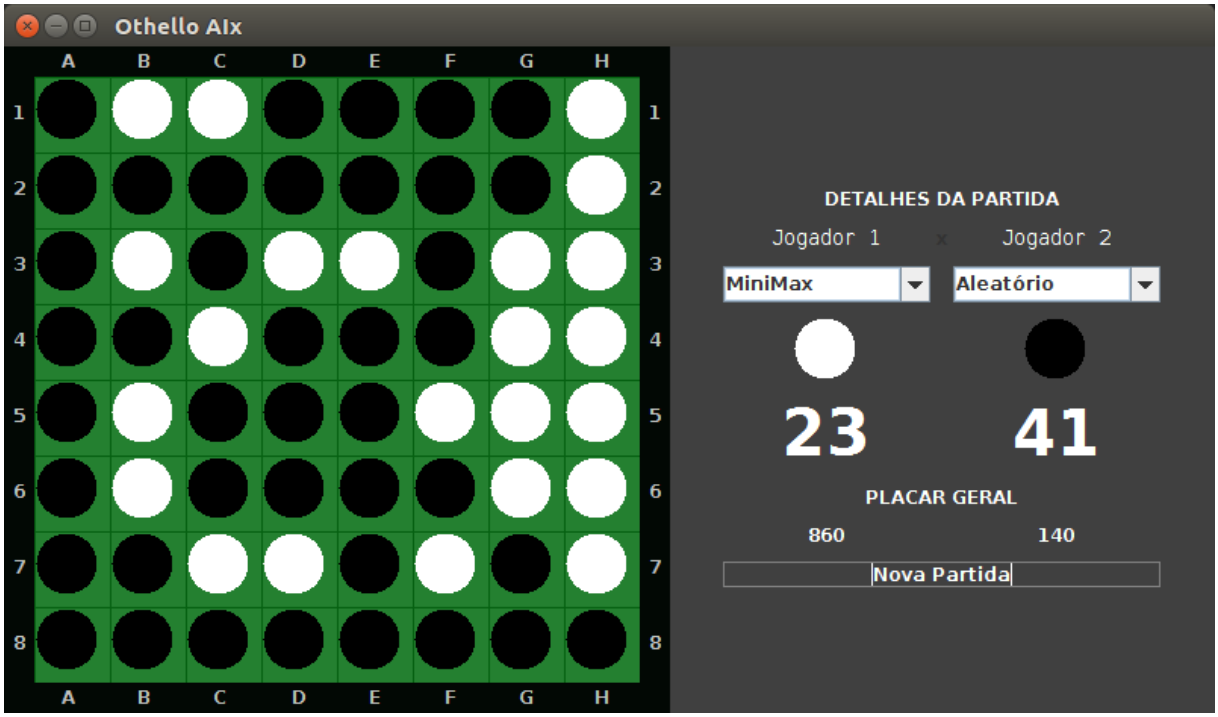
Aumentando o nível para três, ocorreu apenas uma pequena melhora, tendo 87,1% (oitenta e sete virgula um por cento), como mostra o placar geral de vitórias na Figura 12:

Figura 12 - MINIMAX nível três



No nível 7 (sete), se manteve bem próximo da quantidade de vitorias do nível três, com uma pequena redução de menos de um por cento, ficando com 86% (oitenta e seis por cento) de vitorias, como mostra Figura 13:

Figura 13 - MINIMAX nível 7 (sete)



Além de não trazer uma melhora em número de vitórias em relação ao nível três, o MINIMAX com nível de busca 7 (sete), sofreu um aumento exponencial no tempo de resposta de jogadas. Na Tabela 4 e na Figura 14, demonstram o tempo médio e máximo de resposta de uma jogada de todos níveis testados:

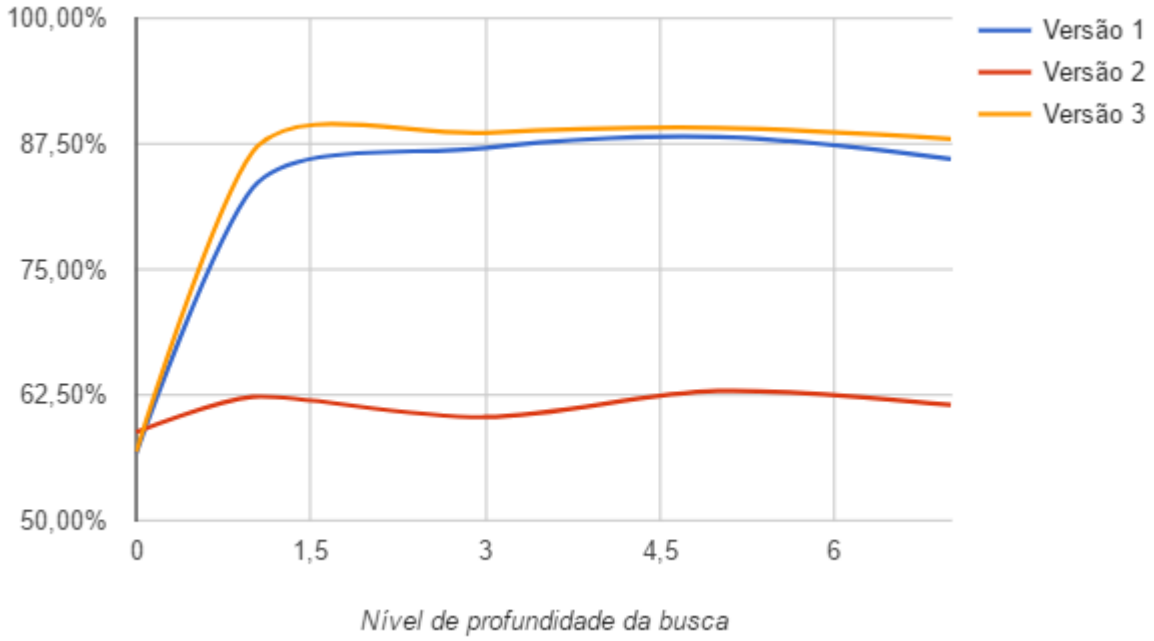
Tabela 4 - Tempo de resposta de jogadas em milissegundos

Nível de profundidade da busca	**0**	**1**	**3**	**5**	**6**	**7**	**10**
Tempo médio	0	1	4	10	13	54	562
Tempo máximo	22	102	75	157	235	1049	8947

Figura 14 - Tempo de resposta de jogadas em milissegundos

Em busca do aprimoramento do algoritmo MINIMAX, foram implementadas mais duas versões, alterando a forma de classificar as jogadas. Sendo a primeira versão com a ideia inicial de avaliar as jogadas de acordo com a Tabela 3, a segunda versão considerando a quantidade de peças capturadas na jogada, e a terceira versão a soma dos valores dessas duas técnicas. Na Figura 15, o gráfico apresenta o demonstrativo da eficiência do percentual de vitórias em cada versão contra o jogador aleatório.

Figura 15 - Comparativo de versões do MINIMAX



A segunda versão obteve o pior resultado, mantendo o percentual de vitórias bem próximo aos 60% (cinquenta por cento), não houve evolução com aumento no nível de busca. Isso demonstra que realizar jogadas, considerando a todo momento a maior captura de peças, não é uma estratégia boa para o Othello, principalmente nas primeiras jogadas, pois capturando muitas peças no início, pode gerar vulnerabilidades de perdê-las no final. Essa implementação só seria eficiente, se a busca não possuísse limitação de nível, sempre simulando até o fim da partida, onde definiria o vencedor.

A primeira e terceira versão, proporcionaram um número de vitorias bem próximos em todos os níveis, com a terceira sobressaindo um pouco melhor em todos níveis.

6. CONCLUSÃO

As técnicas de inteligência artificial estão cada vez mais aprimoradas, fazendo com que a máquina aprenda como ou até melhor que os humanos, em casos específicos. É necessário que o mundo dos jogos absorva essa evolução, ações inteligentes em elementos do jogo fazem toda diferença em seu atrativo.

No desenvolvimento desse projeto, notamos que o desenvolvimento de aprendizado de máquina para um jogo, pode ser muito custoso, devido à complexidade de implementação e a quantidade de dados para um treinamento eficiente. Nesse caso, um jogo de regras simples, com poucos elementos e apenas dois jogadores, se mostrou inviável a implantação do algoritmo de aprendizado de máquina escolhido, o Q-LEARNING, devido ao grande espaço amostral de estados possíveis do tabuleiro do jogo Othello.

Com o algoritmo de buscar em árvore MINIMAX, foi possível obter um bom resultado, chegando ao objetivo de obter um agente inteligente, com a disponibilidade de ajuste de níveis de dificuldades diferentes. Com ressalva que não apresentou bom comportamento em níveis muito altos, sendo necessário limitar a nível da busca, assim foram analisados os níveis de zero a 7 (sete) nos testes, o que não chegou proporcionar uma dificuldade elevada para jogador humano.

Para trabalhos futuros, seria interessante o aprimoramento do MINIMAX com cortes de busca e rever a classificação de jogadas acrescentando outros elementos, podendo trazer melhor eficiência e desempenho. Outra sugestão, seria a utilização de técnicas de inteligência artificial mais complexas.

REFERÊNCIAS

BELL, P; BEER, B. **Introdução ao GitHub:** Um guia que não é técnico. São Paulo: Novatec Editora Ltda. 2015. 13p. On-line. Disponível em <https://books.google.com.br/books?id=3z0aBgAAQBAJ>. Acesso em 23 abr. 2017.

BIANCHI, R. A. da Costa. **Uso de Heurística para a Aceleração do Aprendizado por Reforço**. 2004. 197f. Tese (Doutorado em Engenharia da Computação) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2004.

CHUDASAMA, C; TRIPATHI, P; PRAJAPATI, K. Optimizing Search Space of Othello Using Hybrid Approach. **International Journal of Modern Trends in Engineering and Research.** v. 1, jul. 2014. On-line. Disponível em: <http://www.ijmter.com/papers/volume-1/issue-1/optimizing-search-space-of-othello-using-hybrid-approach.pdf>. Acesso em 15 fev. 2017.

COPPIN, B. **Inteligência Artificial**. Rio de Janeiro: LTC – Livros Técnicos e Científicos, 2004. 636 p.

DEITEL, H. M. **Java como programar**. 8 ed. São Paulo: Pearson Education do Brasil, 2009, 8 p.

EIS, D. Iniciando no GIT: parte 1. **Tableless**. 2012. On-line. Disponível em: <https://tableless.com.br/iniciando-no-git-parte-1>. Acesso em: 23 abr. 2017.

FARIA, G; ROMERO, R. A. F. Navegação de robôs móveis utilizando aprendizado por reforço e lógica fuzzy. **Revista Controle & Automação**, São Paulo, v. 13, n. 13, p. 219-230, set. / dez. 2002.

FEDERAÇÃO BRASILEIRA DE OTHELLO. **Campeonatos**. São Paulo: [201-?]. On-line. Disponível em: <http://www.othello.com.br/campeonatos>. Acesso em 17 mar. 2016.

KISHIMOTO, A. **Inteligência Artificial em Jogos Eletrônicos.** 2004. 11p. Artigo (Graduação de Ciência da Computação) – Universidade Presbiteriana Mackenzie. São Paulo, 2004. Disponível em: <http://www.kishimoto.com.br/publications.php>. Acesso em: 4 mar. 2016.

NIELD, T. Introdução à linguagem SQL: abordagem prática para iniciantes. 1 ed. São Paulo. Novatec Editora Ltda, 2016. 25 p. On-line. Disponível em: < https://books.google.com.br/books?id=5HQdDAAAQBAJ>. Acesso em 25 abr. 2017.

LAI, D. **Learning from the stones:** A go approach to mastering China´s strategic concept. 2004. 41p. Disponível em: <https://fas.org/man/eprint/lai.pdf>. Acesso em: 29 abr. 2016.

LUGER, G. F. **Inteligência artificial.** 6 ed. São Paulo: Pearson Education do Brasil, 2014. 321 p.

PINTO, L. Em nova façanha da inteligência artificial, programa do Google derrota gênio sul-coreano do Go. **Veja**, 12 mar. 2016. Disponível em: <http://veja.abril.com.br/noticia/vida-digital/computador-do-google-vence-sul-coreano-lenda-do-jogo-chines-go>. Acesso em: 25 abr. 2016.

QUAGLIO, V. G. **Técnicas de inteligência artificial aplicadas ao jogo Othello:** um estudo comparativo. 2013. 50 p. Trabalho de Conclusão de Curso (Bacharelado em

Ciência da Computação) - Universidade Estadual de Londrina, Londrina. 2013. Disponível em: <http://www.uel.br/cce/dc/wp-content/uploads/TCC-ViniciusQuaglio-BCC-UEL-2013.pdf>. Acesso em: 17 mar. 2016.

RICARTE, I. L. M**. Programação Orientada a Objetos:** Uma Abordagem com Java. Campinas. 2001. 34 p. Disponível em: <http://www.dca.fee.unicamp.br/cursos/PooJava/Aulas/poojava.pdf>. Acesso em: 7 out. 2016.

RUSSELL, S; NORVIG, P. **Inteligência Artificial.** 3 ed. Rio de Janeiro: Elsevier. 2013. 2172p.

SAIBA COMO FUNCIONA A INTELIGÊNCIA ARTIFICIAL capaz de vencer os humanos. **UOL notícias**, 22 abr. 2016. Disponível em: <http://noticias.uol.com.br/ciencia/ultimas-noticias/the-new-york-times/2016/04/22/reconhecendo-os-artificios-da-inteligencia-artificial.htm>. Acesso em: 25 abr. 2016.

SATO, P. O que é inteligência artificial? Onde ela é aplicada? **Revista Escola**. [201-?]. Disponível em: <http://revistaescola.abril.com.br/ciencias/fundamentos/inteligencia-artificial-onde-ela-aplicada-476528.shtml>. Acesso em 25 abr. 2016.

SCHWAB, B. **AI Game Engine Programming.** 2 ed. Boston: Course Technology, 2009. 741.

SELLITTO, M. A**. Inteligência Artificial: uma aplicação em uma indústria de processo contínuo.** São Leopoldo. 2002. 364 p. Disponível em: <http://www.scielo.br/pdf/gp/v9n3/14574.pdf>. Acesso em: 4 mar. 2016.

SILVA, F. **Othello Classic: deleite, lindos tabuleiros.** 2011. On-line. Disponível em: <http://othelloclassic.blogspot.com.br/2011/06/tabuleiro-de-comemoracao-de-35-anos.html>. Acesso em: 25 mai. 2016.

SILVA, F. **Othello – Arthur vs. Campeão Brasileiro.** Produção WAV3. 2013. Vídeo (13 min). On-line. Disponível em: <https://www.youtube.com/watch?v=cg1UyqR6wHs>. Acesso em: 20 mar. 2016.

SQLITE. **About SQLite.** 2000. On-line. Disponível em <https://www.sqlite.org/about.html>. Acesso em 13 nov. 2016.

TEIXEIRA, J. F. **Mentes e máquinas:** Uma introdução à ciência cognitiva. Porto Alegre: Artes Médicas Sul. 1998. 180 p. Disponível em: <http://docente.ifrn.edu.br/avelinolima/disciplinas/filosofia-da-mente/livro-mentes-e-maquinas>. Acesso em 7 abr. 2016.

WEISSTEIN, E. W. **CRC Concise Encyclopedia of Mathematics.** 2 ed. Boca Raton: Champman & Hall/CRC. 2003. 1915 p. Disponível em: <https://books.google.com.br/books?id=D_XKBQAAQBAJ&lpg>. Acesso em: 30 set. 2016.