

UNIVERSIDADE FEDERAL DO PAMPA

DIÉSSICA DE SOUZA GOULART

UM ESTUDO SOBRE A APLICAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGOS

**Alegrete
2014**

DIÉSSICA DE SOUZA GOULART

UM ESTUDO SOBRE A APLICAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGOS

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal do Pampa, como requisito para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Fábio N. Kepler

**Alegrete
2014**

DIÉSSICA DE SOUZA GOULART

UM ESTUDO SOBRE A APLICAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGOS

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal do Pampa, como requisito para obtenção do Título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 26 de agosto de 2014.


Banca examinadora:



Prof. Dr. Fábio Natanael Kepler
Orientador
UNIPAMPA



Prof. Dr^a. Aline Vieira de Mello
UNIPAMPA



Prof. Me. Jean Felipe Patikowski Cheiran
UNIPAMPA

Dedico este trabalho a Elenir de Souza Goulart, minha mãe, exemplo de pessoa humilde e generosa.

AGRADECIMENTOS

A Deus que em todos os momentos está presente, nos acompanha e nos ampara quando tudo parece tão difícil.

À minha família, pelo apoio e por acreditar em mim.

Pai gostaria que tu estivesse aqui para poder me dar um abraço, mas onde o senhor estiver, saiba que te agradeço muito por me apoiar, tu foste muito importante. Mãe, teu amor e carinho incondicional, sempre me apoiando em todos os momentos me deram as forças necessárias para não desistir e só te admirar ainda mais.

Ao Acilio, meu padrinho querido, meu grande amigo que tem grande importância, pois sempre me incentivou e apoiou nos estudos.

Aos colegas e professores. Nesta jornada cada um ensinando um pouco do seu conhecimento e todos aprendendo juntos, fizeram cada semestre valer a pena.

A todas as pessoas que fazem parte desta Universidade que nos acolhe a todos os anos com ensino, novidades e sempre pensando no crescimento do aluno, muito obrigada.

“Agradeço todas as dificuldades que enfrentei; não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito.”.

Chico Xavier

RESUMO

Os jogos são ambientes ideais para uso de métodos de Inteligência Artificial (IA), pois existem modelos matemáticos bem definidos que auxiliam na implementação de soluções de problemas de busca. Por outro lado, cada problema deve ser analisado individualmente, pois deve-se efetuar uma modelagem específica para os dados e informações que serão utilizadas na solução de um algoritmo. Para minimizar o processamento, *scripts* com ações são aplicados a jogos para que os personagens executem estratégias definidas perante determinadas situações. Este trabalho estuda a aplicação do algoritmo Minimax que efetua busca em árvore para resolver problemas de decisões em jogos. Foram desenvolvidos sete protótipos com diferentes estratégias de avaliação e busca para aplicação no jogo 2048. O desempenho dos resultados foram analisados em relação ao tempo de execução e pontos obtidos. Com o Minimax foi possível resolver o 2048, mas em alguns casos a busca teve que ser limitada para que o computador não travasse a execução. Esse estudo mostra que é possível aplicar IA para resolver um jogo, mas a busca em árvore deve ser limitada para evitar travamentos.

Palavras-chave: Inteligência Artificial, Jogos, Problemas, Minimax, Desempenho.

ABSTRACT

Games are ideal environments for using Artificial Intelligence (AI) methods because there are well-defined mathematical models that help in implementing solutions for search problems. However, each problem must be analyzed individually, because there must be a specific modeling for data and information that will be used in the solution of an algorithm. To minimize the processing cost, scripts with actions are applied to games in order to make characters perform defined strategies in specific situations. This work studies the application of the Minimax algorithm for performing search in trees for solving decision problems in games. Seven prototypes were developed with different evaluation and search strategies in an implementation of the 2048 game. Performance results were analyzed in relation to runtime and score. The Minimax algorithm solves the 2048 game, but in some cases the search had to be limited so the computer would not lock execution. This study shows that it is possible to apply AI to solve a game, but the tree search should be limited to avoid locks.

Keywords: Artificial Intelligence, Games, Problems, Minimax, Performance.

LISTA DE ILUSTRAÇÕES

Figura 1 - Visão do agente e ambiente.....	29
Figura 2 - Agente baseado em objetivos.....	30
Figura 3 - Agente baseado em aprendizagem.....	31
Figura 4 - Personagem deve encontrar o caminho até sua recompensa.....	33
Figura 5 - Busca em profundidade.....	33
Figura 6 - Árvore Minimax.....	36
Figura 7 - Exemplo árvore Minimax para jogo da velha.....	37
Figura 8 - Algoritmo Minimax com limite de profundidade de pesquisa.....	38
Figura 9 - Poda Alfa-Beta.....	39
Figura 10 - Exemplos de jogos TD.....	45
Figura 11 - Protótipo para ambiente de jogo.....	46
Figura 12 - Jogo TD original.....	47
Figura 13 - Jogo TD modificado, protótipo 1.....	48
Figura 14 - Jogo 2048 original.....	49
Figura 15: Teclas para efetuar as jogadas.....	52
Figura 16: Estado inicial da partida.....	53
Figura 17: Primeira jogada.....	54
Figura 18: Segunda jogada.....	54
Figura 19: Terceira jogada.....	55
Figura 20: Tabuleiro completamente preenchido.....	57
Figura 21 - Jogo 2048 modificado.....	58
Figura 22 - Jogo 2048 em execução.....	59
Figura 23 - Vitória no jogo.....	60
Figura 24 - Tela de fim de jogo.....	60
Figura 25 - Protótipo D.....	64
Figura 26 - Protótipo E.....	65
Figura 27 - Pontos obtidos em todas as execuções.....	69
Figura 28: Tempos de execução dos protótipos.....	70
Figura 29 - Relação de médias de pontos por tempo de execução dos protótipos...	71
Figura 30: Vitórias obtidas.....	72

LISTA DE TABELAS

Tabela 1 - Protótipo A, resultados.....	61
Tabela 2 - Protótipo B, limites de profundidade.....	62
Tabela 3 - Protótipo B, resultados.....	62
Tabela 4 - Protótipo C, limites de profundidade.....	63
Tabela 5 - Protótipo C, resultados.....	63
Tabela 6 - Protótipo D, limite de profundidade.....	64
Tabela 7 - Protótipo D, resultados.....	65
Tabela 8 - Protótipo E, limite de profundidade.....	66
Tabela 9 - Protótipo E, resultados.....	67
Tabela 10 - Protótipo F, limites de profundidade.....	67
Tabela 11 - Protótipo F, resultados.....	68
Tabela 12 - Protótipo G, limites de profundidade.....	68
Tabela 13 - Protótipo G, resultados.....	69

LISTA DE SIGLAS

IA – Inteligência Artificial

TD – *Tower Defence*

AS 2 – ActionScript 2

3D – Três Dimensões

GPS - *Global Positioning System* - (Sistema de Posicionamento Global)

DVD - *Digital Versatile Disc* - (Disco Digital Versátil)

IBM - *International Business Machines*

CPU - *Central Processing Unit* – (Unidade Central de Processamento)

2D – Duas Dimensões

J2ME - *Java 2 Micro Edition*

mGBL - mobile Game-Based Learning

NPC - *Non-Player Character* - (personagem não jogável/manipulável)

IDE - *Integrated Development Environment* - (Ambiente Integrado de Desenvolvimento)

SO – Sistema Operacional

RAM - *Random Access Memory* – Memória de Acesso Aleatório.

SUMÁRIO

1 INTRODUÇÃO.....	25
1.1 Objetivos.....	26
1.2 Organização.....	27
2 FUNDAMENTOS.....	28
2.1 Agentes.....	28
2.2 IA e aprendizagem.....	31
2.3 Algoritmos de Busca de Caminho.....	32
2.3.1 Busca em profundidade.....	33
2.4 Função de avaliação.....	34
2.5 Algoritmo Minimax.....	34
2.6 Poda Alfa-Beta.....	38
2.7 Trabalhos relacionados.....	40
3 MÉTODOS, FERRAMENTAS E RECURSOS.....	44
3.1 Jogo Tower Defense.....	44
3.1.1 Recursos.....	46
3.1.2 Modelagem e objetivo do problema.....	47
3.1.3 Protótipo TD.....	47
3.1.4 Problemas encontrados.....	48
3.2 Jogo 2048.....	49
3.2.1 Objetivos do jogo.....	50
3.2.2 Ferramentas e recursos.....	50
3.2.3 Ambiente de desenvolvimento e testes.....	51
3.2.4 Notação.....	52
3.2.5 Como jogar.....	52
3.2.6 Blocos que podem ser formados.....	55
3.2.7 Pontuação.....	55
3.2.8 Criar blocos em posições vazias.....	56
3.2.9 Fim do jogo.....	56
3.2.10 Estratégia para evitar o fim do jogo.....	57
4 EXPERIMENTOS E RESULTADOS.....	58
4.1 Protótipo A: jogadas aleatórios.....	61
4.2 Protótipo B: Minimax.....	61

4.3 Protótipo C: Minimax com poda Alfa-Beta.....	62
4.4 Protótipo D: Minimax com poda Alfa-Beta e min modificado.....	63
4.5 Protótipos E, F e G: Minimax com poda Alfa-Beta e função de avaliação que recompensa os blocos no canto direito superior da tela.....	65
4.5.1 Protótipo E: limite profundidade de pesquisa 5.....	66
4.5.2 Protótipo F: limite profundidade de pesquisa 4.....	67
4.5.3 Protótipo G: limite profundidade de pesquisa 4 e 5.....	68
4.6 Considerações sobre 2048.....	69
5 CONSIDERAÇÕES FINAIS.....	73
REFERÊNCIAS.....	75

1 INTRODUÇÃO

Para muitos, o início dos jogos eletrônicos deve-se ao físico William "Willy" A. Higinbotham que em 1958 criou o primeiro jogo de videogame (BATISTA et al, 2007). Com uso de um osciloscópio para visualização da “quadra de tênis” e com um controle por botões podia-se jogar o famoso *Tennis for Two*, programado em um computador analógico militar em Nova Iorque (KALNING, 2008). A história dos jogos eletrônicos inicia-se com jogos executados em máquinas que não eram específicas para esta finalidade. Os gráficos eram simples, exibidos em um tubo de televisão com computadores espaçosos, porém identificava-se o grande potencial comercial visto o sucesso dos fliperamas na década de 70 (BATISTA et al, 2007).

O envolvimento dos jogadores acontece não apenas pela novidade, mas pelas emoções que o desafio de vencer uma máquina proporcionam e a cada nova partida melhorar seu *ranking* de pontuação. Para agregar valor ao jogo outros fatores devem ser levados em consideração como: estória apresentada, recursos de áudio, personagens da estória e imagens. O realismo dos gráficos também tem grande influência na aceitação pelos jogadores. Outra característica importante é a interação dos personagens com o ambiente de jogo. Não é aceitável encontrar um personagem “caminhando” contra uma parede ou executando movimentos incomuns à realidade proposta pelo jogo. Fatos como esse fazem com que o jogo perca sua credibilidade, conseqüentemente desapontando a expectativa do usuário.

Existem diversos consoles criados especificamente para executar jogos, mas atualmente a maioria dos dispositivos eletrônicos (televisores, GPS, leitores de DVDs, celulares, etc.), possuem jogos instalados. Independente da plataforma que o jogo foi disponibilizado, espera-se que ele seja executado de forma adequada e que o usuário tenha uma experiência agradável e sem erros.

Os jogos estratégicos ganham atenção e destacam-se pela disponibilidade em dispositivos de processamento limitado. Neste caso os gráficos não precisam ser elaborados, mas o jogo tem que propor algo que o torne desafiador e viciante. Pode-se citar como exemplo jogos de carta, cujo objetivo e regras geralmente são simples, mas depende-se das cartas que estão no monte, das cartas dos adversários e das próprias cartas recebidas.

Existem algoritmos específicos que podem ser utilizados para se resolver problemas de lógica, mas o computador não irá se tornar inteligente no sentido

objetivo da palavra, mas será capaz de analisar sistematicamente os grafos de estados em busca da melhor solução (KOVÁSZNAI et al, 2011).

Problemas que são difíceis para o ser humano resolver, como o cubo mágico (cubo de Rubik), que pode ser representado utilizando-se grafos, são facilmente resolvidos por um computador através de uma busca (KOVÁSZNAI et al, 2011). Com o uso do computador é possível se obter uma solução para o cubo em pouco tempo devido à capacidade de processamento da máquina ser mais rápida para avaliar os dados presentes no grafo de estados. No entanto, pode-se encontrar problemas com representações de estados muito grandes até mesmo para o processamento dos computadores mais rápidos (KOVÁSZNAI et al, 2011). Nesta situação entra a criatividade humana para elaborar estratégias de abstração de informações, tornando problemas grandes em problemas menores.

Para resolver um jogo com IA, deve-se primeiramente modelar o problema real e então partir para resolver o problema sobre as regras modeladas. Nesse estudo foram modelados dois jogos com diferentes características. O jogo no estilo *Tower Defense* – TD (torre de defesa, em tradução livre) que foi trabalhado, porém o projeto foi descontinuado. As explicações sobre o jogo TD podem ser encontradas na sessão 3.1 (onde são explicados as características do jogo e os motivos que levaram a esse protótipo ser descontinuado). O segundo jogo trabalhado no desenvolvimento deste estudo é um jogo de quebra-cabeça chamado 2048. O 2048 foi utilizado como base para aplicação do algoritmo de busca da área de IA denominado Minimax.

1.1 Objetivos

O objetivo principal deste trabalho é estudar o desempenho de um jogo executando algoritmos de Inteligência Artificial. Para isso, deve-se estudar a possibilidade de efetuar modificações nos algoritmos escolhidos para que se possa fazer comparações entre os resultados obtidos após a execução dos testes, e então efetuar discussões sobre esses resultados para identificar a viabilidade dessas execuções.

Pretende-se que a implementação do jogo com IA utilize o máximo de processamento da máquina e ainda assim apresente a execução do jogo na tela do computador sem travamentos. Caso sejam identificados travamentos de

processamento do ponto de vista do jogador, deve-se procurar soluções para minimizar esse problema.

Com o jogo 2048, utilizando o código disponibilizado por Virbel (2014), pretende-se mostrar o desempenho do jogo ao executar jogadas com a implementação do algoritmo Minimax com diferentes configurações de busca. Foram desenvolvidos sete protótipos, detalhados na sessão 4.

1.2 Organização

Este trabalho está organizado da seguinte forma: no capítulo 2 descreve-se os fundamentos teóricos do estudo de Inteligência Artificial, alguns conceitos necessários para a explicação dos algoritmos utilizados e alguns trabalhos relacionados na área de Inteligência Artificial aplicada a jogos. No capítulo 3 faz-se a explicação do jogo TD, um pouco sobre a história desse estilo de jogo, o protótipo trabalhado e os motivos que levaram à descontinuação desse jogo. Também no capítulo 3 apresenta-se o jogo 2048, as ferramentas e recursos utilizados, os objetivos do jogo e o sistema de pontuação. No capítulo 4 apresenta-se o desenvolvimento dos protótipos, as funções utilizadas e os resultados obtidos. E no capítulo 5 faz-se algumas considerações sobre os protótipos desenvolvidos, as dificuldades encontradas durante o desenvolvimento do trabalho e potenciais trabalhos futuros.

2 FUNDAMENTOS

Não há um consenso sobre a definição exata de Inteligência Artificial. Seriam sistemas que podem pensar e agir como seres humanos (RUSSELL et al, 2003)? Se IA significa que um carro pode dirigir por um percurso em um deserto sem motorista, então estamos perto da Inteligência Artificial. Se depender do sistema criar uma estória divertida e de forma independente, então estamos longe da IA (RUSSELL et al, 2003).

Uma das primeiras tentativas de se criar um agente inteligente para jogar xadrez aconteceu no ano de 1770, onde foi apresentado para sociedade europeia um jogador mecânico que podia mover peças em um tabuleiro¹ e formar estratégias perfeitamente sozinho, nomeado “O Turco”. Criado por Wolfgang Von Kempel, a máquina possuía um agente que podia mover as peças do jogo de xadrez e elaborar suas jogadas mais complexas. Porém em 1820 foi revelado que havia um operador humano para esta elaborada máquina e ele manipulava a mecânica para os movimentos. Esta ilusão contribuiu para que novos jogos surgissem inspirados neste pensamento de “máquinas inteligentes”. Percebeu-se que desenvolver jogos para apenas um jogador (*single-player mode*) poderia ser lucrativo, pois até o século XVIII existiam apenas jogos para dois ou mais jogadores (*versus mode* ou *multiplayer*).

Um dos primeiros jogos criados com técnicas de IA (SCHWAB, 2004) foi o *Space Invaders* 1978. Popularizado na sua época, foi desenvolvido por Tomohiro Nishikado. No entanto, estas estratégias apresentavam movimentos aleatórios ou mesmo *scripts* pré definidos como sendo a Inteligência Artificial aplicada.

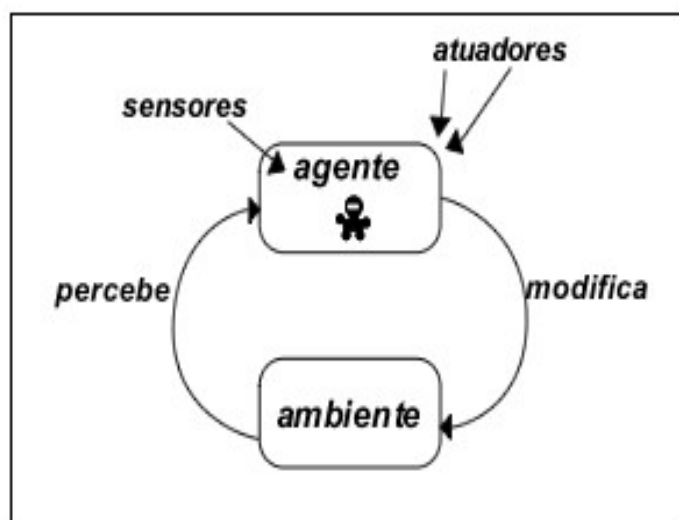
Existem muitas diferenças entre a IA acadêmica e IA para jogos. Enquanto a primeira busca o estudo e reconhecimento de padrões para problemas complexos, a segunda tem por objetivo a diversão dos jogadores com aplicações ilustrativas (DUBINA et al, 2009).

1 Fonte: http://pt.wikipedia.org/wiki/O_Turco

2.1 Agentes

Segundo Russell et al (2003) um agente é algo que pode ser visto como tendo percepção de um ambiente através de sensores e agindo no ambiente através de atuadores. Verifica-se na Figura 1 a esquematização de um agente idealizado por Russell et al (2003).

Figura 1 - Visão do agente e ambiente



Fonte: Baseado em Russell et al (2003)

Para Maes (1995) agentes inteligentes são sistemas computacionais que residem em ambientes dinâmicos e complexos. Eles percebem e atuam autonomamente, logo realizam um conjunto de objetivos e tarefas para os quais foram designados.

Segundo o Dicionário Michalis¹, agente é algo que exerce alguma ação; que produz algum efeito.

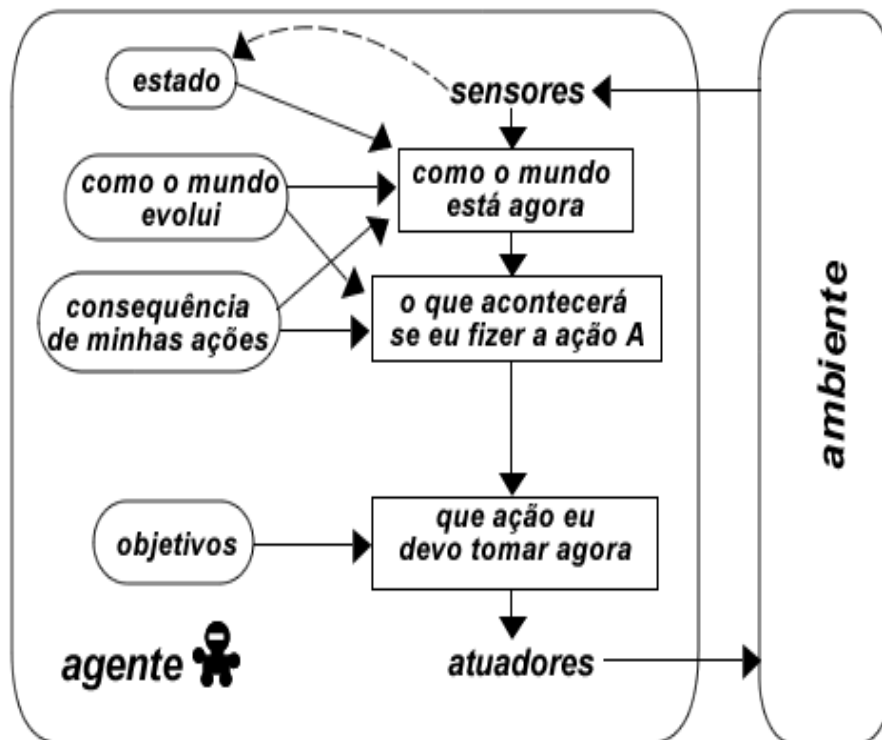
Já segundo Smith et al (1994) agente é uma entidade de software persistente que se caracteriza pela capacidade de possuir métodos próprios, dedicado a um propósito específico.

¹ Definição de agente disponível em: <http://michaelis.uol.com.br/moderno/portugues/index.php>. Acesso em 10 jun. 2012.

Para uma abordagem mais generalizada, pode-se acrescentar ao conceito de agentes a ideia de que um agente é todo personagem que possui objetivo(s). Como agentes desse tipo, chamados de **agentes baseados em objetivos**, pode-se citar inimigos que atrapalham o personagem controlado pelo jogador. Automaticamente o inimigo pode realizar ações como mudar de direção e usar alguma de suas habilidades para atacar.

A Figura 2 demonstra o fluxo de análise de estado de um agente baseado em objetivos conforme as informações que precisam ser consideradas no estado atual e o ambiente do jogo. O estado atual caracteriza como o mundo está neste instante de tempo. Através dos sensores o agente verificará as informações necessárias para gerar uma lista de ações possíveis e para cada ação considerar como o mundo pode evoluir em relação a como ele está. Após fazer a análise das consequências das ações possíveis o agente escolherá a ação conforme seus objetivos definidos previamente. Os atuadores representam os mecanismos que o agente utiliza para

Figura 2 - Agente baseado em objetivos

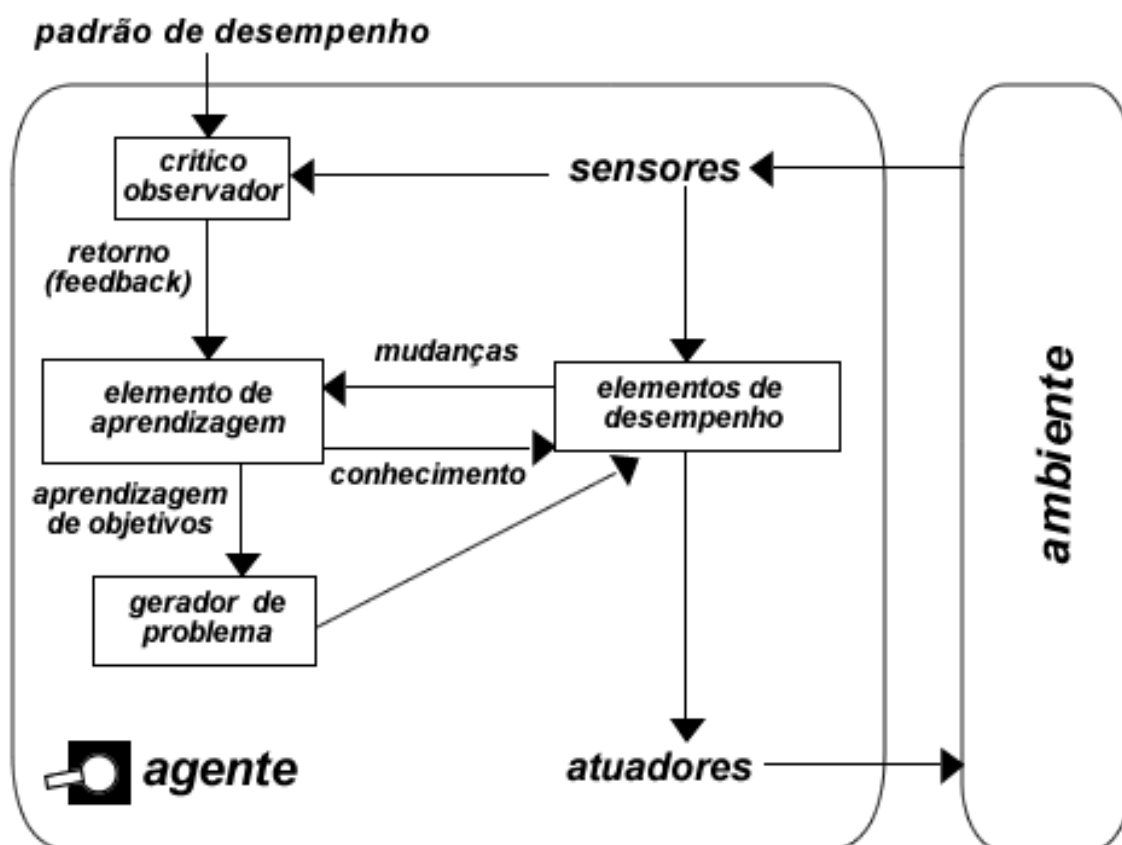


interagir com o ambiente do jogo e desta forma o fluxo de análise de estados do agente se repete a cada novo estado.

Existem também **agentes baseados em aprendizagem**, que podem atuar em um mundo inicialmente desconhecido e ao decorrer de suas ações aprenderão conforme as informações coletadas e seu desempenho, avaliando quão boa foi uma ação tomada em relação as outras executadas.

O elemento de aprendizagem contribui para tornar o agente mais eficiente e faz uso de *feedbacks* que avaliam as ações. O elemento de desempenho é responsável por selecionar as ações que o agente executará e o elemento geração de problemas sugere ações que podem trazer informações úteis para o contexto. Observe na Figura 3 o comportamento esperado de um agente baseado em aprendizagem.

Figura 3 - Agente baseado em aprendizagem



Fonte: Baseado em Russell et al (2003)

2.2 IA e aprendizagem

A inteligência e o aprendizado têm uma aplicação em comum, a intenção de se criar algo que possa aprender e interagir com o seu ambiente da melhor forma possível, ou seja, ter as capacidades de um ser humano.

Conseqüentemente algo dotado de Inteligência Artificial necessita de formas de aprendizagem para filtrar o que foi aprendido, assim como acontece com a inteligência natural. Porém para aplicar este entendimento a um sistema inteligente temos de abstrair muito do que a inteligência e o aprendizado realmente significam.

Desde a década de 70 projetos são elaborados para representar a inteligência humana. Tem-se como exemplo o *Deep Blue*, um computador com sistema criado pela IBM, em 1996, para a única função de jogar xadrez (SILVA, 2013). Com 256 processadores, era uma supercalculadora capaz de prever mais de 200 mil posições por segundo Silva (2013). Após algumas execuções e armazenando as jogas efetuadas pelo adversário Kasparov (considerado o maior jogador de xadrez de todos os tempos), o sistema da IBM venceu sob as regras normais de tempo.

O aprendizado humano está associado à capacidade de adaptação, correção, interação e representação do conhecimento adquirido (OSÓRIO, 1999). É justamente nestas características que a IA está evoluindo, pois muitos estudos estão sendo desenvolvidos para aprimoramento de sistemas inteligentes.

2.3 Algoritmos de Busca de Caminho

A busca de caminho é o processo de mover a posição de um personagem do jogo de sua localização inicial para um destino desejado (BOURG et al, 2004).

Existem vários métodos de busca. A solução para um problema depende de um estudo dos requisitos definidos para o mesmo. Alguns fatores devem ser considerados como: obstáculos presentes, a situação do ambiente, ações permitidas e proibidas, se o caminho mais curto é sempre o desejável, etc.

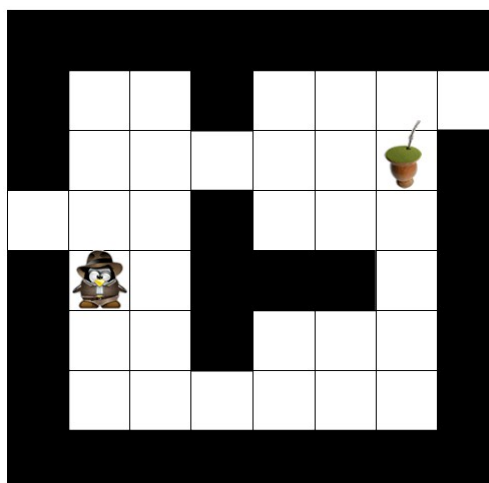
Pode-se entender melhor analisando as seguintes situações para os personagens adversários controlados pela máquina, por exemplo em um jogo de corrida:

- É admissível que os adversários corram sobre a água, ou eles devem encontrar uma ponte?

- Se houver uma parede, o veículo pode simplesmente atravessar, ou deve contornar o obstáculo?
- Para chegar à linha final, existem colinas que podem ser atravessadas. Este é o caminho curto, porém acidentado. O caminho longo é asfaltado e possibilita maior velocidade. Qual caminho é viável?

Após responder estas questões pode-se ter uma ideia mais refinada de qual algoritmo deve ser aplicável ao problema. Na Figura 4 tem-se um personagem em um labirinto e seu objetivo é encontrar a recompensa. Observando, pode-se concluir que ele tem duas opções para encontrar a recompensa, logo ele deve escolher o melhor, porém depende do tipo de busca que foi implementada para o agente.

Figura 4 - Personagem deve encontrar o caminho até sua recompensa

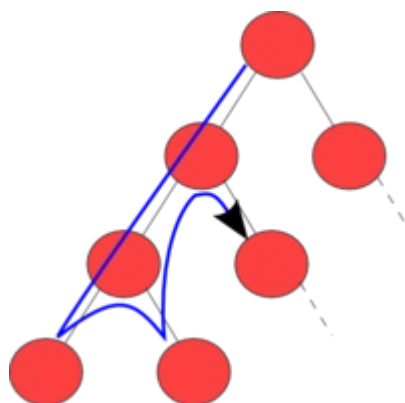


2.3.1 Busca em profundidade

A busca em profundidade caracteriza-se por explorar um ramo inteiro da árvore antes de tentar explorar um ramo vizinho, conforme Figura 5. É utilizada em problemas que possuem muitas soluções, mas não pode ser utilizado em árvores com profundidade infinita, pois pode ficar presa a um ramo (RUSSELL et al, 2003). Pode-se aplicar limites de profundidade para evitar que durante a execução do algoritmo não seja gerada toda a árvore de estados do problema estudado. Esse método de busca é denominada de Pesquisa com Profundidade Limitada¹.

1 Fonte: <http://www.gsigma.ufsc.br/~popov/aulas/ia/modulo3/index.html>

Figura 5 - Busca em profundidade



Fonte: http://pt.wikipedia.org/wiki/Busca_em_profundidade

2.4 Função de avaliação

Para se utilizar uma árvore de jogos, o algoritmo de busca precisa de uma função de avaliação de estados (SILVA, 2013). A Função de avaliação é uma função matemática criada para programas de jogos para estimar a utilidade de um estado s no Minimax e outros algoritmos relacionados (LAGO, 2006). A função de avaliação é modelada para ser simples e rápida, precisão não é um critério e calcula apenas o estado atual não considerando movimentos possíveis. O desempenho do programa depende da qualidade da função que quando projetada de forma errada pode levar o agente à derrota (LAGO, 2006).

O modelo mais comum para construir funções de avaliação é ponderar fatores que influenciam no estado do jogo (RUSSELL et al, 2003).

$$AVALIAÇÃO(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n (s)$$

Onde w = peso, f = característica avaliada e s = estado.

Por exemplo (LAGO, 2006), a avaliação material de cada peça em um jogo de xadrez:

r = número de rainhas

c = número de cavalos

p = número de peões

t = número de torres

b = número de bispos

$$pontuação = 9r(s) + 5t(s) + 3c(s) + 3b(s) + p(s)$$

$r(s) = (\text{número de rainhas brancas}) - (\text{número de rainhas pretas}), \text{ etc.}$
(RUSSELL et al, 2003).

2.5 Algoritmo Minimax

O Minimax é um método usado na Teoria da Decisão, Teoria dos Jogos, Estatística e Filosofia para minimizar a perda máxima possível na escolha de uma ação (CALADO et al, 2009). O algoritmo Minimax é uma busca em profundidade em uma árvore genérica que representa os estados possíveis do jogo, onde deve-se avaliar estados e tomar uma decisão como ação (SILVA, 2013).

A **árvore de jogo** é uma estrutura não linear composta por um conjunto de nós. Trata-se de uma relação de parentesco entre os nós (PAIVA, 2012). Segundo (SILVA, 2013), os elementos de uma árvore de jogo possuem as seguintes definições:

Nó raiz: estado inicial antes de qualquer movimento no jogo.

Nós internos da árvore: possíveis estados futuros do jogo.

Arcos ou **arestas:** representam os movimentos dos jogadores.

Níveis da árvore: movimentos dos dois jogadores. Ex.:

- Arcos no primeiro nível = movimentos do primeiro jogador.
- Arcos no segundo nível = movimentos do segundo jogador.

Nós folha: nó terminal, estado final de um jogo (vitória, derrota ou empate).

Nó objetivo: pode ser um estado final onde um jogador vence a partida.

O algoritmo Minimax baseia-se na suposição de que o adversário sempre irá escolher o movimento ideal e nunca incidir ao erro (SILVA, 2013).

Este teorema surgiu a partir da *Zero-Sum Game Theory* (Teoria dos Jogos de Soma Zero) e foi apresentado pelo matemático John Von Neumann, considerado o pai da teoria dos jogos em 1926 (FERGUSON, 2005). Na teoria *Two-person Zero-Sum Games*, entende-se que para qualquer jogo o ganho (recompensas) de uma pessoa é equivalente à perda (penalidades) de outra. Pode-se citar como exemplo: jogo da velha, xadrez, damas, etc (RUSSELL et al, 2003). A soma¹ dos valores conquistados por uma pessoa é igual as perdas combinadas do outro participante, ou seja, se os ganhos totais dos participantes são somados e as perdas totais são subtraídas, essa conta vai resultar em zero (RUSSELL et al, 2003).

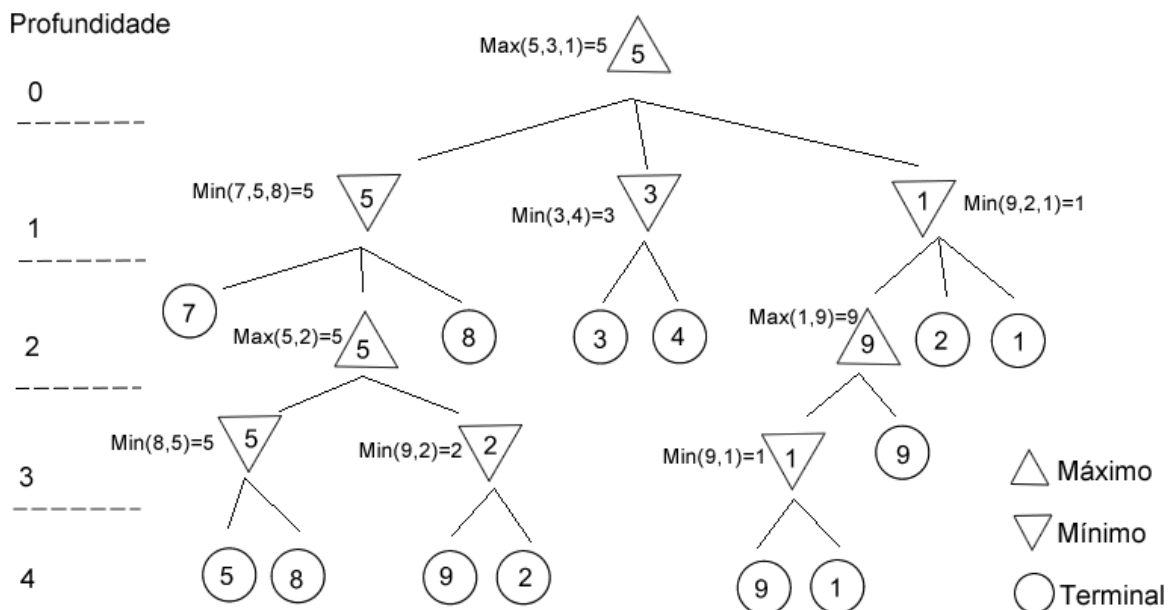
1 Fonte: <http://www.investopedia.com/terms/z/zero-sumgame.asp>

O Minimax é comumente aplicado a jogos com dois jogadores onde tem-se o conhecimento completo do jogo, as regras são simples e o estado atual é totalmente observável (FERGUSON, 2005).

Um jogador é chamado de **max** e deve avaliar todas as jogadas possíveis para o estado atual e executar a jogada que maximize o ganho. O outro jogador chamado **min**, deve avaliar todas as jogadas possíveis para o estado atual e executar a jogada que minimize as chances de **max** pontuar (RUSSELL et al, 2003).

A Figura 6 demonstra uma árvore Minimax que foi totalmente expandida e possui profundidade quatro. O **max** é representado por um triângulo, o **min** é indicado por um triângulo invertido e o nó folha ou terminal é representado por um círculo. Nesta ilustração pode-se identificar que cada nó terminal tem um valor atribuído. Após percorrer a árvore **max** escolhe o ramo com valor 5.

Figura 6 - Árvore Minimax

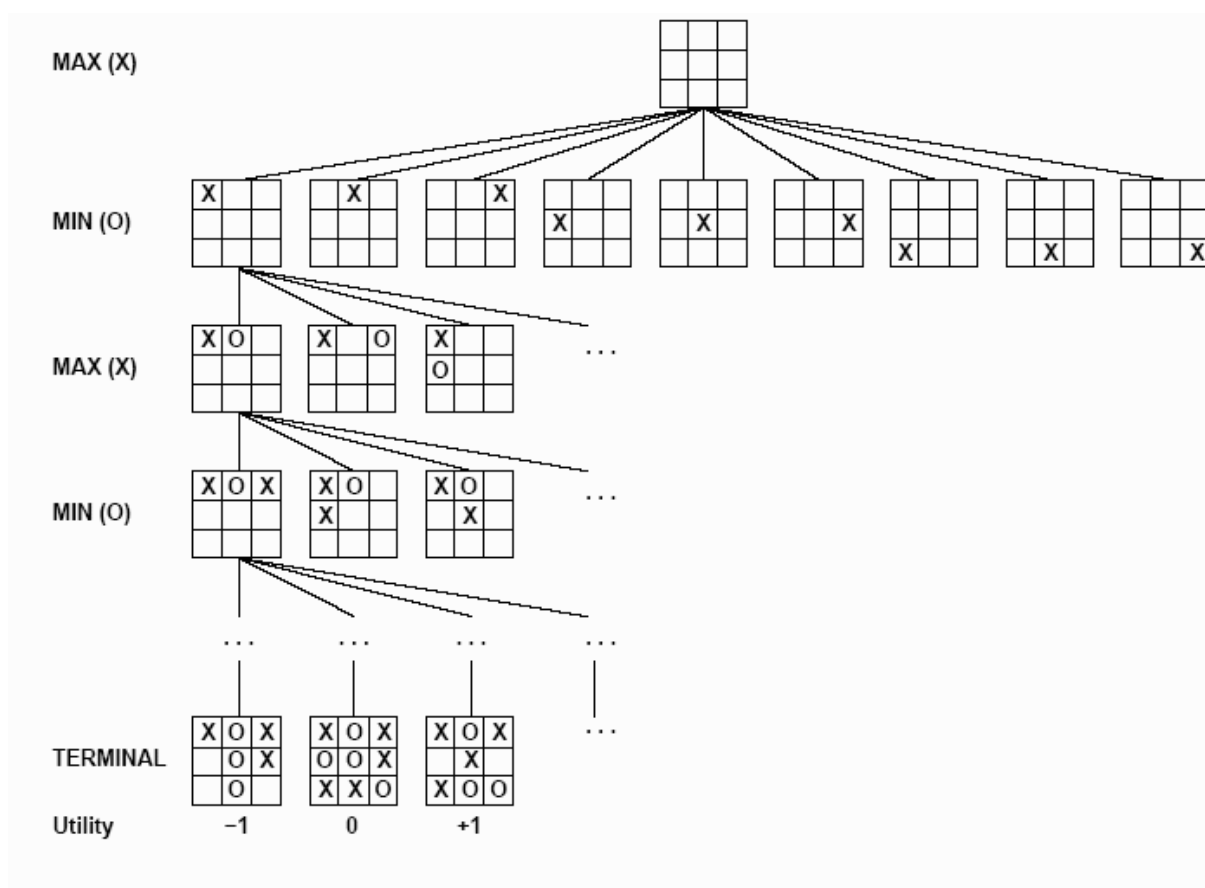


Fonte: <http://upload.wikimedia.org/wikipedia/commons/d/d6/Minimax2.png>

A Figura 7 demonstra um exemplo de árvore com os possíveis estados para o jogo da velha utilizando o Minimax como algoritmo de busca (RUSSELL et al, 2003). A profundidade 0 demonstra o estado inicial do tabuleiro onde **max** não efetuou nenhuma jogada, ou seja, o tabuleiro está vazio. Na profundidade 1 estão descritas todas as jogadas que o **min** prevê efetuar para minimizar as chances de **max**. Na

profundidade 2 apresenta-se os estados que **max** prevê para a segunda jogada do **min**. Na última profundidade apresenta-se três estados terminais e suas utilidades, ou seja, a avaliação de cada um dos terminais. O estado terminal a esquerda com valor (-1) “menos um” demonstra o fim da partida onde **min** vence com uma sequência de “O” alinhados, o estado ao lado com valor (0) “zero” demonstra que nenhum dos jogadores conseguiu criar uma sequência e o estado à direita com valor (+1) “mais um”, demonstra que **max** criou uma sequência de “X” alinhados.

Figura 7 - Exemplo árvore Minimax para jogo da velha



Fonte: <http://homepages.ius.edu/RWISMAN/C463/html/Chapter6.htm>

A Figura 8 apresenta um algoritmo recursivo que utiliza uma variável nomeada “profundidade” para receber o limite de profundidade de pesquisa a ser realizado e outra variável denominada “nó” que recebe o nó do estado atual. Primeiramente é realizada a verificação se o nó recebido é terminal ou se a profundidade de pesquisa na árvore é igual a zero, se uma dessas condições for verdadeira retorna-se o valor da heurística de avaliação do nó. Se o nó for falso para estas condições prossegue com a busca recursiva do valor **min** e **max** para

encontrar a melhor jogada. A condição que verifica se o nó representa a jogada de algum adversário é equivalente a verificação do **min** utiliza-se uma variável auxiliar nomeada de α “alfa” para receber o nó escolhido. A próxima condição retorna o resultado da busca pelo nó **max**.

Figura 8 - Algoritmo Minimax com limite de profundidade de pesquisa.

```

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
     $\alpha \leftarrow +\infty$ 
    PARA CADA filho DE nó
       $\alpha \leftarrow \min(\alpha, \text{minimax}(\text{filho}, \text{profundidade}-1))$ 
    FIM PARA
    RETORNE  $\alpha$ 
  SENÃO
     $\alpha \leftarrow -\infty$ 
    PARA CADA filho DE nó
       $\alpha \leftarrow \max(\alpha, \text{minimax}(\text{filho}, \text{profundidade}-1))$ 
    FIM PARA
    RETORNE  $\alpha$ 
  FIM SE
FIM ROTINA

```

Fonte: <http://pt.wikipedia.org/wiki/Minimax>

2.6 Poda Alfa-Beta

Mesmo limitando a profundidade de avaliação na árvore de busca, para cada nó surgem vários nós filhos em cada nível da árvore (RUSSELL et al, 2003). Neste caso, pode-se aplicar um método de limitação de exploração em alguns ramos da árvore de estados (KOVÁSZNAI et al, 2011).

Para identificar os nós que não precisam ser explorados, utiliza-se a poda Alfa-Beta (RUSSELL et al, 2003), que é uma alteração do algoritmo Minimax. Esta técnica não afeta o resultado final do algoritmo, ela serve para determinar os nós que não precisam ser explorados, eliminando processamentos desnecessários (SILVA, 2013).

Alfa é o maior valor para **max**, beta é o menor valor para **min** (RUSSELL et al, 2003). O algoritmo interrompe a avaliação de um ramo quando ele identifica, de maneira comprovada, que seu valor associado é pior do que outra jogada previamente examinada. Sendo assim, as jogadas posteriores não necessitam ser avaliadas (LAGO, 2006).

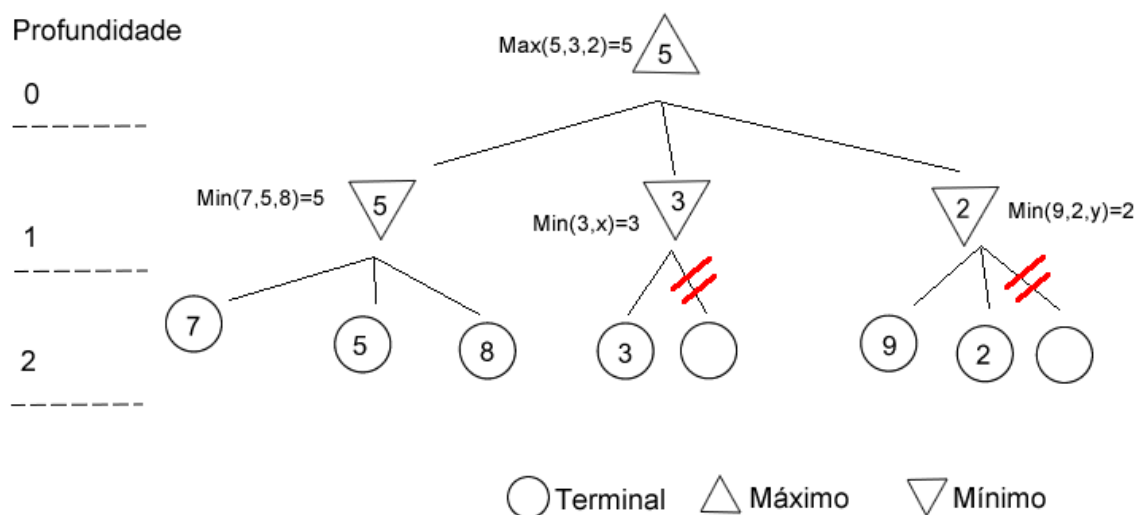
A Figura 9 apresenta um exemplo de busca na árvore de estados utilizando a poda. Partindo do estado inicial da árvore as comparações seguem a seguinte ordem:

$$\text{Min}(7) = 7$$

$$\text{Min}(7, 5) = 5$$

$$\text{Min}(5, 8) = 5$$

Figura 9 - Poda Alfa-Beta



Fonte: Imagem baseada na figura 12

O primeiro ramo foi verificado completamente, por enquanto **max** recebe o valor 5. No segundo ramo o nó folha visitado é igual a 3, **min** tem a informação de que **max** vale 5, então o restante do ramo não é verificado. No terceiro ramo o primeiro nó visitado tem valor 9, **min** tenta o segundo nó e faz mais uma comparação $\text{Min}(9, 2)$, agora **min** vale 2. Como 2 é menor que 5, o restante do ramo é descartado. Após o fim das buscas o nó raiz **max** faz sua última avaliação e escolhe o ramo com valor 5.

2.7 Trabalhos relacionados

Nesta sessão são apresentados e discutidos alguns trabalhos de pesquisa na área de Inteligência Artificial aplicada a jogos. Esses trabalhos são citados por abordarem o tema de jogos e Inteligência Artificial, não implicando, entretanto, que alguma de suas técnicas tenha sido incorporada no desenvolvimento deste trabalho.

O trabalho desenvolvido por Tatai (2006) foi relacionado ao estudo de técnicas aplicadas no desenvolvimento de jogos de computador. Ele relata a importância de criar personagens inteligentes que proporcionem um diferencial durante uma partida. Também descreve alguns gêneros de jogos, mas esta linha voltada para jogos para serem executados em hardwares específicos para execução de jogos.

O estudo feito por Karlsson (2005) aborda a principal dificuldade de pesquisas relacionadas à criação de jogos no Brasil: a falta de material e documentação para desenvolvimento de estudos sobre esta área. É um trabalho voltado ao estudo de padrões de projetos que possam ser adotados para criação de jogos. Verifica-se a relevância dos sistemas e as técnicas de aprendizagem e busca. É um estudo detalhado sobre as técnicas e conceitos acadêmicos sobre cada abordagem aplicadas ao *middleware* para desenvolvimento de jogos criado por Karlsson (2005).

O trabalho desenvolvido por Conceição (2009) apresenta um estudo sobre redes neurais para o jogo da velha de pinos em 3D. O sistema de redes neurais é aplicada a programas de aprendizagem, onde a medida que as informações, são analisadas o programa consegue aprender a jogar sozinho conforme as recompensas e punições sobre as ações efetuadas. Inicialmente as redes não sabem as regras do jogo da velha de pinos, o juiz é um algoritmo criado que define se as jogadas estão corretas. A partir de vários treinamentos as redes neurais aprendem as regras. A aprendizagem usada é por correção de erros, ou seja, as redes irão aprendendo as jogadas que são mais adequadas e possibilitam a vitória. Esta abordagem destaca as vantagens de se programar usando redes neurais, pois não é necessário que ela conheça o problema.

O estudo desenvolvido para o jogo da velha de pinos em 3D também relata a dificuldade de execução dos algoritmos de redes, pois dependendo da complexidade a execução necessita de hardware com processamento elevado. Algoritmos de

redes neurais têm a desvantagem de exigir poder de computação. No trabalho de Conceição (2009), foram utilizados computadores com recursos de hardware que possibilitaram os testes para esse jogo.

O trabalho desenvolvido por Yang (2010) apresenta um estudo sobre a possibilidade de melhorar o desempenho de execução de jogos em celulares. Para otimizar o desempenho do algoritmo de IA aplicado a um jogo de bilhar para celular, foi desenvolvido uma técnica para detecção eficiente de colisões. O algoritmo divide em duas fases os cálculos de colisão, dependendo dos resultados obtidos na primeira fase, há uma economia de processamento para segunda onde nem todos as bolas de bilhar sofrem colisão. Para garantir a melhora de desempenho, a programação do jogo foi feita em linguagem *assembly*, que é muito próximo da linguagem de máquina, o que garante economia em tempo durante a compilação do jogo. Segundo Yang (2010), os principais problemas para desenvolvedores de jogos para dispositivos móveis são: memória pequena para o jogo e CPU lenta para IA e gráficos 3D.

O estudo de Jhingut (2010) aborda programação para jogos executados em celulares. Para Jhingut (2010), os telefones móveis são como pequenos computadores, porém com recursos de processamento limitado. Mas são dispositivos compactos ideais para computação móvel e de jogos graças à sua popularidade. O estudo de Jhingut (2010) é direcionado aos jogos 3D que exigem maior resolução, desempenho e memória no telefone. O tempo de desenvolvimento e o número de recursos necessários pode ser até três vezes mais do que a necessária para um projeto 2D. Uma das principais preocupações é a memória, pois ao ultrapassar seu limite o jogo simplesmente não funcionará. Cada dispositivo tem um limite de memória utilizável por aplicativos, então deve ser assegurado que o jogo projetado pode ser executado no espaço disponível, sem ignorar a capacidade de processamento do telefone móvel (JHINGUT, 2010). Neste estudo foi desenvolvido um jogo *single player* com tema de *snowboard* e um jogo de tênis de mesa *multiplayer*. Ambos com detecção de colisões desenvolvidos com a linguagem de programação J2ME (*Java 2 Micro Edition*) e testados com a ferramenta *Sun Wireless Toolkit* para aplicações em Java. Ambos os jogos foram avaliados em relação a desempenho, gráficos, detecção de colisões, randomização, interatividade e portabilidade. Segundo Jhingut (2010) os principais problemas foram a portabilidade dos jogos e a complexidade de programação.

O estudo desenvolvido por Xin (2008) apresenta as influências de jogos que têm por objetivo o ensino de determinada temática através de aplicações para celulares. Denominados *Serious Game* (Jogo SériO, em tradução literal), eles tornaram-se muito populares inclusive, proporcionando o desafio para os programadores que precisam desenvolver desafios do contexto real para os jogos, como exemplo *SimCity* e *Civilization*, que são jogos de estratégia (XIN, 2008). Segundo o estudo, uma das principais dificuldades para este tipo de jogo é a dificuldade para coletar informações e manter o jogo atualizado. Outro fator importante é a programação da IA e aprendizagem do jogo, pois além do conteúdo a ser ensinado, as respostas do jogo devem ser planejadas durante o desenvolvimento da metodologia adotada.

No trabalho desenvolvido por Zaibon (2010) foi proposta uma estratégia de avaliação heurística que pode ser implementada para avaliar qualquer mobile Game-Based Learning - mGBL (aprendizagem baseada em jogo para celular, em tradução livre). Com quatro componentes a serem considerados: usabilidade, mobilidade, jogabilidade e conteúdos de aprendizagem. Os jogos mGBL tornaram-se populares por oferecer um modo prático para aprendizagem através de aplicações em dispositivos móveis (ZAIBON, 2010). Durante o desenvolvimento do jogo o método de heurística foi seguido para identificar inclusive o que o usuário gostaria de aprender. Segundo a avaliação obtida, o jogo obteve resultado satisfatório quanto à aplicação e transmissão do conteúdo de aprendizagem, porém a pesquisa comprovou que os alunos esperavam um jogo com contexto de aventura.

O trabalho desenvolvido por Rummell (2011) relata sua proposta para execução de algoritmos adaptativos simplificados para um jogo *Tower Defense* – TD. Um dos objetivos do estudo é identificar se o desempenho das torres e suas estratégias poderiam se equiparar a um jogador humano. O algoritmo adaptativo permite ao agente inimigo definir sua estratégia (RUMMELL, 2011). As ações das torres não são baseadas em métodos pré-programados, porém foram definidas algumas restrições: o jogo é capaz de construir um tipo de torre apenas; não é possível melhorar as torres. Essas restrições foram aplicadas para que a IA não precisasse se preocupar com compras, assim ele poderia construir uma nova torre sempre que ganhasse 10 pontos em dinheiro. Sem essa restrição as torres perderiam o jogo rapidamente, pois esperariam até juntar um número elevado de

pontos para comprar outras torres mais poderosas, pois seu treinamento ainda não estaria aperfeiçoado (RUMMELL, 2011).

Em seus testes iniciais, um dos problemas foi ensinar os algoritmos sobre a disposição das torres, pois uma torre em uma posição inadequada deveria ser descartada para inclusão de novas torres em posições mais úteis. A Inteligência Artificial foi alterada para que fosse identificado quando uma torre não obteve nenhum progresso. Segundo Rummell (2011), após 60 rodadas o algoritmo apresentou um desempenho de 50% em relação ao objetivo das torres para derrotar 2000 inimigos. Algumas alterações e melhorias foram necessárias para obter um comportamento satisfatório, mas não foi possível comparar ao resultado de um jogador humano.

3 MÉTODOS, FERRAMENTAS E RECURSOS

Alguns métodos de estudos acadêmicos não são aplicáveis ao desenvolvimento de jogos, e um dos motivos é o risco de um personagem adotar como válida uma ação que visualmente é inesperada. Outros pontos a serem considerados são o comportamento dos personagens, as ações e caminhos que serão executados e as estratégias que podem ser efetuadas no decorrer de cada nível de jogo.

Neste capítulo apresentamos na sessão 3.1 uma breve história dos jogos *Tower Defense*, os recursos utilizados para trabalhar nesse jogo, a modelagem criada para possível utilização de aprendizagem por reforço para o treinamento das torres, e os problemas encontrados que levaram à interrupção do projeto TD. Na sessão 3.2 apresentamos o jogo 2048 e suas características, estratégias e funcionamento. Também descrevemos algumas funções, notações utilizadas e as ferramentas e recursos para programação dos protótipos.

3.1 Jogo *Tower Defense*

Um dos jogos mais antigos no estilo *Tower Defense* – TD (torre de defesa, em tradução livre) foi disponibilizado na década de 80 o jogo *Bokosuka Wars* criado por Koji Sumii foi exportado para várias plataformas conforme Figura 5(a). Outro jogo antigo que influenciou jogos no estilo TD publicado em 1990 denominado *Ramparts* conforme Figura 5(b) criado para fliperamas, tinha por objetivo a defesa de territórios contra invasões. Um jogo TD recente e popular desenvolvido para iPhones e outras plataformas, denominado *Plants vs Zombies* (2009) conforme Figura 5(c), foi considerado o mais vendido na lista de downloads da empresa Apple (CTMAS, 2010).

Figura 10 - Exemplos de jogos TD



(a) 1983 - Bokosuka Wars

(b) 1990 - Ramparts

(c) 2009 Plants vs. Zombies

Fonte: (a) <http://darkscarfy.tripod.com/bokosuka/bokosuka.shtml>;

(b) <http://www.gamesdbase.com/game/arcade/rampart.aspx>;

(c) <http://itunes.apple.com/us/app/plants-vs-zombies/id350642635?mt=8>

Essa categoria de jogos é muito popular em sites, pois a qualidade de efeitos de transição, imagens, animações e histórias dos jogos agrada os internautas.

Um jogo TD é composto por vários níveis de dificuldade, o que torna a partida mais empolgante e exige do jogador o planejamento de estratégias rápidas. Conta com personagens controlados pelo computador, denominado *non-player character* - NPC (personagem não jogável, em tradução livre). Os personagens principais desse estilo que são as torres, devem impedir os inimigos de atingir um determinado objetivo ou local.

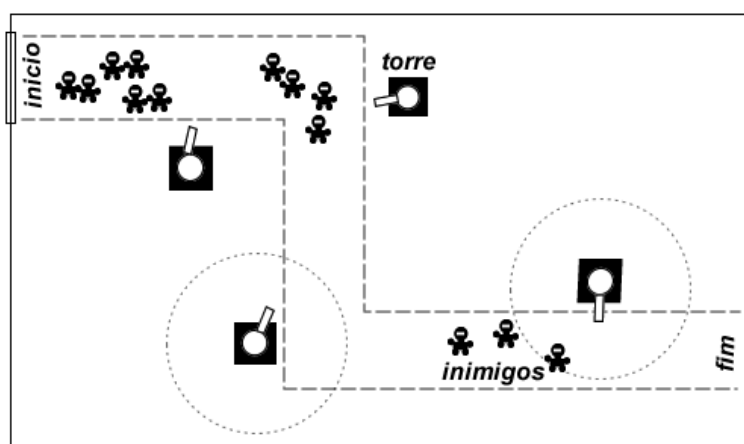
Em um jogo, as torres permitem várias possibilidades para o estrategista. No jogo para computador *Sol Survivor*¹ (2011), existem pesos de defesas que podem ser melhoradas se o jogador tiver pontos para efetuar as modificações de ataques. Pode-se alterar posicionamentos, tipos de armas e a cada nível a dificuldade aumenta. O jogador usa esses recursos para proteger seu território, controlando as torres enquanto o computador comanda os invasores NPC. Mas existem jogos em que o estrategista é o invasor e tem por objetivo destruir as torres, como por exemplo o jogo *Anomaly: Warzone Earth*² (2011), onde o jogador deve comandar comboios de soldados que atacam as torres defensivas.

1 Descrição por SMITH, M. The Three Best 3D Tower Defense Games For The PC, 2011. Disponível em: <http://www.makeuseof.com/tag/3d-tower-defense-games-pc/>. Acesso em 10 jun. 2012.

2 Jogo reverse TD Anomaly: Warzone Earth. Disponível em: http://en.wikipedia.org/wiki/Anomaly:_Warzone_Earth. Acesso em 10 jun. 2012.

A Figura 11 apresenta um protótipo que demonstra o fluxo de jogo geralmente encontrado na categoria de jogos TD. A partir de um ponto inicial os invasores percorrem uma trajetória que pode ser fixa, até seu objetivo que é o ponto final indicado na ilustração. As torres possuem sensores que identificam apenas os inimigos que estão dentro do perímetro de alcance de cada torre, conforme ilustrado na Figura 11, o círculo tracejado indica o alcance de disparo da torre. Apenas inimigos que se encontrarem dentro do círculo poderão ser atingidos.

Figura 11 - Protótipo para ambiente de jogo



3.1.1 Recursos

Pode-se encontrar o tutorial com explicações e o código TD¹ utilizado nesta etapa do estudo no link: <http://www.flashgametuts.com/tutorials/as2/how-to-create-a-tower-defense-game-in-as2-part-1>. A Figura 12 demonstra o jogo original obtido no site indicado. Esse jogo foi desenvolvido com a linguagem *Action Script 2*. Para efetuar as modificações, utilizou-se o programa Adobe Flash CS4, que permite trabalhar com a interface do jogo, efetuar modificações nos *scripts* e executar os testes necessários.

1 Jogo TD. Disponível em: <http://www.flashgametuts.com/tutorials/as2/how-to-create-a-tower-defense-game-in-as2-part-1/>. Acesso em jul. 2012.

Figura 12 - Jogo TD original



Fonte: www.flashgametuts.com

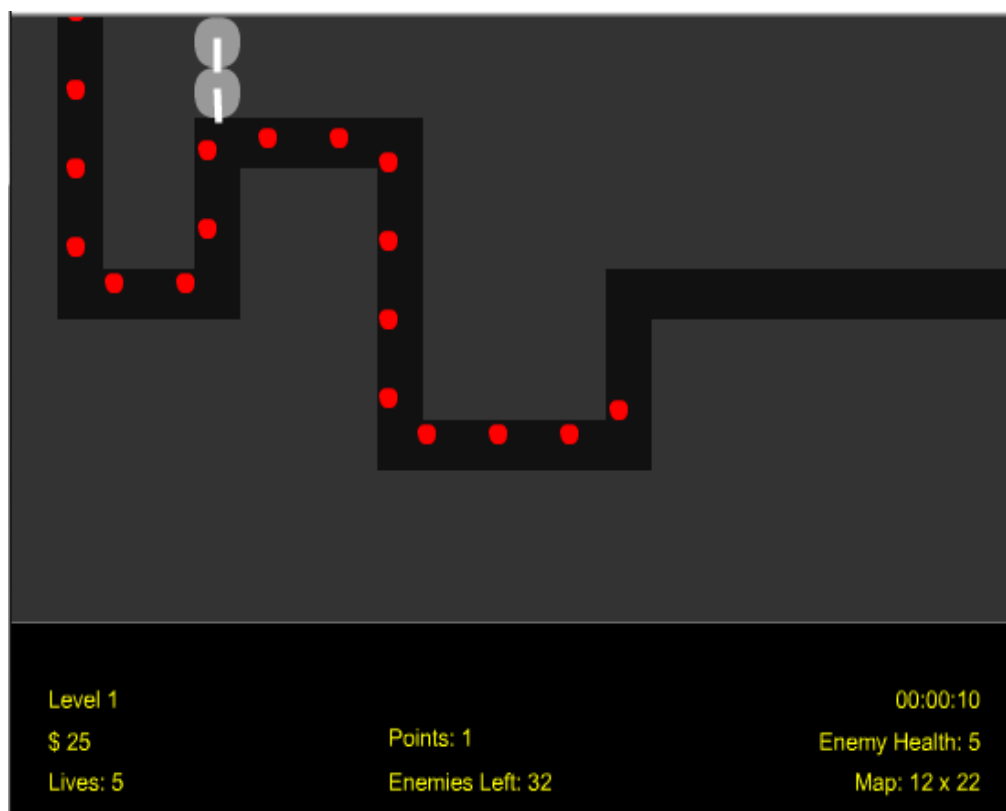
3.1.2 Modelagem e objetivo do problema

Inicialmente o objetivo do estudo era implementar um método de aprendizagem aplicado as torres do jogo TD disponibilizado pelo site www.flashgametuts.com. Para isso, o problema foi modelado para que cada estado guardasse amostras de torres construídas e analisar se elas poderiam ou não estar perto de uma estrada, no meio do tabuleiro, ou em uma saída. Para cada estado deveria executar uma ação conforme o valor de estado obtido.

3.1.3 Protótipo TD

A Figura 13 apresenta o jogo modificado executando sem algoritmo de IA, apenas são sorteadas posições para “criar” uma torre conforme a quantidade de dinheiro que o jogador possui. Pretendia-se implementar agentes com aprendizado por reforço, ou seja, ensinar as torres a identificar o melhor posicionamento no mapa do jogo.

Figura 13 - Jogo TD modificado, protótipo 1



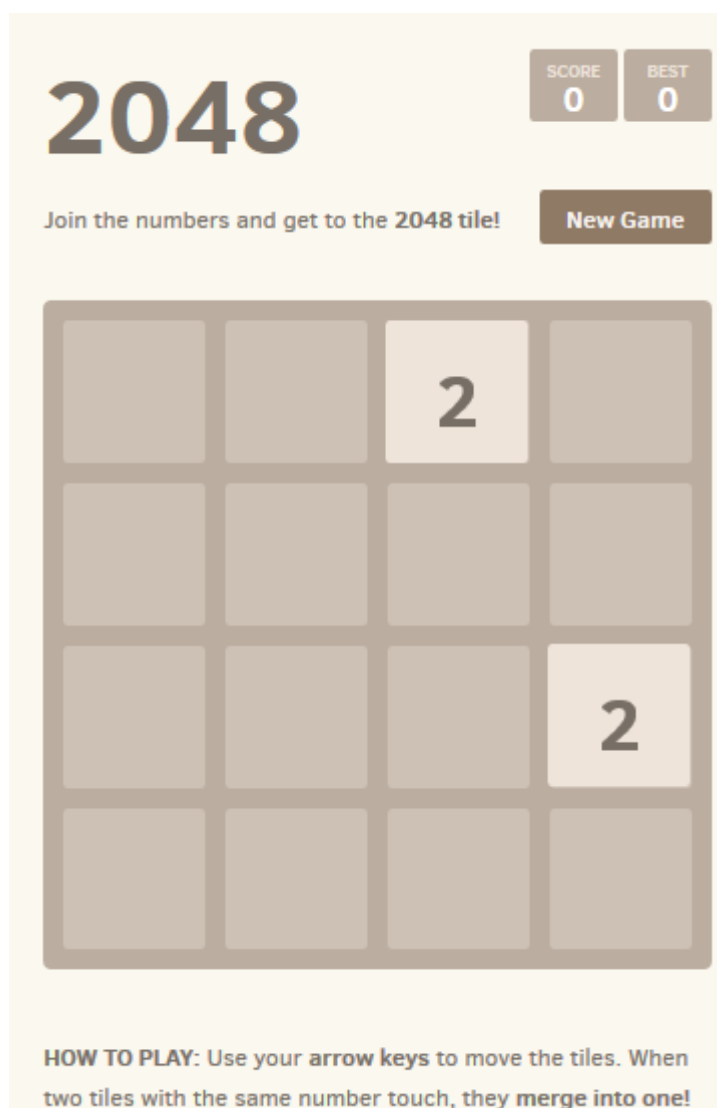
3.1.4 Problemas encontrados

Esse projeto foi descontinuado devido a alguns problemas encontrados. Muitas modificações tiveram que ser feitas para que o jogo pudesse efetuar jogadas randômicas. Mesmo com a modelagem pronta, encontrou-se dificuldade em criar funções específicas para cada agente. A linguagem *Action Script 2* (2003 – 2006) trabalha com códigos separados, ligados a cada animação que representa um agente e cada *frame* pode conter linhas de código, ou seja, a organização dos *scripts* depende das animações a que são atribuídos ou ao *frame* utilizado. Um dos objetivos era implementar o jogo TD para que fosse testado em um celular, porém os celulares que possuíam suporte as jogos em Flash para serem jogados com o uso do teclado do celular não são mais encontrados. Outro fator importante para decisão da troca do projeto para o jogo 2048 é que a linguagem AS2 está em desuso, dessa forma o estudo ficaria limitado.

3.2 Jogo 2048

O 2048¹ é um *single-player puzzle game* (jogo de quebra-cabeça para um jogador, em tradução livre) criado por (CIRULLI, 2014) em março de 2014. A Figura 14 apresenta uma imagem do jogo 2048 que pode ser acessado no seguinte link: <http://gabrielecirulli.github.io/2048/>.

Figura 14 - Jogo 2048 original



Fonte: <http://gabrielecirulli.github.io/2048/>

1 Fonte: http://en.wikipedia.org/wiki/2048_%28video_game%29

O jogo é um quebra-cabeça em que precisa-se utilizar estratégias para alinhar os blocos de mesmo valor, assim possibilitando combinações sem dispor blocos de valores elevados junto a blocos de valores menores. Segundo a revista¹ Info Abril no fórum *Hacker News*², o tópico sobre o jogo relata que alguns internautas ficaram mais de 40 minutos jogando para alcançar o bloco 2048, pois o jogo é viciante.

3.2.1 Objetivos do jogo

O desafio do jogador é formar um bloco com o valor 2048 e a cada nova partida tentar aumentar sua pontuação. A dificuldade do jogo é que após cada movimento efetuado pelo jogador um novo bloco com valor 2 ou 4 é criado pelo programa em um dos espaços vazios do tabuleiro. Quando não há espaços vazios no tabuleiro e nenhum movimento possível para o jogador efetuar, o programa verifica se algum dos blocos possui o valor de 2048: caso existam blocos com valor igual ou superior à 2048, é confirmada a vitória para o jogador; do contrário, o jogador não atingiu a meta e perde a partida. A melhor pontuação obtida fica registrada no campo *best* (conforme Figura 14) e o jogador pode iniciar outra partida para tentar atingir uma pontuação melhor.

3.2.2 Ferramentas e recursos

Para o desenvolvimento do estudo de IA aplicada ao jogo 2048 utilizou-se como base a implementação desenvolvida por Virbel (2014), disponível em <https://github.com/tito/2048>, e por Kepler (2014), disponível em <https://github.com/kepler/2048>. Os códigos do jogo disponibilizado foram desenvolvidos com a linguagem de programação Python 3 e Kivy. Para executar o jogo é necessário instalar o pacote de interpretação da linguagem Python 3.4, disponível em <http://www.python.org/getit/>.

Também é necessário instalar a biblioteca Kivy, pois ela é usada para a interface do jogo. A Kivy³ é uma biblioteca em Python que é utilizada para o desenvolvimento de interfaces de aplicativos que podem ser executados em

1 Fonte: <http://info.abril.com.br/games/noticias/2014/03/voce-e-capaz-de-formar-o-tijolo-numero-2048-neste-viciante-game-online.shtml>

2 Fonte: <https://news.ycombinator.com/news>

3 Fonte: <http://kivy.org/>

dispositivos móveis. Para o estudo utilizamos a Kivy 1.8.0 win 32 py 3.4, disponível em <http://www.lfd.uci.edu/~gohlke/pythonlibs/#kivy>.

Outro requisito para compilação do jogo é o módulo PyGame¹ utilizado para programação gráfica em Python (entre outras funcionalidades), principalmente para jogos. Foi utilizado o módulo Pygame 1.9.2a0 win 32 py 3.4, disponível em <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame>.

Para trabalhar com o projeto do jogo em Python utilizamos uma IDE - (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento), para efetuar correções e a programação dos protótipos. Utilizamos a IDE PyCharm 3.0, disponível em <http://www.jetbrains.com/pycharm/>.

Devido aos repositórios do jogo compartilhado por seus desenvolvedores estarem no site GitHub², foi feito um *fork*³ que pode ser acessado em <https://github.com/diess/2048>. Para isso foram utilizados os programas para controle de versões GitHub 1.3, disponível em <https://windows.github.com/>, e Git 1.9.2, disponível em <http://git-scm.com/>. Para acompanhamento de *commits*⁴ foi utilizado o aplicativo SourceTree 1.5.2, disponível em <http://www.sourcetreeapp.com/>.

3.2.3 Ambiente de desenvolvimento e testes

O ambiente de desenvolvimento utilizado foi um notebook com processador Pentium Dual Core 2.10GHz, 3GB memória RAM, SO de 32 bits Win 7. A escolha para o desenvolvimento e execução dos testes com a configuração dessa máquina deve-se ao propósito deste estudo de verificar a viabilidade de execução do programa, ou seja, testar o jogo em uma máquina comum. Dessa forma não é necessário que o *hardware* seja específico para execução de jogos, podendo ser um notebook. Teoricamente esse jogo criado com Python e Kivy pode ser compilado para gerar um pacote de instalação para dispositivos móveis com Sistema Operacional Android, porém problemas encontrados com o *Framework* para gerar o pacote impossibilitaram a realização de testes do jogo nestes aparelhos.

1 Fonte: <http://www.python.org.br/wiki/PyGame>

2 Fonte: <https://github.com/>

3 Ramificação ou bifurcação com base em um projeto existente.

4 Registros de mudanças experimentais no código fonte.

3.2.4 Notação

O tabuleiro pode ser representado com a notação de uma matriz 4x4 (quatro linhas por quatro colunas). A seguir representamos o tabuleiro onde a_{xy} corresponde a uma posição do tabuleiro na linha x e coluna y .

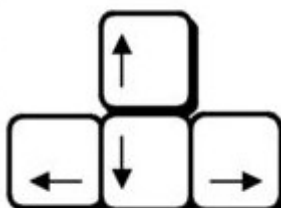
$$\begin{array}{cccc} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{array}$$

Essa notação de representação das posições do tabuleiro como matriz será utilizada nas próximas sessões que explicam o funcionamento do jogo.

3.2.5 Como jogar

Os blocos são “deslizados” (todos movidos de uma vez), e assim existem quatro jogadas possíveis: pode-se mover os blocos para a direita, esquerda, para cima ou para baixo. Para mover os blocos pode-se utilizar as teclas direcionais do teclado do computador conforme ilustrado na Figura 15.

Figura 15: Teclas para efetuar as jogadas



Fonte: http://s14.postimg.org/gs3lwx9c1/keyboard_r1_6.gif

Ao efetuar combinações com os blocos o jogador recebe pontos e consequentemente esvazia posições no tabuleiro que antes estavam ocupadas, desta forma evitando que o tabuleiro fique completo. Após cada movimento efetuado pelo jogador o programa sorteia uma das posições vazias do tabuleiro para criar um bloco com valor 2 ou 4.

As próximas figuras demonstram uma sequência de jogadas que exemplificam o funcionamento do jogo. As imagens foram capturadas durante a

execução do jogo trabalhado durante o desenvolvimento desse estudo. Como pode ser observado na Figura 16 essa imagem é diferente da Figura 14 (jogo original), mas mantém o propósito e jogabilidade do quebra-cabeça.

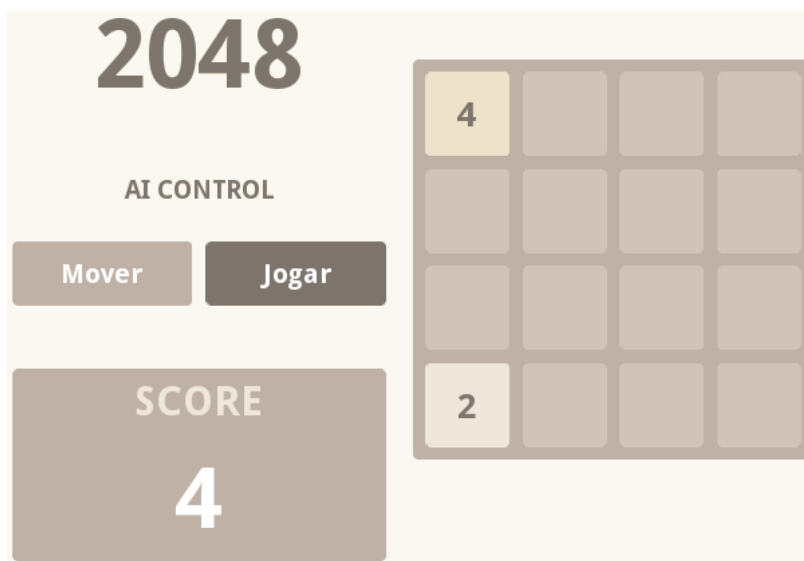
A Figura 16 demonstra a estado inicial de uma partida onde o programa iniciou de forma aleatória as posições a_{00} e a_{10} do tabuleiro com blocos de valor 2. O jogador deve escolher um dos três movimentos possíveis para efetuar, neste caso os blocos podem ser movidos para direita, para baixo ou para cima.

Figura 16: Estado inicial da partida



A Figura 17 demonstra que o jogador escolheu mover os blocos para cima combinando os blocos a_{00} e a_{10} , conseqüentemente criou-se um bloco na posição a_{00} com valor 4. Como a jogada resultou em uma combinação o jogador também recebeu como recompensa o valor da soma conforme demonstra o *score*. Observe também na Figura 17 que após o jogador ter clicado para efetuar a jogada um novo bloco com valor 2 foi criado na posição a_{30} . Observando o estado do tabuleiro existem apenas três jogadas possíveis: direita, para baixo ou para cima.

Figura 17: Primeira jogada



A Figura 18 demonstra que o jogador escolheu mover os blocos para direita e que o programa criou um bloco com valor 2 na posição a_{13} . Como nenhuma combinação foi concretizada o jogador não recebe pontos e seu score continua com valor 4. Nesse estado do tabuleiro o jogador pode mover os blocos para baixo, para cima ou para esquerda.

Figura 18: Segunda jogada



A Figura 19 demonstra que o jogador escolheu mover os blocos para cima combinando o bloco a_{13} com o bloco a_{33} formando um bloco de valor 4 na posição a_{13} . O valor dessa soma é acrescentado ao *score* do jogador que agora possui 8 pontos. É possível identificar que o jogador possui três jogadas possíveis, para cima ou para baixo unindo os dois blocos de valor 4 para um de valor 8 e consequentemente ficar com 16 pontos ou mover para esquerda e não efetuar nenhuma combinação. Dessa forma o jogo segue até o jogador não conseguir efetuar movimentos e a partida termina.

Figura 19: Terceira jogada



3.2.6 Blocos que podem ser formados

Claramente, as combinações sempre formam blocos que são potências de 2: 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, etc.

3.2.7 Pontuação

Cada vez que o jogador consegue combinar dois blocos de mesmo valor e formar um novo bloco que recebe o resultado da soma deles, o valor dessa soma é adicionado aos pontos do jogador no campo *score*.

3.2.8 Criar blocos em posições vazias

A função responsável por criar um bloco após o movimento efetuado pelo jogador recebe uma lista contendo as posições vazias no tabuleiro. Usa-se o método *choice* da linguagem Python para sortear uma posição presente na lista de vazios.

Após a etapa de escolha da posição deve-se escolher o valor do bloco, ou seja, se o bloco será 2 ou 4.

$$value = 2 \text{ if } random() < .9 \text{ else } 4$$

A linha de código acima demonstra que a variável **value** recebe o valor 2 se o valor sorteado pela função **random()** for menor que 0.9, se não **value** recebe o número 4. A função **random()** retorna um número entre 0 e 1, então a chance de o bloco criado ser 2 é de 90% e a chance do bloco ser 4 é de 10%.

3.2.9 Fim do jogo

É necessário que exista pelo menos um espaço vazio para que o programa crie um bloco 2 ou 4 após a última jogada efetuada pelo jogador. Quando não há mais nenhuma combinação possível, ou seja, não existem blocos de mesmo valor para que se possa efetuar a soma deles para liberar uma posição no tabuleiro para que o programa possa inserir um bloco de valor 2 ou 4 o jogo termina. Observe na Figura 20 que o tabuleiro está com todas as posições preenchidas, o jogador possui apenas uma jogada possível que é combinar os blocos sinalizados na imagem. É possível identificar que a melhor jogada é mover os blocos para direita e formar um bloco de valor 4, conseqüentemente este poderá ser combinado com o bloco a_{23} que possui valor 4, dessa forma evitando o fim do jogo.

Figura 20: Tabuleiro completamente preenchido



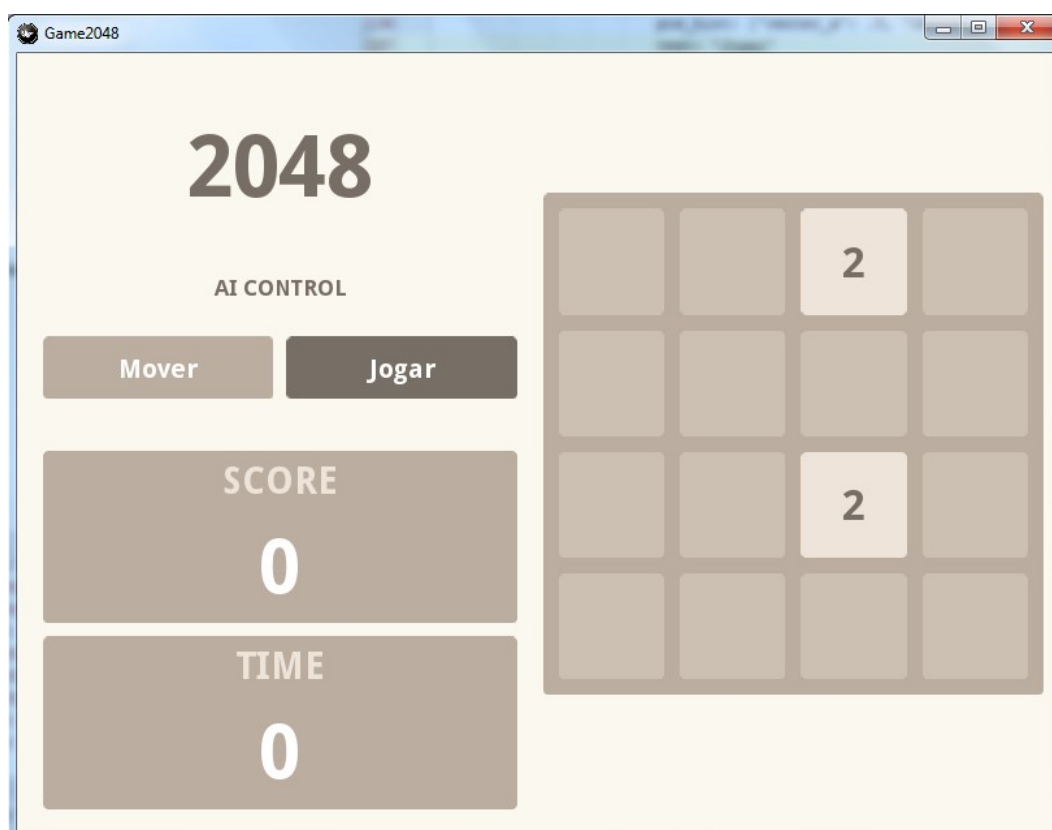
3.2.10 Estratégia para evitar o fim do jogo

Uma das estratégias utilizadas no estudo foi tentar não tirar blocos grandes dos cantos do tabuleiro para evitar que existam posições vazias próximas a esses blocos e, conseqüentemente, que surjam blocos pequenos nessas posições, o que impede a combinação entre blocos pequenos e grandes e assim contribui para que o tabuleiro fique sem espaços rapidamente.

4 EXPERIMENTOS E RESULTADOS

O jogo 2048 utilizado neste estudo apresenta algumas modificações em relação ao 2048 disponibilizado por Curulli (2014). A Figura 21 apresenta a tela inicial do jogo. À direita está o tabuleiro 4x4 com dois blocos dispostos de forma randômica. À esquerda encontra-se os botões, que serão explicados posteriormente. No painel, a sessão “score” apresenta a contagem dos pontos obtidos, e logo abaixo foi incluído um contador “time” que incrementa o tempo de execução. Esta contagem só funciona quando o jogo está executando de forma automática, ou seja, compilando os algoritmos implementados.

Figura 21 - Jogo 2048 modificado

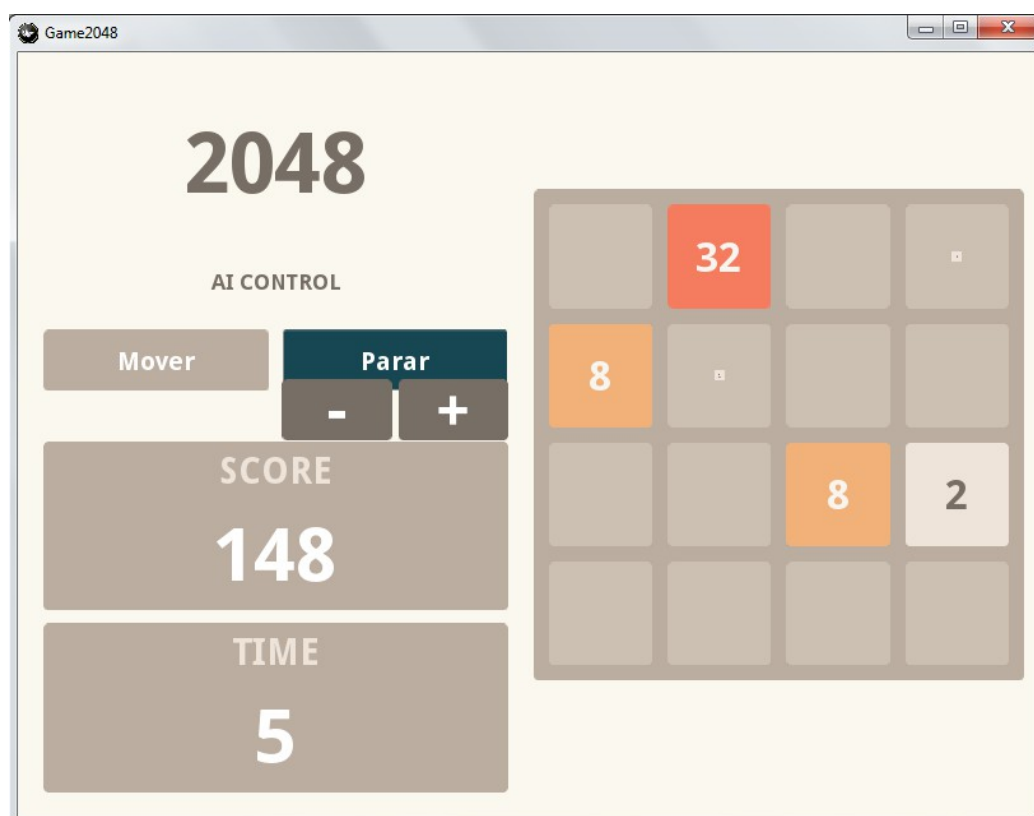


Botão **Jogar**: inicia a partida utilizando o algoritmo de execução implementado, ou seja, executa o jogo até o fim do jogo ou até o jogador clicar no botão **Parar**, Figura 22.

Botão **Mover**: efetua um movimento aleatório ao ser clicado.

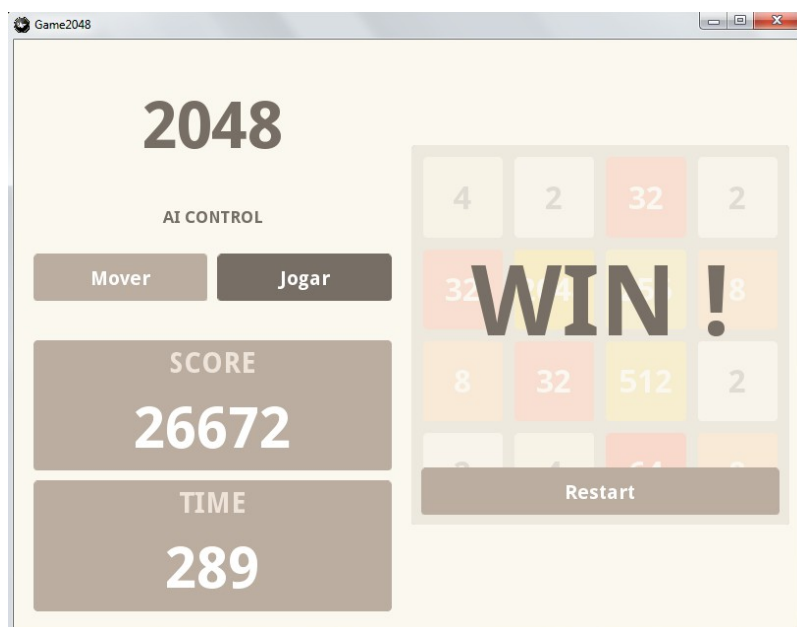
A Figura 22 apresenta uma imagem do jogo em execução, o botão anteriormente com o nome “Jogar” é alterado, ele recebe a cor azul e o nome de “Parar”. O botão **Parar** serve para interromper o jogo e a execução do algoritmo de IA. Abaixo do botão “Parar” encontram-se dois botões pequenos um para aumentar (+) a velocidade de execução do jogo com IA e outro para diminuir (-) a velocidade.

Figura 22 - Jogo 2048 em execução



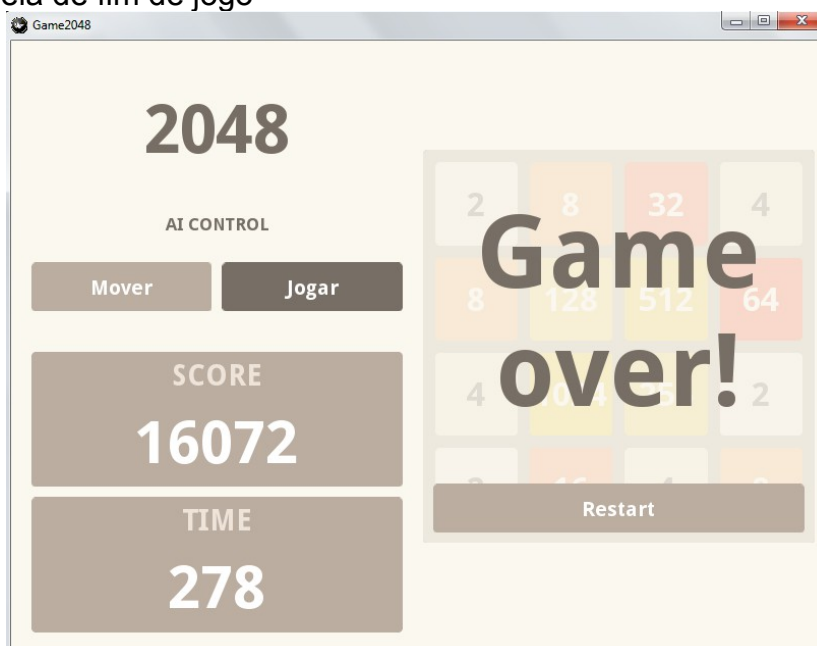
Na Figura 23 é apresentada a tela final do jogo após uma vitória. Sobre o tabuleiro encontra-se o botão **Restart** que ao ser pressionado inicia uma nova partida.

Figura 23 - Vitória no jogo



A Figura 24 demonstra a tela de jogo após uma derrota, semelhante à tela anterior. Essas imagens foram obtidas com a execução do protótipo D.

Figura 24 - Tela de fim de jogo



4.1 Protótipo A: jogadas aleatórios

Na Tabela 1 apresenta-se os resultados das execuções do algoritmo que simula uma pessoa aplicando movimentos escolhidos aleatoriamente (forma randômica). O tempo médio de execução foi de 30 segundos e pontuação (*score*) de 1116, um valor que corresponde a menos que a metade do melhor score que foi 2348. Não obteve-se nenhuma vitória.

Tabela 1 - Protótipo A, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	53	1404	0
2	22	612	0
3	38	1568	0
4	20	580	0
5	43	2348	0
6	32	1448	0
7	19	548	0
8	39	2092	0
9	23	768	0
10	14	272	0
Média	30	1164	0

4.2 Protótipo B: Minimax

Neste protótipo foi aplicada o método Minimax com uma função de avaliação **soma ao quadrado**, ou seja, cada estado será avaliado conforme os valores dos blocos presentes no tabuleiro.

$avaliação(s) = (Vb_{(0,0)})^2 + (Vb_{(0,1)})^2 + \dots + (Vb_{(3,3)})^2$, onde $Vb_{(i,j)}$ é o valor do bloco na posição (i, j) do tabuleiro. Sendo i o número da linha, j o número da coluna.

Também foi definido o limite de profundidade de expansão da árvore de avaliação dos estados. Devido à quantidade de estados possíveis, tornou-se inviável gerar toda a árvore visto que um estado pode gerar outros 16 estados ou mais.

Na Tabela 2 são apresentados os limites de profundidade a serem explorados na árvore, conforme o número de blocos vazios no tabuleiro. Estes foram os limites mais aceitáveis para que fossem evitados os travamentos do computador durante a execução e as jogadas exibidas na tela fluíssem o mais naturalmente possível. No caso de apenas 1 ou 0 bloco vazio, forçou-se a descida na árvore até o terceiro nível

Tabela 2 - Protótipo B, limites de profundidade

Vazios	Profundidade
14 a 10	1
9 a 4	1
3 a 2	2
1 a 0	3

para tentar melhorar as chances de vitória, com isto, em muitos casos o computador apresentou travamentos, mas o jogo foi resolvido até o final da partida.

Observando a Tabela 3 pode-se verificar que comparado ao protótipo A, a média de pontuação teve uma melhora de praticamente 90%. O tempo de execução também teve um aumento significativo. Neste caso todas as possibilidades de estados são verificadas, ou seja, mesmo que um estado apresente uma avaliação inferior se comparado a outros estados, este nó será expandido. Neste caso ocorre uma verificação desnecessária cujo processamento poderia ser usado para explorar os nós com avaliações superiores. Mesmo com esta estratégia, não obteve-se vitória.

Tabela 3 - Protótipo B, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	176	3916	0
2	538	11328	0
3	757	16088	0
4	271	7060	0
5	537	14436	0
6	524	14192	0
7	392	7344	0
8	299	11976	0
9	110	3268	0
10	271	7344	0
Média	388	9695	0

4.3 Protótipo C: Minimax com poda Alfa-Beta

No protótipo C utilizou-se o Minimax com poda Alfa-Beta. Neste caso foi possível aumentar a profundidade de expansão da árvore como pode ser visto na Tabela 4. Também utilizou-se a função de soma dos blocos ao quadrado. Somente nos casos em que o algoritmo expande até o terceiro nível foi possível observar

alguns travamentos, o que aumentou o tempo de execução, mas sem prejudicar a fluidez das jogadas exibidas na tela.

Tabela 4 - Protótipo C, limites de profundidade

Vazios	Profundidade
14 a 10	1
9 a 4	2
3 a 2	3
1 a 0	3

Conforme pode ser visto na Tabela 5, desta vez obteve-se a primeira vitória com mais de 27 mil pontos, e conseqüentemente o jogo ficou em execução por um tempo superior de aproximadamente 16 minutos. A média de tempo de execução foi de 598 s, em torno de 9 minutos.

Tabela 5 - Protótipo C, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	160	3332	0
2	501	13504	0
3	1016	27380	1
4	396	7384	0
5	653	16292	0
6	817	16312	0
7	520	12276	0
8	745	15516	0
9	739	14556	0
10	432	11956	0
Média	598	13851	0,1

4.4 Protótipo D: Minimax com poda Alfa-Beta e min modificado

Neste protótipo foram efetuadas modificações na função que gera os próximos estados para cada jogada do **min**. Essa função considera apenas jogadas em que um bloco 2 surge no canto superior direito do tabuleiro. Esta estratégia foi elaborada para limitar a quantidade de jogadas.

Essa modificação aplicada na função **min** permitiu que para todos os casos de número de blocos vazios a pesquisa fosse expandida até a profundidade 4, conforme apresentado na Tabela 6.

Tabela 6 - Protótipo D, limite de profundidade

Vazios	Profundidade
14 a 10	4
9 a 4	4
3 a 2	4
1 a 0	4

A função de avaliação utilizada foi a de soma dos blocos ao quadrado. Na Figura 25, pode-se observar que os blocos estão dispostos de forma irregular, ou seja, embora se tenha obtido quatro vitórias conforme apresentado na Tabela 7, essa função de avaliação não contribuiu para que os blocos com valores maiores fiquem alinhados nas posições laterais do tabuleiro. Ao deixar as posições centrais livres torna-se possível efetuar melhores jogadas.

Figura 25 - Protótipo D.

2	32		2
4	512		
64	4	4	2
4	2	2	2

Com esta nova abordagem obteve-se também um tempo médio inferior aos protótipos B e C, com execuções em torno de 4 minutos. O tempo reduziu e a pontuação aumentou, como pode ser conferido na Tabela 7.

Tabela 7 - Protótipo D, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	262	25364	1
2	267	25508	1
3	329	30524	1
4	207	15992	0
5	189	16048	0
6	101	7156	0
7	226	16212	0
8	149	12112	0
9	280	26740	1
10	202	15976	0
Média	221	19163	0,4

4.5 Protótipos E, F e G: Minimax com poda Alfa-Beta e função de avaliação que recompensa os blocos no canto direito superior da tela

Os protótipos E, F e G foram modificados para executar a função de avaliação que recompensa os blocos de maior valor que estão no canto direito superior da tela, como pode ser observado na Figura 26. Essa estratégia é uma indução do conhecimento sobre o jogo, pois o próprio autor do jogo Curulli (2014) indicou essa e outras estratégias¹ para solucionar o problema.

Figura 26 - Protótipo E

4	2	64	256
4	8	32	128
2			4
2	8		

1 Fonte: <http://m.softonic.com.br/artigos/como-vencer-jogo-2048-criador-revela-segredos>

$$avaliação(s) = \sum_{i=0, j=0}^3 ((i+1)*(j+1) * Vb_{(i,j)})$$

avaliação(s) = ((0+1)*(0+1)*Vb_(0,0)) + ...+ ((3+1)*(3+1)*Vb_(0,0)), onde **Vb_(i,j)** é o valor do bloco na posição (**i**, **j**) do tabuleiro. Sendo **i** o número da linha e **j** o número da coluna.

Com a função **min** modificada e a função de avaliação que recompensa os blocos induzimos o algoritmo a dispor os blocos maiores no canto direito do tabuleiro. Essa estratégia evita que um bloco de valor 2 fique entre os blocos maiores, o que normalmente acontece e leva o jogador a derrota rapidamente. A diferença básica entre os protótipos E, F e G é o limite de profundidade de pesquisa na árvore de estados. Esta abordagem foi adotada justamente para identificar se existem alterações significativas de tempo de execução e pontuações obtidas com essas alterações.

4.5.1 Protótipo E: limite profundidade de pesquisa 5

Nesta abordagem foi possível limitar a profundidade de expansão da árvore para o nível 5, como descrito na Tabela 8. A Figura 26 demonstra o algoritmo em execução já com os blocos sendo dispostos no canto do tabuleiro, conforme a estratégia de avaliação e a função **min** modificada.

Tabela 8 - Protótipo E, limite de profundidade

Vazios	Profundidade
14 a 10	5
9 a 4	5
3 a 2	5
1 a 0	5

A Tabela 9 apresenta os resultados das execuções do protótipo E. Neste protótipo obteve-se 6 vitórias, com melhor pontuação se comparadas aos protótipos anteriores. Porém o tempo de execução aumentou devido ao número de estados que são gerados, visto que o algoritmo está percorrendo a árvore até a profundidade 5, que foi o limite encontrado para que o jogo executasse com alguns travamentos aceitáveis. O maior tempo obtido nesta tarefa foi de 671 s. ~ 11 minutos.

Tabela 9 - Protótipo E, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	518	27560	1
2	591	32616	1
3	597	27552	1
4	376	15208	0
5	584	27100	1
6	415	16712	0
7	607	27088	1
8	426	16760	0
9	671	32424	1
10	371	11548	0
Média	516	23457	0,6

4.5.2 Protótipo F: limite profundidade de pesquisa 4

Nesta abordagem utilizou-se exatamente o mesmo algoritmo do protótipo E, no entanto, limitou-se a profundidade de pesquisa na árvore para o nível 4, conforme apresentado na Tabela 10.

Tabela 10 - Protótipo F, limites de profundidade

Vazios	Profundidade
14 a 10	4
9 a 4	4
3 a 2	4
1 a 0	4

Na Tabela 11 encontram-se os resultados desta abordagem. Os resultados obtidos foram melhores que os do protótipo D, mas inferiores aos resultados do protótipo E. Neste caso obteve-se 50% de vitórias e uma média de pontuação de 19 mil. Também resultou em um tempo melhor que o protótipo E, pois as buscas até o nível 4 apresentam poucos travamentos e menos nós para avaliar.

Tabela 11 - Protótipo F, resultados

Ordem	Tempo (s)	Pontos	Vitórias
1	377	35964	1
2	175	11688	0
3	300	27872	1
4	279	26468	1
5	147	12140	0
6	140	11632	0
7	238	22956	1
8	293	27324	1
9	190	15868	0
10	104	7264	0
Média	224	19918	0,5

4.5.3 Protótipo G: limite profundidade de pesquisa 4 e 5

Neste protótipo utilizou-se uma configuração mista para busca na árvore, conforme demonstra a Tabela 12. Para 4 a 14 blocos vazios foi determinado que a árvore fosse expandida até o nível 4. Para 0 a 3 vazios a árvore deve ser expandida até o nível 5. Esta abordagem foi adotada para verificar se os resultados obtidos seriam melhores ou intermediários se comparados com os protótipos E e F.

Tabela 12 - Protótipo G, limites de profundidade

Vazios	Profundidade
14 a 10	4
9 a 4	4
3 a 2	5
1 a 0	5

Diferente do que se poderia supor, os resultados obtidos foram inferiores em relação aos dois últimos protótipos. Na Tabela 13 pode-se observar que apenas em uma execução se obteve uma pontuação boa que levou à vitória com 28 mil pontos. O restante das execuções apresentaram tempos e pontos semelhantes, o que demonstra que esse algoritmo modificado obtêm melhores resultados apenas com a busca com mesmo limite para todos os casos de blocos vazios.

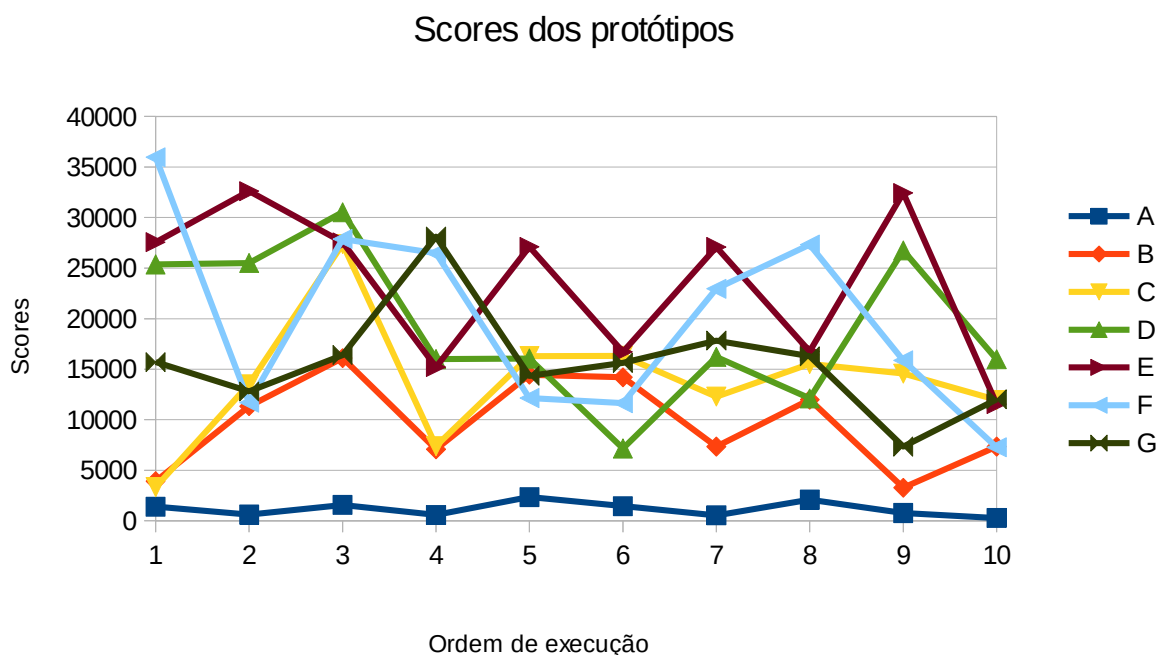
Tabela 13 - Protótipo G, resultados

Ordem	Tempo(s)	Pontos	Vitórias
1	226	15692	0
2	175	12792	0
3	227	16388	0
4	369	28120	1
5	221	14336	0
6	236	15624	0
7	259	17816	0
8	226	16280	0
9	113	7344	0
10	160	12024	0
Média	221	15642	0,1

4.6 Considerações sobre 2048

Os resultados das execuções ilustradas na Figura 27 mostram que os algoritmos testados apresentaram resultados variáveis. Apenas o protótipo A obteve resultados mais homogêneos, porém inferiores aos outros. Tentar resolver o problema com jogadas aleatórias não é uma solução aceitável. O uso de estratégias comprova a sua necessidade para este problema.

Figura 27 - Pontos obtidos em todas as execuções

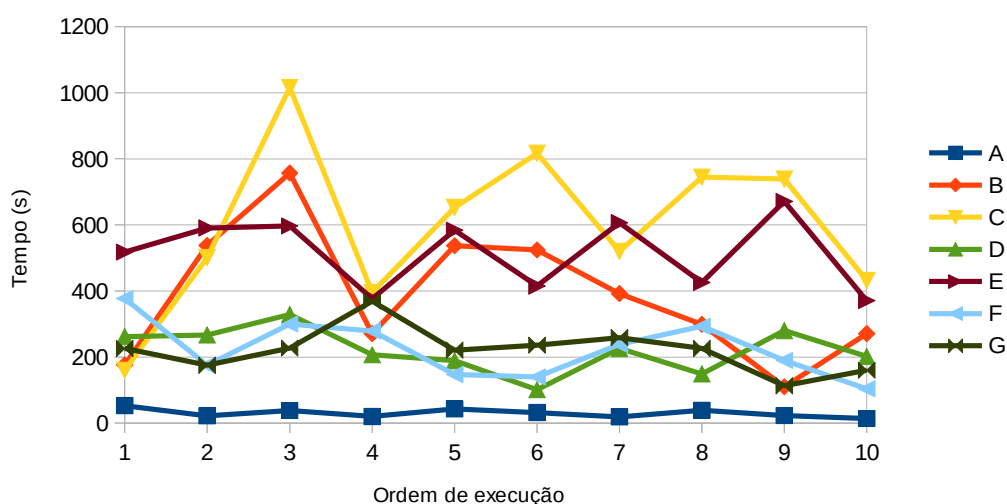


O protótipo B, que usa o algoritmo Minimax e avalia o estado conforme a soma dos blocos, além de analisar a melhor jogada para obter a vitória, mostrou-se um gráfico com pontuações variáveis. Por exemplo, na terceira execução obteve-se 16 mil pontos enquanto na quarta execução foram obtidos 7 mil pontos. Essas variações podem ser identificadas nos outros protótipos, embora a pontuação tenha melhorado. É provável que as funções de avaliação não tenham considerado características suficientes dos estados para que obtivessem pontuações constantes.

Na Figura 27 observa-se que a pontuação mais elevada foi obtida na primeira execução do protótipo F com mais de 35 mil pontos, no entanto as execuções posteriores não obtiveram o mesmo sucesso.

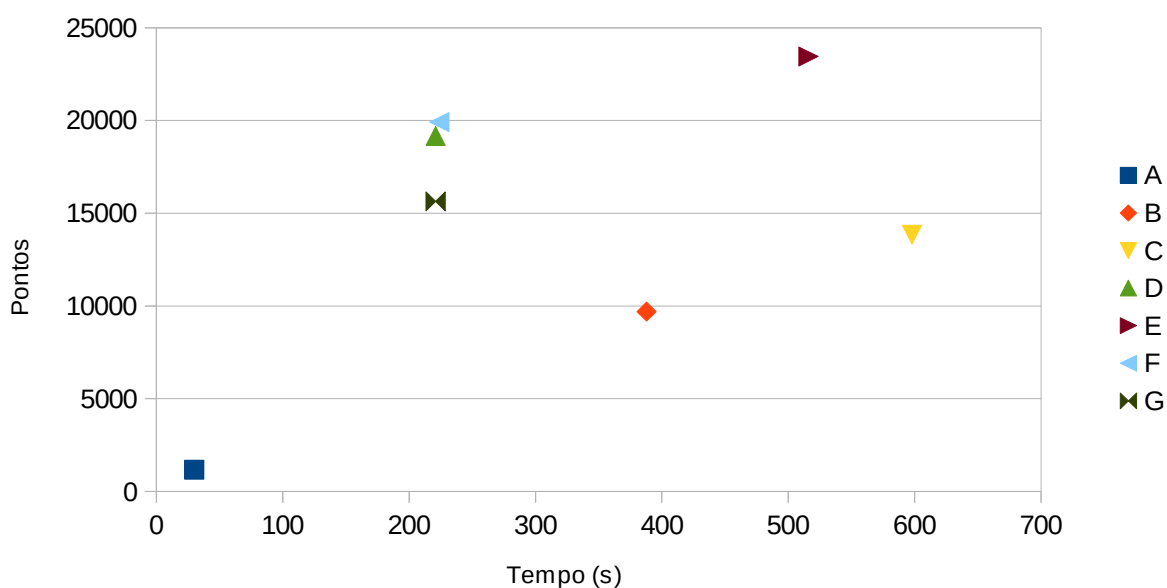
A Figura 28 apresenta o gráfico dos tempos de execução para cada protótipo. Podemos identificar que o protótipo com maior custo de tempo foi o C em relação aos demais e o segundo protótipo com custo elevado de tempo foi o E. Os protótipos D, F e G foram os que executaram em tempos relativamente aproximados. O protótipo B demonstra um gráfico com tempos elevados nas primeiras execuções e posteriormente tempos inferiores não apresentando um padrão de tempo. O protótipo A apresenta um gráfico praticamente linear, o mais rápido, ou seja, com menor tempo em relação aos demais.

Figura 28: Tempos de execução dos protótipos



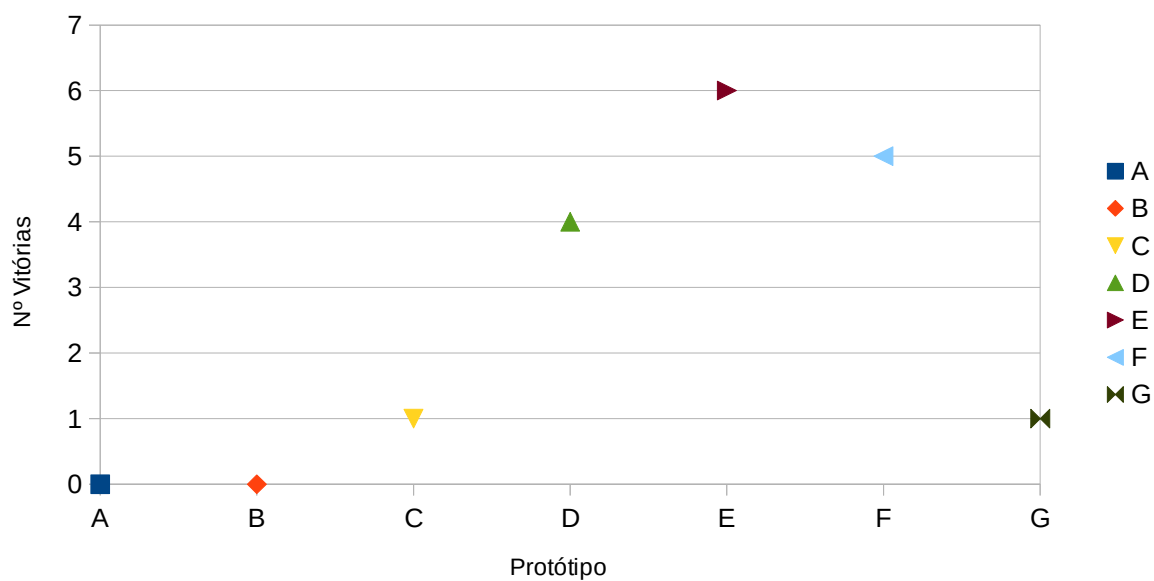
A Figura 29 apresenta o gráfico com as médias de pontos de cada protótipo em relação à média de tempo de execução de cada um. Os protótipos F e D resultaram em médias de tempo e pontos próximas, conforme pode ser confirmado analisando o gráfico da Figura 29. O protótipo E demonstrado na Figura 29 obteve a melhor média de pontos, mas o segundo pior tempo com cerca de 9 minutos para cada execução. O protótipo G apresentou o segundo menor tempo, porém com pontuações inferiores aos dos protótipos F e D, diferente do que se poderia supor adotando a estratégia de profundidade mista. Esperava-se que haveria diminuição de tempo e aumento dos pontos. Neste caso, acredita-se que ao obter avaliações diferentes devido às variantes de consultas com as profundidades 4 e 5, a função de avaliação levou o jogo ao término rápido com jogadas de pouca pontuação. O protótipo C foi o mais demorado com cerca de 10 minutos de execução. Em relação ao protótipo B, houve aumento na pontuação.

Figura 29 - Relação de médias de pontos por tempo de execução dos protótipos



A Figura 30 ilustra o número de vitórias para cada protótipo, observe que o protótipo E obteve a melhor colocação com 6 vitórias. Os protótipos A e B não resultaram em vitórias. Os protótipos C e G possibilitaram apenas uma vitória. O protótipo D com quatro vitórias e o protótipo F com 5 vitórias.

Figura 30: Vitórias obtidas



5 CONSIDERAÇÕES FINAIS

A programação de jogos depende de muitos testes durante seu desenvolvimento. Durante o estudo do algoritmo Minimax, uma das dificuldades foi a definição da função de avaliação dos estados. As características de avaliação utilizadas devem ser simples, mas se forem ponderadas de forma errada, não contribuirão nos resultados.

Uma dificuldade encontrada inicialmente foi o estudo dos métodos presentes nos algoritmos em Kivy, um recurso que pode ser considerado novo para programação de jogos e aplicativos para dispositivos móveis.

Como visto nos resultados dos protótipos, é possível efetuar a busca de soluções para jogos com algoritmos de IA, porém em certos casos a solução pode não ser satisfatória. Nas tentativas de pesquisa com profundidade 6 os travamentos apresentados pelo computador impossibilitaram a execução do algoritmo. Percebe-se que uma das principais influências na solução do problema é a definição da função de avaliação do estado. Dependendo dos critérios utilizados, quando se interrompe o Minimax em determinada profundidade, a ação escolhida, que parece ser a ideal, futuramente pode levar à derrota.

Deve-se salientar que o jogo 2048 é representado por blocos que são calculados e exibidos na tela utilizando as funções do próprio Python e da biblioteca Kivy, não sendo aplicadas imagens criadas com programas de edição. Neste caso, as imagens utilizadas são muito pequenas e praticamente não influenciam na utilização de memória. Os jogos que utilizam *Tiles*¹ (azulejo, em tradução livre) para criar a tela de um jogo são muito utilizados em jogos destinados a máquinas com processamento limitado. Dessa forma o processamento e a memória do computador não sofreram grande influência para criar a parte gráfica do jogo. Como mencionado anteriormente, não foi possível gerar os pacotes para executar o 2048 em dispositivos móveis com SO Android. Esta ideia fica como trabalho futuro, incluindo a identificação de *bugs* e problemas de *Framework* que impossibilitaram a portabilidade do jogo.

Do ponto de vista do jogador, foi possível identificar alguns travamentos de execução, quando o número de blocos vazios fica reduzido e o algoritmo de busca precisa expandir até a profundidade determinada. O Minimax pode ser aplicado a

1 Fonte: <http://www.tonypa.pri.ee/tbw/tut00.html>

jogos *multi-player* e neste caso não seria aceitável, durante uma partida de jogos de ação por exemplo, que o computador travasse visivelmente até efetuar o(s) movimento(s) do(s) agente(s). Podemos identificar que é possível aplicar IA a jogos para execução em máquinas convencionais, porém deve-se aplicar métodos e exaustivos testes para garantir a execução sem travamentos.

Os protótipos utilizados neste trabalho podem ser consultados no seguinte link: <https://github.com/diess/2048>. Apenas o arquivo “main.py” foi alterado para receber as funções de IA. As modificações de interface foram efetuadas no arquivo “game2048.kv”.

REFERÊNCIAS

BATISTA, M. L. S.; QUINTÃO, P. L.; LIMA, S. M. B.; CAMPOS, L. C. D.; BATISTA, T. J. S. **Um Estudo Sobre a História dos Jogos Eletrônicos**. 2007. Disponível em: <http://www.dcc.ufrj.br/~rlopes/trabalhos/conhecimento_inovacao.pdf>. Acesso em 05 mar. 2014.

BOURG, D.M.;SEEMANN, G. **AI for Game Developers**, 1 ed. O'Reilly Media, 2004.

CALADO, L. C.; SOUZA, J. C.; CAMPOS, J. C.; SILVA, R. S. **Algoritmo MiniMax**. 2009. Disponível em: <http://paginas.fe.up.pt/~eol/IA/IA0809/APONTAMENTOS/Alunos_MiniMax.pdf>. Acesso em 23 de jun. 2014.

CONCEIÇÃO, D. T. **Redes Neurais Artificiais Aplicadas ao Jogo da Velha 3D em Pinos**, 2009. 85 f. Trabalho de Conclusão de Curso (Graduação em Tecnólogo da Informação da Comunicação), Instituto Superior de Tecnologia em Ciência da Computação de Petrópolis, Petrópolis, 2009.

CIRULLI, G. **2048**. 2014. Disponível em: <<http://gabrielecirulli.github.io/2048/>>. Acesso em 10 mai. 2014.

CTMAS. **Understanding Tower Defense games**, 2010. Disponível em: <<http://www.loopinsight.com/2010/03/30/understanding-tower-defense-games/>>. Acesso em 25 abr. 2012.

FERGUSON, T. S. **Game Theory**. 2005. Disponível em: <http://www.math.ucla.edu/~tom/Game_Theory/mat.pdf>. Acesso em: 12 jan. 2014.

JHINGUT, M.Z.; GHOORUN, I.M.; NAGOWAH, S.D.; MOLOO, R.; NAGOWAH, L. **Design and Development of 3D mobile games**. In: Third International Conferences on Advances in Computer-Human Interactions, 3, 2010.Saint Maarten. Conference Publication. Reduit, Mauritius: Computer Science & Engineering Department, University of Mauritius, 2010. Disponível em: "<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5430114>", acesso em julho de 2012.

KALNING, K.; **The Anatomy of the First Video Game. 2008**. Disponível em: <http://www.nbcnews.com/id/27328345/ns/technology_and_science-games/t/anatomy-first-video-game/#.U96JyqPNq_Q>. Acesso em: 13 jun. 2014.

KARLSSON, B. F. F. **Um Middleware de Inteligência Artificial para Jogos Digitais**. 2005. 126 f. Dissertação de Mestrado (Pós-graduação em Informática) - Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 2005.

KEPLER, F. **2048 game with Kivy**. 2014. Disponível em: <<https://github.com/tito/2048>>. Acesso em 10 mai. 2014.

KOVÁSZNAI, G.; KUSPER, G. **Artificial Intelligence**. 2011. Disponível em: <http://www.tankonyvtar.hu/en/tartalom/tamop425/0038_informatika_MestInt-EN/index.html>. Acesso em 9 mai. 2014.

LAGO, S. **Jogos e Busca**. 2006. Disponível em: <<http://www.ime.usp.br/~slago/slago-jogos.pdf>>. Acesso em: 21 jul. 2014.

MAES, P. **Artificial Life Meets Entertainment: Lifelike Autonomous Agents**. 1995. In: Communications of the ACM, 38, 11, 108-114. 1995. Disponível em: <<http://www.nada.kth.se/kurser/kth/2D1381/ArtificialLifeMaes.pdf>>. Acesso em 12 mai. 2014.

OSÓRIO, F. **Redes Neurais Artificiais: Do Aprendizado Natual ao Aprendizado Artificial**. 1999. I Fórum de Inteligência Artificial: I Seminário de Inteligência Artificial. ULBRA. Disponível em: <<http://osorio.wait4.org/oldsite/IForumIA/fia99.pdf>>. Acesso em 12 mai. 2012.

PAIVA, J. C. **Árvore Genérica**. 2012. Disponível em: <http://dietinf.ifrn.edu.br/lib/exe/fetch.php?media=corpodocente:jailton:ed2011.2_aula0_arvoregenerica.pdf>. Acesso em 5 jun. 2014.

RUMMELL, P. A. **Adaptive AI to play tower defense game**. In: The 16 th International Conference on Computer Games, 16, 2011. Louisville, KY. Conference Publication. Victoria, British Columbia, Canada: Department of Computer Science, University of Victoria, 2011. Disponível em: "<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6000357>", acesso em julho de 2012.

RUSSELL, S.; NORVING, P. **Inteligencia Artificial uma Abordagem Moderna**, 2 ed. São Paulo: Campus, 2003.

SILVA, J. R. C. **Árvore de Jogos, Minimax e Poda Alfa-Beta**. 2013. Disponível em: <<http://jeiks.net/wp-content/uploads/2013/05/MiniMax.pdf>>. Acesso em 4 jun. 2014.

SMITH, D. C.; CYPHER, A.; SPOHRER, J. C. **KIDSIM: Programming Agents Without a Programming Language**. 1994. In: Communications of the ACM, Vol. 37, pp. 34 - 67. 1994. Disponível em: <<http://www-ui.is.s.u-tokyo.ac.jp/~takeo/course/2006/media/papers/kidsim.pdf>>. Acesso em 11 jun. 2014.

SCHWAB, B. **AI Game Engine Programming**, 1 ed. Hingham, Massachusetts: Charles River Media, 2004.

TATAI, V. K. **Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador**. 2006. 129 f. Dissertação de Mestrado em Engenharia Elétrica, Departamento de Engenharia de Computação, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2006.

VIRBEL, M. **2048 game with Kivy**. 2014. Disponível em:<<https://github.com/tito/2048>>. Acesso em 10 mai. 2014.

XIN, C. **Influence from the Serious Games on Mobile Game Developers' Commercial Strategies**. In: International Seminar on Business and Information Management - ISBIM, 1, 2008. Wuhan. Conference Publication. Guangzhou, China: School of International Trade & Economics, Guangdong University of Foreign Studies, 2008. Disponível em: "<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5117465>", acesso em julho de 2012.

YANG, B.; ZHANG Z. **Design and implementation of high performance mobile game on embedded device**. In: International Conference on Computer Application and System Modeling (ICCASM), 8, 2010. Taiyuan. Conference Publication. Hangzhou, China: College of Computer Science & Information Engineering, Zhejiang Gongshang University, 2010. Disponível em: "<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&tp=&arnumber=5619347>", acesso em julho de 2012.

ZAIBON, S. B.; SHIRATUDDIN, N. **Heuristics Evaluation Strategy for Mobile Game-Based Learning**. In: The 6th IEEE International Conference on Wireless, Mobile, and Ubiquitous Technologies in Education (WMUTE), 6, 2010. Kaohsiung. Conference Publication. Sintok, Kedah, Malaysia: College of Arts & Sciences, Universiti Utara Malaysia, 2012. Disponível em: "<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5476548>", acesso em julho de 2012.