

**FACULDADE DE TECNOLOGIA DE PRAIA GRANDE**

**I.A. EM JOGOS**

**A BUSCA COMPETITIVA ENTRE O HOMEM E A MÁQUINA**

**AUTOR: ROBERTO TENGAN DE SANTANA**

**ORIENTADORA: PROF<sup>a</sup> FERNANDA SCHMITZ LEITÃO  
ALMEIDA**

Praia Grande

2006

**I.A. EM JOGOS**  
**A BUSCA COMPETITIVA ENTRE O HOMEM E A MÁQUINA**

**ROBERTO TENGAN DE SANTANA**

Monografia apresentada à Faculdade de Tecnologia de  
Praia Grande, como parte dos requisitos para a obtenção  
do título de Tecnólogo em Informática com ênfase em  
Gestão de Negócios.

Orientador: Prof. (a) Fernanda Schmitz Leitão Almeida

Praia Grande

2006

*Do or not Do. There is no try  
Master Jedi - Yoda*

*Dedico*

*A todos que ainda não conheci.*

## AGRADECIMENTOS

A realização deste trabalho teve o apoio de muitas pessoas: professores, amigos, colegas, familiares. Cada um contribuiu do seu modo, para a realização desse trabalho e para que eu pudesse encontrar a força e o incentivo necessários para transpor os obstáculos e as dificuldades encontradas no desenvolvimento de um trabalho como este. Por isso, aqui vai meu sincero agradecimento a todos que, de alguma forma, contribuíram para a realização deste trabalho.

Algumas pessoas, porém, foram fundamentais:

Fernanda Schmitz Almeida Leitão, minha orientadora, fornecendo o estímulo inicial e os subsídios necessários para uma primeira reflexão sobre o tema. Obrigado pela sua ajuda, interesse, dedicação e contribuição.

Cybelle Croce Rocha pelas aulas e orientação que ajudaram a tornar a pesquisa realizada em algo formatado para toda a comunidade acadêmica.

Aos meus professores que me transmitiram conhecimentos valiosos, sempre me dando apoio e motivação.

Pessoal da UNIDEV que me ajudou a compreender alguns pontos importantes ao trabalho.

Aos amigos Domingos, Cibele e Vivian que sem eles eu não entregaria a parte burocrática do trabalho.

As bandas System of a Down e Rammstein, pois fizeram a trilha sonora para a confecção deste trabalho.

Finalmente, a minha família, que proporcionaram o apoio moral para o término do trabalho.

# SUMÁRIO

LISTA DE FIGURAS .....	x
LISTA DE TABELAS .....	xi
LISTA DE SIMBOLOS .....	xii
RESUMO .....	xiii
ABSTRACT .....	xiv
INTRODUÇÃO.....	15
1. HISTÓRIA DA INTELIGÊNCIA ARTIFICIAL .....	17
1.1. O nascimento .....	17
1.2. Era Clássica .....	18
1.3. Era Romântica .....	20
1.4. Era Moderna .....	21
2. INTELIGÊNCIA ARTIFICIAL.....	27
2.1. Filosofia .....	27
2.1.1. IA Fraca .....	27
2.1.1.1 Argumento da Inaptidão.....	28
2.1.1.2 Argumento da Objeção Matemática.....	29
2.1.1.3 Argumento da Informalidade .....	29
2.1.2 IA Forte.....	30
2.1.2.1 O Problema de mente/corpo.....	32
2.1.2.1.1 “Cérebro em uma cuba” .....	32
2.1.2.1.2 Prótese Cerebral .....	33
2.1.2.1.3 O quarto chinês.....	35
2.2 Abordagens .....	36
2.2.1 Abordagem Simbolista.....	37
2.2.1.1 Metodologia Simbolista .....	38
2.2.1.2 Hipóteses Simbolistas .....	40
2.2.2 Abordagem Conexionista.....	41
2.3 Ramificações.....	41
2.3.1 Sistemas Especialistas.....	42
2.3.1.1 Estrutura de um Sistema Especialista .....	42
2.3.1.2 Aquisição do conhecimento .....	43
2.3.1.3 Sistemas Especialistas Conhecidos .....	44

2.3.2	Robótica .....	45
2.3.2.1	Hardware .....	46
2.3.2.2	Robôs conhecidos .....	48
2.3.3	Sistemas Visuais .....	48
2.3.4	Processamento de Linguagem Natural .....	49
2.3.5	Sistemas de Aprendizado .....	50
2.3.6	Redes Neurais Artificiais .....	51
2.3.6.1	Neurônio Artificial .....	51
2.3.6.2	Arquitetura .....	52
2.3.6.3	Aprendizagem .....	53
2.3.6.4	Topologia .....	54
2.3.6.5	Etapas para Desenvolvimento .....	54
2.3.6.6	Aplicação .....	56
3.	HISTÓRIA DOS JOGOS ELETRÔNICOS .....	57
3.1	A Primeira Era .....	57
3.2	A Segunda Era .....	59
3.3	A Terceira Era .....	61
4.	JOGOS ELETRÔNICOS .....	65
4.1	Desenvolvimento de Jogos Eletrônicos .....	65
4.1.1	Equipe de Desenvolvimento .....	66
4.1.2	Ciclo de Desenvolvimento .....	67
4.1.3	Game Design .....	69
4.1.4	Regras básicas para um bom jogo .....	70
4.1.5	O que os jogadores querem e esperam .....	72
4.1.6	Puzzles .....	73
4.2	Arquitetura de um Jogo .....	74
4.2.1	Reusabilidade .....	74
4.2.1.1	Engines .....	74
4.2.2	Arquitetando o Jogo .....	75
4.2.2.1	Abstração de Hardware .....	75
4.2.2.2	Abstração do jogo .....	76
4.2.3	Máquinas de Estados Finitos (FSM) .....	78
4.3	Programação de um Jogo .....	79
4.3.1	Gráficos .....	79

4.3.1.1 Superfícies.....	80
4.3.1.2 Função Blit.....	82
4.3.1.3 Transparência.....	83
4.3.1.4 Espelhamento.....	84
4.3.1.5 Técnicas de Animação.....	84
4.3.1.5.1 Efeito flicker.....	85
4.3.1.5.2 Page Flipping.....	86
4.3.1.5.3 Double Buffering.....	87
4.3.1.6 Paleta.....	87
4.3.1.7 Formato de pixel (pixel format).....	88
4.3.2. Tiles, Bricks e Layers.....	88
4.3.2.1 Tiles.....	88
4.3.2.2 Bricks.....	88
4.3.2.3 Layers.....	89
4.3.3 Sprites.....	90
4.3.3.1 Animação de um Sprite.....	90
4.3.4 Dispositivos de Entrada.....	91
4.3.5 Som.....	92
4.3.5.1 Buffers de Som.....	93
4.3.5.1.1 Buffer secundário.....	93
4.3.5.1.2 Buffer primário.....	94
4.3.6 Tratamento de tempo.....	94
4.3.6.1 Gerenciador de Tempo.....	94
4.3.6.2 Taxa de Quadros.....	95
4.3.6.3 Acumuladores de Tempo.....	95
4.3.7 Física.....	95
4.3.7.1 Colisão.....	96
4.3.7.1.1 Detecção de Colisão.....	96
4.3.7.1.2 Tratamento de Colisões.....	97
4.3.7.2 Gravidade.....	98
4.3.8 Visualizações de Jogos.....	99
4.4 Gêneros de Jogos.....	99
4.5 Mercado dos Jogos Eletrônicos.....	102
4.5.1 Mercado Brasileiro.....	103

4.5.2 Indústria de desenvolvimento de jogos eletrônicos no Brasil.....	104
5. BUSCA COMPETITIVA.....	107
5.1. Histórico .....	107
5.2 Definição.....	108
5.3 IA tradicional X IA em Jogos .....	119
5.4 Motores de IA .....	110
5.4.1 Unreal Engine 3 .....	110
5.4.2 Source Engine .....	111
5.4.3 Reality Engine.....	112
5.5 IA em jogos.....	113
5.5.1 Agentes Inteligentes.....	114
5.5.1.1 Tipos de Agentes.....	115
5.5.1.1.1 Agentes Reativos.....	115
5.5.1.1.2 Agentes Cognitivos .....	116
5.5.1.1.3 Agentes Híbridos.....	117
5.5.1.2 Agentes Inteligentes em Jogos .....	117
5.5.2 Máquinas de Estados Finitos .....	118
5.5.2.1 Máquinas de Estados Finitos como IA em jogos.....	118
5.5.3 Conjuntos e lógica <i>Fuzzy</i> (Difusa).....	123
5.5.3.1 Desfuzzificação .....	125
5.5.3.2 Center of Gravity.....	125
5.5.3.3 Mean of Maximum.....	126
5.5.3.4 Máquina de Estados Finitos <i>Fuzzy</i> (FuSM).....	129
5.5.4 Algoritmos de Busca.....	130
5.5.4.1 Algoritmo MiniMax .....	131
5.5.4.1.1 Poda alfa-beta .....	134
5.5.4.2 Algoritmo A* (A-Star) pathfinding .....	137
5.5.4.2.1 Algoritmo A* (A-Star) pathfinding nos jogos .....	143
5.5.4.3 Algoritmos Genéticos .....	147
5.5.4.3.1 Codificação.....	148
5.5.4.3.1.1 Cruzamento.....	148
5.5.4.3.1.2 Mutação .....	149
5.5.4.3.2 Função Objetivo .....	149
5.5.4.3.3 Algoritmo Genético Simples .....	150

5.5.4.3.4 Algoritmos Genéticos em Jogos .....	151
5.5.5 IA Hierárquico .....	152
5.5.6 Linguagens de Script .....	153
5.5.6.1 Classificação .....	154
5.5.6.2 Scripts em Jogos .....	155
5.5.6.3 Linguagem Lua.....	156
CONCLUSÃO.....	158
ANEXOS .....	159
REFERÊNCIAS BIBLIOGRÁFICAS .....	177

## LISTA DE FIGURAS

Figura 1: Arquitetura básica de um jogo .....	77
Figura 2: Exemplo de uma Máquina de Estados Finitos .....	78
Figura 3: Exemplo de superfície offscreen.....	81
Figura 4: Exemplo da função Blit .....	82
Figura 5: Exemplo de Transparência.....	83
Figura 6: Exemplo de Mirroring.....	84
Figura 7: Exemplo do efeito flicker.....	85
Figura 8: Esquema do Page Flipping.....	86
Figura 9: Esquema do Double buffering .....	87
Figura 10: Exemplo de Tiles .....	88
Figura 11: Exemplo da utilização de bricks .....	89
Figura 12: Exemplo de Layer .....	89
Figura 13: Exemplo de Sprites .....	90
Figura 14: Animação Contínua e Finita .....	91
Figura 15: Escolha da forma geométrica.....	97
Figura 16: fsm do guarda.....	119
Figura 17: Método Center of Gravity .....	126
Figura 18: Funções do Método Mean of Maximum.....	127
Figura 19: Exemplo da Aplicação de MiniMax .....	132
Figura 20: Algoritmo MiniMax com poda alfa-beta .....	134
Figura 21: Mapa rodoviário simplificado da Romênia.....	139
Figura 22: Algoritmo A* (A-Star) em ação .....	140
Figura 23: Mapa escaneado em quadrados.....	143
Figura 24: Obtenção dos primeiros valores $f(n)$ , $g(n)$ , $h(n)$ .....	144
Figura 25: Quadrado abaixo da parede sendo ignorado .....	145
Figura 26: Caminho completo .....	146
Figura 27: IA Hierárquico .....	153

## LISTA DE TABELAS

Tabela 1: Histórico da utilização da IA em Jogos .....	108
Tabela 2: Funções de Transição de uma FSM.....	120
Tabela 3: Ação de Cada Estado.....	120
Tabela 4: Distâncias $h_{DLR}$ das cidades da Romênia .....	139

## LISTA DE SIMBOLOS

- $\wedge$  - E lógico
- $\vee$  - OU lógico
- $\neg$  - NÃO lógico

## **RESUMO**

O presente trabalho trata da utilização de algoritmos de Inteligência Artificial para o desenvolvimento de jogos, devido ao fato de esta área está em grande ascensão no mercado.

O maior desafio agora para os desenvolvedores de jogos é a criação de um jogo com um bom desafio para o jogador, já que os jogos estão num nível gráfico e sonoro próximo da realidade. Diversos algoritmos já estão sendo usados pelas produtoras de jogos para a simulação de uma inteligência humana convincente. Este trabalho demonstra exemplos de lógica para o desenvolvimento da inteligência artificial, demonstrando a aplicabilidade dos mesmos para o atual mercado de desenvolvimento de jogos.

## **ABSTRACT**

The present work deals with the use of algorithms of Artificial Intelligence for the development of games, due to the fact that area is in great ascension in the market. The biggest challenge now for the developers of games is the creation of a game with a good challenge for the player, since the games are in a graphical and sonorous level next to the reality. Several algorithms are already being used for the producers of games for the simulation of an intelligence convincing human being. This work demonstrates examples of logic for the development of artificial intelligence, demonstrating the applicability of the same ones for the current market of development of games.

## INTRODUÇÃO

É justificado que com o crescimento do mercado de jogos eletrônicos utilizando a tecnologia de Inteligência Artificial. Verificou-se a escassez de coletâneas de matéria sobre o assunto. Diante desta realidade a elaboração de uma pesquisa completa sobre o tema torna-se imprescindível.

O objetivo do trabalho é adquirir conhecimentos sobre essa nova área em crescimento no mercado.

A metodologia utilizada foi a pesquisa em livros do assunto, além de artigos publicados em revistas e sites especializados no assunto.

O Trabalho foi dividido em cinco capítulos e esses capítulos foram divididos em três partes.

A primeira parte constando os capítulos um e dois aborda a inteligência artificial de forma geral.

O primeiro capítulo trata da história da Inteligência Artificial, como ela foi criada e sua evolução até os dias de hoje.

O segundo capítulo apresenta a Inteligência Artificial como um todo seus aspectos filosóficos, suas metodologias e demonstra suas principais áreas de destaque.

A segunda parte, constando os capítulos três e quatro, apresenta uma visão geral sobre jogos eletrônicos.

O terceiro capítulo trata da história dos jogos eletrônicos começando do *Odissey* chegando até o Nintendo Wii.

O quarto capítulo fala sobre os jogos eletrônicos e seu desenvolvimento, com as técnicas utilizadas atualmente para projetos de jogos e programação de um jogo.

A terceira e última parte contendo o quinto capítulo fala da utilização da inteligência artificial para o desenvolvimento de jogos.

O quinto capítulo demonstra os algoritmos mais utilizados para a criação de uma inteligência artificial para os jogos.

## 1 História da Inteligência Artificial

A Definição de Inteligência Artificial ou IA, vem de muito tempo atrás e a cada geração ela muda um pouco, a cada algoritmo<sup>1</sup> desenvolvido e testado uma nova área é aberta no campo da IA. O objetivo principal da IA é descobrir o funcionamento do raciocínio humano e várias foram as tentativas de simulação durante o seu início até os dias atuais. Diversos filmes abordam este tema, como: *The Matrix* (WACHOWSKI, 1998), *Blade Runner* (SCOTT, 1982), 2001 – Uma odisséia no espaço (KUBRICK, 1968), A.I. - Inteligência Artificial (SPIELBERG, 2001), todos demonstram um futuro próximo em que máquinas podem pensar e também ter emoções como nós, além de poderem competir conosco no mesmo nível ou um nível acima.

*Com bastante rapidez, eles poderiam nos exterminar. Não estou tão alarmado quanto muitos outros por essa última possibilidade, pois considero essas máquinas do futuro nossa progênie, 'filhos da mente' construídos a nossa imagem e semelhança, nós mesmos em uma forma mais poderosa. Como crianças biológicas de gerações anteriores, eles encarnarão melhor a esperança da humanidade de um futuro em longo prazo. É nossa responsabilidade lhes dar toda vantagem, e nos retirarmos calmamente quando não pudermos mais colaborar. (MORAVEC, 2000 apud RUSSEL & NORVIG, 2004:931)*

Para definir o nível em que a IA está e quanto falta para chegar neste “futuro próximo” é preciso conhecer toda sua evolução histórica.

### 1.1 O nascimento

Segundo RUSSEL & NORVIG (2004: 18) o primeiro trabalho a respeito de IA foi realizado por Warren McCulloch e Walter Pitts em 1943, em que demonstrava um modelo de neurônios<sup>2</sup> artificiais, em que cada neurônio poderia estar “ligado” ou “desligado”, isso dependeria dos estados em que os neurônios vizinhos estariam.

---

<sup>1</sup> Seqüência de instruções executadas até que determinada condição seja alcançada

<sup>2</sup> célula do sistema nervoso responsável pela condução do impulso nervoso.

Donald Hebb demonstrou em 1949 uma regra de atualização simples para definir a intensidade de conexão entre neurônios, essa regra é influente ainda nos dias de hoje (RUSSEL & NORVIG, 2004: 18).

Em 1950 Alan Turing demonstrou a IA em seu artigo “*Computing Machinery and Intelligency*”. Neste artigo ele apresentou diversos algoritmos de IA e seu teste, o “Teste de Turing”. (*Idem*, 18).

Para CARVALHO (2002:45) o teste de Turing é elaborado da seguinte forma, em uma sala ficaria a máquina a ser testada, enquanto em outra sala ficaria uma pessoa, as duas entidades conversariam entre si, a máquina passaria no teste se a pessoa não percebesse que estaria conversando com uma máquina.

Em 1951, no departamento de matemática de Princeton, Marvin Minsk e Dean Edmond apresentaram o primeiro computador de rede neural, o SNARC era composto por 3000 válvulas e simulava uma rede de 40 neurônios. (*Op. Cit.*,18)

Segundo RUSSEL & NORVIG (2004: 18-19), cinco anos após o SNARC John McCarthy realiza um seminário em Dartmouth, reunindo todos os grandes pesquisadores do conhecimento. Nesse seminário Allen Newell e Herbert Simon do *Carnegie Tech*<sup>3</sup> apresentam o *Logic Theorist* o primeiro programa de raciocínio, podendo resolver todos os teoremas de matemática ou até a escrever um artigo científico. Nesse seminário John McCarthy juntamente com os outros seminaristas percebeu que a IA devia ser um campo em separado e não uma ramificação da matemática ou estar sob o nome de pesquisa operacional ou teoria de controle, então eles definiram o nome de Inteligência Artificial e após este seminário a IA é definida como o campo em que tenta construir máquinas que funcionem de forma autônoma e em ambientes mutáveis.

Este é o início da Inteligência Artificial, os pesquisadores ainda estavam tímidos com seus trabalhos, mas todos conseguiam êxito em suas demonstrações, dentro dos padrões daquela época.

## 1.2 Era Clássica

Os primeiros anos da nova ciência foram repletos de sucessos, considerando ainda as configurações dos computadores serem bem baixas, em geral, a classe intelectual acreditava que um computador nunca conseguiria realizar uma determinada ação. Os pesquisadores de

---

<sup>3</sup> Agora conhecida como Carnegie Mellon University (CMU)

IA demonstrava uma ação após a outra, por exemplo os críticos da IA dizia que uma máquina não pode jogar xadrez os pesquisadores de IA mostrava que as máquinas podem jogar xadrez. (RUSSEL & NORVIG, 2004:19)

Newell e Simon continuaram a terem sucesso desenvolvendo o GPS *General Problem Solver* (Solucionador de problemas gerais). Diferente do *Logic Theorist* o GPS foi projetado para resoluções de problemas de uma forma humana, observou que a forma que ele encarava os problemas era realmente semelhante à de um humano, com isso ele pode ser considerado o primeiro programa a “pensar de forma humana”, com este sucesso Newell e Simon em 1976 formularam a hipótese de sistemas de símbolos físicos em que afirma “um sistema de símbolos físicos tem os meios necessários e suficientes para uma ação inteligente geral”, ou seja, qualquer sistema seja humano ou máquina que exiba algum nível de inteligência deve operar manipulando dados com símbolos em suas estruturas. (*Idem*, 20)

Na IBM o pesquisador Rochester e seus colegas desenvolveram vários softwares de IA, entre eles o *Geometry Theorem Prover*, podendo demonstrar a resolução de vários Teoremas matemáticos. Arthur Samuel desenvolveu vários jogos de damas com a capacidade de aprender, nessa época ele contestou que uma máquina só pode executar as tarefas que lhe foram programados, seu programa conseguiu atingir um nível de jogador amador melhor que o de seu criador. (*Ibidem*, 20)

Em 1958, McCarthy foi para o MIT, lá ele desenvolveu a linguagem de programação *Lisp*<sup>4</sup>, e escreveu o artigo *Programs with common sense*, descrevendo o programa *Advice Tacker*, como o *Logic Theorist* e o *Geometry Theorem Prover*, ele foi projetado para usar o conhecimento para buscar soluções para os problemas encontrados, porém com a capacidade de adquirir novas competências em diversas áreas sem a necessidade de reprogramação. (RUSSEL & NORVIG, 2004:20)

Minsky foi para o MIT e não colaborou muito com McCarthy, pois McCarthy dava valor a representação e raciocínio, enquanto Minsky se interessava em fazer os programas funcionarem, em 1963, McCarthy funda o laboratório de IA de Stanford, para desenvolver seu *Advice Tacker*, McCarthy pensava em usar a lógica, porém A. J. Robinson desenvolve o método de resolução. O trabalho em Stanford se apoiou no raciocínio, tendo vários sistemas desenvolvidos, tendo como destaque o projeto *Shakey* de robótica em que foi o primeiro a demonstrar a integração de raciocínio lógico e atividade física. (*Idem*, 20-21)

---

<sup>4</sup> List Processing (Processamento de lista)

Minsky orientou diversos alunos que escolhiam problemas limitados cuja solução poderia exigir inteligência. Esses problemas limitados se tornaram conhecidos como micromundos, o mais famoso foi o mundo dos blocos em que diversos blocos são colocados sobre uma mesa e é passada uma tarefa ao computador, geralmente a tarefa é organizar de uma forma lógica todos os blocos, utilizando uma mão robótica podendo erguer somente um bloco por vez. Este mundo foi a base para diversos pesquisadores em seus projetos como David Huffman (projeto de visão, 1971), David Waltz (projeto de restrições, 1975), Patrick Winston (teoria do aprendizado, 1970), Terry Winograd (processamento de linguagem natural, 1972) e Scott Fahlman (agenda eletrônica, 1974). (RUSSEL & NORVIG, 21-22)

O trabalho pioneiro com redes neurais de McCulloch e Pitts prosperou nessa época como o trabalho de Winograd e Cowan que demonstrava que com um grande número de elementos podia representar um conceito individual. Os métodos de Hebb foram melhorados por Bernie Widrow, denominando suas redes *adelines*<sup>5</sup>, e por Frank Rosenblatt com os *perceptrons*<sup>6</sup>. Rosenblatt provou o teorema de convergência de *perceptron*, demonstrando que seu algoritmo poderia ajustar os valores de cada peso para a entrada de qualquer dado em seu sistema. (*Idem*, 22)

Como uma área distinta, a IA começou a crescer e mais pesquisadores colaboraram com suas teorias e projetos nas diversas ramificações que ela tinha abrangência, diversas dessas teorias ainda são usadas nos dias de hoje.

### 1.3 Era Romântica

Segundo RUSSEL & NORVIG (2004, 22), após essa primeira fase da IA, vários pesquisadores estavam entusiasmados com a evolução tecnológica proporcionada pela IA, nos anos 60 Herbert Simon fez a previsão de que dentro de dez anos uma máquina venceria um campeão de xadrez, errou por 40 anos. Este excesso de confiança demonstrado por Simon é devido ao grande sucesso que os sistemas de IA faziam, resolvendo os problemas que lhe eram passados. Porém isso era porque os problemas eram de complexidade menor se lhe forem passados problemas de uma complexidade bem superior eles falhariam.

Isso era devido a duas dificuldades. Inicialmente, os programas de IA manipulavam as informações sintaticamente sem conhecimento de seu significado ou contexto em que estava inserido, outra dificuldade foi que os programas resolviam os diversos pequenos problemas

---

<sup>5</sup> Adaptive Linear Neuron ou Adaptive Linear Element

<sup>6</sup> Rede Neural Artificial criada em 1957

testando diferentes combinações até encontrar a solução. Como os problemas eram pequenos eles conseguiam resolver rapidamente, para esses dois problemas era necessária uma configuração de hardware bem superior nas máquinas existentes na época. (RUSSEL & NORVIG, 22-23).

Uma terceira dificuldade para os sistemas de IA foi a de representar seu conhecimento, o livro *Perceptrons* (MINSKY & PAPERT, 1969), provou que os *perceptrons* não conseguiriam representar tudo que aprendiam isso fez as pesquisas de redes neurais acabarem na época.

A ilusão de poder computacional ilimitado também afetou a área de evolução de máquina, não demonstrando nenhum avanço naquela época (*Idem*, 23).

Segundo RUSSEL & NORVIG (2004: 23) a incapacidade da IA conviver com a explosão combinatória foi a principal crítica que a IA recebeu do relatório de *Lighthill* (1973), com base nesse relatório vários governos pararam a investir em projetos envolvendo a IA.

Entre os anos 70 e 80 a IA reduz seu crescimento devido a grande limitação de hardware, isto influenciou todas as áreas da IA, isto fez com que parassem os investimentos em pesquisas de IA realizados pelos governos.

## 1.4 Era Moderna

As técnicas para resolução de problemas apresentadas inicialmente, eram de busca geral, em que procurava reunir passos elementares para solucionar o problema, esses métodos eram considerados de métodos fracos, pois cada vez mais complexo o problema mais difícil era encontrar a solução, para isso precisava ter um conhecimento do próprio problema antes de começar a solucioná-lo, para resolver um problema grande era necessário saber a solução antes. (*Op. Cit*, 2004: 23)

Para RUSSEL & NORVIG (2004:23) o programa DENDRAL (Buchanan *et. Al.*, 1969), foi um exemplo inicial dessa abordagem. Desenvolvido em Stanford com a finalidade de descobrir a estrutura molecular a partir de informações coletadas por um espectrômetro<sup>7</sup> de massa, com base do conhecimento de químicos de Stanford o programa gerava todas as possíveis estruturas moleculares e depois comparava as estruturas formadas com a estrutura dada no início.

---

<sup>7</sup> Instrumento óptico utilizado para medir as propriedades da luz em uma determinada faixa do espectro eletromagnético

O DENDRAL era eficiente, pois tinha um conhecimento técnico de uma área específica em formas de regra, para depois aplicar o raciocínio para a resolução do problema, ou seja, os desenvolvedores do sistema separaram o conhecimento técnico da base de raciocínio, sendo esta uma abordagem gerada por McCarthy em seu *Advice Taker*. (RUSSEL & NORVIG. 2004:24)

Com o sucesso do DENDRAL seus desenvolvedores iniciaram o projeto HPP (*Heuristic Programming Project*) com o intuito de enxergar até que ponto os sistemas especialistas poderiam ser aplicados em outras áreas, com isso eles criaram o projeto denominado MYCIN em que realizava diagnósticos para infecção sanguínea. A abordagem dele de resolução de problemas era baseado em cerca de 450 regras, e diferente do DENDRAL ele não se baseava de um modelo geral para trazer a solução e sim com base no conhecimento adquirido de entrevistas com os especialistas do ramo, além de experiência a cada consulta realizada, outra diferença era que ele utilizava de um algoritmo de incerteza associada ao conhecimento médico, simulando assim a forma como os médicos da época faziam seus diagnósticos. O MYCIN gerou diagnósticos até melhores que os próprios especialistas. (*Idem*, 24)

Os sistemas especialistas começam a ganhar espaço, tendo agora ajuda dos especialistas humanos para preencherem as bases de dados com os seus conhecimentos, alguns sistemas conseguiam ser melhores que diversos especialistas do ramo.

O conhecimento específico de uma determinada área também foi utilizado no campo de processamento de linguagem natural, o sistema SHRDLU (Winograd) de reconhecimento de linguagem natural, despertou muito interesse, apesar disso ele ainda possuía as mesmas falhas que seus antecessores, além de conseguir reconhecer ambigüidades e referências pronominal, mas isso era por causa dele ser desenvolvido para uma única área. Diversos pesquisadores sugeriram que para se ter um sistema para compreensão de linguagem era necessário conhecimento geral sobre o ambiente e um método genérico para a utilização destes conhecimentos. (*Op Cit*, 24-25)

Para RUSSEL & NORVIG (2004:25) em Yale o lingüista e pesquisador de IA Roger Schank afirmou o seguinte: “Não existe essa coisa de sintaxe.”. Isso irritou vários lingüistas e criou uma grande discussão nos anos seguintes, Schank e seus alunos desenvolveram diversos sistemas de reconhecimento de linguagem, com grande ênfase nos problemas de representação do raciocínio para a compreensão da linguagem, esses problemas incluíam

representações de situações estereotípicas<sup>8</sup>, descrição da organização da memória humana e compreensão de planos e metas.

O grande crescimento dos sistemas de resolução de problemas reais causou um aumento na demanda de esquemas de representação do conhecimento. Foram desenvolvidas diversas linguagens para representação de raciocínio, como por exemplo, a linguagem *Prolog*<sup>9</sup> muito popular na Europa, e a família PLANNER nos Estados Unidos. Outras linguagens seguiram o padrão de *frames* de Minsky, adotando uma abordagem estruturada, reunindo fatos e organizando-os em uma hierarquia taxonômica<sup>10</sup>. (RUSSEL & NORVIG, 2004:25)

Para suprir a demanda, houve um grande investimento na área de sistemas especialistas, onde foram desenvolvidas novas linguagens para a demonstração e processamento do conhecimento adquirido nos sistemas.

Segundo RUSSEL & NORVIG (2004:25) o primeiro sistema especialista bem-sucedido foi o R1, utilizado pela *DEC (Digital Equipment Corporation)* em 1982. Este sistema ajudou a empresa economizar cerca de 40 milhões de dólares por ano, em 1986 a *DEC* tinha 40 sistemas especialistas em funcionamento e outros a serem produzidos, a *Du Pont* tinha 100 sistemas especialistas em uso e 400 em desenvolvimento economizando cerca de 10 milhões de dólares, maioria das grandes empresas americanas começaram a investir em IA.

Em 1981, os japoneses anunciam o “*Fifth Generation*”, um plano para que em 10 anos fossem construídos computadores inteligentes, os Estados Unidos responderam criando o *MCC (Microelectronics and Computer Technology Corporation)* sendo um consórcio para pesquisas assegurando a competitividade. Porém os dois projetos não alcançaram as metas estabelecidas. Na Inglaterra, o relatório *Alvey* retira as críticas geradas pelo *Lighthill*. (*Idem*, 25)

De algum modo, a IA expandiu de milhões de dólares em 1980 para bilhões de dólares em 1988, após essa explosão as empresas sofreram à medida que deixaram de atingir suas metas extravagantes. (RUSSEL & NORVIG, 2004:26)

Para RUSSEL & NORVIG (2004:26) o campo de redes neurais estava abandonado pela área da computação, porém outras áreas investiram nas redes neurais, como a Física em que foi desenvolvido um estudo para técnicas de mecânica estatística e de otimização das redes, os psicólogos deram continuidade ao estudo de modelos de memória de redes neurais,

---

<sup>8</sup> Imagem preconcebida para a situação

<sup>9</sup> Acrônimo do francês *Programmation en Logique* (Programação em Lógica)

<sup>10</sup> Forma de classificar os seres vivos

mas a grande evolução foi na metade dos anos 80, quando quatro grupos de pesquisadores recriaram o algoritmo de aprendizado por retropropagação desenvolvido inicialmente por Bryson e Ho em 1962, o algoritmo foi utilizado em vários problemas e seus resultados apresentados na coletânea *Parallel Distributed Processing* (Rumhart & McClelland, 1986) causou grande excitação.

Os modelos conexionistas eram vistos por todos como concorrentes dos modelos simbólicos idealizados por Simon e Newell e da abordagem logicista desenvolvido por McCarthy. O livro *The Symbolic Species* (1997) afirma que os seres humanos podem ser manipuladores de símbolos, mas os conexionistas mais fervorosos ainda questionam a abordagem simbolista. Pela visão atual as abordagens conexionista e simbólica são complementares e não concorrentes. (RUSSEL & NORVIG, 2004:26)

O campo da IA encantou os empresários, fazendo empresas economizarem milhões, os investimentos retornaram e a IA voltou a ter um crescimento significativo, áreas como redes neurais que estava abandonada voltam a crescer.

Para RUSSEL & NORVIG (2004:26) nesses últimos anos a IA teve um grande crescimento tanto no conteúdo e na metodologia, pois agora é bem comum usar teorias já existentes como base, em vez de criar novas teorias, fundamentar as afirmações em teoremas rigorosos e em evidências experimentais rígidas, em vez de utilizar como base a intuição e destacar a relevância de aplicações em vez de exemplos de brinquedos.

A IA adota o método científico, para a aceitação das hipóteses, cada hipótese agora devia passar por testes empíricos<sup>11</sup> e seus resultados avaliados de acordo com a sua importância (COHEN, 1995 *apud* RUSSEL & NORVIG, 2004:27), repositórios na Internet e compartilhamento de códigos, ajudam a realização de experimentos.

A área de reconhecimento de linguagem natural ilustra essa mudança em toda a IA, inicialmente diversos algoritmos eram utilizados até o aparecimento dos Modelos ocultos de Markov (MOMs) que dominaram a área, por dois motivos primeiro o modelo é baseado totalmente em expressões matemáticas e é utilizado uma grande base de dados reais em fala, a área do reconhecimento de linguagem natural está fazendo a transição para aplicações industriais e de consumo a larga escala.

As Redes Neurais também seguem essa tendência, com grande parte do trabalho realizado nos anos 80 ajudou a descobrir o nível de abrangência e a capacidade da área, agora ela é comparada com técnicas de estatísticas, reconhecimento de padrões e aprendizado de

---

<sup>11</sup> Conhecimento é o resultado de nossas experiências

máquina, o resultado do desenvolvimento das redes neurais cria a indústria de DataMining. (RUSSEL & NORVIG, 2004:27)

Jude Pearl com sua obra, *Probabilistic Reasoning in Intelligent Systems* (1988), leva a uma diferente aceitação da probabilidade e da teoria da decisão na IA, criando as redes bayesianas, criadas para permitir a representação de conhecimento incerto e raciocínio rigoroso, esse tipo de abordagem supera todos os problemas encontrados nas décadas de 60 e 70 na área de sistemas probabilísticos, e agora domina as pesquisas de raciocínio incerto e sistemas especialistas. A idéia permite o aprendizado a partir da experiência e combina o melhor da IA clássica e das redes neurais. Isso promoveu as idéias de sistemas normativos (sistemas que agem racionalmente de acordo com as leis da teoria da decisão e não procuram imitar especialistas humanos), o sistema operacional Windows possui diversos desses sistemas para diagnósticos de problemas. (*Idem*, 27)

Nos campos da robótica, da visão computacional e da representação de conhecimento também ocorreram revoluções semelhantes as que ocorreram em toda a ciência da IA, compreensão melhor dos problemas e de suas propriedades de complexidade aliada a sofisticação matemática, resultou em métodos robustos. Essas revoluções separaram as áreas do foco principal da IA e estão cada vez mais isoladas. (*Ibidem*, 2004:27)

Agora a IA está mais madura e não possui mais aquele ar de pioneirismo do início, o pesquisador de IA deve provar suas teorias com diversos experimentos empíricos e a demonstração dos resultados, a rede da Internet fez com que os pesquisadores compartilhassem as evoluções de suas pesquisas.

Encorajados pela resolução dos subproblemas da IA, os pesquisadores começaram a examinar o problema do “agente como um todo”. O trabalho de Allen Newell, Jhon Laird e Paul Rosenbloom no SOAR, é o exemplo mais conhecido que represente uma arquitetura completa de um agente inteligente. O movimento estabelecido tem como objetivo entender o funcionamento interno de agentes incorporados a ambientes reais com entradas sensoriais contínuas. Um dos ambientes mais importantes para os agentes inteligentes é a Internet, os sistemas de IA se tornaram tão comuns na *web* que o sufixo “-bot” já está no cotidiano dos usuários da Internet, além disso, diversas ferramentas utilizam a Inteligência Artificial na Internet como mecanismo de pesquisa, sistemas de recomendação e sistemas de construção de *web sites*. (RUSSEL & NORVIG, 2004:27-28)

Na tentativa de construir agentes percebeu-se que os subcampos que anteriormente isolados da IA, precisam de uma reorganização para o melhor aproveitamento de seus resultados, agora sistemas sensoriais não podem oferecer informações confiáveis sobre o

ambiente, e os sistemas de raciocínio e planejamento devem trabalhar com a incerteza. Outra consequência foi que a IA se aproximou de outras áreas, como a teoria de controle e a economia, áreas que também lidam com agentes. (RUSSEL & NORVIG, 2004:28)

No século XXI há um grande interesse em Agentes Inteligentes uma nova ramificação que pretende reunir todas as subáreas da IA, pois se percebeu que todas juntas poderiam criar um Agente perfeito realizando o principal objetivo da IA, entender o pensamento humano.

## 2 Inteligência Artificial

Como visto anteriormente, a IA tem uma grande história e abrange diversas áreas, para um entendimento melhor do funcionamento da IA, é preciso ter conhecimento de seu objetivo, contexto e suas principais ramificações como Sistemas Especialistas, Robótica, Sistemas Visuais, Processamento de Linguagem Natural, Sistemas de Aprendizado e Redes Neurais Artificiais.

### 2.1 Filosofia

Para RUSSEL & NORVIG (2004: 915) os filósofos já muito antes dos computadores procuravam a resposta para o funcionamento da mente humana, o mesmo objetivo da IA.

Os filósofos definiram que se as máquinas talvez pudessem simular uma inteligência são consideradas detentoras de uma IA Fraca, ou se as máquinas talvez pudessem realmente pensar seriam consideradas detentoras de IA Forte. (*Idem*, 915)

Grande parte dos pesquisadores de IA desenvolvem para atender o modelo de IA Fraca esquecendo a IA Forte. (*Ibidem*, 915)

A seguir será definido um pouco mais a fundo os dois modelos.

#### 2.1.1 IA Fraca

Segundo RUSSEL & NORVIG (2004:915-916), vários filósofos tentaram provar que máquinas não têm possibilidades de agir de forma inteligente. Porém o fato de provar que a IA é impossível depende de como ela é definida. Em termos gerais a IA é a busca pelo melhor agente inteligente em uma dada arquitetura. Com essa definição a IA é possível: para qualquer arquitetura digital de  $k$  bits<sup>12</sup> de espaço, existem  $2^k$  programas de agentes, e o que precisa ser feito é testar todos eles para descobrir o melhor programa que se encaixe na arquitetura.

---

<sup>12</sup> Simplificação de dígito binário “*Binary digiT*” em inglês

Essa definição funciona para problemas de engenharia, Porém os filósofos querem comparar as duas arquiteturas a humana e a máquina, formulando a questão “Máquinas podem pensar?”. (RUSSEL & NORVIG, 2004:916)

Alan Turing, em 1950, sugeriu que ao invés de perguntar se as máquinas podem pensar, devemos perguntar se elas podem passar por um teste de inteligência comportamental, sendo chamado de Teste de Turing. O teste consiste em fazer uma “máquina pensante” desenvolver uma conversa via on-line com um interrogador por 5 minutos. O interrogador deve adivinhar se teve uma conversa com uma máquina ou com um humano, o programa passará no teste se enganar o interrogador durante 30% de seu tempo. Turing previu que em 2000 uma máquina devidamente programada passaria no teste, até os dias de hoje diversos programas conseguiram enganar pessoas, como o programa ELIZA e o *chatbot*<sup>13</sup> da Internet MGONZ. Porém, até agora nenhum programa conseguiu os 30% contra julgadores treinados e a IA dedicou-se muito pouco para o desenvolvimento de máquinas para passarem no Teste de Turing. (*Idem*, 916)

Turing examinou três maneiras de negar a possibilidade que máquinas poderiam ser inteligentes.

### 2.1.1.1 Argumento da Inaptidão

Conforme RUSSEL & NORVIG (2004:917) Turing definiu ações que máquinas não conseguiriam executar como exemplo dessas ações Turing lista algumas:

- Ser amável;
- Senso de humor;
- Apaixonar-se;
- Aprender a partir de experiências;
- Distinguir o certo e o errado;

E mais outras, no século XXI nós sabemos que máquinas podem fazer ações que antes só humanos poderiam executar como jogar xadrez e outros jogos, reconhecimento de peças em linha de montagem, dirigir automóveis, fazer diagnósticos médicos, além de outras tarefas que são executadas tão bem ou até melhores do que os humanos.

Computadores podem realizar tarefas humanas até as que exigem perspicácia e compreensão humana, porém isso não prova que máquinas possuem esses dois atributos

---

<sup>13</sup> Pequeno programa capaz de manter uma conversa com uma pessoa

humanos, mas quebra a suposição sobre os processos mentais exigidos para formar um comportamento inteligente, há tarefas em que computadores não executam tão bem, como uma conversação ilimitada com um ser humano. (RUSSEL & NORVIG, 2004:917)

### **2.1.1.2 Argumento da objeção matemática**

Segundo RUSSEL & NORVIG (2004: 917) certas questões matemáticas são insolúveis para sistemas formais específicos, o teorema da incompleteza de Gödel é o exemplo mais famoso, no geral, qualquer sistema denominado F que consiga efetuar operações aritméticas é possível desenvolver a “sentença de Gödel”  $G(F)$  com as propriedades:

- $G(F)$  é uma sentença de F, mas não pode ser provada dentro de F.
- Se F é consciente, então  $G(F)$  é verdadeira.

Esse teorema demonstra que máquinas são inferiores aos humanos mentalmente, pois máquinas são sistemas formais limitados pelo teorema da incompleteza, enquanto um humano não possui essa limitação. (LUCAS, 1961 *apud* RUSSEL & NORVIG, 2004:918) essas afirmações geraram discussões em diversos pontos.

Um dos pontos é que um agente inteligente não deve ficar envergonhado se não conseguir afirmar a verdade em uma sentença em que outros agentes conseguem fazer. Um agente não pode afirmar que uma sentença é válida se essa mesma sentença se contradiz, pois se ele afirmar ele estaria contradizendo ele mesmo, com isso foi encontrado uma sentença em que esse determinado agente não possa afirmá-lo, mas outros agentes podem. Outro exemplo é que os humanos demorariam a vida inteira para calcular 10 bilhões de números de 10 dígitos, enquanto um computador demoraria segundos, isso não é uma limitação da habilidade matemática humana, pois os seres humanos se comportaram de forma inteligente durante milhares de anos antes da criação da matemática, isso prova que a matemática só possui um papel não muito vital para definição de um ser inteligente. (RUSSEL & NORVIG, 2004:918)

### **2.1.1.2 Argumento da informalidade**

Para RUSSEL & NORVIG (2004, 919), Turing definiu que o comportamento humano é muito complexo para ser capturado e representado por sistemas de regras simples e como os

computadores seguem um conjunto de regras básicas, eles não conseguiriam ter um comportamento igualmente ao dos seres humanos.

O filósofo Hubert Dreyfus apoiava essa visão. Ele e seu irmão criticavam a seguinte posição, que todo comportamento inteligente poderia ser absorvido por um sistema que raciocine a partir de regras básicas, isso era denominado de GOFAI (“*Good Old-Fashioned AI*”). Dreyfus criticava o modo como o computador era programado para desenvolver um comportamento humano. (RUSSEL & NORVIG, 919)

Dreyfus dava como exemplo o jogo de xadrez em que um mestre de xadrez não precisaria pensar muito para desenvolver uma jogada, essa jogada sairia pelo motivo de o mestre de xadrez ter total domínio das regras do xadrez, que foi aprendido durante toda a sua carreira como enxadrista. (*Idem*, 919)

Dreyfus propõe um sistema de aquisição de experiências, uma rede neural com uma vasta biblioteca de regras. Agora essas idéias estão sendo incorporadas a projetos de agentes inteligentes, demonstrando que a IA está progredindo e não prova sua impossibilidade de acontecer. (*Ibidem*, 919)

Os filósofos comparavam homens com as máquinas, perguntando se máquinas pensam, enquanto Alan Turing levanta a questão se as máquinas podem passar em um teste comportamental, com esse teste Turing levanta três argumentos para a negação de inteligência, ou seja, se a máquina não atingir esses três argumentos ela não será considerada inteligente.

## 2.1.2 IA Forte

Alguns filósofos afirmam que mesmo que uma máquina passasse no Teste de Turing ela ainda não estaria realmente pensando, mas sim simulando um pensamento.

*Somente quando uma máquina conseguir escrever um soneto ou compor um concerto em consequência de ter pensado e sentido emoções, e não pela disposição aleatória de símbolos, poderíamos concordar que a máquina irá se equiparar ao cérebro isto é, se ela não apenas escrever, mas souber o que escreveu (GEOFFREY JEFFERSON, 1949 apud RUSSEL & NORVIG, 2004:920)*

Turing chama esse argumento de consciência, ou seja, a máquina deve estar ciente de seus atos e de seus estados mentais, apesar da consciência ser um assunto importante Jefferson se relaciona na verdade à fenomenologia, que é o estudo da experiência direta, a

máquina precisa realmente sentir emoções, outros se concentram na intencionalidade, as máquinas crêem em algo existente no mundo real. (RUSSEL & NORVIG, 2004:920)

Turing responde os filósofos com a seguinte idéia de que não devemos utilizar um padrão de inteligência para máquinas mais alto que nós seres humanos, pois nós não sabemos os estados mentais de outro ser humano. (*Idem*, 921)

Turing admite que aplicar a consciência em máquinas é difícil mas isso não significa que a IA seja impossível ele se importa em fazer programas que simulem o pensamento humano e não em saber se pessoas consideram esses programas como seres pensantes, mas alguns filósofos estão interessados em considerar se um programa pensa ou não. (*Ibidem*, 921)

Segundo RUSSEL & NORVIG (2004:921) há áreas em que elementos produzidos artificialmente são considerados iguais a elementos naturais, e outras em que os elementos artificiais não são considerados naturais.

Tudo depende da forma como é feita análise, em alguns casos o comportamento do artefato em si influencia se ele é artificial ou natural já em outros a forma de ser produzida. Como por exemplo, um adoçante artificial é considerado um adoçante, pois ele tem o mesmo comportamento de um adoçante normal, já uma pintura de Picasso artificial não é considerada real, pois não foi Picasso que produziu, porém no caso de mentes artificiais, não há uma regra de como avaliarmos se uma mente artificial seja considerada natural, então é preciso se basear em intuições. O filósofo John Searle (1980) afirma que, ninguém supõe que uma simulação de computador de uma tempestade irá molhar as pessoas, então como simulações de processos mentais, podem ser considerados reais. (*Idem*, 921)

Em Hollywood uma simulação de uma tempestade, com certeza molha os atores. Grande parte das pessoas confirma que a simulação de um jogo de xadrez realizada por um computador é considerada um jogo de xadrez real. Os processos mentais podem ser semelhantes a um jogo de xadrez ou a uma simulação de tempestade. A definição da semelhança depende da teoria utilizada por nós. (*Ibidem*, 922)

Para RUSSEL & NORVIG (2004:922) a teoria do funcionalismo representa que o estado mental é uma condição causal intermediária entre entrada e saída. Sob isso dois sistemas que possuem a mesma condição teriam os mesmos estados mentais. Um computador poderia ter os mesmos estados mentais de uma pessoa.

Contra essa teoria, há a teoria do naturalismo biológico, que afirma que os estados mentais são resultados de manifestações neurológicas de baixo nível realizadas pelos neurônios. Por isso os estados mentais produzidos por um programa com a mesma estrutura

de outro programa serão diferentes, para serem iguais eles devem estar também em um mesmo ambiente. (RUSSEL & NORVIG, 2004:922)

Para a avaliação dos dois pontos de vista é examinado um dos mais antigos problemas da filosofia mental o problema de mente e corpo.

### **2.1.2.1 O Problema de mente/corpo**

Conforme RUSSEL & NORVIG (2004:922) o problema de mente e corpo é como estão relacionados os processos mentais a processos físicos. René Descartes considerou que a alma é um ser imortal que interage com um corpo físico concluindo que alma e corpo são coisas distintas teoria dualista. Contra essa idéia há a teoria de que não existe alma imaterial, só objetos físicos, os estados mentais são considerados estados do cérebro, teoria materialista.

Grande problema para o materialismo seria o conceito de livre-arbítrio, pois um objeto físico tem suas ações regidas pelas leis da física, não tendo mais nenhuma liberdade de escolha. Outro problema é a questão de consciência, pois é possível experimentar certos estados mentais e não experimentar certos estados físicos. (*Idem*, 922)

Para resolver contornar esses problemas é preciso examinar os estados cerebrais em níveis mais abstratos do que apresentados até agora. Por exemplo, quando um ser humano pensa em algo seu cérebro sofre certas mudanças, porém nenhuma é qualitativa para o estado mental. Para resolver isso é preciso definir os tipos dos estados cerebrais para podermos julgar se dois estados são diferentes. (*Ibidem*, 923)

Considerando o tipo de estado mental: os estados intencionais, são estados relacionados a algum aspecto do mundo exterior como acreditar, conhecer, desejar e assim por diante, ou seja, estados intencionais devem possuir uma ligação com objetos do mundo exterior, porém a identificação de estados mentais deve ser gerada “dentro da cabeça”, para examinar esse dilema será apresentado um experimento que tenta separar estados intencionais e seus objetos externos. (RUSSEL & NORVIG, 2004:924)

#### **2.1.2.1.1 “Cérebro em uma cuba”**

O experimento do “Cérebro em uma cuba” demonstra o seguinte cenário: seu cérebro é retirado após o seu nascimento e colocado em uma cuba especialmente projetada para o seu desenvolvimento, sinais eletrônico gerados por uma simulação de computador são enviados

ao seu cérebro, e seus sinais de resposta são capturados para modificar a simulação da maneira apropriada. Nesse caso, o estado mental MorrendoDeVontade(Eu,Hambúrguer) seria o mesmo estado mental de um cérebro em um corpo. (RUSSEL & NORVIG, 2004:923)

O conteúdo de um estado mental pode ser interpretado de dois pontos de vista diferentes “conteúdo largo” e “conteúdo estreito”. A visão de “conteúdo largo” interpreta em que há um observador externo onisciente com acesso a toda a situação, capaz de distinguir diferenças nos estados mentais, sob essa visão os estados mentais gerados dentro de uma cuba são diferentes dos estados gerados por uma pessoa normal. A visão de “conteúdo estreito” considera o ponto de vista subjetivo interno, sob essa visão todos os estados mentais são idênticos. (*Idem*, 923)

A crença que um hambúrguer é delicioso tem uma natureza intrínseca, do mesmo a ter essa crença. Esse é o domínio da qualia<sup>14</sup>, por exemplo, existem duas pessoas daltônicas uma vê verde na cor vermelha e a outra vice-versa, quando ambas virem o mesmo semáforo, elas agirão do mesmo modo, mas a experiência será diferente em algum aspecto. As duas pessoas podem concordar que tiveram a mesma experiência. Isso não define se as duas pessoas estariam no mesmo estado mental ou em estados diferentes. (*Ibidem*, 924)

Para se ter uma outra idéia será apresentada outra experiência que levará a pensarmos se só humanos podem ter estados mentais.

### **2.1.2.1.2 Prótese Cerebral**

Conforme RUSSEL & NORVIG (2004, 924), o experimento de Prótese Cerebral foi introduzido na década de 70 por Clark Glymour e foi estudado por Jhon Searle. O experimento demonstra a seguinte situação: A neurofisiologia se desenvolveu até um ponto em que o cérebro humano está perfeitamente compreendido. E ainda possamos construir dispositivos que reproduzem um neurônio perfeitamente, por fim é desenvolvida uma técnica cirúrgica em que é possível a substituição de neurônios individuais pelos neurônios artificiais, sem interromper a atividade cerebral. O experimento tem por objetivo substituir todos os neurônios naturais pelos neurônios artificiais e depois inverter o processo devolvendo os neurônios naturais à pessoa e seu estado biológico normal.

O experimento visa analisar o comportamento externo e a experiência interna da pessoa durante e após a operação, segundo o experimento o comportamento externo deve

---

<sup>14</sup> Qualidades ou sentimentos, como a vermelhidão, quando consideradas independentemente de seus efeitos no comportamento

permanecer inalterado, embora a presença da consciência durante a operação não seja assegurada, deverá ter a possibilidade da pessoa registrar as mudanças de sua experiência consciente. Isso gera um choque de intuições sobre o que aconteceria, Moravec pesquisador de robótica afirma que a consciência permaneceria intacta, já o filósofo Searle defende que perderíamos pouco a pouco a nossa consciência. (RUSSEL & NORVIG, 2004:924)

Para o comportamento exterior permanecer o mesmo enquanto é feita a operação é necessário reduzir por completo a vontade do indivíduo. Essa remoção das vontades do indivíduo é uma hipótese improvável. (*Idem*, 924)

Para RUSSEL & NORVIG (2004, 925) se fizermos perguntas ao indivíduo relativas à sua experiência consciente durante a operação obteremos respostas consistentes, ou se cutucarmos o indivíduo com um bastão pontudo ele nos daria uma resposta que o bastão machuca, os céticos podem dizer que programamos regras para obtermos essas respostas. Porém conseguimos reproduzir as propriedades funcionais de um cérebro humano, supomos que um cérebro eletrônico não possui essas propriedades, então devemos achar uma explicação das manifestações realizadas pelo cérebro eletrônico que apelem para as propriedades dos neurônios, chegamos a duas conclusões:

- Os mecanismos causais de consciência que geram esses tipos de saída em cérebros normais ainda estão operando na versão eletrônica, que é portanto consciente.
- Os eventos mentais conscientes no cérebro normal não têm nenhuma conexão causal com o comportamento, e foram omitidos do cérebro eletrônico, que portanto não é consciente.

Considere agora que a operação seja desfeita e o indivíduo volte a ter um cérebro normal. Seu comportamento deve ser igual se a operação não fosse realizada. Porém o indivíduo deverá possuir na memória a exata consciência de suas experiências até aquele momento. (*Idem*, 925)

Porém o experimento não foi realizado da maneira apropriada se os neurônios retirados fossem colocados em animação suspensa do momento em que foram retirados até o momento em que são recolocados no cérebro, eles não lembrarão da operação, é preciso ter certeza de que os estados dos neurônios artificiais sejam atualizados aos neurônios naturais, se os neurônios naturais causarem um comportamento diferente ao que os neurônios artificiais provocaram, significaria que os neurônios artificiais não são equivalentes aos naturais. (*Ibidem*, 925)

Patricia Churchland afirma que a operação também pode ser realizada com qualquer unidade funcional maior que um neurônio como um módulo mental, um hemisfério ou até um

cérebro inteiro. Isso significa que se aceitarmos o experimento de prótese cerebral, aceitaríamos que a consciência se mantém mesmo quando o cérebro é trocado por um circuito que faz mapeamento de entrada/saída em uma tabela de busca, isso vai contra a idéia de Turing, de que as tabelas de buscas não são conscientes. Isso sugere que o experimento de prótese cerebral não pode substituir um cérebro inteiro, porém não significa que se deve usar a substituição de um átomo de cada vez. (RUSSEL & NORVIG, 2004:925-926)

### 2.1.2.1.3 O quarto chinês

O último experimento e o mais famoso entre eles, é o do quarto chinês desenvolvido inteiramente por Jhon Searle, ele descreve um sistema que passa pelo Teste de Turing, porém esclarece que o programa não compreende suas entradas e saídas, concluindo que a execução de um programa apropriado não é suficiente para ser uma mente. (RUSSEL & NORVIG, 2004:926)

O sistema consiste em um ser humano que só compreenda inglês, equipado com um livro de regras folhas de papel algumas em branco e outras com símbolos indecifráveis. Ele está em um quarto com uma pequena abertura e de lá passam folhas de papel com símbolos. O ser humano encontra esses símbolos no livro de regras e executa os procedimentos descritos neles, esses procedimentos podem ser desenhar símbolos em outras folhas e repassar para o mundo exterior. (*Idem*, 926)

Do exterior do quarto percebe-se que o sistema está recebendo dados sob a forma de sentenças em chinês e gera respostas em chinês “inteligentes”. A pessoa no quarto não compreende chinês, o livro de regras e os papéis também não. Então não está sendo compreendido o dialeto chinês. (*Ibidem*, 926)

Como Turing, Searle recebeu várias críticas uma delas é que, embora se possa perguntar se o ser humano no quarto entende chinês, isso é o mesmo que perguntar se a CPU pode efetuar raízes cúbicas. Ambos os casos as respostas seriam negativas, e em ambos os casos os sistemas tem a capacidade em questão. Caso se pergunte ao quarto se ele sabe chinês, a resposta seria positiva em chinês fluente. Isso é o suficiente para Turing. Searle quer afirmar que a compreensão não está no ser humano nem no papel, não podendo haver qualquer compreensão. Mesmo se o ser humano decorasse o livro de regras e o conteúdo dos papéis se fosse perguntado para ele em inglês a resposta será negativa. (RUSSEL & NORVIG, 2004:926)

Essa afirmação levanta quatro axiomas<sup>15</sup>:

- Os programas de computador são entidades sintáticas formais;
- Mentes têm conteúdo mental, ou semântica;
- Sozinha, a sintaxe não é suficiente para a semântica;
- Cérebros causam mentes;

Com os três primeiros axiomas conclui-se que os programas não são suficientes para mentes. Ou seja, um Agente que executa um programa pode não ser uma mente e não precisa ser uma mente para executar o programa. A partir do quarto axioma ele conclui que qualquer outro sistema capaz de causar mentes deve possuir poderes causais equivalente aos do cérebro. A partir daí ele deduz que qualquer cérebro artificial teria de reproduzir os poderes causais do cérebro, e não apenas executar um só programa específico, e que os cérebros humanos não produzem fenômenos em virtude de um programa. (RUSSEL & NORVIG, 2004:927)

A finalidade do argumento do quarto chinês é fornecer intuições para o terceiro axioma, porém a reação desse argumento mostra que ele fornece intuições para aqueles que já estão inclinados a aceitar a idéia de que programas não geram compreensão verdadeira. (*Idem*, 927)

Searle admite a possibilidade de que o cérebro realmente esteja implementando um programa de IA tradicional. Ele nega a crença de que máquinas não podem ter mentes, porém afirma que algumas máquinas têm mentes. Somos máquinas biológicas com mentes. (*Ibidem*, 927)

Para termos a IA Forte é preciso discutir o problema de mente/corpo que está muito longe de acabar, tendo duas correntes filosóficas discutindo isso a materialista e a dualista. Tendo como demonstração três experimentos percebe-se a dificuldade de diferenciação de um estado mental de um estado físico, para depois iniciar uma implementação de uma IA Forte.

## 2.2 Abordagens

Com base nos diversos campos que a IA rege existem duas abordagens (SIMON 1988 *apud* FERNANDES, 2005:3).

A Abordagem cognitiva ou conhecida como Simbolista, dá ênfase em como o ser humano raciocina. Ela tem por objetivo encontrar uma explicação para comportamentos

---

<sup>15</sup> Reivindicação que pode ser vista como verdadeira sem nenhuma prova

inteligentes baseados em aspectos psicológicos e processos algorítmicos, tem como pioneiros Jhon McCarthy, Marvin Minsky, Newell e Simon. (FERNANDES, 2005:3)

A Abordagem Conexionista ou conhecida como biológica, dá ênfase no funcionamento do cérebro, dos neurônios, e das conexões neurais, tendo como pioneiros McCulloch, Pitts, Helder, Rosenblatt e Widrow. (*Idem*, 3)

### 2.2.1 Abordagem Simbolista

A Inteligência Artificial Simbolista surgiu da união da Psicologia comportamentalista dos anos 50 e do recente computador digital dos anos 60. Sendo a ciência que busca os meios necessários à imitação da inteligência humana. (Carvalho 2002:61)

A inteligência humana possui muitas facetas. Sendo impossível ter a exata definição do que é a inteligência humana. (*Idem*, 61)

Como não conseguimos ter uma definição da inteligência humana então descreveremos algumas de suas características:

- Capacidade de comunicação;
- Conhecimento próprio;
- Conhecer o mundo;
- Traçado de planos ou metas;
- Criatividade;

Capacidade de comunicação: quanto mais versátil o uso das linguagens para expressar as suas idéias e sentimentos, mais “inteligente” parece o ser humano, porém existem seres humanos com um grande nível de inteligência e uma pequena capacidade de comunicação com seus semelhantes. (CARVALHO, 2002:62)

Conhecimento próprio ou a consciência: diferem-nos dos seres irracionais, sabemos o que somos quem somos o que sabemos e o que não sabemos, o grau de inteligência é dependente desta capacidade, que talvez seja a maior expressão de inteligência. (*Idem*, 62)

Conhecer o mundo: Máquinas que executam tarefas complexas, mas não tem conhecimento do que está fazendo (o problema do “quarto chinês”), não parecem ser inteligentes. Quanto mais o conhecimento das relações do ser com o seu mundo, mais ele será eficiente na manipulação do mundo exterior. Isto requer várias formas de memórias para que o mundo daquele ser seja armazenado para uso futuro, máquinas sem memória não podem seguramente demonstrar inteligência. (*Ibidem*, 62)

Traçados de planos ou metas: Todo movimento do ser humano inteligente é uma meta, seja um plano de casamento ou até o de atravessar uma rua. O ser inteligente é capaz de encontrar a solução, subdividir a meta, vencer etapas intermediárias, até a conclusão de seu objetivo. A definição de planos também mostra que o ser tem autonomia, desejos, idéias e consciência. (CARVALHO, 2002:62)

Criatividade: A mais inexplicável das propriedades da inteligência, o poder criativo que retira do “nada” de idéias antigas um conceito novo que resulta em um salto qualitativo que acrescenta conhecimento. (*Idem*, 62)

Definidas estas propriedades, conclui-se que a Inteligência Artificial Simbolista assim como a biologia estuda algo que não sabe exatamente, em outras palavras a Inteligência Artificial Simbolista seria “o estudo de como criar máquinas que realizam tarefas em que, no momento, as pessoas são melhores”. Portanto se não existir uma máquina que consiga reconhecer pessoas numa foto como um ser humano, este será o objetivo da IA Simbolista. (*Ibidem*, 63)

Com esta definição a IA Simbolista entra num paradoxo insolúvel: pois se gerarmos uma máquina que consiga realizar uma tarefa semelhante ao ser humano, essa máquina já não será alvo do estudo da IA Simbolista, pois a tarefa foi mecanizada. Ou seja, a IA Simbolista considera que uma tarefa mecanizada, não será mais inteligente pelo fato de ter sido mecanizada. Se a realização de grandes cálculos com ajuda da tábua de logaritmos há séculos atrás, era sinal de inteligência, hoje fazemos isso com nossas pequenas calculadoras, fazendo com que isto seja uma tarefa mecânica deslocando a inteligência para outros processos ainda não mecanizados. (CARVALHO, 2002:63)

### **2.2.1.1 Metodologia Simbolista**

Conforme CARVALHO (2002, 63) devido às influências da Psicologia e da estrutura dos novos computadores a IA Simbolista, assume que os processos inteligentes sempre são uma seqüência de operações controladas por um supervisor. As operações devem ser representadas por símbolos. A própria inteligência estaria guardada em símbolos especiais de alto nível, denominados de heurísticas.

A metodologia da IA Simbolista pode ser descrita em três fases:

- Escolher uma atividade inteligente para estudo;
- Desenvolver uma estrutura lógico-simbólica capaz de imitá-la;

- Comparar a eficiência dessa estrutura com a atividade inteligente real;

Devido ao fato de que a metodologia Simbolista tem fundamentos na escolha de uma atividade inteligente, surgem diversas subáreas específicas:

- Processamento de Linguagem Natural
- Sistemas Especialistas
- Planejamento
- Solução de Problemas
- Reconhecimento de Padrões
- Aprendizado de máquina

Sistemas de Processamento de Linguagem Natural: Trata de criar algoritmos para o entendimento da linguagem humana escrita e falada. A linguagem envolve mecanismos complexos ainda não entendidos até hoje. Esses sistemas devem possuir um conhecimento grande embutido e de fácil acesso. Alguns sistemas conseguem dialogar dentro de certos temas, resumirem textos ou até mesmo responderem perguntas feitas para consultas a banco de dados. (CARVALHO, 2002:64)

Sistemas Especialistas: Imitam o raciocínio de um especialista em certo ramo do conhecimento. Vários especialistas são consultados e suas ações frente a certas situações são representados e programados no sistema. O sistema passa então a agir como se fosse um especialista. Já foram desenvolvidos diversos sistemas especialistas nas áreas da medicina, financeira e gerencial. (*Idem*, 64)

Sistemas de Planejamento: São capazes de traçar estratégias na área administrativa, outros podem gerar planos de como ligar e desligar redes de equipamentos sem causar danos. Dentro desta subárea há a programação automática que estuda como criar programas capazes de programar quase sem nenhuma interferência humana. (*Ibidem*, 64)

Sistemas de Solução de Problemas: Desenvolve novas metodologias para resolver problemas matemáticos. Muitos problemas são complexos que não são resolvidos em um tempo considerável, então passa ser objetivo encontrar soluções aproximadas em menor tempo. (CARVALHO, 2002:65)

Sistemas de Reconhecimento de Padrões: Reconhece qualquer tipo de padrão, padrões auditivos podem ser reconhecidos para que a máquina execute um comando de voz. Também reconhecem padrões visuais para perceberem peças defeituosas na linha de montagem ou uma falha na chapa de um raio-X. Padrões econômicos parta falência de bancos ou empresas também são detectados por técnicas de reconhecimento de padrões. (*Idem*, 65)

Sistema de Aprendizado de Máquina: Preocupa com a criação de algoritmos que permitam á máquina aprender com o ambiente ao redor dela. Se um algoritmo de aprendizado receber uma grande massa de dados ele poderá tirar conclusões das relações dos dados e transformará esses dados em regras para ele. (CARVALHO, 2002:65)

A metodologia Simbolista está preocupada em “imitar” a inteligência. Essa preocupação em excesso a distanciou das ciências do cérebro.

### 2.2.1.2 Hipóteses Simbolistas

A IA Simbolista possui duas hipóteses que a definem:

- Hipótese do Sistema de Símbolos;
- Hipótese da Busca Heurística<sup>16</sup>;

Hipótese do Sistema de Símbolos: Afirma que um sistema simbólico possui meios necessários para a ação inteligente, afirmando que sistemas simbólicos são necessários à expressão inteligente é cabível, pois nós realmente processamos símbolos a todo o momento, porém dizer que eles são suficientes para qualquer ação inteligente vai contra a abordagem Simbolista. (CARVALHO, 2002:66)

Hipótese da Busca Heurística: Afirma que um sistema de símbolos demonstra a inteligência na solução de problemas por meio de uma busca guiada por heurísticas, ou seja, ele gera novas formas simbólicas com base em experiências já vividas até que seja encontrada a solução. (*Idem*, 66)

Sistema de símbolos é um conjunto de entidades padronizadas que podem ser componentes de outras entidades simbólicas maiores. A sintaxe do sistema estabelece como os símbolos serão definidos e quais expressões possíveis. Deve possuir operadores capazes de criar, modificar, reproduzir e destruir seus símbolos. Possui dois processos para a semântica dos símbolos o de designação onde procedimentos computacionais são designados para cada símbolo e interpretação onde os procedimentos computacionais relacionados a cada símbolo são executados pelo computador. (*Ibidem*, 66)

A abordagem Simbolista é a mais ampla da IA, nela está quase todas as subáreas da IA, tendo como base o processamento de símbolos e regras de heurísticas, ela esquece a compreensão da mente humana, e busca a sua simulação.

---

<sup>16</sup> Série de conhecimentos que proporcionam uma rápida solução para algum problema ou dificuldade, com o menor gasto de energia ou esforço

### **2.2.2 Abordagem Conexionista**

A abordagem Simbolista dominou a IA entre os anos 60 a 80, quando John Hopfield biólogo e físico construiu uma rede de neurônios com a capacidade de simular a memória humana. Sua rede neural, junto com os novos computadores paralelos, levantou questões importantes contra a abordagem simbolista, pois em 20 anos eles não conseguiram resultados relevantes de emulação da inteligência. (CARVALHO, 2002 99)

O maior questionamento à abordagem Simbolista foi a hipótese da suficiência de estruturas lógicas para o entendimento da mente humana, desprezando totalmente o maquinário cerebral. (*Idem*, 100)

A abordagem Conexionista, afirma que a estrutura do cérebro é fundamental para a compreensão da mente. Para os conexionistas os processos mentais surgem do comportamento de uma população de elementos simples, que se conectam e trocam sinais entre si. (*Ibidem*, 101)

A maneira com que esses elementos se comunicam é fundamental para o surgimento do processo mental, daí o nome conexionista adotado por esta comunidade de cientistas. Assim o processamento da informação não seria mais centralizado e seqüencial, mas, sim paralelo e distribuído no espaço, pois cada neurônio executará suas ações em diferentes locais e sem sincronia com os demais. A inteligência para os conexionistas não vem em regras de heurísticas e sim nas conexões entre os neurônios. (CARVALHO, 2002:101)

A mais nova das abordagens tem como objetivo o estudo do cérebro humano, para isso utilizam-se da construção de neurônios artificiais e das interações de suas conexões para a geração da inteligência.

### **2.3 Ramificações**

A IA abrange atualmente diversos sub-campos desde áreas de uso geral como aprendizado e percepção, até tarefas específicas como jogar xadrez, demonstrações de teoremas matemáticos, ela sistematiza e automatiza tarefas intelectuais, portanto sendo importante em qualquer área da atividade intelectual humana, sendo um campo universal.

Alguns desses sub-campos serão apresentados mais adiante.

### 2.3.1 Sistemas Especialistas

São sistemas que solucionam problemas que podem ser resolvidos por pessoas especialistas. Também conhecidos como sistemas baseados em conhecimento. (FERNANDES, 2005:13)

O processo para a construção de um sistema especialista é chamado de “Engenharia do Conhecimento”, onde envolve a interação entre o desenvolvedor do sistema conhecido como “Engenheiro do Conhecimento”, e um ou mais especialistas. O Engenheiro do Conhecimento “extrai” do especialista seus procedimentos, estratégias e regras para a solução de problemas e constrói seu conhecimento dentro do sistema. (WATTERMAN, 1986 *apud* FERNANDES, 2005:13-14)

Para tomar uma decisão sobre um determinado assunto, um especialista consulta em sua memória um conhecimento, armazenado no período de sua formação e no decorrer de sua vida profissional sobre esse assunto, e toma a sua decisão de acordo com a sua experiência. (SCHWABE & CARVALHO, 1987 *apud* FERNANDES 2005:14)

Durante esse processo o especialista verifica a importância dos fatos levantados, faz uma comparação entre eles e seu conhecimento acumulado. Formula novas hipóteses e verifica novos fatos, esses fatos podem influenciar a decisão. Com esse processo o especialista pode não chegar a uma conclusão se seu conhecimento prévio for muito baixo. Podendo chegar a uma conclusão errada, sendo justificado pelos fatos apurados e conhecimento prévio. (FERNANDES, 2005:14)

O Sistema especialista deve ter a capacidade de aprender novos conhecimentos, melhorando o desempenho de raciocínio e a qualidade das decisões. (*Idem*, 14)

#### 2.3.1.1 Estrutura de um Sistema Especialista

Segundo FERNANDES (2005:16) um sistema especialista apresenta cinco componente básicos:

- Base de conhecimento;
- Máquina de inferência;
- Subsistema de aquisição de conhecimento;
- Subsistema de explicações;
- Interface do usuário;

Base de Conhecimento: Local onde os fatos e regras que representam o conhecimento do especialista residem. Muitos sistemas utilizam regras como a base de conhecimento sendo considerados “sistemas baseados em regras”. Outros sistemas utilizam de outras técnicas para a representação do conhecimento, como redes semânticas e *frames*. Pelo fato da base de conhecimento ser separada da máquina de inferência o conhecimento contido na base é fácil de ser modificado. Podendo adicionar regras novas, remover ou modificar as regras antigas. (FERNANDES, 2005:17)

Máquina de inferência: Mecanismo que procura as respostas na base de conhecimento. Encontra regras necessárias para o problema e ordena-as de uma forma lógica. Funciona como um “supervisor”, que dirige a operação sobre toda a base de conhecimento do sistema. Toma decisões e julgamentos baseados em dados simbólicos existentes na base de conhecimento. (*Idem*, 17-18)

Subsistema de aquisição de conhecimento: Utilizado para alimentar a base de conhecimento. Meio por onde são adicionadas novas regras e modificam ou eliminam regras antigas. (*Ibidem*, 18)

Subsistema de explicações: Designado para explicar ao usuário a linha de raciocínio que o sistema utilizou para encontrar a solução. Usuários dos sistemas podem perguntar “Por quê?” ou “Como?” e o sistema apresentará uma resposta. Este subsistema é ótimo para usar em situações instrutoriais e para *debugging*<sup>17</sup> durante o desenvolvimento do sistema. (FERNANDES, 2005:18)

Interface do usuário: Utilizada para estabelecer um meio de comunicação entre o usuário e o sistema, podendo ser na forma de menus, perguntas e representação gráfica exibidas na tela do computador. Também exibe todas as perguntas, repostas e resultados de consultas. (*Idem*, 18)

### **2.3.1.2 Aquisição do conhecimento**

Cada vez mais usados nas organizações, os sistemas especialistas auxiliam na tomada de decisões, porém o processo de aquisição de conhecimento passa despercebido e é tida como um “gargalo” no desenvolvimento do sistema, o responsável por essa aquisição é o Engenheiro do conhecimento. (ROOK & CROGHAN, 1989 *Apud* FERNANDES 2005:18-19)

---

<sup>17</sup> Identificação e correção de erros no *software*

O engenheiro do conhecimento é a figura central tanto na aquisição de conhecimento, como no desenvolvimento. Por isso a pessoa deve ter um preparo especial e algumas qualidades inatas. (ROLANDI 1986 *apud* FERNANDES 2005:19)

Esse processo compreende a aquisição, análise e representação do conhecimento, representado hoje o principal gargalo para o desenvolvimento de sistemas especialistas. (FERNANDES, 2005:19)

Conforme GONÇALVES (1986) *apud* FERNANDES (2005:19) Existem duas abordagens para a engenharia do conhecimento:

- Abordagem psicológica;
- Abordagem baseada em modelos.

Abordagem psicológica: Procura obter sistemas especialistas que “imitam” o especialista. O conhecimento modelado é resultante do processo mental interno do especialista.

Abordagem baseada em modelos: Parte de modelos de tarefas básicas como diagnósticos. Abordagem é a mais recente tendo suas ferramentas de implantação ainda em processo de desenvolvimento. (GONÇALVES *apud* FERNANDES 2005:19)

### **2.3.1.3 Sistemas Especialistas Conhecidos**

Serão apresentados a seguir os sistemas especialistas mais conhecidos:

- MYCIN;
- Prospector.

MYCIN: Um dos primeiros Sistemas Especialistas, com o objetivo de diagnosticar doenças infecciosas. Muito útil, pois nem sempre o médico responsável é especialista em infecções principalmente num ambiente hospitalar. O sistema coleta informações do paciente como nome, idade, sexo, tempo de manifestação dos sintomas e resultados de exames. A partir dessas informações e de suas base de regras o sistema é capaz de definir um diagnóstico e encontrar o melhor tratamento. A base de regras possui cerca de 450 regras para diagnosticar e determinar tratamentos para doenças infecciosas. A aquisição do conhecimento é facilitada pelo fato do sistema explicar o seu raciocínio. (BITTENCOURT, UFSC/GIA, 2005)

Prospector: Desenvolvido na SRI *International* com o objetivo de auxiliar geologistas em prospecção mineral. A Principal função do sistema é determinar correspondência entre

dados com modelos de descrição de classes disjuntas de situações possíveis. O sistema possui cinco modelos montados com ajuda de cinco geólogos. O sistema funciona em duas etapas: inicialmente o usuário fornece dados do local para a prospecção, esses dados são comparados com os modelos já existentes e depois de definido um modelo juntamente com as informações fornecidas é feita uma refinação da análise. (BITTENCOURT, UFSC/GIA, 2005)

Sistemas Especialistas é a subárea de IA de mais fácil implementação, pois o “engenheiro do conhecimento” precisa transformar todo o conhecimento do especialista em regras de negócio do sistema. Na forma de um banco de dados em que são colocadas todas as alternativas para certas situações. Em que o sistema irá consultar para tomar a melhor decisão.

### 2.3.2 Robótica

Os robôs são agentes físicos que executam tarefas manipulando o mundo físico. Para isso são equipados com atuadores como braços, pernas, rodas, garras e articulações. Esses atuadores têm o propósito de exercer força física sobre o ambiente. Os robôs também possuem sensores que permitem perceber o mundo ao redor dele. Entre os sensores estão câmeras, ultra-som, giroscópios<sup>18</sup> e acelerômetros<sup>19</sup>. (RUSSEL & NORVIG, 2005:870)

Os robôs são divididos em três categorias diferentes:

- Manipuladores;
- Robô Móvel;
- Robô Humanóide.

Manipuladores: Também conhecidos como braços robôs, ficam fisicamente ancorados no local de trabalho como em uma linha de montagem ou na estação espacial internacional. Seus movimentos são causados por diversas articulações controláveis, permitindo que o robô alcance qualquer posição dentro do local de trabalho. É o tipo mais comum de robôs com cerca de um milhão de unidades em todo o mundo. Alguns são usados em hospitais para auxiliar os cirurgiões, poucas fábricas de carro sobreviveriam sem eles. (*Idem*, 870)

Robô Móvel: Se deslocam pelo ambiente através de rodas, pernas ou mecanismos semelhantes. Foram projetados para uso na entrega de alimentos em hospitais, movimento de containeres nos portos e tarefas similares. Porém há três diferentes tipos entre os robôs móveis o ULV (*unmanned land vehicle*) capaz de navegar de forma autônoma em altas estradas. UAV (*unmanned air vehicle*) usados para vigilância, pulverização e operações militares.

---

<sup>18</sup> Dispositivo usado para orientação de navios, aviões e espaçonaves

<sup>19</sup> Dispositivo usado para medir a aceleração

AUV (*autonomous underwater vehicle*) usados em explorações submarinas. Também os viajantes exploratórios como o Sojourner. (RUSSEL & NORVIG, 2005:870-871)

Robô Humanóide: São robôs híbridos, um, robô móvel equipado com manipuladores, cuja estrutura imita o torso humano. Eles podem aplicar seus efetadores em um campo mais amplo, porém pelo fato de não possuírem uma rigidez física do ponto de fixação, as tarefas se tornam mais difíceis. (*Idem*, 871)

A robótica também inclui dispositivos protéticos como membros artificiais, olhos e orelhas para seres humanos, ambientes inteligentes como casas equipadas com sensores e efetadores e sistemas com vários corpos, onde a ação robótica ocorre por enxames de pequenos robôs. (*Ibidem*, 871)

Para RUSSEL & NORVIG (2004, 871) os robôs devem lidar com ambientes parcialmente observáveis, dinâmicos e contínuos. Nem todos os ambientes de robôs são sequenciais ou de multiagentes. O robô não pode ver em torno de obstáculos, e os comandos estão sujeitos à incerteza, devido ao deslizamento de engrenagens, à fricção. Os sistemas robóticos precisam incorporar o conhecimento anterior sobre o robô, seu ambiente físico e suas tarefas a serem executadas, de forma que o robô possa aprender com rapidez e executar as tarefas de segurança.

### **2.3.2.1 Hardware**

O grande sucesso dos robôs depende do projeto de sensores e efetadores para a execução da tarefa. (RUSSEL & NORVIG, 2004:872)

Segundo RUSSEL & NORVIG (2004:872) o sensor é a interface perceptiva entre robôs e seus ambientes. Existem os sensores passivos e os sensores ativos. Os sensores passivos são observadores do ambiente, eles captam sinais gerados por fontes do ambiente. Sensores ativos enviam energia ao ambiente e aguarda a reflexão do sinal, como um sonar<sup>20</sup>.

Os sensores ativos coletam mais informações que os sensores passivos, porém possuem um custo maior e podem sofrer com interferência. Independente disso existe três tipos de sensores:

- Telêmetros;
- Sensores de tratamento de imagem;
- Sensores Proprioceptivos.

---

<sup>20</sup> Aparato capaz de emitir ondas de som em objetos para captar seus ecos

Telêmetros: São Sensores que medem a distância de objetos que estão ao seu redor. O tipo comum é o sensor de sonar. Emitem ondas sonoras direcionais que serão refletidas pelos objetos e capturadas novamente pelo sonar, esse sinal retornado traz as informações necessárias para encontrar a distância de objetos próximos. Os sonares subaquáticos é a tecnologia utilizada pelos AUV's, em terra os sonares são utilizados para evitarem colisões com objetos próximos. (RUSSEL & NORVIG, 2004:872)

Sensores de tratamento de imagens: As câmeras fornecem imagens do ambiente e utilizando de técnicas de visão computacional podem-se tirar modelos e características do ambiente. A visão estereoscópica é importante na robótica, pois com ela são obtidos dados sobre a profundidade e entre outros. Porém com as novas tecnologias de reconhecimento de imagem que estão dando certo podem deixá-las obsoletas. (*Idem*, 873)

Sensores Proprioceptivos: Informam ao robô sobre seus próprios estados. Para medir a configuração de uma articulação o robô utiliza de decodificador de eixos que contam a revolução de motores. Em robôs móveis os decodificadores de eixos são utilizados para a odometria em que mede a distância percorrida, funcionando somente para distâncias curtas. (*Ibidem*, 873)

Conforme RUSSEL & NORVIG (2004, 873) os efetadores são os meios pelos quais os robôs se movem e alteram seu corpo no ambiente. Os robôs móveis possuem diversas formas de se locomoverem como rodas, esteiras e pernas. Há robôs móveis de tração diferencial em que possuem duas rodas ou esteiras uma de cada lado do robô que são acionadas independentemente uma da outra. Se as duas estiverem na mesma velocidade e mesma direção o robô irá pra frente se estiverem em sentido oposto o robô girará em torno do próprio eixo. Uma alternativa é utilizar a tração sincronizada, no qual cada roda age independente em torno de seu próprio eixo, porém precisa-se de restrições como as rodas devem apontar para a mesma direção e velocidades para evitar o caos.

As pernas, diferente das rodas, conseguem andar em terrenos acidentados, porém em terrenos planos as pernas são mais lentas. Já foram realizadas pesquisas com robôs de uma, dez pernas, sendo construídos para caminhar, correr e até mesmo saltar. Outros robôs móveis utilizam de diferentes mecanismos para se locomoverem com turbinas, propulsores e os dirigíveis robóticos utiliza-se de efeitos térmicos para se manterem no ar. (*Idem*, 874)

Um robô, além de usar sensores e efetadores necessita também de um motor elétrico para fornecer energia para locomoção. Eles também precisam de um meio de comunicação, como rede sem fio. Além de ter uma estrutura corporal para as peças serem encaixadas. (*Ibidem*, 875)

### 2.3.2.2 Robôs conhecidos

Serão apresentados a seguir alguns dos robôs mais conhecidos:

- Spirit;
- Aibo.

Spirit: Veículo que a NASA utiliza para a exploração juntamente com o Opportunity explora o terreno de Marte analisando rochas e terreno e enviando imagens e vídeos de suas explorações para a NASA (PIRES, Universidade de Coimbra, 2004)

Aibo: Robô da Sony, não só interage com o dono como “aprende” com ele desenvolvendo uma personalidade única, as ações podem mudar conforme o tempo como os outros animais, estão disponíveis para a venda somente no Japão e Estados Unidos. (MESSA & PAIM, 1999)

Robótica é a área em que se desenvolvem agentes autômatos físicos utilizados para exploração espacial e locais onde o homem não conseguiria alcançar ou ficar como vulcões e geleiras, na linha de montagem de carros, além da área doméstica como guias de deficientes físicos, babás e animais de estimação.

### 2.3.3 Sistemas Visuais

Segundo STAIRS & REYNOLDS (2006:424), sistemas de visão envolvem hardware e software que permitem os computadores capturar e armazenar e manipular imagens visuais.

O departamento de justiça dos Estados Unidos usa sistemas de visão para identificação de impressões digitais. A velocidade que o sistema percorre a base de dados para encontrar impressões digitais trouxe soluções rápidas para casos antigos. (*Idem*, 424)

Sistemas de visão são capazes também de reconhecer características faciais, Canesta uma empresa da Califórnia, usa raios infravermelhos para que seus computadores capturem imagens tridimensionais de objetos. Os raios são direcionados ao objeto e refletidos de volta, a diferença no tempo da luz retornar dá ao sistema a imagem do objeto tridimensional. (*Ibidem*, 424)

Também podem ser usados juntamente com a robótica para dar visão aos robôs. A visão aumenta a capacidade dos robôs deixando tomar decisões com base na entrada visual.

Os robôs com sistema visual podem reconhecer nas cores pretas, brancas e alguns tons de cinza, sem uma boa visão colorida ou tridimensional. (STAIRS & REYNOLDS, 2006:424)

Outros sistemas se concentram apenas em algumas características de uma imagem ignorando o resto, vai demorar que um sistema possa “ver” e tirar conclusões do modo de um humano. (STAIRS & REYNOLDS, 2006:424)

Área de sistemas de visão é a que dá “visão” às máquinas, muito usada em robôs para a tomada de decisão, além de ser utilizada em sistemas biométricos para reconhecimento de íris, e de peças defeituosas em linha de montagens.

### **2.3.4 Processamento de Linguagem Natural**

Conforme STAIRS & REYNOLDS (2006:424), processamento de linguagem natural permite ao computador reconhecer comandos de voz em uma linguagem natural. Existem três níveis para o reconhecimento: comandos (reconhece de dezenas a centenas de palavras), discreto (reconhece fala ditada e com pausas entre as palavras) e contínuo (reconhece a fala natural). Processamento de linguagem natural pode ser usado para recuperar informações sem digitar comandos ou procurar palavras-chave. Pode-se falar em um microfone conectado ao computador e o computador converte a fala em arquivos de textos ou comandos. Sistemas simples conseguem associar uma palavra digitada a uma palavra falada pelo microfone, sistemas mais avançados não precisam gravar as palavras. A empresa Upstart Natural Machine está desenvolvendo um sistema em Java o IA verbal universal. Ela espera o beneficiamento de todos por tornar o código aberto e disponível.

As corretoras de ações utilizam bastante os sistemas de linguagem natural. A Schwab usa uma máquina de linguagem natural para ajudar os seus clientes a navegar em seu site. A T. Rower Price usa reconhecimento de voz para que seus clientes obtenham dados de pensões, ações, saldos de conta-corrente pelo telefone utilizando de comandos de voz simples, ela espera ainda que possa ser realizadas transações pelo telefone futuramente. A TDWaterhouse usa uma máquina de linguagem natural para responder as dúvidas dos clientes, isso fez cair a demanda de chamadas ao departamento de serviços a clientes. (*Idem*, 424-425)

A Hewlett-Packard usa um programa da SpechWorks em suas centrais de atendimento para a assistência e redução de custos. Uma pessoa em geral possui um vocabulário de 20 mil palavras o sistema que a Hewlett-Packard usa possui um vocabulário de 85 mil palavras. A SpeechWorks tem como clientes a FedEx e Continental Airlines. A empresa alega que seu

software é tão bom que alguns usuários esquecem que estão falando com um computador e começam a discutir sobre o clima ou resultados esportivos. (STAIRS & REYNOLDS, 2006:425)

Processamento de Linguagem Natural é muito usado para a área de biometria como os sistemas visuais, para o reconhecimento de voz, diversas corretoras de ações utilizam para o reconhecimento de seu cliente e outras funcionalidades, os sistemas podem chegar a interpretar comandos de voz até falas naturalmente.

### 2.3.5 Sistemas de Aprendizado

Área da IA, onde há uma combinação de hardware e software permitindo que o computador reaja a situações com base em seu *feedback*<sup>21</sup>, ou seja, extrair regras heurísticas em grande massas de dados. Os algoritmos de aprendizado são interessantes, pois além de modelarem os dados bem, fornecem regras de heurísticas que explicam os padrões existentes. Um dos algoritmos mais usados são os chamados algoritmos de particionamento sucessivo. Em que particionam a massa de dados várias vezes formando subgrupos até o momento em que chegue ao ponto em que se consegue extrair regras heurísticas sobre os padrões encontrados. (CARVALHO, 2002:165)

Normalmente, esses subgrupos são gerados através de uma regra heurística aplicada em um grupo. Sendo assim a melhor forma de representar o particionamento sucessivo é uma árvore binária chamada de árvore de decisão, pois em cada um de seus nós é preciso tomar uma decisão para onde os dados vão. Com a árvore já totalmente definida qualquer dado novo pode ser classificado em algum subgrupo desde que ele seja inserido a partir do nó raiz da árvore, ele vai passando pelas regras heurísticas decidindo seu caminho até chegar ao nó terminal conhecido de folha da árvore. (*Idem*, 165)

As árvores de decisões mais usadas são ID3 (ou C4.5) e CART, ambas buscam tornar os subgrupos cada vez mais homogêneos com o objetivo de atingir classes de dados bem definidas e organizadas. O segredo está em como o algoritmo escolhe a regra para definir os subgrupos mais homogêneos possível. (*Ibidem*, 166)

Técnica muito utilizada na área de redes neurais em que passa por um treinamento e em jogos eletrônicos como futebol e principalmente de estratégia em que o computador registra suas jogadas e resultados para que no futuro não cometa o mesmo erro.

---

<sup>21</sup> Resposta, retorno de uma ação para que seja feita uma correção que futuramente reduz ou aumenta a resposta do sistema

### 2.3.6 Redes Neurais Artificiais

Para FERNANDES (2005:57) as redes neurais artificiais podem ter várias definições, porém as três palavras-chave: neurônio, arquitetura e aprendizagem, requerem um entendimento, mais concentrado. Em qualquer definição de rede neural, o neurônio é a unidade básica da rede, a arquitetura é a estrutura de como os neurônios são conectados e a aprendizagem é um processo que adapta a rede de modo a realizar uma tarefa.

*As Redes Neurais Artificiais são sistemas físicos que podem adquirir, armazenar e utilizar conhecimentos experimentais, que podem alcançar uma boa performance, devido à sua densa interconexão entre os nós da rede. Elas também são conhecidas por: modelos conexionistas, modelos de processamento paralelo distribuído e sistemas neuromorfológicos (LIPPMANN 1997 apud FERNANDES 2005:57).*

Atualmente, há um grande interesse em Redes Neurais Artificiais, psicólogos se interessam pelas semelhanças com a mente humana. Os pesquisadores de IA buscam nas redes neurais soluções para o aprendizado de máquina. (FERNANDES, 2005:57)

#### 2.3.6.1 Neurônio Artificial

Uma rede neural é composta por várias unidades de processamento (neurônios artificiais) conectadas por canais de comunicação que possuem um determinado peso. As unidades fazem operações somente com os seus próprios dados recebidos pelos canais. A inteligência da Rede Neural vem das interações entre as unidades de processamento. (FERNANDES, 2005:59-60)

Conforme FERNANDES (2005:60) McCulloch e Pitts em 1943 levantaram uma proposta sobre o funcionamento de um neurônio com os seguintes passos:

- Dados são entregues à entrada;
- Cada dado é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade;
- É feita a soma ponderada dos produtos, resultando em um nível de atividade;
- Se este nível de atividade exceder certo limite (*threshold*<sup>22</sup>) a unidade produz uma determinada resposta de saída;

---

<sup>22</sup> Valor mínimo de alguma quantidade

Suponha que temos os seguintes valores de entrada  $X_1, X_2, \dots, X_p$  e com os seguintes pesos  $w_1, w_2, \dots, w_p$  e um limitador  $t$ , que define valores booleanos<sup>23</sup>. Portanto, o neurônio terá a saída da seguinte forma. (FERNANDES, 2005:60)

Será feito a soma dos produtos obtidos entre os valores de entrada e os seus pesos correspondentes após isso será feito um teste junto ao limitador em que será definido o nível de atividade, ou seja o resultado da soma seja maior que o limitador o neurônio será ativado com valor 1, senão ele ficará desativado com o valor de 0. (*Idem*, 60)

Cada Neurônio é capaz de processar um sinal de entrada e transformá-lo em um sinal de saída.

### 2.3.6.2 Arquitetura

Segundo FERNANDES (2005:61) a Arquitetura de uma rede neural é organizada em formas de camada em que o neurônio possa estar conectado com, outros neurônios. Existem três camadas:

- Camada de Entrada;
- Camadas Intermediárias ou Escondidas
- Camada de Saída;

Camada de Entrada: por onde obtém os dados. (*Idem*, 61)

Camadas Intermediárias ou Escondidas: Onde é realizada quase toda a regra de negócio, podem ser consideradas como extratoras de características. (*Ibidem*, 61)

Camada de Saída: onde o resultado é concluído e apresentado. (FERNANDES, 2005:61)

As redes neurais podem ser identificadas pela direção pelo qual o valor do dado flui, sendo existente somente dois tipos. *feedforward*, e *feedback*. (*Idem*, 61)

Nas redes *feedforward* os valores só seguem um caminho, começando pela camada de entrada, passando pela camada intermediária até a camada de saída. Nas redes *feedback*, os sinais de entrada podem propagar de qualquer neurônio para qualquer outro neurônio. (*Ibidem*, 61)

A Rede Neural é especificada pela sua topologia, características dos nós e pelas regras de treinamento. Para ANDERSON (1972) um modelo de redes neurais deverá constituir-se de

---

<sup>23</sup> Valores verdadeiro ou falso

uma rede de neurônios, relativamente autônomos, com capacidade de processamento. (FERNANDES, 2005:62)

Os neurônios são ligados por conexões com pesos, que demonstra o grau de importância à conexão tem. Pontos positivos correspondem a um reforço do sinal, e pontos negativos corresponde ao fator inibição. (ANDERSON, 1972 *apud* FERNANDES, 2005:62)

### 2.3.6.3 Aprendizagem

A propriedade mais importante das Redes Neurais é a habilidade de “aprender seu ambiente” e melhorar ainda o ambiente. (FERNANDES, 2005:62)

A habilidade de aprender ocorre através de um processo de ajustes aplicados a seus pesos. O aprendizado chega ao fim quando a rede neural atinge uma solução generalizada para o problema. (ACKLEY, 1985 *apud* FERNANDES, 2005:62)

O algoritmo de aprendizado é um conjunto de regras para a solução de um problema de aprendizado. Há muitos algoritmos específicos para determinados modelos de redes neurais que diferem entre si. (*Op. Cit.*, 63)

Uma rede Neural tem as seguintes formas de aprendizado:

- Aprendizado Supervisionado;
- Aprendizado Não-Supervisionado;
- Reforço.

Aprendizado Supervisionado: é utilizado um agente externo exibindo a resposta para o padrão de entrada. (FERNANDES, 2005:63)

Aprendizado Não-Supervisionado: quando não existe um agente externo apontando para as respostas correspondentes aos padrões de entrada. (*Idem*, 63)

Reforço: um crítico avalia a resposta fornecida pela rede. (*Ibidem*, 63)

O algoritmo de aprendizado ocorre num ciclo onde todos os pares de entrada e saída são verificados um a um. (FERNANDES, 2005:63)

Após a execução do algoritmo de aprendizado a rede neural deve corrigir os pesos de suas entradas, para isso existem dois modos:

- Modo Padrão;
- Modo Batch.

Modo Padrão: A correção é realizada após cada apresentação do resultado de treinamento. Cada correção é baseada somente no erro apresentado, podendo ter várias correções em cada ciclo. (FERNANDES, 2005:63)

Modo Batch: é feita uma correção por ciclo. Os erros são apresentados para a rede, o erro médio é escolhido e usado para a realização das correções. (*Idem*, 64)

As redes neurais podem ser classificadas de acordo com o tipo de aprendizado que utiliza supervisionado, não-supervisionado. (HAYKIN, 1994 *apud* FERNANDES 2005:64)

#### **2.3.6.4 Topologia**

Uma rede neural deve possuir pelo menos duas camadas de neurônios a de entrada e a de saída, somente com isso a rede apresenta um desempenho limitado, portanto precisa-se de uma camada adicional (intermediária) onde todos os neurônios estejam ligados com todos os outros neurônios da camada vizinha, fazendo a comunicação unidirecional apresentando um comportamento estático. (RUMMELHART, 1986 *apud* FERNANDES, 2005:72)

Para Hecht-Nielsen (1987) com apenas uma camada intermediária a rede consegue extrair resultados a partir dos dados fornecidos. (FERNANDES, 2005:72)

Segundo Cybenko (1989) são necessárias duas camadas intermediárias para obtenção de resultados. Com a seguinte condição que a cada três neurônios da primeira camada intermediária, é preciso um neurônio na segunda camada. (*Idem*, 72)

A RNA de Hopfield (1982) apresenta comportamento dinâmico e fluxo multidirecional, devido à integração total de seus neurônios, evitando a idéia de camadas. Isso torna seu funcionamento complexo, tendo complicações na fase de aprendizado ou de testes, seu uso é para problemas de minimização e otimização de percursos. (*Ibidem*, 72)

Independente da forma em que a Rede é construída, quanto mais camadas de neurônios, melhor é seu desempenho, pois a capacidade de aprendizado é aumentada. Melhorando a precisão da rede de definir uma resposta (FERNANDES, 2005:73)

#### **2.3.6.5 Etapas para Desenvolvimento**

Segundo Gourney (1997), as etapas de desenvolvimento de uma aplicação de redes neurais são:

- Coleta de dados e separação em conjuntos;
- Configuração da rede;
- Treinamento;
- Teste;
- Integração.

Coleta de dados e separação em conjuntos: Esses dois primeiros passos requerem uma análise bem cuidadosa para reduzir erros e minimizar ambigüidades nos dados. Além disso os dados devem ser significativos englobando totalmente o problema, cobrindo também as exceções. Esses dados são separados em dois tipos: dados de treinamento utilizados para o treinamento da rede e dados de teste utilizados para testarem o desempenho da rede. Podem ser feitas subdivisões dentro desses grupos, para melhor classificar os dados. Determinados esses conjuntos, eles são colocados em ordem aleatória para evitar tendências associativas quanto à apresentação dos dados, além de poderem ser pré-processados para uma melhor utilização na rede. (FERNANDES, 2005:81)

Configuração da rede: É dividida em três etapas: (i) – Seleção do paradigma neural apropriado à aplicação; (ii) – Determinação da topologia da rede ser utilizada; (iii) Determinação de parâmetros do algoritmo de treinamento; A realização dessas etapas possui dicas e truques, normalmente a escolha entre quais usarem é feita de forma empírica. A configuração de uma rede requer bastante experiência dos projetistas. (*Idem*, 81)

Treinamento: Será ajustado o peso das conexões. Sendo importante considerar aspectos como inicialização de uma rede, a escolha de certos valores iniciais dos pesos pode diminuir o tempo de treinamento; o modo, o mais usado é o padrão, pois tem um menor armazenamento de dados e pouco suscetível a problemas; e o tempo de treinamento, vários fatores influenciam, a sua duração, é preciso ter algum critério de parada. O treinamento deve ser parado no momento em que a rede ter uma capacidade boa de generalização e os erros forem bem menores. (*Ibidem*, 82)

Teste: Utilizado para determinar a performance da rede, podendo ser estendido para testes de comportamento quanto a entrada de dados em que pode-se ter um dado muito pequeno sendo ignorado ou dados maiores causando um *over-training*<sup>24</sup> na rede. (FERNANDES, 2005:82-83)

Integração: A Rede Neural se integra com um sistema depois de treinada e avaliada. O sistema deve ter facilidades na aquisição de dados para um melhor funcionamento da rede

---

<sup>24</sup> Excesso de treinamento

neural, além de periodicamente monitorar o seu desempenho para realizar manutenções da rede ou indicar aos projetistas a necessidade de um re-treinamento. (FERNANDES, 2005:83)

### **2.3.6.6 Aplicação**

Conforme (TATIBANA & KAETSU, GSI UEM, 2005) redes neurais podem ser aplicadas em diversas áreas, Grupos de investimentos utilizam redes neurais para previsões de mercado.

Outra aplicação é o de reconhecimento ótico de caracteres, além de controles de processos industriais, aplicações climáticas, identificação de fraude de cartão de crédito. (*Idem*, 2005)

O Banco Mellon Bank dos EUA utilizou um sistema de redes neurais que diminuíram os prejuízos, conseguindo cobrir o investimento do banco em seis meses. (*Ibidem*, 2005)

Redes Neurais é utilizada para grandes aplicações com várias finalidades, como Mineração de Dados, em que busca um perfil em uma grande massa de dados, vários cientistas contribuíram para a construção de redes neurais apresentando modelos de neurônios e de redes neurais.

## 3 História dos Jogos Eletrônicos

Para a melhor compreensão do que é um jogo precisamos saber de onde eles surgiram e com que propósito, fazendo um levantamento histórico dos jogos.

Várias pessoas com mais de 20 anos já tiveram seus olhos lacrimejando ao jogar *Super Mario World* (Nintendo, 1991), *Donkey Kong Country* (Rareware, 1994) e *Sonic* (SEGA, 1990) e para os mais velhos, jogos como *Enduro* (Activision, 1983), *Pac-Man* (Midway, 1980) e *Pong* (Atari, 1972). Agora em pleno início do século XXI o mercado de jogos tem um grande interesse em um gráfico perfeito, efeitos sonoros quase que próximo da realidade. Esquecendo a jogabilidade e o desafio, somente em alguns jogos estas características aparecem com certo destaque.

*Nós precisamos de gráficos. Nós precisamos de uma boa interface, limpeza visual para as informações que constarão no jogo, e de gráficos que façam isso. Mas quando um projetista de jogo é questionado como o jogo dele fará a diferença, espero que ele fale de jogabilidade, diversão e criatividade – em oposição a uma idéia de que simplesmente foca em como ele é bom visualmente* (Sid Meyer, Edge Magazine, 1997 apud PERUCIA e.t al., 2005:34).

Para compreender como os Jogos chegaram a esse ponto de tecnologia e o que eles contribuíram para a sociedade é necessário conhecer sua história e evolução.

### 3.1 A Primeira Era

Segundo Merkel (2005:21,22), em 1952, A. S. Davis escreveu seu *PhD* (Doutorado) na universidade de Cambridge demonstrando um jogo da velha, rodando no *mainframe* EDSAC (*Eletronic Delay System Automatic Computer*).

Segundo Normand (RetroSpace, 2001), em 1958 o físico Willy Higinbotham desenvolveu em suas horas vagas o primeiro jogo, no osciloscópio<sup>25</sup> do observatório de BrokHaven EUA.

No MIT, em 1961, um grupo de pesquisadores chefiados por Stephen Russel desenvolve o *Spacewar*, ele era executado no *mainframe*<sup>26</sup> PDP-1 (MERKEL, 2005:22).

Para Cunha (2001) Os primeiros jogos foram desenvolvidos para *mainframes* com o objetivo de testarem algoritmos de busca, como clássicos de xadrez e damas, a preocupação dos desenvolvedores era mais o visual e descartava as técnicas de IA e a jogabilidade.

No início os idealizadores dos primeiros jogos, criavam com o intuito de testar lógica de programação ou de aumentar a popularidade do local onde trabalhavam até o momento os jogos não tinham fundamento comercial.

Para Duarte (1998:34) o primeiro console da história foi o *Odissey 100* da empresa *Magnavox*.

Em 1971, Nolan Bushnell desenvolveu uma versão *arcade* de *Spacewar* chamada de *Computer Space*, no ano seguinte ele funda a *Atari*, principal empresa de entretenimento da história, seu primeiro jogo foi o *Pong* que foi uma mania na época. (NORMAND, RetroSpace:2001).

Em 1976, a *Farchild* lançou o *Channel F*, primeiro console a começar a utilizar cartuchos, essa tendência é seguida até hoje. (*Idem*, RetroSpace:2001)

Em 1977, a *RCA* lançou o *Studio II* que vinha com quatro jogos na memória principal, além dos jogos extras vendido em cartuchos. (*Ibidem*, RetroSpace:2001)

Após a *Atari* fazer sucesso com o *Pong*, diversas empresas começaram a investir em projetos para o mercado de jogos, a principal novidade foi o cartucho trazido pela *Farchild* em que continha o jogo, isso facilitava ao usuário que poderia trocar o jogo no console sem muita complicação.

Em 1976, a *Warner Communications* comprou a *Atari*. Ao final do ano seguinte é lançado o *Atari 2600*, Nolan Bushnell começa a lançar diversos jogos, pela primeira vez na história é lançado um jogo pornô. (*Op. Cit*, RetroSpace:2001)

Em desenvolvimento na mesma época do *Atari 2600*, a *Bally* lançou o *Professional Arcade*, nessa mesma época a *Magnavox* lança uma versão melhorada do *Odissey* denominada *Odissey 2*. (NORMAND, RetroSpace:2001)

---

<sup>25</sup> Instrumento de medida eletrônica, usado para identificar comprimento de onda eletromagnética.

<sup>26</sup> Computador de grande porte

O nome do primeiro game portátil da história foi *Microvision*, desenvolvido pela *Milton Bradley* no final dos anos 70 sendo o pioneiro dos games portáteis<sup>27</sup>. (NORMAND, RetroSpace:2001)

Em 1980 a *Mattel*, lança o *Intellivision (Intelligent Television)*, grande inimigo do *Atari 2600*, dois anos depois a *Emerson* lança o *Arcádia 2001*. (*Idem*, RetroSpace:2001)

Em 1982 a *Conneticut Leather Company* (Companhia de Couro de Conneticut), lança o *ColecoVision* considerado o “melhor console de todos os tempos“. No mesmo ano a *Milton Bradley* lança o *Vectrex*, projetado para mostrar imagens vetoriais ao invés de *pixels*<sup>28</sup> como a maioria dos consoles da época. (*Ibidem*, Retrospace:2001)

Ameaçada pelo *IntelliVision* e com o lançamento do *ColecoVision* a *Atari* desenvolve uma atualização do *Atari 2600* denominada de *Atari 5200*. (NORMAND, RetroSpace:2001)

Em 1983 o grande número de jogos ruins no mercado, trazida pela *Atari*, espantou os consumidores e no ano seguinte aconteceu o “*crash dos Jogos*”, quebrando todas as empresas do ramo. (*Idem*, Retrospace:2001)

Para FERRARI (N-Planet, 2006) o “*crash dos Jogos*” aconteceu por dois fatores importantes, a *Atari* lançou diversos jogos ruins para o mercado concorrendo com a nova empresa *Activision* e os computadores pessoais estavam em alta sendo melhor comprar um computador que entreteria as crianças e ajudaria nas pesquisas de trabalho, além os pais no trabalho de casa.

Ao final da primeira era o mercado americano começava a ficar saturado com os diversos jogos lançados pela *Atari* dando um efeito repulsivo aos consumidores. Os computadores também estavam começando a se popularizarem sendo barateados. Esses dois fatores acabaram quebrando todas as empresas do ramo. Este fato ficou conhecido como o “*crash dos Jogos*” fazendo alusão à quebra da bolsa de Nova Iorque em 1922.

### 3.2 A Segunda Era

Enquanto acontecia o “*crash dos Jogos*” no Japão a *Microsoft* juntamente com a *Ascii* lança o computador *MSX* considerado mais um vídeo game do que um computador normal. (NORMAND, Retrospace:2001)

Nessa mesma época no Japão a *Nintendo* lança o *Nintendo Famicom (Family Computer)* conhecido como *NES (Nintendo Entertainment System)*. A partir desse momento

<sup>27</sup> Game portátil é todo game que pode ser facilmente utilizado em qualquer lugar e a qualquer momento.

<sup>28</sup> O menor elemento em um dispositivo de imagem, em que pode ser atribuída uma cor.

as empresas japonesas começam a dominar o mercado de games. (NORMAND, RetroSpace:2001)

Com o fracasso do *Atari 5200* e o grande sucesso que a *Nintendo* estava fazendo com o *NES* a *Atari* ficou desesperada e iniciou uma pesquisa intensiva sobre o que os usuários queriam em um console. Resultando no *Atari 7800* em 1986. (*Idem*, RetroSpace:2001)

Embalada pelo sucesso do *NES*, a *Nintendo* resolve investir seus lucros na área de portáteis, então surge o *Game Boy*. (*Ibidem*, Retrospace:2001)

Com o grande sucesso da *Nintendo* nesse novo mercado de jogos domésticos, a *SEGA* (*Service Games*) gigante japonesa de fliperamas, lança o *Master System*, tendo a finalidade de competir com o *NES*. (NORMAND, RetroSpace:2001)

Em 1987, a empresa *NEC* lança o *PC Engine* um dos consoles mais amados de todos os tempos, foi lançado com a difícil tarefa de superar *Nintendo Famicom (NES)*, fazendo sucesso somente no Japão. (*Idem*, RetroSpace:2001)

Em 1989, a inexperiente *SNK*, resolve entrar no mercado de games e lança uma poderosa placa para arcade chamada *MVS*, permitindo a troca dos jogos através de cartuchos. A placa tinha o poder de produzir gráficos 2D muito bonitos para a época, fazendo muito sucesso, permitindo que a *SNK* lance seu primeiro console o *Neo Geo*, fazendo sucesso somente no Japão. (*Ibidem*, RetroSpace:2001)

Após esses fatos o mercado de Jogos começa a se erguer novamente e cada vez mais as grandes empresas de eletrônicos, principalmente as Japonesas começam a investir no segmento.

A *Sega* vendo o *Master System* perder para o *NES* e também para o *PC Engine*, resolve contra atacar lançando o *Mega Drive* primeiro console de 16 bits. (*Op. Cit*, RetroSpace:2001)

A *Nintendo* Alarmada com a popularidade do *PC Engine* e com o novíssimo console da *SEGA*, lança o sucessor do *NES* o *Super Famicom* mais conhecido como *SNES (Super Nintendo Entertainment System)*. (NORMAND, RetroSpace:2001)

Nessa mesma época a *Atari* sempre atrás da *Nintendo*, lançou o portátil *Lynx*, baseado no *Handy*, era o melhor da época com uma capacidade equiparada ao *SNES*, com isso a *Atari* conseguiu uma grande margem de lucro. (*Idem*, RetroSpace:2001)

Embalada com o sucesso do *PC Engine* no Japão e a popularização do *Game Boy* da *Nintendo*, a *NEC* lançou o portátil *PC Engine GT* em 1990. Uma versão compacta do *PC Engine*. (*Ibidem*, RetroSpace:2001)

Em 1991 a *Sega* lançou o *Game Gear*, uma cópia descarada do *Game Boy*, porém sendo superior ao *Game Boy*. (NORMAND, RetroSpace:2001)

No final de 1991 a *Philips* lançou o *CD-I*, um padrão de CD onde os usuários poderiam ter vídeos, imagens e sons em um disco de 8 cm. (*Idem*, RetroSpace:2001)

Nesse momento a *Nintendo* e a *SEGA* se tornam líderes do mercado e começam uma competição entre si lançando diversos periféricos para os seus consoles.

A fabricante de computadores canadense *Commodore*, vendo seu grande sucesso na Europa com computadores especializados para conteúdo multimídia e principalmente para jogos, lança em 1993 o *AMIGA CD32* baseado no computador *AMIGA 1200*. (NORMAND, RetroSpace:2001)

Em setembro de 1993, surgiu a *3DO Company* idéia de Trip Hawkins, fundador da *Electronic Arts*, criou um padrão de hardware único para jogos. (*Idem*, RetroSpace:2001)

Em outubro de 1993, surgiu o *LaserActive*, mais uma tentativa de definir um padrão para jogos. Formato desenvolvido em conjunto entre a *NEC*, *Sega* e *Pioneer*. (*Ibidem*, RetroSpace:2001)

Muito atrás no mercado, a *Atari* lança em 1993, o console chamado *Jaguar*. (Normand, RetroSpace:2001)

O início dos anos 90 foi a época de ouro dos Jogos Eletrônicos, os consumidores esqueceram o “*crash* dos Jogos” que aconteceu no início dos anos 80, e as empresas cada vez mais zelavam pela qualidade dos jogos que eram oferecidos.

### 3.3 A Terceira Era

Em 1994 a *SEGA* lançou o *SEGA Saturn* em substituição ao *Mega Drive*, foi o primeiro a utilizar ambientes 3D. (*Op. Cit.*, RetroSpace:2001)

Em Novembro de 1994 a *Nintendo* apresentou um novo portátil para revolucionar a forma de Jogar, embalado pela realidade virtual, o *Virtual Boy* novo portátil da *Nintendo* podia exibir gráficos 3D. (NORMAND, RetroSpace: 2001)

O *Virtual Boy* não fez sucesso, pois era preciso de uma área plana para se jogar, cansava a vista e não houve suporte por parte da *Nintendo*. (*Idem*, RetroSpace:2001)

Em dezembro de 1994 a *Sony* lançou o *Playstation*, fazendo uma parceria com a *LSI* para o desenvolvimento dos chips para o novo console, para que ele pudesse competir com os consoles da época como *3DO*, *Jaguar* e *Saturn*. (*Ibidem*, RetroSpace:2001)

O *Playstation* impressionou o mundo com gráficos superiores e a ótima jogabilidade, tendo um grande suporte das *softhouses* devido à facilidade de desenvolvimento. (NORMAND, RetroSpace:2001)

Em 1995 a *Sega* lança o *Nomad* outro portátil, seu objetivo era ganhar do *Game Boy*, seu hardware era um clone do *Mega Drive*, possuía seis botões, uma tela matriz e um cabo para ligar em outro *Nomad*, era compatível com 99% dos jogos do *Mega Drive*. (*Idem*, RetroSpace:2001)

A *Nintendo* lança em 1996 o *Nintendo 64* console com 64 bits fazendo sucesso com as diversas Franquias criadas até o momento como *Mario* (Shigeru Miyamoto,1981), *Donkey Kong* (Shigeru Miyamoto,1981) e *Zelda* (Shigeru Miyamoto,1986). (*Ibidem*, RetroSpace:2001)

A *Nintendo*, a *SEGA* e a *Sony* lançam diversos periféricos alguns inovadores, encabeçada pela *Sony*. Foram lançados periféricos de armazenamento, dispositivos de vibração no controle entre outros. (NORMAND, RetroSpace:2001)

Em 1996 a *Sega* já percebia que o *Playstation* iria dominar o mercado e vendo que a *Nintendo* estava lançando o *N64*, bem mais poderoso e com franquias de renome como *Mario*, *Zelda* e *Pokémon* (Satoshi Tajiri, 1996), inicia o desenvolvimento de seu próximo console, então em Maio 1998 foi lançado o *DreamCast*, com o periférico *VMU* uma mistura de cartão de memória e um mini-videogame. (*Idem*, RetroSpace:2001)

Embalada com o sucesso do *Game Boy*, a *Tiger*, entra no mercado de portáteis lançando o *Game.com* em 1997. Possuía tela sensível ao toque, calendário, agenda e calculadora, acesso com Internet e envio de e-mails. (*Ibidem*, RetroSpace:2001)

A *SNK* vendo a *Nintendo* dominar o mercado de portáteis, lança o *Neo Geo Pocket* em 1998, utilizando o seu charme de jogos 2D e do velho *Neo Geo*, o portátil possuía 16 bits. (NORMAND, RetroSpace:2001)

No final de 1999 a *Sony* já soltava detalhes do sucessor do *Playstation*, o nome ficou sendo *Playstation 2* e prometia uma revolução no entretenimento doméstico. Então em 4 de março de 2000 ele é lançado no Japão, o console aceitava dois formatos de mídia CD e DVD.(Villiegas, 2000:14-15)

Na época a *Nintendo* lançava o *Game Boy Color* compatível com o *Game Boy*, e a *SNK* lança logo em seguida o *Neo Geo Pocket Color*, também compatível com o modelo anterior. (*Op. Cit* , RetroSpace:2001)

Em 2000 a *Nintendo* na feira de games *Nintendo SpaceWorld*, exhibe o *GameCube*. As grandes franquias da *Nintendo* como *Zelda*, *Super Metroid* e *Mario* ajudam o *GameCube* a

crescer nas vendas. Utilizando uma mídia menor que o DVD denominado de miniDVD feito exclusivamente para o console. (PANCHERI & MORATO, UOL jogos: 2005)

Nos Estados Unidos a *Microsoft* lança o *XBOX*, sendo esgotado em poucas semanas. (*Idem*, UOL Jogos: 2005)

Em 2003 a *Nintendo* lança o *Game Boy Advance SP*, uma nova versão do *Game Boy*, o lançamento foi um sucesso, não conseguindo atender a demanda a *Nintendo* em nota a Imprensa pediu desculpas para os consumidores. A *Nokia* lança no mesmo ano o *N-Gage* um celular-videogame, além de ser celular é possível escutar MP3 e jogar, em Novembro a *Sony* mostra imagens de seu portátil o *PSP (PlayStation Portable)* com a capacidade de rodar MP3, vídeos além dos jogos. (*Ibidem*, UOL jogos: 2005)

Em 2004 a *Nintendo* mostra novidades de seu novo portátil *Nintendo DS*, possuindo duas telas, as *softhouses* demonstraram interesse em desenvolver jogos para o portátil. No final do mesmo ano a *Sony* lança o *PSP*. (PANCHERI & MORATO, UOL jogos: 2005)

Na época atual os consoles que dominam o mercado são pertencentes a essas três empresas *Sony*, *Nintendo* e *Microsoft* depois do lançamento de seus consoles cada inovação que uma empresa lançava as outras duas lançavam seguidamente.

A próxima geração de games já começou com a *Microsoft* lançando em 2005 o *XBOX 360*, sucessor do *XBOX*, com grandes inovações como um disco rígido. A *Nintendo* anuncia o desenvolvimento do novo console com o nome de *Wii* e a *Sony* começa a mostrar demos de seu *Playstation 3*. (*Op. Cit*, UOL Jogos: 2005)

O *Playstation 3* foi lançado em 13 de novembro de 2006, nesse primeiro lote saíram alguns com defeito no Japão de não conseguirem ler os jogos de seus antecessores *PS2* e *PSone*, além de ser um dos mais caros. Um dos maiores medos da empresa é que a reserva de um milhão de consoles já produzidas sejam esgotadas rapidamente nos Estados Unidos. (Martin, YahooNoticias, 2006)

O *Nintendo Wii* foi lançado em 19 de novembro de 2006, trazendo como novidade o *Wiiote* controle remoto sensível ao movimento.

As três grandes empresas do mercado estão com estratégias de Marketing distintas.

A *Nintendo* está vendendo o seu console com um preço bem baixo sendo um dos mais baratos custando US\$250,00. A multinacional acredita que o novo controle sensível ao movimento gera uma interação com o jogo bem maior que existe até agora. O controle força a pessoa a jogar com o corpo e usa-lo como um taco de golfe. (MARTIN, YahooNoticias, 2006)

A Sony investe num poder de computação muito superior trazendo gráficos realistas e efeitos sonoros avançados, afirma-se que o processador fará a cabeça de todos. (MARTIN, YahooNoticias, 2006)

Já a Microsoft investe que o XBOX 360 é um potente centro de lazer e entretenimento da casa podendo reproduzir vídeos, músicas, além de ter a capacidade de se integrar com a câmera digital, computadores e a própria Internet. A Microsoft ainda aposta na tecnologia do *XBOX live*, onde pode-se baixar filmes, músicas, ou jogar via rede *on-line*. (*Idem*, YahooNoticias, 2006)

Essa nova geração de videogames será marcada pelo custo benefício a empresa que conseguir vender por um preço baixo, mas que seja equivalente ao que se espera do console irá começar com o pé direito.

## 4. Jogos Eletrônicos

Visto a história dos Jogos Eletrônicos e suas trajetórias, parece que é fácil desenvolver um jogo e começar a vender, porém é preciso ter uma grande base em conhecimentos específicos de multimídia e também de gestão de projetos, além de conhecer seu público alvo e o andamento do mercado.

### 4.1 Desenvolvimento de Jogos Eletrônicos

Segundo PERÚCIA *et. al.* (2005:22-25), as empresas desenvolvedoras de jogos têm a imagem de um ambiente de trabalho desorganizado, em que cada um trabalha do jeito que bem entende. O negócio de desenvolvimento de jogos não é levado a sério pelas pessoas e nem pelos próprios desenvolvedores, a imagem que se tem é um grupo de desenvolvedores enfiados em um ambiente passando horas programando sem planejamento ou objetivos claros, dando início a um processo conhecido como *code like hell*<sup>29</sup>.

Esse modelo no início funcionava até o momento em que a indústria começou a se profissionalizar, e esse paradigma teve que ser quebrado. Os *publishers*<sup>30</sup> exigiam prazos cada vez menores e orçamentos limitados. A competitividade começou e as equipes mais organizadas começaram a se destacar. (*Idem*, 25)

A melhor forma de se iniciar no desenvolvimento de um jogo é executar essas duas ações pensar e planejar. Qualquer mudança no jogo durante a fase de produção do jogo pode ser fatal chegando a finalizar o projeto. No entanto quanto mais tempo usar na fase de planejamento menor serão as chances de isso acontecer. (*Ibidem*, 23)

---

<sup>29</sup> “Programar como no inferno” ou “programar que nem um doido”

<sup>30</sup> Publicadores de jogos

### 4.1.1 Equipe de Desenvolvimento

Para PERÚCIA *et. al.* (2005:26), atualmente os Jogos Eletrônicos são como grandes produções de cinema com orçamentos de milhões de dólares. Para criar um jogo de última geração e que faça sucesso é preciso três anos e uma equipe de cerca de 20 a 50 pessoas com os seguintes perfis:

- Programadores;
- Artistas;
- Projetistas de níveis/fases;
- Projetistas de Jogos;
- Planejador de software;
- Arquiteto Chefe;
- Gerente de Projeto;
- Músicos e sonoplastas;
- Testadores;

Programadores: responsáveis pelo desenvolvimento do jogo. Geralmente vindo de cursos de Informática ou Ciência da Computação, eles implementam técnicas de computação gráfica, inteligência artificial, efeitos sonoros e interação para implementar o jogo. (PERÚCIA *et. al.*, 2005:26)

Artistas: responsáveis pelo Layout, são criadores dos objetos, personagens, ilustrações e animações. (*Idem*, 26)

Projetistas de níveis/fases: vindo das mais diversas áreas, são responsáveis pela estruturação dos níveis dos jogos estruturando seus desafios e surpresas. Dependendo do tamanho da empresa o projetista de níveis pode ficar responsável pela fase de design ou atuar em todas as fases. (*Ibidem*, 27)

Projetistas de Jogos: Conhecidos como *Game Designers*, por estarem envolvidos em quase todas as áreas da produção de um jogo são fundamentais durante a construção dos projetos. Possui a responsabilidade da elaboração e cumprimento do *design document*, documento contendo características e especificações do jogo, precisando manter a comunicação com todos os membros da equipe para o cumprimento do *design document*. (PERÚCIA *et. al.*, 2005:27)

Planejador de Software: conhecido como *Software Planner*, tem a tarefa de dividir o projeto desenvolvido pelo *game designer* em conjuntos de requisitos técnicos e estimar o tempo e esforço para implementação das características. (PERÚCIA *et. al.*, 2005:27)

Arquiteto-chefe: Conhecido como *Lead Architect*: tem a tarefa de trabalhar em conjunto com o *Software Planner* para produzir módulos especificados com base nos requisitos técnicos levantados pelo *Software Planner*. O *Lead Architect* é responsável pela arquitetura geral do projeto. (*Idem*, 2005:27)

Gerente de Projeto: Conhecido como *Project Manager*, possui a tarefa de balancear a carga de trabalho gerada pelo *Software Planner* e *Lead Architect* produzindo um cronograma com as tarefas de cada membro da equipe e após monitorar, cobrar essas tarefas. Se possível o *Project Manager* não deve estar envolvido com a parte operacional do desenvolvimento podendo ter uma ampla visão da situação. (*Ibidem*, 27)

Músicos e sonoplastas: vindos das áreas de arte e música são responsáveis pela trilha sonora, vozes e feitos sonoros. (PERÚCIA *et. al.*, 2005:27)

Testadores: incumbidos de testar o jogo, procurando falhas e erros (*bugs*), entram no processo de desenvolvimento quando o jogo está quase pronto na fase de *Beta Testing*, costuma-se usar pessoas que nunca jogaram o jogo ou que não participaram da fase de desenvolvimento. A equipe que está desenvolvendo o jogo fica “viciada” nos procedimentos corretos do jogo e não encontra erros importantes. (*Idem*, 27)

A Equipe de desenvolvimento de um jogo é grande, pois são diversas responsabilidades e o desenvolvimento de um jogo envolve várias áreas como audiovisuais e de informática, além de gestão para gerenciar o projeto.

#### **4.1.2 Ciclo de Desenvolvimento**

Conforme PERÚCIA *et. al.* (2005:28), o ciclo é iniciado com uma reunião em que é discutido todas as idéias de jogos expostas por todos os participantes. Cada idéia é analisada em diversos aspectos como originalidade, público-alvo, inovação, plataforma e possibilidades de mercado. São necessárias diversas dessas reuniões até que se defina um projeto de jogo para se investir. Escolhido o projeto inicia o processo de *game design*.

O *game design* é o processo onde as características principais do jogo como jogabilidade, controles, interfaces, personagens, armas, golpes, inimigos, fases são descritas. Durante essa fase é elaborado o *design document*, um documento que descreve todas as

características detalhadamente, funcionando como um roteiro de cinema. (PERÚCIA *et. al.*, 2005:28)

Durante a fase de *game design* os artistas começam a trabalhar na concepção dos personagens e na aparência visual do jogo. Nessa fase os programadores fazem a modelagem do software e implementam a estrutura básica de classes do produto. (*Idem*, 28)

Com o *design document* pronto o gerente de projeto tira uma idéia do tamanho do projeto e tempo para o seu desenvolvimento. Os diretores de arte e programação definem suas metas as equipes e passam os cronogramas ao gerente de projeto para que ele organize um cronograma geral com as metas de cada equipe e para cada integrante da equipe. (*Ibidem*, 28)

Em projetos pequenos, apenas o *design document* é necessário para os artistas e programadores a iniciarem desenvolver o projeto. Em projetos maiores e que possuem muitas fases e missões, é necessário que cada fase tenha um planejamento para que programadores e artistas tenham a clara idéia de como irão trabalhar. (PERÚCIA *et. al.*, 2005:28)

O *level design* é um mapa geral com os desafios que o jogador deve cumprir para completar a fase. Serve de referência para que os artistas trabalhem na criação dos cenários de cada fase. Enquanto não estiver pronto o documento os artistas e programadores trabalham em cima de características mais gerais do jogo. Com o documento concluído, eles começam a dedicar na produção das fases. (*Idem*, 28)

Dependendo do tamanho da empresa são geradas versões intermediárias a cada semana ou mês, permitindo ao gerente de projeto acompanhar a evolução do jogo e possa cobrar as metas de cada um. Essas versões possibilitam a detecção de *bugs* que são corrigidos para a próxima versão, que acaba a aumentar a estabilidade do produto até a versão final. (*Ibidem*, 29)

O projeto evolui a cada versão gerada, até chegar a versão *Beta*. A versão *Beta* contém todas as fases do jogo e toda a interatividade. Quando a versão *Beta* é concluída inicia-se o ciclo de detecção de *bugs* e coleta de sugestões para melhora do produto (*Beta Testing*). Este processo gera refinamentos tanto na programação quanto na arte do jogo. Esses refinamentos continuam até a conclusão da versão *Gold* o produto final. (PERÚCIA *et. al.*, 2005:29)

Para desenvolver um jogo a equipe deve estar em sintonia para cumprir o cronograma geral imposto pelo gerente de projeto, seguindo o cronograma e o *game design document* corretamente o jogo será desenvolvido com perfeição.

### 4.1.3 Game Design

Para PERÚCIA *et. al.* (2005:29), antes de iniciar o desenvolvimento do jogo, é preciso que seu projeto esteja bem definido. Para isso existe o *game design*. Existem várias definições para *game design*, a principal é que o *game design* determina a jogabilidade, as escolhas que o jogador terá dentro do jogo e as variações que suas escolhas podem fazer ao resto do jogo. Inclui também o que faz o jogador perder ou ganhar, como ele vai controlar o jogo e quais informações serão exibidas para ele, em geral o *game design* descreve cada detalhe de como funcionará o jogo.

A seguir uma lista dos principais conceitos que o *game design* deve abordar:

- Idéia;
- Rascunhos do jogo;
- Detalhamento do jogo;
- *Game Design Document*.

**Idéia:** A idéia de um Jogo surge de um pequeno conceito que deve ser expandido com técnicas de *brainstorm*<sup>31</sup>. Muitas vezes boas idéias surgem de pensamentos que no momento parecem ridículos. Por esse motivo, deve-se evitar jogar fora uma idéia antes de ela estiver bem madura. Questões como “Qual o objetivo do Jogo?”, “O que o jogador terá de fazer”, “Como ele vai fazer?”, “O que tornará este jogo divertido?” podem facilitar o processo de desenvolvimento das idéias. (PERÚCIA *et. al.*, 2005:29)

**Rascunhos do jogo:** Criar desenhos que mostrem algumas fases, personagens e itens com pequenas descrições para cada um. Faça alguns rascunhos de telas, menus, fluxo de telas. Nesta fase é interessante realizar testes de jogabilidade, que podem ser feitos sem nenhuma linha de código, dependendo do tipo do jogo, como um jogo de *Puzzle*<sup>32</sup> por exemplo, o *Tetris* (1985), pode-se usar pedaços de papel que representam as peças do jogo. (*Idem*, 30)

**Detalhamento do jogo:** Nesse momento o jogo fica mais detalhado e complexo. É necessário pensar em cada detalhe importante e escrever algo a esse respeito. O desenvolvedor é soberano ao fazer um jogo, ou seja, se não programar um detalhe que foi definido pelo *game design* ele não acontecerá e vice-versa. (*Ibidem*, 32)

**Game Design Document:** Agora o próximo passo é juntar todos os conceitos em um grande documento conhecido como *design document*, ele é como um *script* de um filme, que informa todos os detalhes do jogo. Escrever um *design document* é muito trabalhoso, pois se

---

<sup>31</sup> “tempestade de idéias”

<sup>32</sup> Quebra-cabeça

deve detalhar todo o jogo, porém, é muito útil para repensar decisões já tomadas, validar alguns conceitos e suprimir ou adicionar regras. O jogo pode ser visualizado com antecedência. Esse documento é muito exigido pelos *publishers* para analisar um “jogo demo”. Assim eles podem ter idéia de como será o produto final. Normalmente estes documentos constam o cronograma, as metas e o orçamento para o projeto. (PERÚCIA *et. al.*, 2005:32-33)

Game design é a fase mais importante do desenvolvimento de um jogo nela cria-se o *game design document*, nessa fase é discutido diversos pontos do projeto para ser posto em prático logo após.

#### 4.1.4 Regras básicas para um bom jogo

Para PERÚCIA *et. al.* (2005:34), bons gráficos e sons não são suficientes para um jogo. Muitos jogos com recursos de última geração não venderam bem. Outros não vendem por falta de marketing ou por não agradarem o público-alvo. Conhecer o jogador é fundamental para o sucesso de um jogo.

O segredo é um bom *game design* e, acima de tudo, o jogo deve ser consistente e divertido. Para isso existem algumas regras básicas para criar um bom jogo:

- Comece com uma boa idéia;
- Escreva o *design* no papel;
- Comece pequeno;
- Cuidar do público;
- Use uma nova idéia;
- Seja flexível;
- Projete seu jogo;
- Pense em séries;
- Conteúdo é tudo;
- Dê objetivo ao jogador.

Comece com uma boa idéia: Pense no jogo, fazendo *brainstorms* com base nele, criando com coerência e lógica. Mostrem a idéia para os jogadores. Não desanimam com as críticas o objetivo é coletar novas idéias que podem ser aplicadas ao jogo caso necessário. (PERÚCIA *et. al.*, 2005:34)

Escreva o *design* no papel: Colocar a idéia no papel, descrevendo elementos, regras, alguns desenhos, fluxos de telas, *storyboards*<sup>33</sup> etc. Fica mais fácil perceber se o jogo é bom ou ruim se pode ser melhorado, se é viável. Muitos dos jogos não saíram do papel. (PERÚCIA *et. al.*, 2005:35)

Comece pequeno: Não criar o que não conhece, ou nem sabe se é possível fazer. Optar pelo simples. Muitas empresas entram em falência por terem problemas de concepção e desenvolvimento em projetos muito audaciosos. (*Idem*, 2005:36)

Cuidar do público: Desenvolver o jogo para entreter o público-alvo. Fazendo o jogo encanta-los, invocando as emoções deles. Não agradar a todos e sim o público-alvo. (*Ibidem*, 36)

Use uma nova idéia: Desenvolver uma idéia própria e criativa. Inspirar em outros jogos sem copiá-los. Adicione ingredientes pessoais, ter idéias mesmo que sejam ridículas num primeiro momento. (PERÚCIA *et. al.*, 2005:36)

Seja flexível: Quando está desenvolvendo um jogo, pode ser que ele não funcione do jeito que foi planejado. Sistema gráfico atual não suporta os efeitos ou falta de tempo ou recursos para implementá-lo. Adaptar o jogo a realidade. (*Idem*, 36)

Projete seu jogo: Pensar em que tipo de tecnologia estará disponível no momento que o jogo seja lançado. Não inserir recursos de última geração, pois ninguém terá acesso. Fazer o jogo para a plataforma padrão de mercado quando o jogo for lançado. (*Ibidem*, 36)

Pense em séries: Guardar idéias para seqüências e expansões. Ter em mente outras versões dos jogos. Podendo reutilizar código e arte para as próximas versões. Projete o jogo para ter expansões futuras. (PERÚCIA *et. al.*, 2005:36)

Conteúdo é tudo: Oferecer gráficos, sons, jogabilidade. Ter certeza que o jogo é divertido. (*Idem*, 36)

Dê objetivo ao jogador: Jogos sem objetivos são odiáveis. Dando sentido ao jogo, para que o jogador saiba para onde está indo. Premiar o jogador para cada objetivo conquistado. (*Ibidem*, 36)

Essas são regras básicas para que qualquer jogo consiga sucesso, se seguidas a risca o jogos será um grande sucesso entre o seu público chegando a se tornar mais tarde um clássico da área.

---

<sup>33</sup> Seqüências de cenas cinematográficas

#### 4.1.5 O que os jogadores querem e esperam

Conforme PERÚCIA *et. al.* (2005:37), os desenvolvedores de jogos devem pensar no público-alvo para isso eles devem abordar uma lista atingindo os que os jogadores querem e esperam de um jogo:

- Desafio;
- Socializar;
- Respeito;
- Experiência emocional;
- Fantasia;
- Entender os limites do mundo;
- Direção;
- Imersão;
- Falha;
- Não gostam de repetição;
- Não deixar o jogador trancado;
- Querem fazer e não ver.

Desafio: é a verdadeira essência do jogo. Os desafios geram experiências de aprendizado além, de emoções ao serem superados. (PERÚCIA *et. al.*, 2005:37)

Socializar: jogos em geral geram experiências sociais com amigos ou família. Videogames oferecem *singleplayer*<sup>34</sup> e *multiplayer*<sup>35</sup> e ambos são sociáveis. (*Idem*, 37)

Respeito: jogadores querem ganhar e obter, com o resultado disso, respeito. Quando estão entre os melhores do ranking de um jogo, sentem-se orgulhosos. Isso gera disputas acirradas em torno dos jogos. (*Ibidem*, 37)

Experiência emocional: Todos procuram um tipo de emoção. Adrenalina em *Quake* (*Id Software*,1996), Suspense e medo em *Resident Evil* (*CAPCOM*, 1996), Heroísmo em Mario. (PERÚCIA *et. al.*, 2005:38)

Fantasia: Os jogadores querem escapar do mundo atual para uma realidade diferente. O jogador poderá voar, atirar, mergulhar, matar alienígenas. Podendo praticar ações proibidas no nosso mundo, como em *Grand Theft Auto* (*Rokstar North*,1997), em que se pode ser um ladrão de carro e assassino. (*Ibidem*, 38)

---

<sup>34</sup> Somente uma pessoa joga até o final do jogo

<sup>35</sup> Vários jogadores participam simultaneamente do jogo.

Entender os limites do mundo: o que o jogador pode ou não pode fazer deve estar bem explícito. (PERÚCIA *et. al.*, 2005:38)

Direção: Sempre mostrar a direção ao jogador. Dando dicas para que ele saiba onde chegar. Dando objetivos e não deixando ele perdido. (*Idem*, 38)

Imersão:Fazer o jogador entrar no mundo do jogo em todos os sentidos. (*Ibidem*, 38)

Falha: Fazer os jogadores falharem algumas vezes. Usualmente quando o jogo é fácil o jogador para. Porém não pode exagerar na dificuldade, pois o jogador pode desistir de jogá-lo. (PERÚCIA *et. al.*, 2005:38)

Não gostam de repetição: Não oferecer desafios iguais ao jogador. Isso torna o jogo desagradável e cansativo. Não fazer o jogador retornar ao início do jogo se ele falhar em um ponto adiante. (*Idem*, 38)

Não deixar o jogador trancado: Não colocar um buraco em uma fase que impeça o jogador de continuar. (*Ibidem*, 38)

Querem fazer e não ver: Evitar colocar *cutscenes*<sup>36</sup> longos, tal ação tira a interatividade do jogo. Se for necessário permitir que os jogadores possam cancelar a sua execução. (PERÚCIA *et. al.*, 39)

Em complemento às regras de um bom jogo, se a equipe seguir as dicas do que um jogador quer de um jogo aumenta as possibilidades do jogo se tornar um grande clássico.

#### 4.1.6 Puzzles

Para PERÚCIA (2005,39), a utilização de *puzzles* é fundamental para qualquer jogo, porém seu uso deve ser bem balanceado, para não torná-los chatos e entediantes. Deve-se colocar quando necessário e tendo coerência com a temática do jogo.

*Myst* (Cyan, 1993) é um Grande *puzzle*, como *Indiana Jones and the Fate of Atlantis* (Lucas Arts, 1994), já em jogos como *Duke Nukem* (3D Realms, 1996), *Doom* (Id Software, 1993) e *Quake*, os *puzzles* estão na forma de interruptores que abrem portas equipamentos que devem ser usados em determinados momentos. O bom desse *puzzles* que estão inseridos no jogo tão claramente que o jogador nem percebe que está resolvendo-os. (*Idem*, 39)

Os quebra-cabeças num jogo é utilizado para que ele não fique maçante, tendo só um tipo de ação como sair atirando pra qualquer lado.

---

<sup>36</sup> Técnica utilizada para contar uma parte da história do jogo

## 4.2 Arquitetura de um Jogo

Conforme PERÚCIA (2005: 40), o processo de arquitetar começa logo após a etapa de *game design*. É um processo que proporciona uma visão de alto nível do *software*. Um jogo é considerado um *software*, então qualquer padrão de desenvolvimento para *softwares* se encaixa para o desenvolvimento de um jogo.

### 4.2.1 Reusabilidade

A palavra já diz tudo, deve-se arquitetar o software para que futuramente possa se aproveitar do código já desenvolvido. Para isso utiliza-se de bibliotecas que podem ser usadas em diversos projetos. (PERÚCIA *et. al.*, 2005:42)

Existem diversas maneiras de se utilizar isso, sendo com *libraries*<sup>37</sup> ou com o conceito de objetos COM, que é uma estrutura de módulos que funcionam em diversas aplicações, independente da linguagem de programação. Esta arquitetura permite a atualização das bibliotecas sem perder as funcionalidades antigas. O *DirectX* da *Microsoft* é um exemplo desta arquitetura. (*Idem*, 43)

Nos jogos essas bibliotecas são chamadas de *engines*, pois servem como um “motor”. Atualmente existem diversas *engines* que reúnem diversos recursos para quase todas as finalidades de um jogo completo como gráficos, sons, entradas, rede entre outros. Há *engines* específicas para física e detecção de colisão. (*Ibidem*, 43)

A utilização desta técnica reduz o tempo de desenvolvimento dos jogos seguintes, pois alguns códigos não precisam ser implementados, pois é aproveitado o código gerado do primeiro projeto.

#### 4.2.1.1 Engines

Segundo SANTEE (2005: 381), o termo mais conhecido por qualquer desenvolvedor de jogos é “*engine*”, conhecido como motor. Pode-se dizer que é o pilar sustentador de um jogo, e não um programa que faz jogos como o *3D Game Studio*.

Grande parte dos motores é constituída de código, geralmente armazenados em um arquivo DLL. Instruções como “desenhe o personagem”, “ilumine a superfície se está

---

<sup>37</sup> Conjunto de algoritmos com uma finalidade me comum agrupados

apontada para a luz” entre outras, então elas são chamadas e executadas no jogo. (SANTEE, 2005:381)

Eles são desenvolvidos em muitas vezes por linguagens de programação de alta performance, como C++. Os códigos das rotinas criadas são compiladas criando *DLL's* ou *LIB's*, arquivos binários que armazenam todo o código compilado, para ser utilizado pelo jogo. (*Idem*, 381)

Existem também *engines* do tipo software, ou *framework*<sup>38</sup>, onde oferece aplicações com interface visual ou simples linguagens de programação/*scripting*<sup>39</sup> para a criação e edição dos eventos e efeitos. (*Ibidem*, 381)

Diversas características peculiares são encontradas ao analisarmos os diversificados tipos de *engines*. A maioria deles envolve programação de baixo nível, como *Half-Life 2* (*Valve Software*, 1998), é desenvolvida para executar apenas uma ou poucas rotinas. Processar o controle de física e colisão, renderização, animação facial e controle de rede são algumas das funções que as *engines* especializadas possuem. (SANTEE, 2005:382)

*Engines* é outro conceito de reusabilidade, porém totalmente voltada para o desenvolvimento de jogos, uma das principais características é a diminuição do código por exemplo, onde é preciso mais de cem linhas para desenhar uma figura na tela, pode ser feito em cinco linhas.

## 4.2.2 Arquetando o Jogo

Segundo PERÚCIA *et. al.* (2005:45), arquetar um jogo é o processo mais complicado no desenvolvimento, requer que projetistas modelem o sistema por completo. Envolvendo estruturas e fluxo de dados, define interações entre todos os componentes do sistema. A principal utilidade é para que toda a equipe tenha uma visão do que está sendo desenvolvido. Ela é dividida em duas partes: abstração de hardware e abstração de software, ambas as camadas são divididas em subsistemas.

### 4.2.2.1 Abstração de *Hardware*

A idéia é a de facilitar o uso de recursos de acesso ao *hardware* como gráficos, dispositivos de entrada (teclado, mouse, *joystick*<sup>40</sup>), som, rede, etc., ou seja, tudo que esteja

---

<sup>38</sup> Estrutura de suporte definida, onde um projeto de software pode ser trabalhado em cima

<sup>39</sup> Linguagens de programação interpretadas

relacionado ao *hardware* da plataforma de desenvolvimento. Este conceito é muito útil, pelo motivo em que o desenvolvedor não precisa se preocupar com as configurações e condições do hardware, utilizando apenas de comandos intuitivos como *windowed* o jogo será exibido em uma janela no computador ou *fullscreen* o jogo será exibido em tela cheia, a camada de abstração fará as configurações para esses dois modos liberando o programador de realizar outros ajustes. (PERÚCIA *et. al.*, 2005: 46)

O isolamento do hardware na camada de abstração permite as seguintes vantagens para a arquitetura do jogo:

- Isola especificidades de hardware;
- Facilita a atualização da camada de *hardware* sem interferir no código do jogo;
- Facilita portar o jogo para outra plataforma;
- Permite o desenvolvimento de componentes reusáveis de fácil uso e configuração;
- Permite ao desenvolvedor preocupar-se somente com o jogo;
- Torna o código do jogo mais limpo e consistente.

A abstração de Hardware facilita e muito o desenvolvedor, tirando dele o peso de se preocupar no *hardware* em que o jogo está sendo colocado.

#### **4.2.2.2 Abstração do jogo**

A abstração do jogo serve para ter uma visão global de como funcionará o jogo, com os seus subsistemas e interações, sem se preocupar com o *hardware*. Não existe um padrão para o desenho dessa abstração o desenvolvedor fará da melhor forma em que o restante da equipe de desenvolvimento compreenda. (PERÚCIA *et. al.*, 46-47)

Em um nível básico de abstração, os jogos apresentam os seguintes subsistemas:

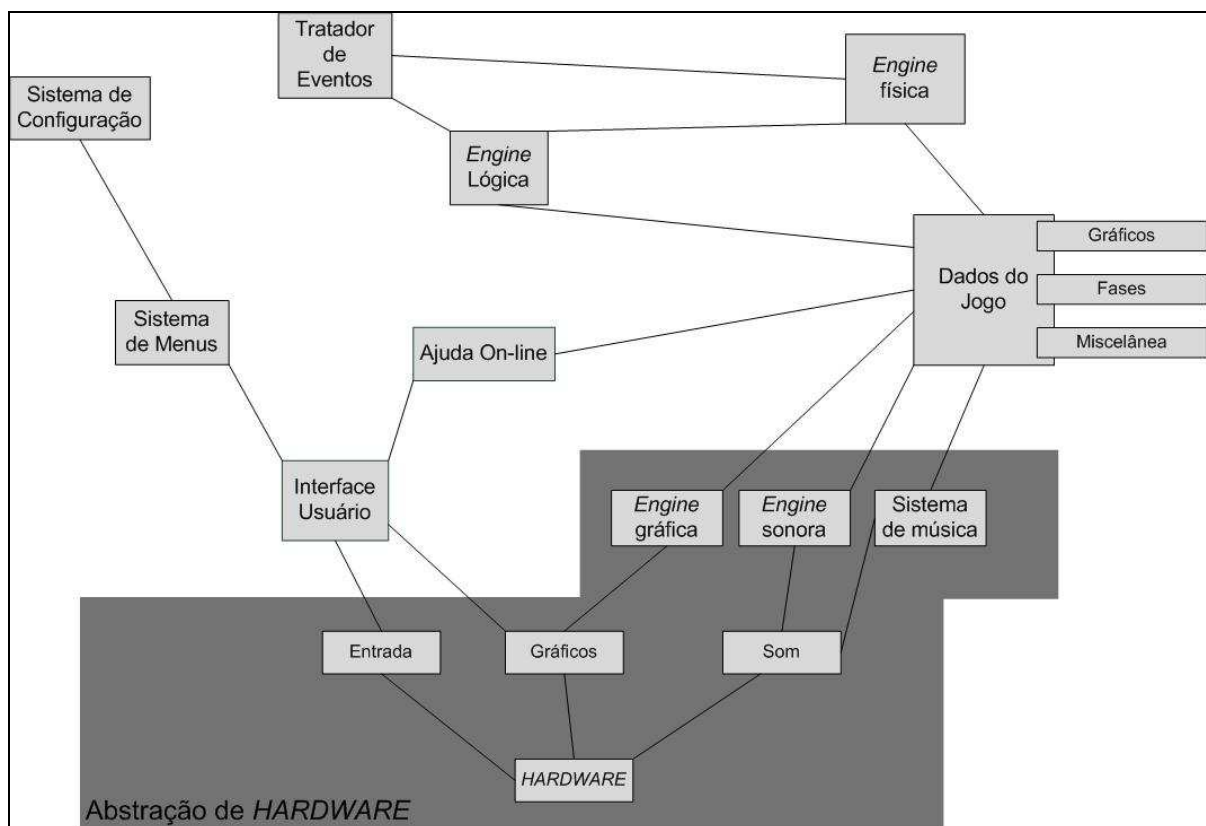
- Interface (menus, botões, controles etc);
- Tratador de eventos (identificar uma tecla pressionada, um botão abaixado etc);
- Gerenciador de dados (gráficos, fases e miscelâneas);
- Gerenciador de Física (movimento, velocidade, colisão, gravidade etc);
- *Engine* gráfica (exibição de imagens, rotação, efeitos especiais etc);
- *Engine* sonora (efeitos sonoros);
- *Engine* lógica (o coração do jogo);

---

<sup>40</sup> Dispositivo de entrada utilizado somente para jogos

- Camadas de abstração de hardware (interfaces com gráficos, sons e entradas do hardware);
- Sistema de configuração do jogo (opções de jogo, salvamento de jogo etc.);
- Sistema de menus;
- Sistema de ajuda;
- Sistema de música.

A figura abaixo exemplifica a arquitetura básica de um jogo, demonstrando a integração entre a abstração do jogo com a de *hardware*. Pode-se adicionar ou retirar itens dependendo do tipo de jogo.



**Figura 1: Arquitetura básica de um jogo**

FONTE: Adaptado de PERÚCIA *et. al.* (2005:47)

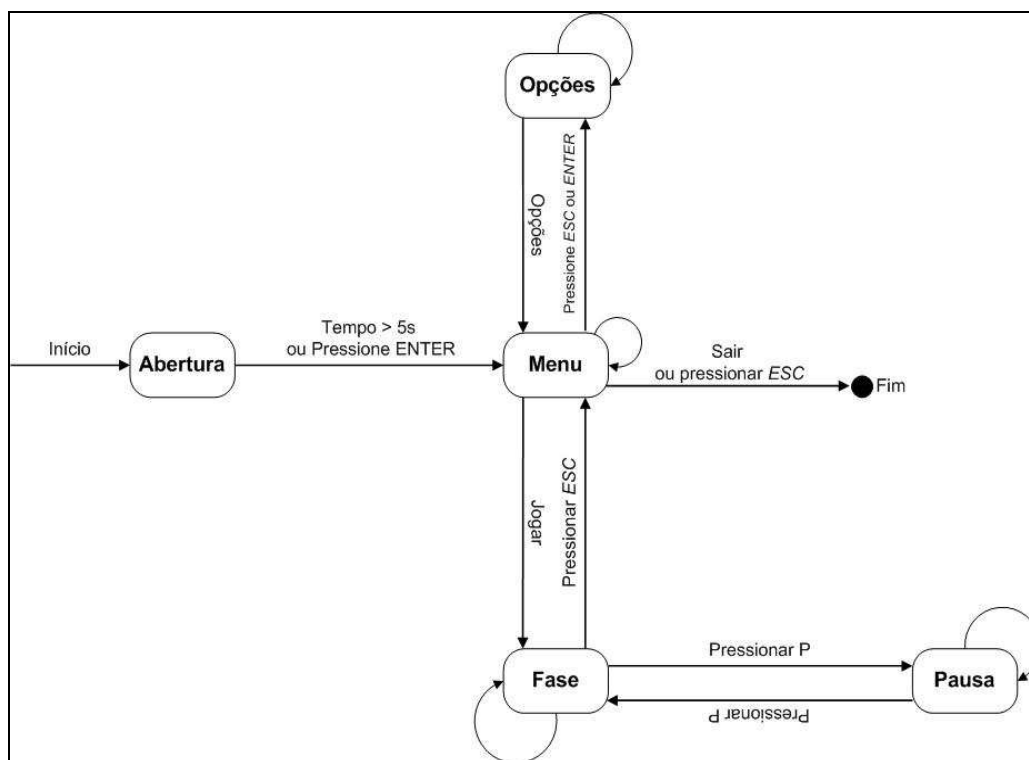
A Abstração do jogo facilita o desenvolvedor. Quanto mais completa e organizada for a abstração mais fácil fica o desenvolvimento do jogo.

### 4.2.3 Máquinas de Estados Finitos (FSM)

Conforme PERÚCIA *et. al.* (2005:47), máquinas de estados finitos, ou *Finit State Machines* (FSM), são ferramentas importantes para o planejamento de um jogo. O objetivo principal é modelar o funcionamento do jogo antes da codificação. Por isso você já deve ter uma visão bem clara do funcionamento do jogo.

Precisa pensar de forma modular. Por exemplo, um jogo vai ter uma abertura, um menu com as opções Jogar, Opções e Sair e uma fase. Considerar cada item como um estado do jogo. (PERÚCIA *et. al.*, 2005:48)

A figura abaixo é exemplo de uma máquina de estados bem simplificada.



**Figura 2: Exemplo de uma Máquina de Estados Finitos**

FONTE: Adaptado de PERÚCIA *et. al.* (2005:48)

De acordo com a máquina observa-se a apresentação de cinco estados de máquinas:

- Estado Abertura;
- Estado Menu;
- Estado Opções;
- Estado Fase;
- Estado Pausa;

Estado Abertura: estado inicial do jogo, normalmente será exibido o nome do jogo e o produtor. De acordo com a regra a tela permanecerá durante cinco segundos ou até que o jogador pressione *ENTER*. (PERÚCIA *et. al.*,2005:48)

Estado Menu: representa o menu do jogo oferecendo três opções simulando três botões, se o jogador pressionar “Jogar”, a máquina irá para o estado fase, se pressionar “Opções”, irá para o estado Opções, caso escolha “Sair”, a máquina de estados termina e encerra o jogo. (*Idem*, 48)

Estado Opções: mantém o jogo em uma tela de opções até o jogador pressionar a tecla *ESC* ou *ENTER*. Quando isso acontecer, o jogo retornará para o estado de Menu. (*Ibidem*, 48)

Estado Fase: Onde o jogo realmente acontece. Caso seja pressionada a tecla P o jogo será pausado. Caso o jogador pressione a tecla *ESC*, a máquina retornará para o estado Menu. (PERÚCIA *et. al.*,2005:48)

Para PERÚCIA *et al.* (2005:49), um jogo não é só isso, o que se fez aqui foi uma demonstração em alto nível do jogo. O que se faz após isso é expandir cada estado macro em máquinas de estados menores.

Geralmente um jogo contém inúmeras máquinas de estados finitos e são muito úteis, principalmente pela facilidade de codificação sendo uma ferramenta muito poderosa, podendo ser utilizada para o planejamento e desenvolvimento do jogo. (*Idem*, 29)

As máquinas de estados finitos servem para a arquitetura do jogo, em que o desenvolvedor desenhará em diagramas os acontecimentos (estados) do jogo, vendo a interligação entre eles.

### **4.3 Programação de um Jogo**

Serão apresentadas os principais conceitos para o desenvolvimento de um jogo de duas dimensões ou jogo 2D.

#### **4.3.1 Gráficos**

Segundo PERÚCIA *et. al.* (2005:63), antigamente o processamento de vídeo era feito pelo próprio processador do computador. Hoje, o processo foi mudado passando a ser a placa de vídeo que processa o vídeo deixando o CPU mais “livre” para outras tarefas.

Monitores possuem diversos modos de vídeo. Eles são compostos por três atributos:

- Resolução;

- Formato;
- Taxa de atualização.

Resolução (*resolution*): define a quantidade de *pixels* na tela esse número é obtido pela multiplicação entre largura e altura, quanto mais *pixels* possuir, melhor será a visualização de imagens, porém maior será o consumo de memória e processamento. (PERÚCIA *et. al.*, 2005:65)

Formato (*format*): define a profundidade de cores (*color depth* ou *bit depth*) de um pixel, ou seja a quantidade de bits que o pixel usará. Quanto maior a quantidade de cores, maior será o processamento. (*Idem*, 65)

Frequência (*refresh rate*): número de vezes por segundo (*Hertz*) que é atualizado o monitor, ficando entre uma faixa de 60 a 85 *Hertz*. (*Ibidem*, 65)

As cores dos jogos eletrônicos são formadas por uma combinação de três cores vermelho, verde e azul, variando as intensidades dessas três cores obtém-se todas as cores. Os computadores identificam uma cor dando um valor numérico chamado de tripla RGB(*Red, Green, Blue*). Essa tripla é um conjunto de três valores de 8 bits onde cada valor representa a intensidade de sua cor correspondente os valores variam de RGB(0,0,0) a RGB(255,255,255). (PERÚCIA *et. al.*, 2005:65)

#### 4.3.1.1 Superfícies

Segundo PERÚCIA *et. al.* (2005:67), superfícies são regiões de memória que guardam dados de imagens lidas de arquivos *bitmaps*<sup>41</sup>. Podem ser alocadas na memória da placa de vídeo ou na memória do sistema, sendo estas mais lentas, pois são processadas pelo processador. A superfície não precisa ter imagens prontas para serem visualizadas: podem-se guardar animações, *tiles*, fontes ou pedaços de um cenário. A superfície é gerenciada como uma “folha” virtual em que a aplicação embaralha, copia, combina e rearranja as imagens com outras superfícies compondo assim o ambiente visual do jogo, produzindo animações.

As superfícies são divididas em três categorias:

- Superfície primária;
- Superfície secundária;
- Superfície *offscreen*.

---

<sup>41</sup> Mapa de bits

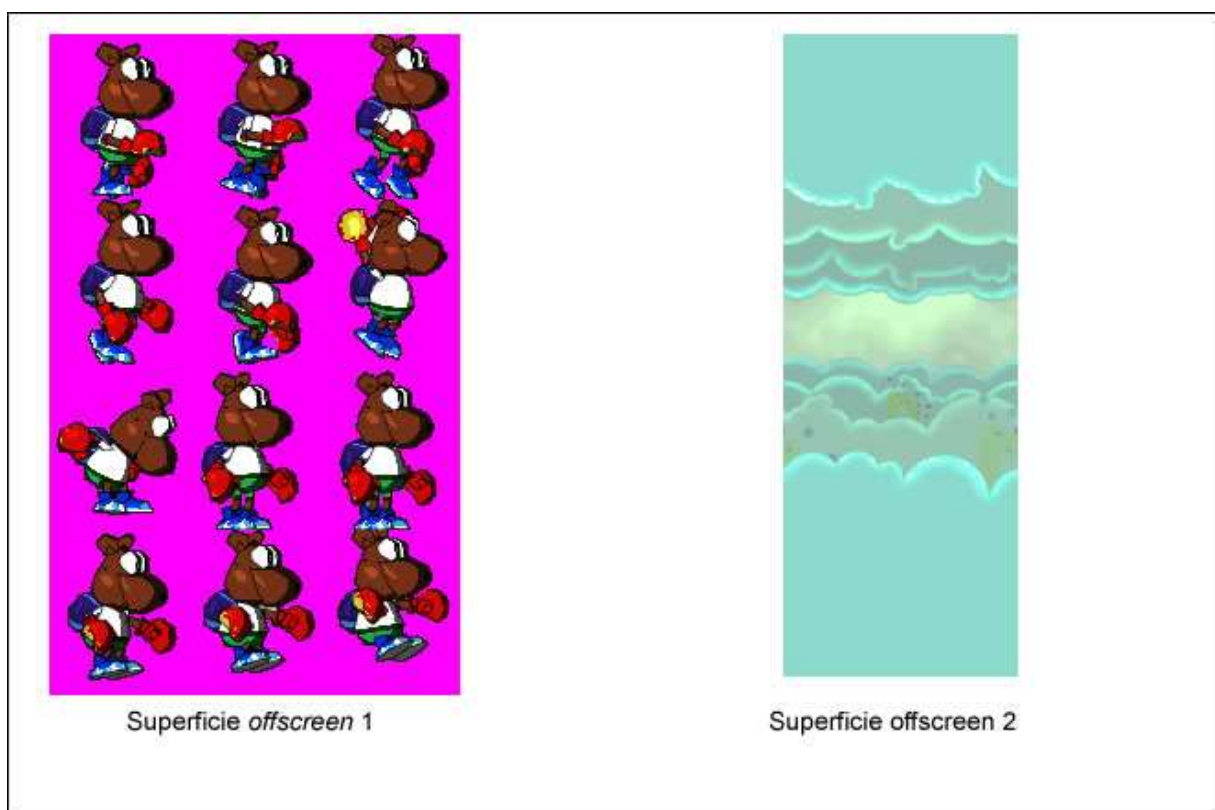
Superfície primária: conhecida como *front buffer*, é a área visível do monitor, gerada diretamente na placa de vídeo. Qualquer dado escrito nela aparecerá na tela imediatamente. (PERÚCIA *et. al.*, 2005:67)

Superfície secundária: conhecida como *back buffer*, utilizada da mesma forma que a primária. Sendo utilizada como espaço de trabalho para geração de imagens, que posteriormente são enviadas para a superfície primária. (*Idem*, 67)

Superfície *offscreen*: superfície usada para guardar imagens de *sprites*, itens do jogo ou outros objetos. Os dados são arranjados e copiados para o *back buffer* e posteriormente mostrado na tela. Um jogo pode ter mais de uma superfície *offscreen* podendo assumir um determinado tamanho de 8, 16, 32 bits. (*Ibidem*, 67)

Criam-se as superfícies em ordem hierárquica primeira é a primária após a secundária e em seguida cria-se as superfícies *offscreen*. (PERÚCIA *et. al.*, 2005:67)

A figura abaixo demonstra duas superfícies *offscreen*, pois as superfícies *back buffer* e *front buffer* são a própria tela do jogo, não precisa ser demonstrada aqui.



**Figura 3: Exemplo de superfície *offscreen***

FONTE: Adaptado de PERÚCIA *et. al.*, 2005:67

A superfície é um recurso gráfico em que o desenvolvedor desenhará o quadro do jogo e depois enviará para a tela, a própria tela é uma superfície, ou seja, superfície é uma camada

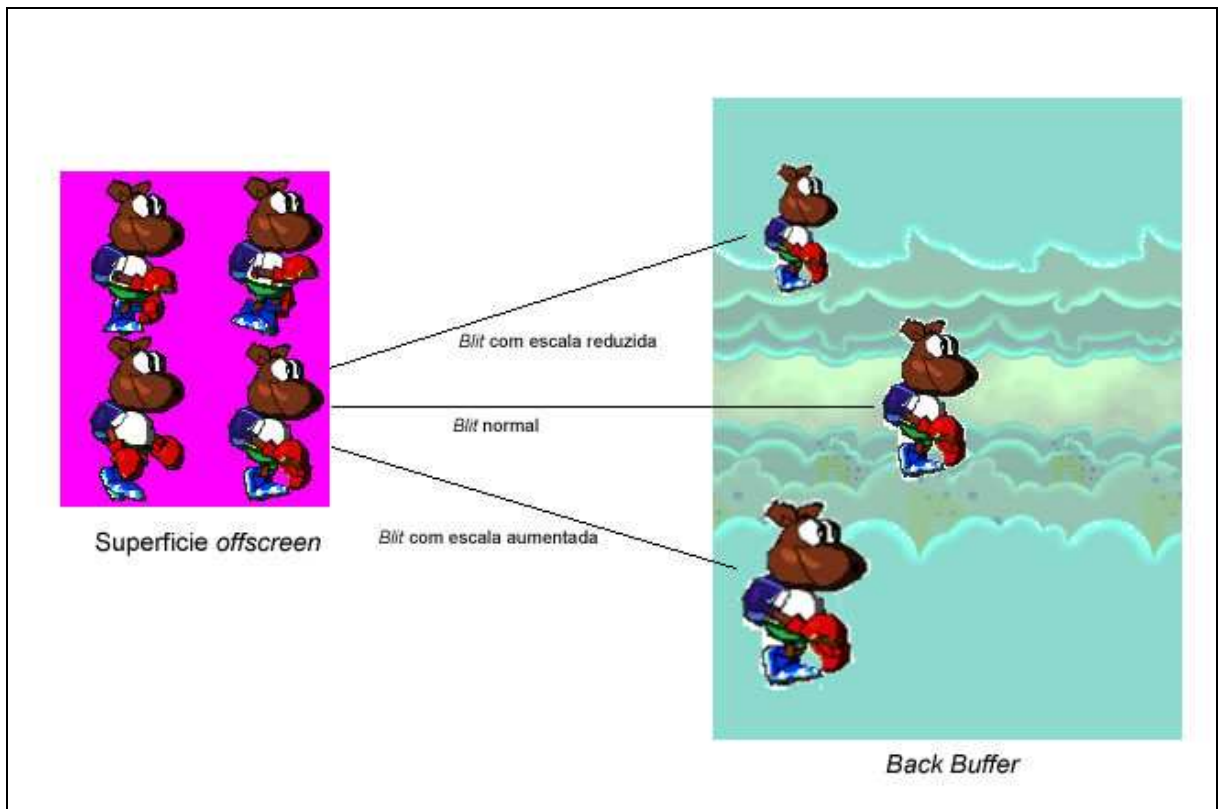
do jogo em que são guardados os arquivos gráficos para serem usados ou exibidos posteriormente.

#### 4.3.1.2 Função *Blit*

Para PERÚCIA *et. al.* (2005:68), a função mais importante do processamento gráfico. Sendo uma cópia de dados de uma superfície para outra, em que se define o *frame* de origem e o *frame* de destino que será o local em que será tirado uma cópia e o local para onde será gravada a cópia respectivamente.

Em funções *Blits* às vezes utiliza-se parâmetros nulos, sem valor nenhum de endereço, indicando a cópia de toda a superfície. Se o tamanho dos *frames* de origem e de destino for diferente será feita uma escala permitindo a imagem aumentar ou diminuir. (PERÚCIA *et. al.*, 2005:68)

A figura abaixo demonstra a função *Blit* com a escala aumentada e escala reduzida, além de sem escala.



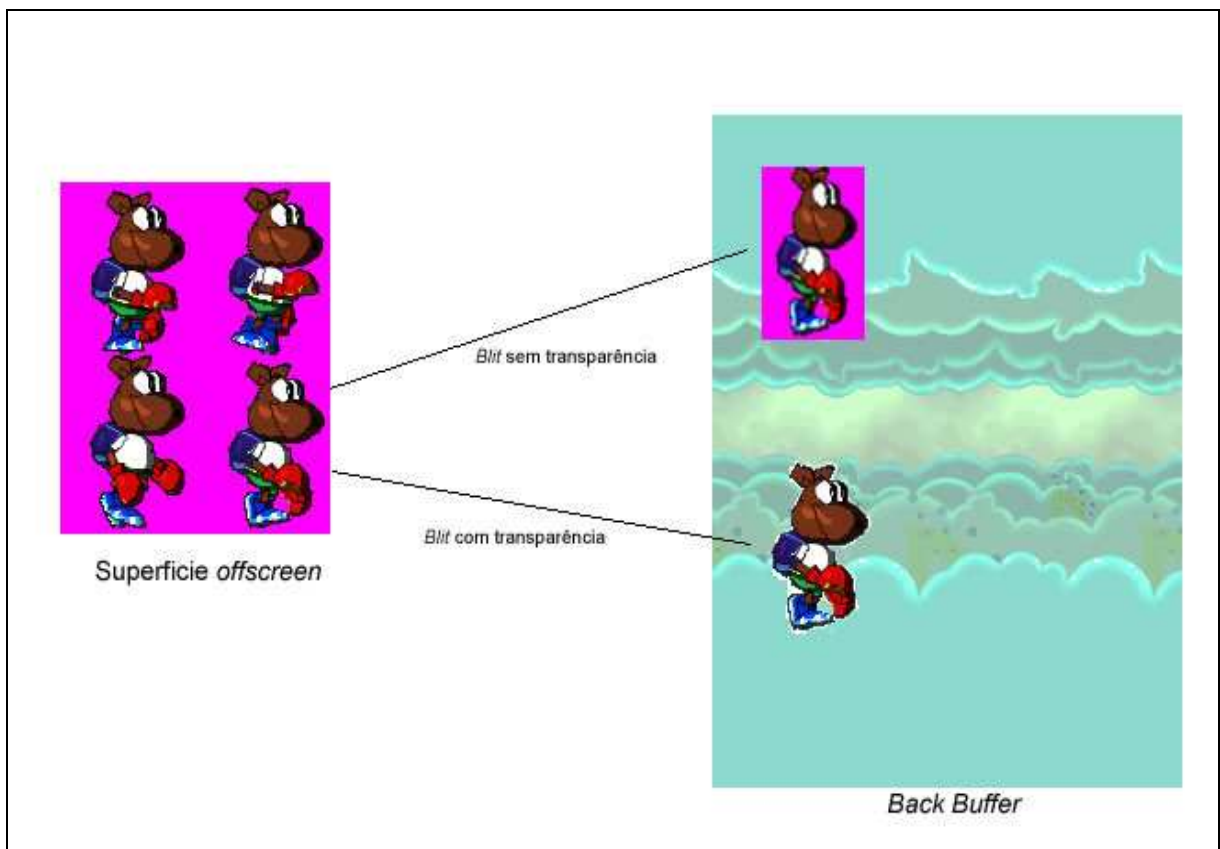
**Figura 4: Exemplo da função *Blit***  
 FONTE: Adaptado de PERÚCIA *et. al.*, 2005:68-69

A função *Blit* é a mais importante para jogos 2D, é com ela que o desenvolvedor desenhará cada quadro do jogo.

### 4.3.1.3 Transparência

Segundo PERÚCIA *et. al.* (2005:69), uma das principais funções de processamento gráfico de um jogo é a transparência. Ele é utilizado quando é necessário exibir imagens que possuem regiões transparentes. Para isso é definida uma cor denominada como *color key* ou “cor-chave”. Todos os *pixels* dessa cor não serão copiados para o quadro de destino dando o efeito de transparência.

A figura abaixo demonstra a função *Blit* utilizando o recurso de transparência, perceba que a cor-chave é a rosa.



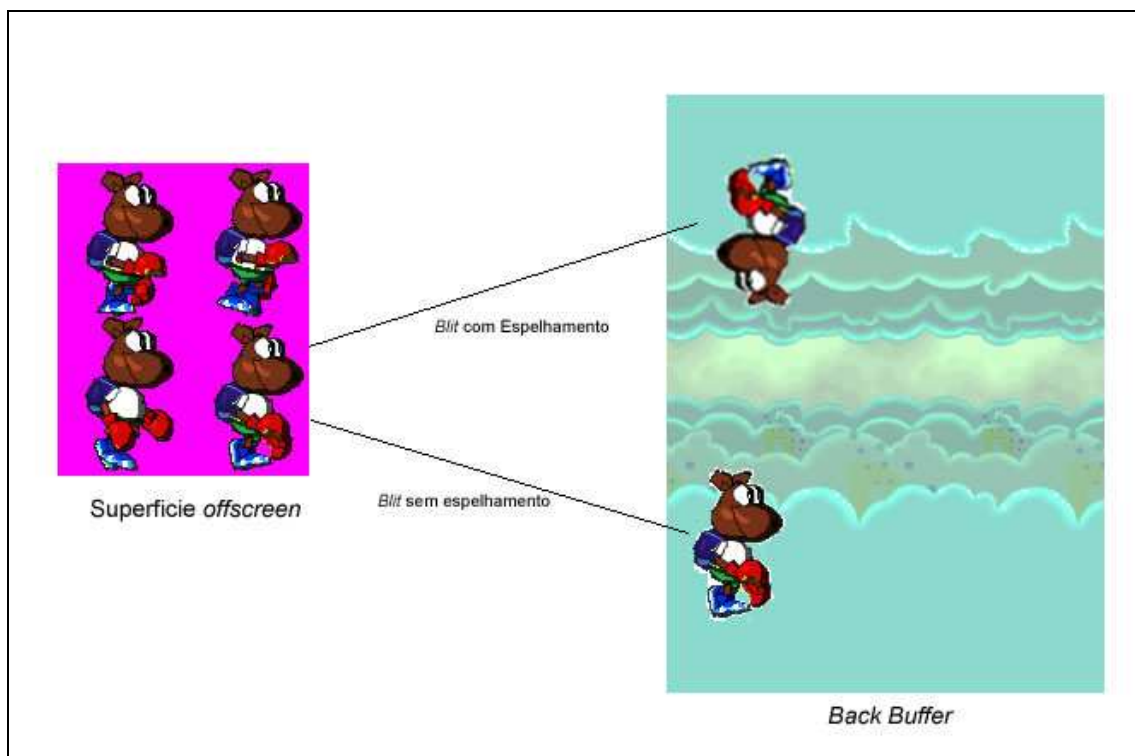
**Figura 5: Exemplo de Transparência**  
FONTE: Adaptado de PERÚCIA *et. al.*, 2005:69

A Transparência é uma função a mais da *Blit* para que o desenvolvedor possa trabalhar com imagens que não tem o fundo transparente, normalmente escolhe-se como cor de fundo a cor rosa.

#### 4.3.1.4 Espelhamento

Para PERÚCIA *et. al.* (2005:70), espelhamento ou *mirroring*, é uma técnica de “espelhar” a imagem. Esse efeito é obtido pela inversão dos *pixels* da imagem a ser desenhada em um ou dois eixos (X e Y). Essa técnica pode ser aplicada junto com a função *blit* passando um parâmetro indicando o espelhamento.

A figura abaixo demonstra a função *Blit* com o recurso de espelhamento.



**Figura 6: Exemplo de *Mirroring***  
FONTE: Adaptado de Perúcia *et. al.*, 2005:70

#### 4.3.1.5 Técnicas de Animação

Segundo PERÚCIA *et. al.* (2005:70-71), existem duas principais técnicas de animação para jogos a *page flipping* e *double buffering* ambas utilizam a superfície *back buffer* para desenhar as imagens da animação, pois se desenharmos diretamente na superfície *front buffer*, ocorrerá um efeito indesejado chamado de *flicker*.

#### 4.3.1.5.1 Efeito *flicker*

Conforme PERÚCIA *et. al.* (2005:71), o monitor é um tubo de raios catódicos, ou *cathode-ray tube*(CRT), que contem um canhão de elétrons que reflete tudo que está escrito na superfície primária.

Quando o canhão de elétrons chega ao fim de uma linha, ele desliga e segue em direção ao início da próxima linha para recomençar a exibir a cena. Este intervalo de tempo é chamado de *Horizontal Blank* ou *HBlank*. Quando o canhão chega ao fim da última linha ele retorna para o início da primeira linha para recomençar a operação o intervalo de tempo gerado nessa operação é chamado de *Vertical Blank* ou *VBlank*. (*Idem*, 71)

O efeito *flicker* ocorre justamente porque os dados, presentes na superfície primária são atualizados com uma velocidade maior da do canhão. Ocorrendo assim o canhão de elétrons começa a desenhar um quadro de uma animação de um objeto e quando estiver terminando desenhar outro quadro, desenhando na tela de forma assíncrona. (*Ibidem*, 71)

A figura abaixo demonstra uma situação do efeito *flicker*.



**Figura 7: Exemplo do efeito *flicker***

FONTE: Adaptado de PERÚCIA *et. al.*, 2005:72

A solução para esse problema é fazer uma sincronização da superfície primária com o canhão do monitor, ou seja, a escrita na superfície primária deve ser feita somente quando o canhão parar de desenhar na tela. Esse momento será somente em intervalos *VBlank*. Isto garante que o *front buffer* será atualizado somente quando o monitor estiver parado e retornando para recomençar o desenho. (PERÚCIA *et. al.*, 2005:72)

Para essa solução temos duas técnicas a *Page Flipping* e a *Double Buffering*

#### 4.3.1.5.2 Page Flipping

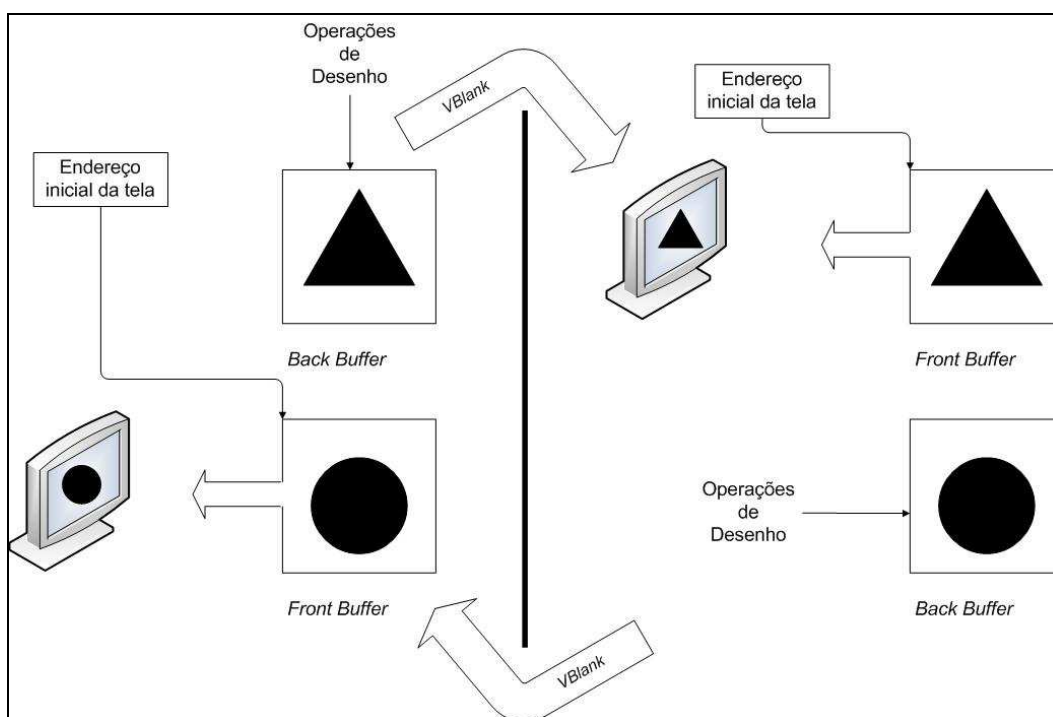
Conforme PERÚCIA *et. al.* (2005:73), essa técnica consiste dos seguintes passos:

- Desenhar as imagens do jogo no *back buffer*;
- Quando for gerado um *VBlank* do monitor, fazer o *back buffer* virar o *front buffer*;
- Repetir a primeira ação.

Desta forma, quando o canhão estiver em seu *VBlank*, o *front buffer* torna-se o *back buffer* e vice-versa, garantindo que o canhão tenha as imagens que não vão ser alterados Até o próximo *VBlank*. (*Idem*, 73)

A técnica é alterar o endereço de memória das duas superfícies alternadamente para que o canhão sempre pegue a superfície em que nada vai ser escrito até o próximo *VBlank*. Essa técnica é usada somente em modo tela cheia. (*Ibidem*, 73)

A figura abaixo esquematiza o funcionamento do *page flipping*.



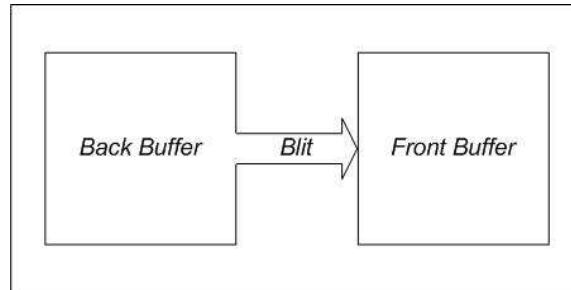
**Figura 8: Esquema do Page Flipping**  
 FONTE: Adaptado de PERÚCIA *et. al.*, 2005:73

#### 4.3.1.5.3 Double Buffering

Para PERÚCIA *et. al.* (2005:73), a técnica do *Double Buffering*, consiste em ter as duas superfícies, em que na *back buffer*, todas as imagens do jogo são desenhadas lá, E ao

final do laço do jogo, tudo é passado para o *front buffer* fazendo uma cópia de dados. Essa técnica permite a visualização no modo janela.

A figura abaixo esquematiza o funcionamento do *double buffering*.



**Figura 9: Esquema do *Double buffering***

FONTE: Adaptado de PERÚCIA et. al., 2005:74

#### 4.3.1.6 Paleta

Segundo PERÚCIA *et. al.* (2005:75), Paleta é um mecanismo utilizado pelos programadores para economizar memória. Em uma imagem sem paleta, o buffer de memória contém todos os dados dos *pixels* da imagem. Quanto maiores forem as dimensões da imagem e maior for o *bit depth*<sup>42</sup>, maior será a quantidade de memória utilizada para guardar a imagem.

Com uma imagem “paletizada” em vez do *buffer* conter os dados dos *pixels* da imagem ele possui índices da tripla RGB. Isto reduz o consumo de memória, porque em vez de serem gastos os 3 bytes em um *pixel* é gasto 1 *byte* correspondendo o índice da tripla na paleta. (*Idem*, 75)

Este método resume-se em reduzir o custo computacional. Porém hoje em dia com as máquinas atuais e periféricos gráficos bons, não é mais necessária a utilização desta técnica. (*Ibidem*, 75)

O recurso de paleta já está sendo deixado para trás, porém é uma ótima opção para reduzir o consumo de memória deixando mais leve o jogo, mesmo que seja por uma quantidade mínima.

<sup>42</sup> Números de bits para cada *pixel*

### 4.3.1.7 Formato de *pixel* (*pixel format*)

Conforme PERÚCIA *et. al.* (2005:76), cada *pixel* é representado pela tripla RGB (*red, green, blue*). O número de cores possíveis é determinado pelo número de bits de cada componente. Por exemplo, um RGB de 16 *bits* utiliza-se dos 16 *bits* para representar as cores possíveis sendo 5 *bits* para o vermelho, 6 *bits* para o verde e 5 *bits* para o azul.

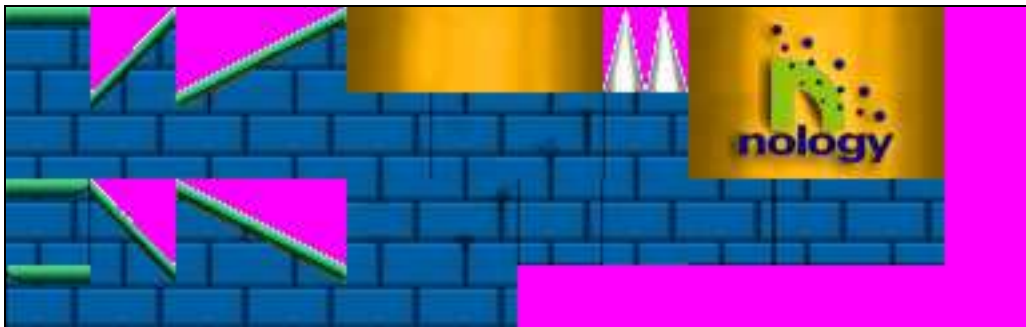
### 4.3.2 *Tiles, Bricks e Layers*

No desenvolvimento de jogos com duas dimensões, tem-se conhecimento dessas três estruturas *tiles, bricks, layers*, são estruturas utilizadas para a montagem geral dos jogos.

#### 4.3.2.1 *Tiles*

São imagens adicionadas em um cenário e divididas em pedaços iguais. Elas são utilizadas pelas *layers* para a construção dos cenários. Jogos como *Super Mario Bros* tem seus cenários construídos com pequenos blocos que se repetem. (PERÚCIA *et. al.*, 2005:83)

A figura abaixo dá um exemplo de *Tiles*.



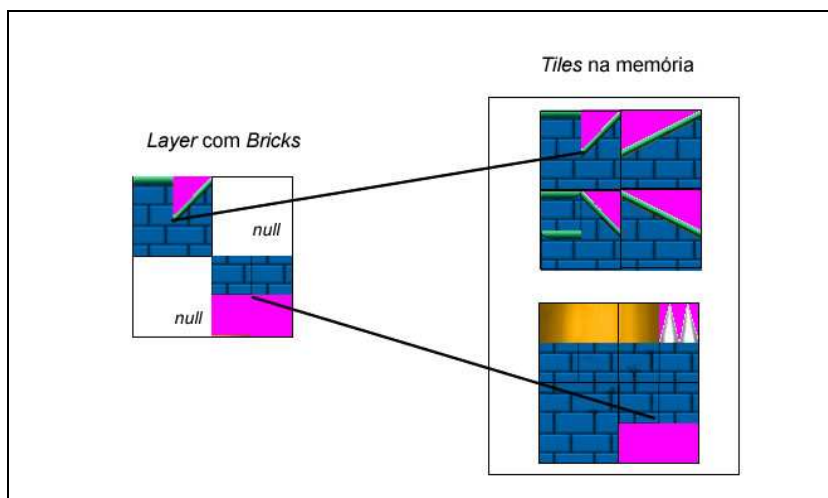
**Figura 10: Exemplo de Tiles**

FONTE: PERÚCIA *et. al.*, 2005

#### 4.3.2.2 *Bricks*

São estruturas que descrevem para a *layers* a disposição dos *tiles* que a compõem. O *brick* contém dois índices um da *layer* e o outro do *tile*. Cada *layer* possui uma matriz de *bricks* que são desenhados na tela. (PERÚCIA *et. al.*, 2005:84)

A figura abaixo demonstra a utilização de *bricks*, percebe-se que dois estão com valores nulos, ou seja, sem nenhum *tile* para ser exibido naquele local.



**Figura 11: Exemplo da utilização de *bricks***

FONTE: Adaptado de PERÚCIA *et. al.*, 2005:85

#### 4.3.2.3 Layers

São camadas de desenho que são compostas por uma matriz de *bricks* que são utilizados para desenhar os cenários e outras telas do jogo. As *layers* sempre preenchem a tela inteira do jogo, caso a *layer* seja menor, será replicada até atingir toda a tela do jogo. (PERÚCIA *et. al.*, 2005:85)

A figura abaixo exemplifica uma *layer*, os pontos vermelhos e pretos serão substituídos por *tiles*, gerando assim o quadro do jogo.



**Figura 12: Exemplo de Layer**

FONTE: Adaptado de PERÚCIA *et. al.*, 2005:85

Essas três estruturas *tiles*, *bricks* e *layers* são muito importantes para a realização de um jogo 2D.

### 4.3.3 Sprites

Conforme (PERÚCIA *et. al.*, 2005:99), são estruturas com imagens próprias permitindo a criação de animação e o livre posicionamento na tela. Todos os objetos que possuem animação ou movimento no jogo são chamados *sprites*. O *sprite* sempre tem uma imagem padrão, podendo ter vários conjuntos de animações configuradas.

A figura abaixo dá um exemplo de *sprites*, todos os objetos que estão marcados na cena é um *sprite*.



**Figura 13: Exemplo de Sprites**

FONTE: Adaptado de PERÚCIA *et. al.*, 2005:99

#### 4.3.3.1 Animação de um *Sprite*

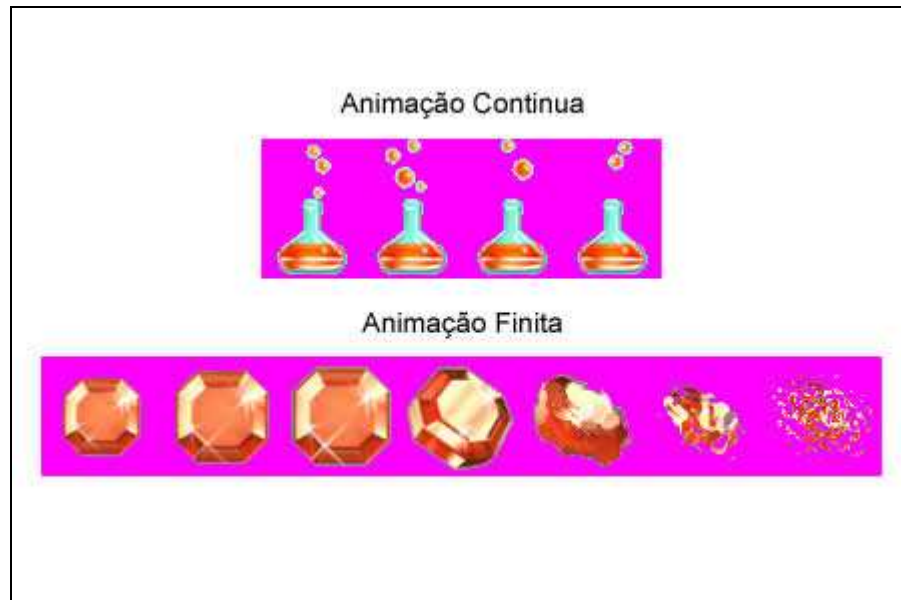
Segundo (PERÚCIA *et. al.*, 2005:102), a animação de um *Sprite* é uma seqüência de quadros desenhados um atrás do outro em um intervalo de tempo constante. Tendo dois tipos de animação:

- Animação Contínua;
- Animação Finita.

Animação Contínua: A principal característica desta animação é a sua continuidade, ou seja, a animação é infinita, tendo o seguinte padrão, exibir a seqüência de quadros até o final; ao chegar o último quadro, voltar ao primeiro, realizando um ciclo. (*Idem*, 107-108)

Animação Finita: Semelhante a animação contínua porém chegando ao último quadro ela para. (*Ibidem*, 108)

A figura abaixo demonstra quadros para os dois tipos de animação, a animação contínua possui quatro quadros podendo ser exibidos em um ciclo, já a animação finita possui sete quadros sendo o último quadro a ser exibido é a destruição do objeto.



**Figura 14: Animação Contínua e Finita**

FONTE: PERÚCIA *et. al.*, 2005:108

Os *sprites* representam os personagens e itens do jogo podendo ser movimentados via programação ou pelo próprio jogador via comandos do dispositivo de entrada conectado ao computador ou console.

#### 4.3.4 Dispositivos de Entrada

Conforme PERÚCIA *et. al.* (2005:109), um dos principais requerimentos para um jogo é a captura de entradas do jogador. Os dispositivos mais comuns são teclado, mouse, *joystick* ou *gamepad* com ou sem *force feedback*, controladores de carro, de vôo e capacetes de realidade virtual. Um jogo deve ser capaz de interpretar as entradas dos usuários, refletindo uma interação do jogo com o usuário.

Essa interação ocorre quando o jogador envia comandos por um dispositivo de entrada no computador ou vídeo game e o jogo responde.

A seguir um exemplo de algoritmo para o tratamento dos dispositivos de entrada.

```
SE (tecla A pressionada){
```

```
    Move o jogador para a esquerda
```

```
}
```

```

SE (tecla D pressionada){
    Move jogador para direita
}
SE (tecla ALT pressionada e tecla F4 pressionada){
    Finaliza o jogo
    Fecha a aplicação
}

```

Para a manipulação de entradas é fundamental a capacidade de manipular adequadamente os eventos dos dispositivos. Sendo três eventos principais mais utilizados em jogos:

- Tecla Abaixada;
- Tecla Pressionada;
- Tecla Largada.

Tecla Abaixada: Este evento indica apenas que uma tecla foi pressionada, independente do intervalo de tempo em que isso ocorreu. Pode ser utilizado para a movimentação de personagens, enquanto a tecla a estiver abaixada ir para a esquerda. (PERÚCIA *et. al.*,2005:110)

Tecla Pressionada: Refere-se ao momento em que a tecla é pressionada. Muito utilizada em jogos em que a ação deve ocorrer somente uma vez ao pressionar uma tecla. Como, o personagem deve dar um tiro somente quando pressionar a tecla ctrl. Utiliza-se o evento de tecla abaixada o jogador teria um tiro com um comportamento de metralhadora. (PERÚCIA *et. al.*,2005:110)

Tecla Largada: Refere ao momento específico em que uma tecla é largada, significando que ela estava abaixada um momento antes e agora não. Ela é muito utilizada em sistemas de menus dos jogos. (*Idem*, 110)

A utilização dos dispositivos de entrada é importante, pois são com eles que o jogador irá interagir com o jogo, controlando seu personagem para realizar suas ações.

### 4.3.5 Som

Para PERÚCIA *et. al.*, (2005:114), uma das características mais importantes de um jogo é sua ambientação musical. Os recursos de áudio dão vida ao jogo, deixando a experiência de jogá-lo mais valiosa e emocionante. A produção sonora deve ser valorizada,

pois os sons também podem enjoar o jogador ou atrapalhá-lo se a música não estiver de acordo com a temática do jogo.

#### 4.3.5.1 *Buffers* de Som

No momento em que um arquivo sonoro é lido pelo computador, um *buffer* de memória é criado. *Buffers* são blocos de memória onde guardam os dados de um determinado som, podendo ser uma música, um efeito sonoro ou outro ruído desejável em um jogo. Esses sons podem ser tantos carregados diretamente na placa de som ou na memória. (PERÚCIA *et. al.*, 2005:117)

Os sons colocados na placa de som são rapidamente usados pelo hardware, e sons alocados na memória são mais lentos pois precisam passar para o hardware. (*Idem*, 117)

Existem dois tipos de *buffers* de som, o *buffer* secundário e o *buffer* primário.

##### 4.3.5.1.1 *Buffer* secundário

Conforme PERÚCIA *et. al.* (2005:117), *buffers* secundários de som é a região da memória que guarda um som alocado. Existem dois tipos de *buffers* secundários.

- *Buffers* estático;
- *Buffers* sob demanda.

*Buffer* estático: Onde o som é pequeno e pode ser totalmente armazenado na memória, esses *buffers* ficam no hardware da placa de som, pois ele deve fazer uma reposta rápida quando requisitado. Ele é muito utilizado para efeitos sonoros simples como tiros, ou pulo do personagem. (PERÚCIA *et. al.*, 2005:117)

*Buffer* sob demanda: Usado geralmente para arquivos de som muito grande, como uma música. Se um jogo tem seis músicas cada uma com 50 MegaBytes de tamanho, dando 300 megabytes de RAM ou da própria placa de vídeo, para reduzir isso utiliza-se o *streaming*, em que consiste em criar um pequeno *buffer*, que é alimentado com a música que está na HD periodicamente. Esse tipo de *buffer* tem um conceito de circular em que quando acaba o *buffer* retorna área o início, esperando outro anuncio para continuar o processo. (*Idem*, 117)

#### 4.3.5.1.2 *Buffer* primário

Para PERÚCIA *et. al.* (2005:119), *buffer* primário é o primeiro *buffer* a ser carregado pelo jogo. Representa a mistura (*mixer*) de sons que pode ser criado em software e hardware, o *mixer* permite escutar os sons simultaneamente, pela placa de som. Podendo ser escutados as músicas, efeitos sonoros e etc.

Toda vez que um som é tocado, o *buffer* secundário é enviado ao primário e somado com ele, que então é passado para o chip DAC (*Digital Analogic Converter*) para ser tocado finalmente. Pode-se também configurar a qualidade do som reproduzido pelo jogo definindo suas características no *buffer* primário, pois todos os sons enviados para ele serão convertidos para o seu formato. (*Idem*, 119)

Embora o *buffer* primário, reproduza tudo em *loop*, ele só receberá os dados do *buffer* secundário quando houver uma requisição para a reprodução do áudio. (*Ibidem*, 119)

O som de um jogo deve ser tratado em especial, pois a música deve ser envolvente, senão o jogador pode enjoar com ela e para de jogar, vários jogos são famosos por suas trilhas outros não chegam a ser famosos por causa da má escolha das músicas.

#### 4.3.6 Tratamento de tempo

Segundo PERÚCIA *et. al.* (2005:126), o gerenciamento de tempo para jogos é vital. Sendo Realizado corretamente pode dar vários benefícios, como economia de memória e processamento, centralização de funções e facilidade em trabalhar com funções de física e simulações.

O grande objetivo deste conceito é determinar o valor do intervalo de tempo entre o quadro atual e o quadro anterior do jogo e utilizar essa informação para realizar o tratamento necessário no quadro atual. (PERÚCIA *et. al.*, 2005:126)

##### 4.3.6.1 Gerenciador de Tempo

Conforme PERÚCIA *et. al.* (2005:126), para obter o intervalo de tempo é preciso obter o tempo do sistema, cada sistema possui uma maneira de marcar o tempo, geralmente é medido em segundos ou milisegundos.

Para obter esse intervalo entre os quadros, subtrai-se o tempo atual com o tempo armazenado no quadro anterior. (PERÚCIA, 128)

#### 4.3.6.2 Taxa de Quadros

Para PERÚCIA *et. al.* (2005:126), a taxa de quadros de um jogo é o número de quadros que o jogo consegue exibir em um segundo. Essa medida é conhecida como FPS ou *frames per second*, indicando os números de quadros exibidos por segundo, esse valor varia de 30 a 200 quadros normalmente, as simulações utilizam esse padrão para realização de testes.

#### 4.3.6.3 Acumuladores de Tempo

Segundo PERÚCIA *et. al.* (2005:129), uma situação muito comum em jogos é ter que esperar um determinado intervalo de tempo para a realização de uma ação, por exemplo:

- Inimigo que atira em intervalos de 5 segundos;
- Item que congela o jogador por 3 segundos;
- Uma mensagem que pisca em intervalos de 200 milissegundos.

Para facilitar o tratamento dessas ações utiliza-se uma estrutura que conta o tempo e informar ao programador quando um limite é alcançado. Essa estrutura é chamada de acumulador de tempo funcionando da seguinte maneira (*Idem*, 130):

- Inicia-se o acumulador informando o tempo de espera;
- A cada laço o acumulador é atualizado;
- Ao atingir o tempo de espera, executar a ação desejada e, se preciso reiniciar o acumulador.

O Gerenciador de tempo além de economizar memória e facilitar para a realização de tarefas periodicamente durante o jogo pode ser utilizado como controle para ações efetuadas pelo jogador.

#### 4.3.7 Física

Segundo PERÚCIA *et. al.* (2005:133), utiliza-se a simulação das leis da física para aumentar a sensação de realidade, por mais que sejam belos os gráficos dos jogos os

jogadores irão acreditar naquele mundo quando a interação dele com o jogo forem considerados como possível.

O programador utiliza-se de física para determinar a posição dos objetos em um cenário e em um determinado momento chamado de *current frame*. Para cada quadro precisam-se atualizar as posições dos objetos em movimento para que o jogador perceba o movimento. (*Idem*, 133)

Em virtude dos sistemas de eixos que o computador utiliza, a distância é medida em *pixels* e não em metros, ocasionando implicações para outras unidades de medida, ou seja, utiliza-se o *pixel* como unidade de medida, sendo *pixels* para distância, *pixels* por segundo para a velocidade e *pixels* por segundo quadrado para a aceleração. (*Ibidem*, 133-134)

#### 4.3.7.1 Colisão

Para PERÚCIA *et. al.* (2005:139), o processo de detecção e tratamentos de colisões consiste em verificar, se após atualização das posições dos objetos, eles não se sobrepõem, caso isto aconteça, executar o tratamento necessário.

Este processo é realizado em duas etapas a primeira é a detecção da colisão, em que consiste em um conjunto de operações que informa ao programador se dois objetos colidiram entre si, a segunda etapa é o tratamento da colisão que é a ação tomada no momento em que se detecta a colisão dos dois objetos para impedir de continuarem a colidirem. (*Idem*, 139)

A detecção e o tratamento de colisão devem ser feitos a cada quadro e para cada objeto presente na cena. Esse processo deve ser rápido e eficiente o bastante, não permitindo que o jogador perceba inconsistências e não provocar queda do desempenho no jogo. (*Ibidem*, 139)

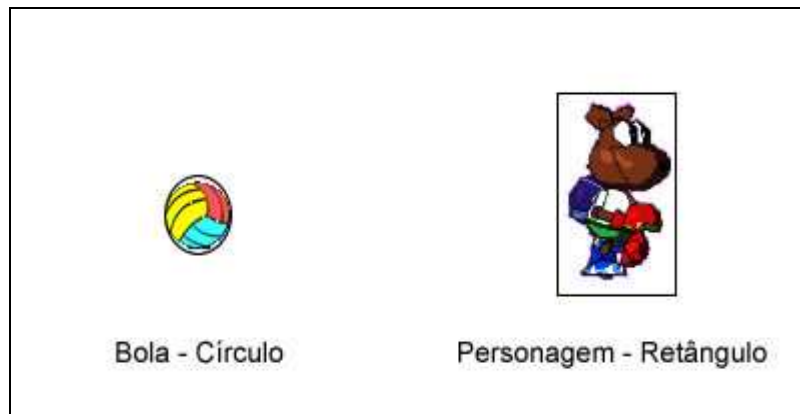
##### 4.3.7.1.1 Detecção de Colisão

Segundo PERÚCIA *et. al.* (2005:140), para realizar a detecção de colisão entre dois objetos, verifica-se se estes objetos se sobrepõem. Nos jogos 2D, a maneira mais precisa de verificar isso seria verificar se os objetos possuem *pixels* que se sobrepõem, sendo muito funcional para dois objetos, porém seria impossível verificar os *pixels* de diversos objetos um uma cena sem causar queda de desempenho.

Praticamente o que se faz é usar formas geométricas que circundam os objetos, então utiliza-se de métodos simples para determinar a colisão das formas geométricas, se isso

ocorrer é considerado que os objetos se colidiram. Deve-se escolher a forma geométrica mais adequada para detectar colisões, o formato do objeto pode ajudar para escolher a melhor forma geométrica. (PERÚCIA, 140)

A figura abaixo demonstra a escolha das formas geométricas para a realização da detecção de colisão.



**Figura 15: Escolha da forma geométrica**

FONTE: Adaptado de PERÚCIA et. al., 2005:140

Depois definido a forma geométrica, utiliza-se do método apropriado para fazer a detecção de colisão, em jogos 2D, os métodos de detecção são:

- Detecção por círculos;
- Detecção por retângulos.

Detecção por círculos: utiliza-se o teorema que dois círculos colidem quando a distância de seus centros é menor que a soma dos seus raios, sendo um método bastante comum em jogos com bolas sinuca, por exemplo. (*Idem*, 140)

Detecção por retângulos: verifica-se em separado os eixos X e Y dos retângulos, só funcionando com retângulos alinhados aos eixos das abscissas e ordenados, em retângulos rotacionados o método não funciona. Adquire um segmento de reta, projetando o retângulo em um eixo X ou Y. A colisão é detectada quando os segmentos da reta de um retângulo colidem entre si tanto no eixo das abscissas como no das ordenadas. É considerado que os objetos se colidiram. (PERÚCIA et. al., 2005:141)

#### **4.3.7.1.2 Tratamento de Colisões**

Conforme PERÚCIA et. al. (2005:145), quando um personagem colide com uma bomba, por exemplo, a bomba explode e o personagem exibirá uma animação de perdendo,

no caso de uma colisão de personagem com *brick*, deve ser feito o reposicionamento do personagem para evitar que não colida mais com o *brick*. Uma vez detectada a colisão deve ser feito o tratamento dos objetos para que o jogador não perceba uma consistência.

Em se tratando de física, dois objetos que se colidem devem ser reposicionados evitando novas colisões, os valores de física dos objetos são alterados, como velocidade. Esse tratamento é simples para colisões onde um dos objetos é estático, pois é necessária a atualização do objeto que se move. (PERÚCIA, 2005:145)

A aplicação de detecção e tratamento de colisão é uma das técnicas mais importantes para um jogo, pois com ela saberá se o personagem foi atingido por algum inimigo ou se ele atingiu o inimigo tendo que realizar o tratamento da colisão resultando em danos ao personagem ou danos ao inimigo.

#### 4.3.7.2 Gravidade

Segundo PERÚCIA *et. al.* (2005:136), no mundo real a gravidade tem aceleração de, aproximadamente  $9,8 \text{ m/s}^2$ , sendo a mesma para todos os corpos. O que faz um corpo cair mais rápido que outro é a intensidade de atrito com o ar que o corpo sofre. Simular a gravidade e um objeto de um jogo 2D significa aplicar uma aceleração no eixo das ordenadas (eixo Y), atualizando a posição e velocidade desse objeto a cada quadro.

Como a distancia em jogos 2D é medida em *pixels*, é um erro utilizar o valor de  $9,8 \text{ m/s}^2$  para a gravidade, o que se faz é definir para a gravidade, valores que tornem o movimento o mais real possível para o jogador. Ao invés de calcularmos o atrito em cada objeto para definir qual cairá mais rápido, especificamos um valor de gravidade diferente para cada objeto, ou seja, se em queda livre, um personagem tem a gravidade como 500, ele teria gravidade 250 se estivesse usando um pára-quedas aberto. (PERÚCIA *et. al.*, 2005:136)

Durante a queda de um corpo no mundo real, existe um momento em que a força que o atrito exerce sobre o corpo se iguala a força gravitacional. Nesse momento com as forças equilibradas a aceleração passa a ser nula e o corpo possui uma velocidade constante. É o que ocorre quando o pára-quedas é aberto, ele equilibra as forças de atrito e a gravitacional fazendo com que o personagem caia com velocidade constante. Para simular essa situação em jogos é definido um valor máximo para a velocidade. No momento em que esse valor é atingido, a aceleração para de agir sobre o objeto em questão. (*Idem*, 136)

A simulação da força da gravidade é muito utilizada em jogos 2D de plataforma que será abordado mais a frente, está simulação o personagem poderá morrer se cair em um buraco na tela ou em cima de algo que lhe tire a vida.

### 4.3.8 Visualizações de Jogos

Segundo MERKEL (2005:49), os jogos possuem três tipos principais de visualização:

- Primeira pessoa;
- Terceira pessoa;
- Isométrica.

Primeira Pessoa: A cena mostra o que os olhos do usuário vêem, como jogos de ação. (*Idem*, 49)

Terceira Pessoa: O jogador atua como uma terceira pessoa, podendo se ver na cena como um personagem, também muito utilizada em jogos de ação. (*Ibidem*, 49)

Isométrica: Conhecida como *God's eye*<sup>43</sup>, onde se tem uma visão geral do jogo, muito utilizada em jogos de estratégia. (MERKEL, 2005:50)

## 4.4 Gêneros de Jogos

Para MERKEL (2005:34), antes de começar a desenvolver um jogo, é preciso definir o tipo dele. Um jogo é criado para as pessoas se entreterem, mas todas as pessoas não possuem o mesmo gosto então há um tipo de jogo para cada tipo de pessoa.

Entre esses gêneros estão:

- Simuladores;
- Aventura;
- Estratégia;
- RPG;
- Esportes;
- Ação;
- Infantil;
- Passatempo;

---

<sup>43</sup> “olhar de Deus”

- Educacional;
- Luta;
- *First Person Shooter / Third Person Shooter (FTPS)*;
- Plataforma;
- Corrida;
- *Online/Massive Multiplayer*.

Simuladores: São jogos baseados em tática e física, tendo como principal objetivo a total inserção do jogador ao um ambiente proposto pelo jogo. Fazem parte dessa categoria simuladores de vôo, simuladores de carro e qualquer simulador de máquina que tente simular a realidade, deixando o jogador em primeira pessoa. (MERKEL, 2005:34)

Geralmente são feitos em 3D (três dimensões), focando um alto grau de qualidade e realismo, devido a tentativa de simulação sua implementação é bem complexa. (*Idem*, 34)

Aventura: Baseados em raciocínio e reflexo, o objetivo do jogador é resolver enigmas e quebra-cabeças, para atingir ao final do jogo. Em sua grande maioria jogos desse gênero são feitos em 2D (duas dimensões), porém em alguns jogos novos tem-se a utilização de elementos 3D. Possui uma programação simples do que de um simulador. (*Ibidem*, 34)

Estratégia: Jogos cujo objetivo é fazer decisões, que possam acarretar grandes conseqüências ao jogador, entre esses jogos temos *SimCity* (Maxis, 1989), em que o jogador deve planejar, construir e manter uma cidade, outros jogos são os de guerra como *WarCraft* (Blizzard Entertainment, 1994) e *Age of Empires* (Ensemble Studios, 1997). A implementação desses jogos tem semelhança com os outros gêneros, porém esses jogos utilizam extensas bases de dados. (MERKEL, 2005:35)

*RPG: Role Playing Games* eletrônicos, tem os mesmos objetivos dos RPG's convencionais. Sua implementação é bem complexa, pois o jogo deve possibilitar ao usuário a escolha de diversos caminhos no decorrer da história. (MERKEL, 2005:35-36)

Esportes: Jogos que simulam esportes, como futebol, tênis, basquete, boxe entre outros esportes, sua interface geralmente é desenvolvida em 3D e a implementação possui a mesma complexidade de simuladores, pois a ação do jogo é em tempo real, deixando a programação diferente da programação habitual. (*Idem*, 36)

Ação: Jogos em que a ação gira em torno de um personagem principal, que podem andar, correr, saltar, atirar, chutar, etc. A visão do jogo pode ser em primeira pessoa ou em terceira pessoa, atualmente os jogos desses gêneros são lançados em 3D. Jogos como *Tomb Raider* (Core Design, 1996), *Doom*, *Quake* são desse gênero. (*Ibidem*, 36-37)

Infantil: Jogos em que as crianças é o público alvo têm como objetivo entreter e divertir o seu público. São jogos simples caracterizados por imagens bonitas e coloridas. Tem prioridade no visual e na facilidade de interação. É formada por jogos de quebra-cabeça e histórias educativas em que a criança deve auxiliar o personagem principal. (MERKEL, 2005:37)

Passatempo: Jogos simples baseados em quebra-cabeças, jogos de tabuleiros como dama e xadrez e jogos de cartas como pôquer e paciência, geralmente possuem gráficos 2D simples, porém sua implementação pode ser complexa. (*Idem*, 36)

Educacionais: Jogos cujo objetivo é ensinar enquanto é jogado, envolve qualquer característica dos gêneros anteriores. Jogos desse gênero levam em conta critérios didáticos e pedagógicos, para atingir realmente o objetivo de ensinar brincando. (*Ibidem*, 38)

Luta: Jogo para duas pessoas, em que cada um controla um personagem, que possui diversas manobras de ataque ou defesa, sendo utilizadas para derrotar o oponente. Um dos jogos mais famosos desse gênero é o *Street Fighter* (CAPCOM, 1987). (MERKEL, 2005:38)

*First Person Shooter/Third Person Shooter* (FPS): Este gênero enfatiza tiros e combate de um ponto de vista específica. No caso dos FPS's o jogador é colocado atrás da arma dando a sensação de “estar lá”. Em sua maioria requer reflexos e velocidade. No caso de TPS's emprega uma perspectiva em terceira pessoa para o jogador normalmente atrás do jogador. Jogos desses gêneros apresentam um tipo de visão padrão podendo ser alternada entre as duas pelo jogador, outros alternam somente em determinados pontos do jogo. Jogos desse gênero são *Counter Strike* (Valve Software, 1998), *Halo* (Bungie Studios, 2001) e *Unreal Tournament* (Epic Games, 1999). (FILHO, 2005:14)

Plataforma: São chamados assim por compreenderem jogos onde o fator gravidade restringe a viagem do jogador através de superfícies horizontais referenciadas como plataformas. Um dos primeiros gêneros tornou-se muito popular, agora com a popularidade dos jogos 3D esse gênero perde espaço. Jogos de plataforma são 2D e apresentam uma visão lateral do mundo. Com o avanço dos gráficos tridimensionais foram criados mundos com plataformas 3D, perdendo assim a simplicidade do controle dos jogos 2D. (FILHO, 2005:16)

Corrida: Um dos gêneros mais tradicionais. Normalmente colocam o jogador no assento do motorista de um veículo de alta performance e realiza uma corrida contra outros jogadores ou as vezes contra o tempo. O gênero apareceu no início dos anos 80 e mantém sua popularidade até os dias de hoje. Possui a característica de sempre revolucionar a área de jogos em termos de jogabilidade e gráficos. Entre os mais populares jogos estão *Gran Turismo* (Poliphony, 1996) e *Need For Speed* (Electronic Arts, 1994). (*Idem*, 24)

*Online/Massive Multiplayer*: Este gênero pode englobar a temática dos gêneros anteriores, porém deve ser jogado em podendo ser via *Internet* ou em *lan houses*. (MERKEL, 2005:39)

Todos esses tipos de jogos possuem cada um o seu tipo de público-alvo e a forma de específica de ser jogado, alguns entram na área de outro, porém cada gênero abrange o seu público-alvo.

## 4.5 Mercado dos Jogos Eletrônicos

Conforme MERKEL (2005:39-40), ao longo dos últimos cinco anos os jogos eletrônicos passaram a ocupar uma fatia bem maior no tempo que antes era ocupado por outras atividades de lazer.

Segundo a VEJA (São Paulo, 26 de novembro de 2003) *apud* MERKEL (2005:40), o jogo *Madden NFL* (Electronic Arts, 2004), que simula partidas da *NFL*, lançado em 2004 faturou cerca de US\$ 100 milhões nas três primeiras semanas. Para comparar o filme *Chicago*, ganhador do Oscar de melhor filme de 2002, precisou de 9 meses para atingir US\$ 171 milhões em bilheteria. A banda *The Strokes* um fenômeno do rock mundial, demorou dois anos para vender 1,8 milhões de cópias de seu primeiro CD, o jogo *FIFA 2004* vendeu cerca de 6 milhões de cópias em um ano 400 mil delas aqui no Brasil.

Uma das explicações para esse tremendo sucesso é o avanço tecnológico, permitindo a interatividade e qualidade na simulação da realidade que ficou interessante para todas as faixas etárias. O jogador Ronaldinho Gaúcho atacante do Barcelona e da Seleção brasileira foi filmado em todas as situações possíveis num jogo inclusive a comemoração de um gol, para ser inserido no jogo *FIFA 2004*. (MERKEL, 2005:40)

A fidelidade de reprodução “É o que cativa o consumidor e faz aumentar as faixas etárias do público interessado” afirma a consultora Cristina Hilsenrath da NDP/Group. (*Idem*, 40)

A faixa de idade dos jogadores chega a 29 anos. Os homens são maioria, porém as mulheres correspondem 32% do universo. Estima-se que 3 milhões de brasileiros frequentam *lan houses*, “Dos vinte games mais vendidos 16 estão na categoria ‘everyone’<sup>44</sup>, e podem ser jogados por toda a família”, diz a americana Ashley Vanarsdall, da Entertainment Software Association. (*Ibidem*, 40)

---

<sup>44</sup> Todas as idades

O jogo *The Sims* (*Electronic Arts*, 2000), o mais popular do mundo com 30 milhões de cópias vendidas, o jogador cria uma família, os personagens e faz o que quiser: vai para o trabalho, realiza encontros amorosos. O segundo lugar do ranking mundial e primeiro no brasileiro é o jogo FIFA. (MERKEL, 2005:41)

Os dois são produzidos pela produtora norte-americana *Electronic Arts*, maior fabricante mundial de jogos, que faturou US\$ 2,5 bilhões em 2004, 40% mais que em 2001. (*Idem*, 41)

Enquanto a indústria do cinema faturou, em 2003, 19 bilhões de dólares. Os jogos com uma trajetória ascendente em 2002, acumularam um faturamento de US\$ 30 bilhões. (*Ibidem*, 41)

A área de jogos cresce cada vez mais, uma prova disso são jogos que possuem um faturamento maior que bandas de *rock* ou de produções cinematográficas.

#### 4.5.1 Mercado Brasileiro

Segundo MERKEL (2005:41), mesmo com o mercado internacional crescendo, porém a situação desse mercado no Brasil anda meio obscura.

Por conta da pirataria e a queda do poder de compra da classe média brasileira, os publicadores nacionais sofrem sérias dificuldades no mercado de jogos nos últimos anos. Empresas multinacionais estão indo embora e a cada dia o mercado informal cresce cada vez mais, fruto da falta de fiscalização e punição por parte dos órgãos competentes. (*Idem*, 41)

Estima-se que 90% dos jogos vendidos por aqui sejam piratas. Em 1996 o mercado de jogos eletrônicos do Brasil gerava US\$ 250 milhões com previsão de US\$ 1 Bilhão, valor próximo a países como Espanha ou Canadá. (*Ibidem*, 2005:41)

Hoje, segundo a ABES (Associação Brasileira de Empresas de *Software*) o Brasil fatura cerca de US\$ 50 milhões de dólares no mercado legal de jogos eletrônicos, as empresas instaladas no país e que investiram no mercado, não faturaram nada, o governo não arrecadou praticamente nada de imposto, comerciantes legais que cumprem com seus deveres legais e tributários, também não lucraram, quem ganhou foi o mercado informal, a pirataria. (MERKEL, 2005:41-42)

Porém, ainda existem no Brasil empresas que estão começando a ter destaque no mercado de jogos eletrônicos, com qualidade para atingir o mercado externo, sendo que

algumas já começaram a exportar os seus produtos. Maioria destas empresas estão situadas no estado do Paraná e em sua maioria instaladas em projetos do governo. (*Idem*, 42)

O maior exemplo é a empresa *Continuum Entertainment* de Curitiba que com o lançamento do jogo em tempo real (RTS - *Real Time Strategy*) *Outlive* vendeu 25 mil cópias em um mês nos estados unidos pela distribuidora *Take Two*. No Brasil foram vendidas 3 mil cópias do jogo. (*Ibidem*, 42)

O Paraná tem grandes possibilidades de liderar a produção nacional de jogos eletrônicos, uma vez que possui o maior número de desenvolvedoras de games do país e de futuramente alcançar o reconhecimento internacional de pólo de excelência no desenvolvimento de *software* de entretenimento. É acreditando nisso que as empresas e o governo criaram uma rede visando à criação de novas empresas e negócios. (MERKEL, 2005: 42)

Com a organização das empresas do setor e o apoio do governo do estado será criada a sinergia necessária para o sucesso desta empreitada. (*Idem*, 42)

O mercado brasileiro não cresce junto com o mercado internacional, pois com a presença da pirataria cada vez mais forte as empresas de jogos saem do mercado nacional para irem a outros mercados.

#### **4.5.2 Indústria de desenvolvimento de jogos eletrônicos no Brasil<sup>45</sup>**

A Associação Brasileira de Desenvolvedoras de Jogos eletrônicos (Abragames), uma entidade sem fins lucrativos, realizou uma pesquisa sobre a realidade das empresas que trabalham nesse ramo. Sendo analisado a partir de agora alguns dados relevantes.

É importante ressaltar que a pesquisa visou somente a área de desenvolvimento tirando de seu alvo empresas participantes dessa área, porém com outra função como distribuidoras, marketing e publicidade.

A pesquisa foi realizada entre os dias de 21 de fevereiro a 3 de Abril de 2005. Os resultados referem-se à idade, distribuição geográfica, segmentação das atividades, crescimentos nos últimos anos, faturamento médio, empregabilidade<sup>46</sup> e perfil dos profissionais da área.

Para maiores informações a pesquisa segue em Anexo neste trabalho.

---

<sup>45</sup> O Governo Federal possui um programa dedicado para jogos eletrônicos desenvolvidos no país endereço [www.jogosbr.org.br](http://www.jogosbr.org.br)

<sup>46</sup> Capacidade de adequação ao profissional às novas necessidades

Com os resultados da pesquisa percebe-se que a indústria de desenvolvimento de jogos começou nos anos 80, prova disso são registros de empresas fundadas a 10 anos atrás mais precisamente entre o final dos anos 80 e início dos anos 90. Mas foi em 97 que o mercado começou a se movimentar e em 99, tem-se o recorde de fundações de empresas de jogos no Brasil (21%). Tendo hoje 55 empresas do ramo.

A pesquisa também demonstrou que o estado que mais abriga empresas do ramo é o Paraná com 33% seguido por São Paulo e Rio de Janeiro ambos com 30% e 12% respectivamente,

Porém o faturamento dos três estados são quase idênticos SP(36%), RJ(26%), PR(16%), ainda com a inclusão de Pernambuco(16%) e Rio Grande do Sul(16%) na análise que também possuem o mesmo faturamento do Paraná.

O mercado de jogos é dividido em quatro nichos, entretenimento puro, *middlewares* (ferramentas para o desenvolvimento ou manutenção de um jogo), *advergames* (jogos com vocação publicitária) e *business games* (Simulações de negócios com fins de aprendizado). Percebe-se pela pesquisa que as empresas nacionais investem no nicho de entretenimento puro (72%), porém *advergames* (16%) está começando a crescer e *middlewares* (8%) e *business games* (6%), começam a se movimentar.

A pesquisa demonstra também, que a dificuldade em obter as licenças dos kits de desenvolvimentos para os grandes *consoles*, impede o avanço das desenvolvedoras nesse nicho, outro fator é a pirataria. Deixando o foco das empresas limitado aos PC's (66%) e celulares (23%).

Conforme a pesquisa, o mercado demonstra um grande crescimento de 40%, o valor do faturamento das empresas gira em torno de 20 milhões de reais, não podendo competir com os grandes centros de desenvolvimento do mundo, porém elas têm crescido em um modo muito competitivo. Além disso, a pesquisa demonstrou que ao mesmo tempo em que empresas antigas crescem novas empresas são fundadas, fazendo uma renovação de mercado boa.

A pesquisa demonstrou que uma empresa de jogos no Brasil emprega aproximadamente 15 pessoas. A pesquisa também demonstrou o perfil desses 15 profissionais, que vão desde programador até administradores. Um fato levantado foi a relação de programador (36%) / artistas (ilustradores (12%) e modeladores 3D (15%)), muito diferente dos mercados consolidados em que a relação é de 2 artistas para cada programador.

O fator para essa adversidade pode ser que os cursos de graduação dão mais ênfase no desenvolvimento do jogo, ou as desenvolvedoras não estão dando ênfase ao conjunto visual

do jogo. Para descobrir os reais motivos dessa adversidade será necessária a realização de outra pesquisa entre os desenvolvedores.

## 5 Busca Competitiva

A inteligência artificial e os jogos caminham juntos desde o início da história dos jogos, porém a inteligência artificial não teve um grande investimento no início por parte dos desenvolvedores. Ela não era muito usada em jogos comerciais para consoles, e sim mais usada para programas específicos de jogos como o *Deep Blue* (IBM, 1997) computador que jogou uma partida de xadrez contra Gary Kasparov campeão mundial de xadrez daquela época.

*O jogo decisivo da competição foi o segundo, que deixou uma cicatriz em minha memória ... vimos algo que estava muito além de nossas expectativas mais fantásticas a respeito da capacidade que um computador teria de prever as conseqüências posicionais de suas decisões a longo prazo. A máquina se recusou a efetuar um movimento para uma posição que teria vantagem decisiva a curto prazo – mostrando um sentido de perigo muito humano.*(KASPAROV 1997 apud RUSSEL & NORVIG, 2005:174)

No início do século XXI a utilização desses algoritmos de IA está crescendo, pois os jogos já chegaram à uma qualidade gráfica sem igual, restando apenas o crescimento do desafio de chegar ao final do jogo. Portanto as empresas de agora que não possuem um bom desenvolvedor de IA em jogos pode estar com risco de se fechar. Este desenvolvedor deve ter o conhecimento dos grandes algoritmos utilizados em jogos como *MiniMax*, *A-Star* entre outros.

### 5.1. Histórico

Segundo FILHO (2005:1), a Inteligência Artificial vem sendo aplicada em jogos desde os anos 70, onde surgiram os primeiros jogos simples e de recursos escassos. A área de IA para jogos ainda influencia a percepção do público em geral. Ainda nos dias de hoje os fantasmas de *Pac-Man* (*Inky*, *Pinky*, *Blink* e *Clyde*), assombram a área. Até recentemente as desenvolvedoras de jogos não desenvolviam nada para melhorar a IA de seus jogos.

A tabela abaixo demonstra o histórico da utilização de IA em jogos eletrônicos.

**Tabela 1 : Histórico da utilização da IA em Jogos**

Nível de IA	Aplicação	Ano
Sem IA	<i>Space War!</i> – primeiro jogo para computador escrito para o minicomputador PDP-1. Requer 2 jogadores.	1962
	<i>Pong</i> – Versão eletrônica do tênis de mesa	1972
Padrões	Inimigos começam a aparecer	1974
	<i>Space Invaders</i> – Inimigos são padronizados, mas atiram de volta. Considerado o primeiro jogo “clássico” com níveis, <i>score</i> <sup>47</sup> , controles simples a dificuldade crescente no decorrer do jogo	1978
	<i>Pac-Man</i> – Fantasmas em movimento padronizado, mas cada fantasma possui uma personalidade	1980
	Um microcomputador vence um jogador profissional de xadrez pela primeira vez	1983
	<i>Karate Champ</i> (Data East, 1984) – um dos primeiros jogos de luta um contra um, com o computador como adversário	1984
FSM's	<i>Herzog zweio</i> (TechnoSoft, 1990) - O primeiro RTS a surgir e o mundo experimenta pela primeira vez uma péssima implementação do algoritmo de <i>pathfinding</i>	1990
	<i>Doom</i> ( <i>id Software</i> , 1993) – início oficial da era dos FPS	1993
Várias técnicas	<i>BattleCruiser: 3000AD</i> ( <i>Take Two Software</i> , 1996)– Primeiro uso de redes neurais em um jogo comercial	1996
	<i>Deep Blue</i> – derrota o atual campeão de xadrez Gary Kasparov	1997
	<i>Half Life</i> – A inteligência artificial em jogos encontra-se em seu auge. O jogo faz grande uso de linguagens de <i>script</i> .	1998
	<i>Black &amp; White</i> ( <i>Lionhead Studios</i> , 2001) – Utiliza criaturas que usam aprendizado por reforço e observação.	2001

FONTE: Adaptado de FILHO, 2005:1-2

No decorrer dos anos a IA em jogos teve um crescimento muito tímido em relação ao crescimento da IA como um todo. No início do século XXI os desenvolvedores de jogos começam a olhar para essa área com interesse, pois agora é crucial um jogo ter uma IA convincente para o jogador.

## 5.2 Definição

Segundo PERÚCIA (2005:152), os jogos podem utilizar em sua implementação diversos algoritmos, podendo alguns simples outros complexos. Dependendo do tipo do jogo e as especificações dele. Jogos como Mario não precisam ter uma IA muito evoluída, pois os

<sup>47</sup> Pontuação

seus inimigos apesar de serem “tolos”, estão em grande quantidade para causar problema ao herói, já jogos como *StarCraft* (*Blizzard Entertainment*, 1998) ou *Age of Empires* (*Ensemble Studios*, 1997), que simulam batalhas ou mundo reais, a IA tem uma importância significativa.

Para SANTEE (2005:371), sempre quando algum desenvolvedor de jogos chega ao ponto de desenvolver a IA de seu jogo os sentimentos de medo, espanto e interesse surgem. Tratar a IA é falar de um ramo importante para a “tecnologia terrestre” e também para simulação de comportamentos de situação, ação e reação.

O primeiro passo é ter todo um conhecimento do comportamento humano, e que esse tipo de informação está baseado em uma simples lógica de verificação e reação, ou seja, a realização de vários comandos condicionais verificando os acontecimentos e aplicando suas conseqüências. (SANTEE, 2005:371)

Dependendo do tipo de jogo a área de IA terá um enfoque diferente, para jogos mais realistas a IA deve ser bem fiel ao mundo real, já jogos de fantasia ou de puro entretenimento não é necessária uma IA bem convincente.

### **5.3 IA tradicional X IA em Jogos**

Como visto anteriormente a definição de IA é a busca de um agente que se sobressaia sobre outros agentes em um determinado ambiente ou situação ou a criação de agentes que simulam ações ou pensamentos. Já a IA para jogos é um algoritmo dentro de um jogo com o objetivo de controlar os inimigos de uma forma que eles aparentam um comportamento inteligente quando surgem cenários com múltiplas escolhas. A palavra aparentar é a palavra chave para a diferença de IA com a IA em jogos. A IA em jogos não se preocupa nos resultados de sua execução e nem como é feito, ele se interessa em como o sistema “jogo” age. (FILHO, 2005:3-4)

Os jogadores não estão preocupados em saber se o seu inimigo foi implementado em cima de uma rede neural ou de um sistema especialista, eles estão preocupados com a aparente inteligência demonstrada. (*Idem*, 4)

## 5.4 Motores de IA

Como dito anteriormente a evolução dos motores e *frameworks* para o desenvolvimento de jogos eletrônicos seguiu a idéia de dividir e conquistar, após a divisão das áreas de *engine* o desenvolvedor só precisaria juntar todas as *engines* para construir o jogo.

Essas *engines* abordam diversas áreas, porém a área de IA ainda não teve um aprofundamento e consta de forma tímida em grande maioria das *engines* para jogos, deixando o trabalho para o desenvolvedor criar a IA. (FILHO, 2005:31)

Alguns motores de jogos se destacam na ênfase que dão para a área da IA, porém ainda que timidamente.

### 5.4.1 Unreal Engine 3

Segundo FILHO (2005:31), *Unreal Engine 3* é um *framework* completo desenvolvido pela *Epic Games*, especialmente feito para os novos consoles como *Playstation 3*, *XBOX 360* e *Nintendo Wii*, além de PC's equipados com *DirectX*, tendo uma abrangência muito grande em diversas áreas do desenvolvimento de jogos, facilitando a vida do desenvolvedor e do artista de jogos. Vários jogos já foram desenvolvidos com essa *engine*, entre eles estão *Unreal Tournament*, *Unreal Championship* (*Infogrames*, 2002), *Splinter Cell* (*Ubisoft*) e *Harry Potter* (*Electronic Arts*, 2001).

O sistema de IA dessa *engine* aborda em vários níveis como:

- Suporte ao *pathfinding*;
- Navegação de alto nível;
- IA baseada em times.

Suporte ao *pathfinding*: comporta objetos complexos como portas elevadores permitindo uma navegação entre os cenários onde os personagens podem interagir com os objetos para alcançar o seu objetivo final. (*Idem*, 32)

Navegação de alto nível: suporta táticas de combate de curto prazo como fazer cobertura. (*Ibidem*, 32)

IA baseada em times: adequada para jogos do gênero de FTGS e Estratégia. Suporta o gerenciamento de times, objetivos e metas de longo prazo. (FILHO, 2005:32)

A criação dos caminhos para os sistema de navegação e de *pathfinding* utiliza a ferramenta *UnrealEd*, onde os designers podem editar os caminhos graficamente, além de possuir recursos de otimização e dar dicas para utilização. (FILHO, 2005:32)

A *engine* ainda conta com um sistema de *script* e uma ferramenta visual para edição a *UnrealKismet*, ela permite o total controle sobre as informações do jogo para os desenvolvedores e designers sem a necessidade de escrever uma linha de código. (*Idem*, 32)

A *Unreal Engine* é um *framework* proprietário tendo três tipos de licença:

- *Royalty-Bearing*;
- *Royalty-Free*;
- *Custom*.

*Royalty-Bearing*: possui o valor de US\$350000,00 mais US\$50000,00 por plataforma de desenvolvimento, além de pagar uma taxa de 3% em cima do faturamento do jogo. (*Ibidem*, 33)

*Royalty-Free*: possui o valor de US\$750000,00 mais US\$100000,00 por plataforma, não cobrando nenhuma taxa por jogo vendido. (FILHO, 2005:33)

*Custom*: essa última licença a empresa interessada deverá entrar em contato com a *Epic Games* para a aquisição do produto. Sendo destinada para projetos não comerciais voltados mais para aprendizado. (*Idem*, 33)

### 5.4.2 *Source Engine*

Para FILHO (2005:34), a *Source Engine* é um *framework*, Desenvolvido pela *Valve Software* para produção de jogos envolvendo animação de personagens, física, baseada em *shaders* e uma avançada IA. O *framework* foi usado para o desenvolvimento de jogos famosos como *Half-Life*, *Counter-Strike* e *Day of Defeat*. A *Source Engine* juntamente com a *Unreal Engine* são as *engines* mais utilizadas dessa área.

Os recursos da *engine* consistem em um sistema de entrada e saída dando o controle da IA para os desenvolvedores, além de possuir um sistema de navegação onde os personagens podem correr, voar, pular etc. também permite tratar eventos relacionado aos sentidos humanos e ainda adicionar relacionamentos que definem o status amigo ou inimigo. (*Idem*, 34)

Outro recurso é o de batalhas em que se pode formar uma esquadra ou pelotão com o objetivo de trabalhar em conjunto, podendo se defender, cobrir alguém, avançar, recuar no momento exato. (FILHO, 2005:34)

O *framework* é proprietário tendo que entrar em contato com a *Valve* para obtenção de informações, pois seus valores não são públicos. (*Idem*, 34)

### 5.4.3 Reality Engine

Conforme FILHO (2005:34), *Reality Engine* é um motor para o desenvolvimento de jogos produzido pela *Artificial Studios*, sendo desenvolvido em cima da plataforma *DirectX9* podendo rodar em *DirectX7* ou 8. Tendo vários jogos desenvolvido com ele como *Armageddon*, *CellFactor* e *Dreamers*.

Foi criada com o objetivo de competir com a *Source Engine* e a *Unreal Engine 3*, tendo uma qualidade gráfica bem superior, ela veio ao mercado como uma *engine* semelhante as líderes do mercado porém com um custo de aquisição menor, isso fez a *EpicGames* comprar a *Relity Engine* acabando com o projeto e contratando parte da equipe para trabalhar na *Unreal Engine*. (*Idem*, 34-35)

A *Reality Engine* possui um suporte a IA com técnicas importantes como, *pathfinding*, tomada de decisões baseadas em máquinas de estados, além de dar aos agentes inteligentes a possibilidade de responder a estímulos sonoros e visuais. (FILHO, 2005:35)

O módulo de IA é totalmente feito em *script* com o propósito de facilitar a extensão da *engine*, outra característica é a arquitetura da *engine*, pois para melhorar a performance a IA é executada no servidor tendo o resultado passado para os clientes. (*Idem*, 35)

Essa *engine* também é proprietário tendo duas versões uma é a venda do código-fonte<sup>48</sup> da *Reality Engine* a outra é a compra da *Reality Engine SDK* que dá acesso somente aos binários<sup>49</sup> da *engine*. (*Ibidem*, 35)

A *Artificial Studios* não disponibiliza publicamente o valor de suas versões.

Esses três motores são os melhores para a criação e desenvolvimento de jogos tendo já alguns jogos famosos desenvolvidos por uma delas, porém a ênfase que a IA recebe dos três é bem em relação a outras funcionalidades que os *engines* têm.

---

<sup>48</sup> conjunto de palavras escritas de forma ordenada, contendo instruções em uma das linguagens de programação existentes no mercado

<sup>49</sup> Programa executável

## 5.5 IA em jogos

Segundo SILVA (2005:11), a IA se desenvolveu muito nos últimos trinta anos, desenvolvendo cada vez mais algoritmos específicos para problemas específicos, não dando enfoque para a construção de sistemas que se aproximem da inteligência humana a IA Forte, também conhecida como “*Humam-Level AI*”<sup>50</sup>.

Os sistemas de *Humam-Level AI* são como os vistos no cinema em *C3PO*, *R2-D2* de *Star Wars* ou HAL de 2001 *Uma Odisséia no Espaço*. Eles demonstram características de inteligência humana como resposta em tempo-real, robustez, interação inteligente autônoma com o ambiente, planejamento, criatividade, comunicação em linguagem natural, raciocínio de senso comum. (*Idem*, 11-12)

Os Jogos eletrônicos são definidos como um “*Killer Application*”<sup>51</sup> para o campo da IA, pois os jogos precisarão cada vez mais de *Humam-Level AI*, além de oferecerem um ambiente de pesquisa para problemas específicos levando para uma pesquisa de integração para obter uma *Humam-Level AI*. (*Ibidem*, 12)

Há diversas razões para que os pesquisadores de IA migrem para a área de desenvolvimento de jogos. (SILVA, 2005:12)

Primeira os desenvolvedores de jogos já perceberam da necessidade de construir cada vez mais personagens inteligentes. (*Idem*, 12)

Segunda a área de jogos eletrônicos é altamente competitiva e a tecnologia que será o grande diferencial a partir de agora é o desenvolvimento de IA. (*Ibidem*, 12)

Terceiro o cargo de programador de IA, já está comum na área de desenvolvimento de jogos. (SILVA, 2005:12)

Quarto como demonstrado no capítulo anterior o mercado de jogos fatura mais que a do cinema. (*Idem*, 12)

Quinto com a evolução dos hardwares de processamento gráfico e a sua otimização sobra mais processamento para os algoritmos de IA. (*Ibidem*, 12)

Sexto e último, a indústria dos jogos precisa da IA acadêmica, pois a ênfase dos jogos na IA é a simulação do comportamento humano em situações limitadas. À medida que os jogos evoluem em termos de gráficos exige-se a construção de personagens mais inteligentes. O pesquisador de IA pode utilizar esse ambiente para o desenvolvimento de agentes cada vez mais inteligentes. (SILVA, 2005:12)

---

<sup>50</sup> “Nível Humano de IA”

<sup>51</sup> “Aplicação Assassina”

O ambiente onde o jogo acontece é virtual, porém não é uma simulação de um problema, ele próprio é o problema a ser resolvido. (SILVA, 2005:12)

Existem diversas técnicas de IA utilizadas para o desenvolvimento de jogos. Dando aos personagens certa inteligência e uma personalidade. Segundo LAMOTHE (1999) *apud* KISHIMOTO (2003:5) um dos princípios básicos para a IA em jogos são os algoritmos determinísticos e padrões de movimento, onde os personagens tem seus comportamentos pré-programados ou pré-processados. Além de outras técnicas como Agentes Inteligentes, Máquinas de Estados Finitos entre outros algoritmos.

A IA em jogos quer cada vez mais se aproximar a inteligência humana, com isso pesquisadores de IA migram para o desenvolvimento de jogos para o desenvolvimento de uma IA cada vez mais forte.

### **5.5.1 Agentes Inteligentes**

Para RUSSEL & NORVIG (2004:33), um agente é todo aquele que consegue interagir com o ambiente a sua volta por meio de sensores e atuadores. O Agente humano tem olhos, ouvidos como sensores e mãos, pernas e boca como atuadores, já um agente robótico teria câmera e detectores da faixa infravermelho. Todo agente pode perceber as próprias ações, porém as conseqüências de suas ações não.

A utilização da palavra do termo percepção, é referente aos sensores perceptivos que os agentes possuem. A seqüência de percepções do agente é a história de tudo que aconteceu com ele. A escolha de uma ação por um agente será com base nas seqüências de percepções obtidas até aquele momento. (*Idem*, 34)

Qualquer sistema pode ser considerado um agente inteligente se possuir as seguintes características:

- Autonomia;
- Habilidade social;
- Reatividade;
- Pró-atividade.

Autonomia: o agente deve funcionar sem intervenção de um humano, baseando no conhecimento do ambiente dele guardado em cima de acontecimentos anteriores. (FERNANDES, 2005:91)

Habilidade Social: Interagir com outros agentes por uma linguagem em comum. (FERNANDES, 2005:91)

Reatividade: Ser capaz de perceber e se adaptar com as mudanças ocorridas em seu ambiente. (*Idem*, 91)

Pró-atividade: o agente não atua com base na sua percepção, porém deve procurar alcançar um objetivo demonstrando iniciativa. (*Ibidem*, 91)

### 5.5.1.1 Tipos de Agentes

Entre os tipos de agentes inteligentes se destacam o agente reativo e o agente cognitivo a diferença entre os dois está em que os agentes reativos consideram somente os dados disponíveis naquele momento, ou seja, eles não possuem memórias. Os agentes cognitivos aprendem com as suas experiências e são deliberativos, tendo a oportunidade de trocar informações de histórico e planos de ataque com outros agentes, criando um sistema mais inteligente. (PERÚCIA *et. al.*, 2005:153)

Porém a vantagem dos agentes reativos está na velocidade das respostas, pois o reativo responde com base nos dados encontrados no momento e o agente cognitivo pode consultar a sua base de experiências, além de trocar informações com outros agentes. O tempo é normalmente curto em aplicações de jogos, por isso que o uso de agentes cognitivos ainda é uma novidade. (*Idem*, 153)

#### 5.5.1.1.1 Agentes Reativos

Conforme RUSSEL & NORVIG (2004, 46), o tipo de agente mais simples. Eles selecionam uma ação com base na percepção, atua ignorando todo e qualquer histórico de percepções. O processamento é realizado de acordo somente com os dados da entrada, com isso é ativado uma conexão entre o programa do agente os dados da entrada e sua resposta a aquela ação essa conexão é conhecida como regra condição-ação.

Os seres humanos têm muitas dessas regras, algumas são aprendidas como dirigir um carro ao parar no sinal fechado, e outras são reflexos inatos como o piscar de olhos quando algo se aproxima de seu olho. (*Idem*, 47)

Os agentes reativos são simples, porém sua inteligência será muito baixa, pois a sua decisão será tomada somente quando todos os dados de entrada vindo do ambiente forem

processados, ou seja, se houver um empecilho na entrada de dados do agente que não seja previsto pelos desenvolvedores do agente, causará diversas dificuldades ao tomar a decisão. (RUSSEL & NORVIG, 2004:47)

Abaixo está o algoritmo de um agente simples:

função AGENTE-REATIVO-SIMPLES(percepção) retorna ação

Variáveis estáticas: regras

estado = INTERPRETAR-ENTRADA(percepção)

regra = REGRA-CORRESPONDENTE(estados, regra)

ação = AÇÃO-DA-REGRA(regra)

retornar AÇÃO

fim.

A função recebe como parâmetro os dados dos sensores, após é obtido o estado do agente, com base no estado obtido e na regra define a ação a ser tomada e retorna ela para ser executada.

### 5.5.1.1.2 Agentes Cognitivos

Segundo FARACO (1998) *apud* (FERNANDES, 2005:102-103), se caracteriza pela complexidade dos agentes, onde demonstram meios de inferência de decisão robusta, interações sofisticadas e alto grau de intenção. Eles são divididos em Agentes cognitivos funcionais e Agentes Cognitivos baseados em estado mental.

Os agentes funcionais são compostos por módulos necessários para a execução do sistema e devem possuir conhecimento, objetivos, além de ter a capacidade de: percepção, comunicação, decisão e raciocínio. (DEMAZEAU *apud* FERNANDES, 2005:103)

Os agentes Cognitivos vêm de uma abordagem psicológica para a definição de sua estrutura. Eles possuem estados mentais a respeito do domínio de crenças, capacidade de escolha e compromisso, por isso que o estado do agente é chamado de estado mental. (FERNANDES, 2005:103)

Esses estados mentais devem estar bem definidos, pois a forma de encontrar um agente cognitivo em um hardware ou software é o fato de ele poder ser analisado e controlado por esses estados mentais. Deixando a questão da definição de Agente para quais entidades podem ser consideradas como possuir um estado mental. (D'AMICO *apud* FERNANDES, 2004:103)

### 5.5.1.1.3 Agentes Híbridos

Esse tipo de agente mescla os dois anteriores tendo o objetivo de tornar mais funcional para a construção de agentes. Nesse caso, os agentes serão dotados de comportamentos reativos com relação aos eventos que acontecem no ambiente naquele momento, e comportamento deliberativo no momento em que for definido um objetivo para ele. (FARACO 1998 *apud* FERNANDES, 2004:104)

Esse tipo de agente soluciona a incapacidade de ação adequada por parte do agente cognitivo para respostas mais rápidas em momentos oportunos, além de dar ao agente reativo a capacidade de raciocínio e planejamento no momento em que ele se depara com uma situação onde o ambiente diverge bastante de seus objetivos iniciais. (YEPES 2002 *apud* FERNANDES, 2004:104)

### 5.5.1.2 Agentes Inteligentes em Jogos

Para a demonstração da utilização de Agentes Inteligentes em jogos tem-se como exemplo o gênero de jogo RPG explicado no capítulo anterior.

Segundo SILVA (2005:13), a IA em um jogo de RPG é utilizada para controlar inimigos, parceiros e personagens de suporte.

Os inimigos seriam considerados os monstros que o jogador encontra pelo caminho ou personagens iguais ao jogador, porém controlados pela IA do jogo. Exige uma tarefa complexa para a construção da IA para os inimigos, eles devem ser autônomos e precisam interagir com o ambiente ao redor deles, precisando de um comportamento reativo, planejamento e senso comum. Para uma demonstração mais fiel eles precisam das mesmas informações sensoriais que o jogador possui. Para se locomoverem pelo ambiente é necessário de um algoritmo de *pathfinding*, raciocínio espacial e raciocínio temporal, podendo ter agentes que se adaptam a novos ambientes ou as estratégias do jogador, chegando a aprender. (*Idem*, 14)

A IA para os parceiros que o jogador possui exige uma tarefa mais complexa ainda, pois o jogador irá encontrar um inimigo poucas vezes e vários deles diferentes já o aliado poderá ficar boa parte do tempo ao lado do jogador. O desafio é fazer que o aliado demonstre um comportamento humano. O jogador poderá conversar com o seu aliado para isso ele

precisará se lembrar dos acontecimentos anteriores, o aliado pode ficar magoado com as ações do jogador com ele, ou criar uma aliança dependendo do tratamento que ele recebe em um período de tempo. O agente aliado precisa simular emoções realmente humanas. O RPG *NeverWinter Nights* é um jogo com agentes aliados. O jogador desenvolve um tipo de relacionamento com seus aliados no decorrer do jogo, chegando a ter vários finais o jogo dependendo da interação que o jogador teve com seus aliados. (SILVA, 2005:14)

Os personagens de suporte são, por exemplo, o dono da taverna, o guarda do portão, o rei, o mercador mesquinho, são personagens que o jogador não irá ver durante todo o jogo, porém a implementação da IA deles deve ser semelhante ao do Aliado. (*Idem*, 15)

Agentes Inteligentes é muito usado em jogos, apesar de ser uma nova área da IA geral, os agentes inteligentes tem o objetivo de unir todas as áreas da IA tradicional, para que o agente inteligente de um jogo consiga simular o comportamento humano ele precisará de um motor ou uma máquina.

### **5.5.2 Máquinas de Estados Finitos**

Como dito no capítulo anterior FSM's são estruturas que facilitam para o planejamento do jogo, podendo ser usada também para o desenvolvimento de sua IA juntamente com a estrutura de Agentes Inteligentes.

Segundo MENDES (UNIDEV, 2002), nessa concepção o estado de uma máquina de estados de uma forma simples é a ação que o agente inteligente estaria executando. O sinal para mudar de ação é chamado de função de transição. Essa função pode avaliar diversas condições para tomar uma decisão um exemplo é a porta eletrônica que se abre com a aproximação do jogador, porém somente se ele estiver carregando uma chave abaixo uma demonstração dessas condições:

Se (jogador se aproximou da porta) e (jogador possuir chave)

Abrir porta

#### **5.5.2.1 Máquinas de Estados Finitos como IA em jogos**

A principal vantagem de utilizar FSM's para o desenvolvimento de IA em jogos é que ela consegue ser adaptável para qualquer tecnologia, sendo a parte principal ou secundária da inteligência do personagem, além de poder fazer que diferentes personagens utilizem da

mesma rotina podendo gerar assim um comportamento diferente para cada personagem. (MENDES, UNIDDEV:2003)

Com essa complementação da definição de máquinas de estados finitos, inicia-se o processo da criação de uma FSM para um jogo.

- Definir como o NPC deve se comportar;
- Identificar os estados;
- Identificar variáveis de controle;
- Definir como os estados se relacionam;
- Definir as funções de transição;
- Definir a ação exercida pelo NPC em cada estado;
- Implementar a máquina de estados.

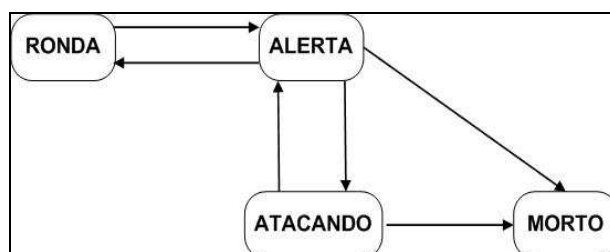
Definir como o NPC deve se comportar: tomaremos como exemplo um guarda, ele faz uma ronda e quando um personagem aproxima-se dele, ele vai a sua direção. Chegando perto o suficiente o guarda ataca, se o personagem morre, ou foge e fica longe do guarda, o guarda volta ao estado de alerta. Se o guarda for atingido ele vai para o estado de alerta, procurando o personagem que atacou, se ele morrer ele morre. (*Idem*, UNIDDEV:2003)

Identificar os estados: com base na explanação acima se encontram os seguintes estados: RONDA, ALERTA, ATACANDO, MORTO. Que definem o comportamento do guarda. (*Ibidem*, UNIDDEV:2003)

Identificar as variáveis de controle: as variáveis de controle para o caso acima são: distância, visibilidade e levar algum dano. Pode-se adicionar mais duas variáveis de controle uma seria a probabilidade de mudar de estado e a outra o tempo que se encontra nesse estado. (MENDES, UNIDDEV, 2003)

Definir como os estados se relacionam: para isso é montado um desenho descrevendo as relações dos estados. (*Idem*, UNIDDEV:2003)

A figura abaixo demonstra os relacionamentos de cada estado onde RONDA é o estado inicial e MORTO é o estado final.



**FIGURA 16: FSM do Guarda**  
 FONTE: Adaptado de MENDES, UNIDDEV

Definir as funções de transição: para facilitar é usada uma tabela com a condição para a função de transição. (MENDES, UNIDDEV:2003)

A tabela abaixo demonstra as funções de transição do guarda.

**TABELA 2: Funções de Transição de uma FSM**

Estado Inicial	Função de Transição	Estado Final
RONDA	((dano = 1) ou (distancia < media e visível = 1)) e (chance < 90%)	ALERTA
ALERTA	Vitalidade < 0	MORTO
ALERTA	((distancia > media) e (chance < 90%)) ou ((visível = 0) e (tempo > tempo máximo))	RONDA
ALERTA	(distancia < perto) e (visível = 1) e (chance < 80%)	ATACANDO
ATACANDO	((distancia > perto) ou (dano > vitalidade x (0,50+sorteie ente (0 e 0,5))) e (vitalidade > 0))	ALERTA
ATACANDO	Vitalidade < 0	MORTO

FONTE: Adaptada de MENDES, UNIDDEV:2003

Definir a ação exercida pelo NPC em cada estado: o comportamento pode ser demonstrado em uma tabela também. (*Idem*, UNIDDEV:2003)

A tabela abaixo demonstra as ações que o guarda realizará para cada estado.

**TABELA 3: Ação de Cada Estado**

Estado	Ação
RONDA	Anda de acordo com um caminho alterando periodicamente o caminho.
ALERTA	Dependendo da vitalidade ele segue em direção ao personagem ou foge dele.
ATACANDO	Procede com o ataque.
MORTO	Demonstra animação de morte.

FONTE: MENDES, UNIDDEV, 2003

Na tabela está a definição do que o guarda irá fazer. Poderíamos colocar sub-máquinas dentro dos estados para ficar mais perfeita a ação. (*Ibidem*, UNIDDEV:2003)

Implementar a máquina de estados: Existem duas formas de implementar a máquina de estados uma é a utilização de *scripts*, a outra maneira é fazer testes verificando o estado atual e executando a ação correspondente. Será feito um exemplo como desta última maneira. (MENDES, UNIDDEV:2003)

O algoritmo é simples:

- Verificar qual é o estado atual através de um descritor;
- Executar ação correspondente;
- Verificar se mudou o estado;
- Atualizar o descritor com o estado atual.

O descritor define qual o estado atual que o personagem está. Muitas vezes a ação correspondente para aquele estado não será executada uma única vez. Nesse caso pode-se utilizar *threads*<sup>52</sup> para que da próxima vez que for atingido aquele estado o guarda continuar do ponto onde parou. (MENDES, UNIDDEV:2003)

Abaixo está o algoritmo da máquina de estados do guarda.

```
função EXECUTA-IA (guarda){
    ESCOLHA (guarda.estado)
        caso RONDA: IARONDA(guarda)
        caso ALERTA: IAALERTA(guarda)
        caso ATACANDO: IAATACANDO(guarda)
        caso MORTO: IAMORTO(guarda)
        caso padrão: estado do guarda = RONDA
    FIM-ESCOLHA
fim
```

Essa função é executada periodicamente a cada quadro do jogo ou em um determinado tempo ele recebe o estado do guarda e dependendo do estado ele executa a função da ação correspondente. As funções IARONDA, IAALERTA, IAATACANDO e IAMORTO são onde estarão as funções de transição e as ações correspondentes aos estados. (*Idem*, UNIDDEV:2003)

Abaixo os algoritmos de cada função.

Função IARONDA.

```
função IARONDA (guarda){
    posição de destino do guarda = ESCOLHA-POSIÇÃO(posição do guarda)
    MOVE-GUARDA(posição de destino do guarda)
    SE ((dano = 1) OU (distancia < media E visível = 1)) E (chance < 90%)
        estado do guarda = ALERTA
    FIM-SE
fim
```

Essa função o guarda recebe uma nova posição para se locomover além de realizar a função de transição.

Função IAALERTA

```
função IAALERTA (guarda, inimigo){
```

---

<sup>52</sup> Programação concorrente

SE(vitalidade > 50)

posição de destino do guarda = posição do inimigo

SENÃO

fugir do inimigo

FIM-SE

SE (vitalidade < 0)

estado do guarda = MORTO

SENÃO SE (((distancia > media) E (chance < 90%)) OU ((visível = 0) E (tempo > tempo máximo)))

estado do guarda = RONDA

SENÃO SE ((distancia < perto) e (visível = 1) e (chance < 80%))

estado do guarda = ATACANDO

FIM-SE

fim

Nessa função o guarda persegue ou foge do inimigo, além de executar as funções de transição.

Função IAATACANDO

função IAATACANDO(guarda)

guarda ataca

SE (vitalidade < 0)

estado do guarda = MORTO

SENÃO SE (((distancia > perto) OU (dano > vitalidade x (0,50+sorteie ente (0 e 0,5)))) E (vitalidade > 0))

estado do guarda = ALERTA

FIM-SE

fim

Nessa função o guarda ataca e continuará atacando se as funções de transições não forem executadas.

Função IAMORTO

função IAMORTO(guarda)

guarda morre

fim

Nessa função o guarda simplesmente morre.

Essas são as funções executadas pelo guarda para cada estado definido.

As FSM's definem o que o personagem irá fazer e não como será feito. Para melhorar uma FSM, a melhor sugestão é acrescentarem mais estados tomando o cuidado de não criar estados redundantes. (MENDES, UNIDDEV:2003)

Há outras soluções também como definir sub-máquinas dentro de cada estado dando um refinamento na inteligência artificial, criar máquinas de estados evolutiva que se adaptam ao jogador, ou máquinas de estados *fuzzy* permitindo o personagem assumir mais de um estado ao mesmo tempo. (*Idem*, 2002)

A Máquina de Estados Finitos em IA sendo utilizada juntamente com o conceito de Agentes Inteligentes torna o jogo mais poderoso, pois o agente consulta a máquina de estados finitos para saber qual a ação a ser tomada, ou seja, a máquina de estados finitos é a inferência que o Agente Inteligente faz em suas regras.

### 5.5.3 Conjuntos e lógica *Fuzzy* (Difusa)

Na lógica tradicional os conjuntos são precisos ou o individuo pertence 100% a um grupo ou não pertence. Um grupo de pessoas com menos de 25 anos podem ser considerados participantes do grupo “jovem”, já as pessoas com mais de 25 anos serão considerados participantes do grupo “adulto”, a lógica *fuzzy* ou difusa permite que a pessoa tenha 100% de participação no grupo jovem e 20% no grupo adulto. (ROMERO & LACERDA, JEDI:2003)

A teoria dos conjuntos difusos (*fuzzy*) é o meio de simplificar o quanto um objeto satisfaz uma descrição vaga, por exemplo, considere a proposição “Nei é alto”. É verdade essa afirmação se Nei tem 1,87m, maioria das pessoas hesitariam para responder se sim ou não outras responderiam talvez. Isso não é uma questão de incerteza por que sabe-se a altura de Nei. A questão é que o termo “alto” não pode ser aplicado para demarcação nítida de objetos. Existem graus de altura, por essa razão, a teoria de conjuntos difusos não é de forma alguma um método para o raciocínio incerto. Em vez disso, a teoria de conjuntos difusos trata Alto como um predicado e afirma que o valor verdade de Alto(Nei) é um número entre 0 e 1, ao invés de ser simplesmente verdadeiro ou falso. O termo “conjuntos difusos” deriva da interpretação do predicado como a definição implícita do conjunto, o conjunto não tem limites precisos. (RUSSEL & NORVIG, 2004:511-512)

Conforme RUSSEL & NORVIG (2004:512), a lógica difusa (*fuzzy*) é um método de raciocínio com expressões lógicas descrevendo a pertinência aos conjuntos difusos. Por exemplo, a sentença  $\text{Alto}(\text{Nei}) \wedge \text{Pesado}(\text{Nei})$  possui um valor-verdade difuso que é uma

função dos valores de seus componentes. As regras para avaliação da verdade difusa,  $V$ , de uma sentença são:

$$V(A \wedge B) = \min(V(A), V(B))$$

$$V(A \vee B) = \max(V(A), V(B))$$

$$V(\neg A) = 1 - V(A)$$

Então, a lógica difusa é um sistema verdade-funcional, fato este que causa sérias dificuldades. Por exemplo, suponha que  $V(\text{Alto}(\text{Nei})) = 0,6$  e  $V(\text{Pesado}(\text{Nei})) = 0,4$ . Com base nas regras acima teremos então  $V(\text{Alto}(\text{Nei})) \wedge V(\text{Pesado}(\text{Nei})) = 0,4$ , parece razoável, mas o resultado também aparece em  $V(\text{Alto}(\text{Nei}) \wedge \neg \text{Alto}(\text{Nei})) = 0,4$ , que não parece nem um pouco razoável. Sem dúvida, o problema surge da inabilidade da abordagem verdade-funcional para levar em conta as correlações e anticorrelações entre os componentes propostos. (RUSSEL & NORVIG, 2004:511-512)

A popularização da lógica difusa vem do fato que os conjuntos difusos podem ser associados a variáveis lingüísticas, ou seja, podemos comprar um produto de um valor  $x$  e podemos possuir uma função que define o quão caro é esse produto. (TATAI, 2005:23)

As variáveis lingüísticas *fuzzy* são conceitos como temperatura, energia, distância. Elas são compostas por conjuntos que definem seus estados, por exemplo, a variável temperatura tem os estados alto, médio, baixo. A definição de cada conjunto é formada por uma função de participação. As formas mais comuns das funções são triangular e a trapezoidal. (ROMERO & LACERDA, JEDI:2003)

Depois que as variáveis lingüísticas são definidas para o modelo, é preciso passar por regras que definirão a ação a ser tomada. As regras geralmente usadas são comandos condicionais básicos como Se...Então. Como exemplo, tem-se duas variáveis representando as energias de dois personagens do jogo sendo feito uma análise definindo as ações que um dos personagens deve realizar. (*Idem*, JEDI:2003)

Ex.

SE (minha energia está baixa E energia do oponente está alta)

fugir

SE (minha energia está alta E energia do oponente está baixa)

atacar

### 5.5.3.1 Desfuzzificação

Ao final da análise é realizado o processo de desfuzzificação com o objetivo de converter o valor *fuzzy* para um valor discreto. Tendo dois métodos para esse processo o *Center of Gravity* e o *Mean of Maximum*. (FERNANDES, 2004:23)

### 5.5.3.2 *Center of Gravity*

O método *Center of Gravity* (Centro da Gravidade) é o mais utilizado. O método se baseia no cálculo do centro de gravidade ou de área da figura resultante das regras utilizadas. (OLIVEIRA & AGUIAR 1999 *apud* FERNANDES, 2004:23)

Com base nos valores de X e Y, obtidos por sensores, descobre-se quais os termos lingüísticos que são selecionados, o valor X está contido nos conjuntos das variáveis lingüísticas A1 e A2, já os valores de Y está contidos nos conjuntos das variáveis lingüísticas B1 e B2. Através da regra min, tem-se o grau de pertinência devendo ser projetado até a saída. Na saída utiliza-se o termo lingüístico descrito em cada uma das regras. (*Idem*, 23-24)

A figura resultante é obtida através da função max sob cada uma das figuras geradas pelas regras. Para obtenção de sua área utiliza-se o calculo simples de áreas geométricas como triângulos, trapézios e retângulos, pois a forma de calculo provou ser mais eficiente em comparação com os cálculos da área de figuras complexas como parábolas e senóides. (*Ibidem*, 24)

O valor da área é calculado sob a nova figura gerada, tendo em conta a pertinência do antecedente de menor valor. Deve ser calculado o ponto médio da base da cada figura do termo lingüístico do termo lingüístico da parte conseqüente da regra e aplicar a seguinte fórmula para se obter o centro da área Z. (FERNANDES 2004, 24)

$$coa = \frac{\sum_{i=1}^r Wi.Ai}{\sum_{i=1}^r Ai}$$

Onde:

r = número de regras

Wi = peso do termo de cada regra i

Ai = área do termo de cada regra i

Esta fórmula é a aproximação de outra mais complexa, que utiliza cálculos de integrais, porém essa provou na prática ser mais eficiente computacionalmente. (FERNANDES 2004, 24)

A figura demonstra todo o processo realizado até agora.

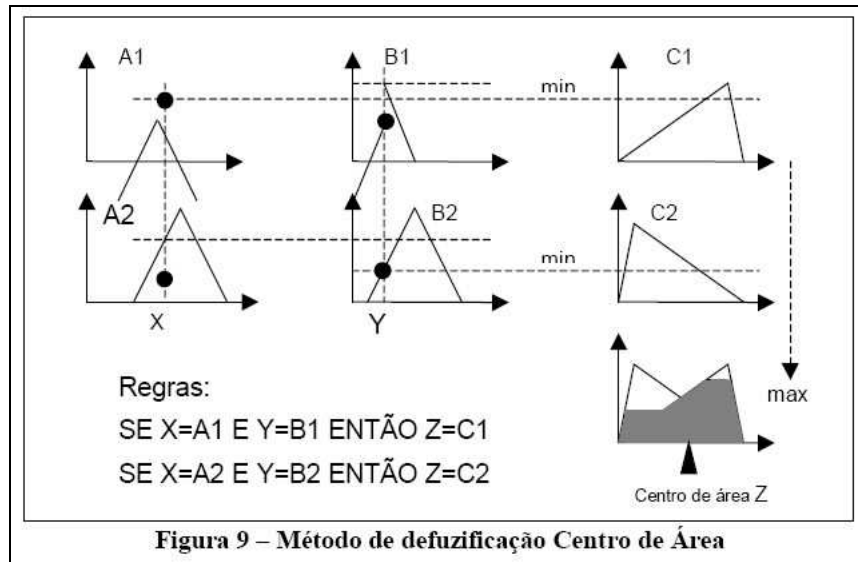


Figura 9 – Método de defuzzificação Centro de Área

### Figura 16: Método *Center of Gravity*

FONTE: FERNANDES, 2004:23

O método *Center of Gravity*, leva em conta o tamanho das figuras geradas. Pequenas variações nos valores passarão despercebidos, pelo sistema. Para alguns sistemas isso não pode acontecer, porém tem sistemas que com esta característica tornam-se robustos e imunes a variações, gerando uma oscilação menor na resposta ao processo. (*Idem*, 24)

Outro fato a ser considerado deve ser o tipo de processo de controle: os processos podem ser divididos em estáticos ou contínuos, no caso de processos contínuos é importante otimizar as rotinas de desfuzzificação, pois o processamento pode tornar lento. Logo a escolha do método do processo de desfuzzificação deve ser escolhida com atenção. (*Ibidem*, 24)

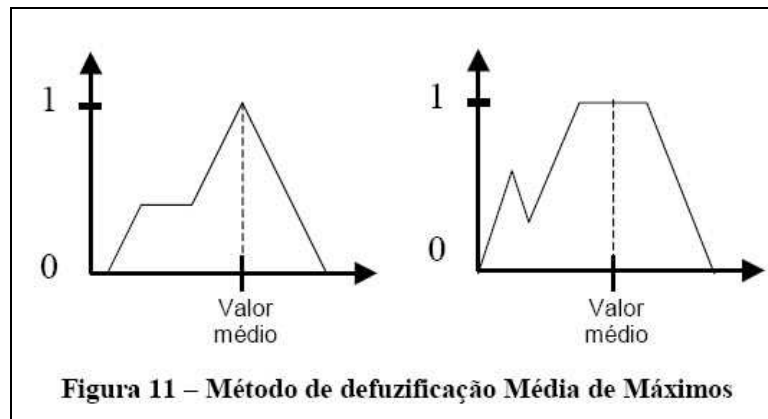
#### 5.5.3.2 *Mean of Maximum*

O método *Mean of Maximum* (Média dos Máximos) é muito usado em função da relação custo-benefício, devido ao seu custo computacional baixo e processamento eficiente em certos sistemas. Neste método podem surgir duas variações, baseada na função de pertinência no termo conseqüente: função de pertinência unimodal ou multimodal. (FERNANDES, 2004:24-25)

Na unimodal é utilizado o maior valor encontrado na função como valor defuzzificado. Quando a função trazer mais de um valor tira-se a média desses valores para definir o valor defuzzificado essa é a função multimodal. (FERNANDES, 2004:24-25)

Diferente da *Center of Gravity* o método *Mean of Maximum* leva em conta somente os valores de pico das funções de pertinência, não importando a forma que a função tome, isso pode acarretar uma representação incompleta da inferência. As ações de controle resultantes tendem a ser mais rápidas e sensíveis com as variações dos valores de entrada. Porém este método pode demonstrar descontinuidades, dependendo da função de pertinência, causando deslocamentos abruptos em decorrência das interações com o sistema. (*Idem*, 25)

A figura abaixo demonstra as duas funções de *Mean of Maximum*.



**Figura 17: Funções do Método *Mean of Maximum***  
 FONTE: FERNANDES, 2004:25

A seguir tem um exemplo de modelo *Fuzzy*, construído com a FFLL(*Free Fuzzy Logic Library*) uma biblioteca de lógica *Fuzzy* feito em C++, baseada nas especificações da FCL (*Fuzzy Control Language*).

```
FUNCTION_BLOCK
VAR_INPUT
    Our_Heath      REAL;
    Enemy_Health  REAL;
END_VAR
VAR_OUTPUT
    Aggressiveness REAL;
END_VAR
FUZZIFY Our_Health
```

```
    TERM Near_Death := (0,0) (0,1) (50,0);
    TERM Good := (14,0) (50,1) (83,0) ;
    TERM Excellent := (50,0) (100,1) (100,0) ;
END_FUZZIFY
FUZZIFY Enemy_Health
    TERM Near_Death := (0,0) (0,1) (50,0) ;
    TERM Good := (14,0) (50,1) (83,0) ;
    TERM Excellent := (50,0) (100,1) (100,0) ;
END_FUZZIFY
FUZZIFY Aggressiveness
    TERM Run_Away := 1;
    TERM Fight_Defensively := 2;
    TERM All_Out_Attack := 3;
END_FUZZIFY
DEFUZZIFY valve
    METHOD: MoM;
END_DEFUZZIFY
RULEBLOCK first
    AND: MIN;
    ACCU: MAX;
RULE 0: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Near_Death) THEN
        (Aggressiveness IS Fight_Defensively)
RULE 1: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Good)
        THEN (Aggressiveness IS Run_Away)
RULE 2: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Excellent) THEN
        (Aggressiveness IS Run_Away)
RULE 3: IF (Our_Health IS Good) AND (Enemy_Health IS Near_Death)
        THEN (Aggressiveness IS All_Out_Attack)
RULE 4: IF (Our_Health IS Good) AND (Enemy_Health IS Good)
        THEN (Aggressiveness IS Fight_Defensively)
RULE 5: IF (Our_Health IS Good) AND (Enemy_Health IS Excellent)
        THEN (Aggressiveness IS Fight_Defensively)
RULE 6: IF (Our_Health IS Excellent) AND (Enemy_Health IS Near_Death)
        THEN (Aggressiveness IS All_Out_Attack)
```

```

RULE 7: IF (Our_Health IS Excellent) AND (Enemy_Health IS Good)
        THEN (Aggressiveness IS All_Out_Attack)
RULE 8: IF (Our_Health IS Excellent) AND (Enemy_Health IS Excellent)
        THEN (Aggressiveness IS Fight_Defensively)
END_RULEBLOCK
END_FUNCTION_BLOCK

```

O modelo define primeiro as variáveis de entrada *Our\_Health*, *Enemy\_Health*. Depois define a variável de saída *Aggressiveness*, após isso ele define os valores dos conjuntos que serão componentes de cada variável. Sendo *Near\_Death*, *Good* e *Excelent* para as variáveis de entrada e *Run\_Away*, *Fight\_Defensively* e *All\_Out\_Attack* para a variável de saída. Sendo assim, ele define como método de desfuzzificação o *Mean of Maximum*, após isso aplica as regras do modelo definindo qual o valor a variável de retorno terá.

A lógica nebulosa está presente atualmente em diversas áreas de aplicação, sendo as principais o controle, automação de processos e recuperação de informações dentre outros, as formas de utilização de lógica *fuzzy* varia muito: tem sistemas de controle nebulosos, bases de inferências nebulosas, máquinas de estados finitos nebulosas entre outras.

### 5.5.3.2 Máquinas de Estados Finitos *Fuzzy* (FuSM)

Conforme TATAI (2003:22), máquina de estados finitos *Fuzzy* é de interesse aos desenvolvedores de jogos, pois a própria é uma evolução da máquina de estados finitos, método muito utilizado em jogos. Existem diversas implementações possíveis para as FuSM's, porém as mesmas são caracterizadas por atribuir valores nebulosos a um determinado estado. Isso tem diversas implicações sendo a principal, o fato de se ter que definir um comportamento dos valores nebulosos quando é executado um evento. Normalmente esse evento também possui um valor nebuloso, tornando assim intuitivamente o uso de regras que combine este valor com o valor atual e para obter o próximo valor de pertinência.

Tipicamente a regra utilizada está associada ao sistema de FuSM, dando a responsabilidade ao projetista determinar qual regra será utilizada por qual evento lançado. (TATAI, 2003:25)

Podemos também utilizar a FuSM de modo que um sistema não assuma somente um estado, mas possa assumir vários estados com graus de pertinências distintos ao mesmo

tempo, porém isso pode gerar complicações na determinação das ações tomadas e na compreensão do funcionamento do sistema. (TATAI, 2003:25)

Em jogos de estratégia a FuSM é utilizada, pois com a pouca informação que se tem do inimigo e a quantidade de micro decisões a serem tomadas faz que o oponente de IA tome decisões quase que com a certeza de vitória. Por exemplo, o adversário não sabe a quantidade de unidades do exército e nem as reservas de ouro do jogador, porém acredita na possibilidade da vitória. (*Idem*, 12)

Em jogos FTPS as FuSM's é utilizado devido ao número de variáveis *fuzzy* ser bem baixo, diminuindo assim a complexidade do sistema, evitando problemas de cálculos prejudiciais ao sistemas *fuzzy*. Os dados obtidos para os personagens de um FTPS não é necessariamente preciso ser regular. Um oponente com 23% de saúde e uma arma ótima e conseguir perseguir o jogador sem ser notado. Ele poderia atacar, mesmo estando machucado ele está com uma arma superior ao do jogador e consegue segui-lo sem ser notado isso deixa as chances dele de ganhar bem grandes. Entretanto isso só acontece quando é utilizado um sistema de entradas *fuzzy*. (*Ibidem*, 15)

Em jogos de esporte são mais fáceis de adaptar com máquinas de estados finitos do que os jogos dinâmicos. Todos os jogos desse gênero já têm regras pré-definidas, por exemplo, o futebol tem os estados: lateral, escanteio, falta, pênalti. A estrutura da IA desses jogos é baseado em estados, porém decisões tomadas pelo técnico ou pelo próprio jogador se tornam nebulosas. Nesse caso as FuSM's podem prover a tomada de decisão nebulosa. (TATAI, 2003:19)

A Lógica fuzzy é bem antiga, porém bem poderosa a utilização dela em jogos é nova, pois seu processamento pode sobrecarregar o jogo , ela está sendo usada como uma evolução da Máquina de Estados Finitos.

#### **5.5.4 Algoritmos de Busca**

Segundo PERÚCIA (2005:154), a utilização da IA para a solução de problemas em jogos é muito difundido. Por exemplo em jogos de tabuleiro (xadrez, damas) o computador deve ser capaz de calcular qual a melhor jogada a ser realizada contra o jogador, para isso tem diversos algoritmos de busca de IA que realizam uma consulta as diversas possíveis soluções para o problema escolhendo a melhor dependendo do nível imposto contra o jogador.

Seguindo essa analogia, ela também pode ser utilizada em jogos com entidades que procuram melhores caminhos para atingir certos objetivos, Jogos como *Warcraft III*, tem diversas unidades que percorrem rios, pontes, montanhas etc, nesses jogos devem haver um algoritmo capaz de gerar o melhor caminho para cada unidade. (PERÚCIA *et. al.*, 2005:154)

Dentre todos esses algoritmos de busca utilizados em jogos os que se destacam são MiniMax e o de A\* (A - Estrela).

#### 5.5.4.1 Algoritmo MiniMax

Um dos algoritmos mais utilizados em jogos de tabuleiro tem como objetivo percorrer todas as possíveis jogadas dependendo de quanto se quer pesquisar para encontrar a melhor jogada a ser realizada. (PERÚCIA, 2004:154)

Considere um jogo com dois jogadores o MAX e MIN, o jogo começa com uma jogada de MAX e os dois jogadores se revezam até o jogo terminar. Ao terminar o jogo pontos são dados ao jogador vencedor e o perdedor sofre penalidades. Um jogo desse tipo pode ser definido como um problema de busca com os seguintes componentes: (RUSSEL & NORVIG, 2004:157)

- Estado inicial;
- Função sucessor;
- Teste de término;
- Função utilidade.

Estado inicial: inclui a posição do tabuleiro e identifica o jogador que fará o movimento. (*Idem*, 158)

Função sucessor: retorna uma lista de pares (movimento, estado), cada qual indicando um movimento válido e o estado resultante (*Ibidem*, 158)

Teste de término: determina quando o jogo termina. Os estados que o jogo é encerrado são chamados estados terminais. (RUSSEL & NORVIG, 2004: 158)

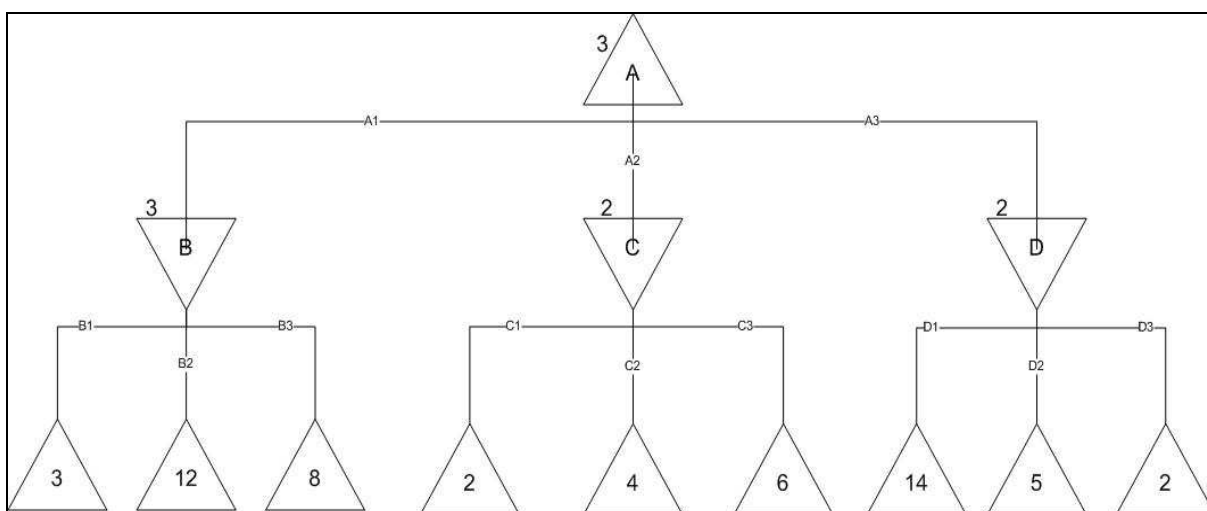
Função utilidade: conhecida também como função objetivo, dá um valor numérico aos estados terminais. No xadrez, o resultado pode ser uma vitória, uma derrota ou um empate tendo os valores 1, -1 ou 0. Alguns jogos tem uma variedade de pontos mais ampla como o gamão que varia de 192 a -192. (*Idem*, 158)

O estado inicial e todos os movimentos possíveis para cada jogador formam a árvore do jogo que corresponde todo o jogo. Num jogo da velha a partir do estado inicial MAX tem-

se nove movimentos possíveis. O jogo vai se alterando com a colocação de um X por MAX e a colocação de um O por MIN até ser alcançado nós de folhas que correspondem ao estado terminal, ou seja, o jogador deve ter três símbolos em uma linha ou todos os quadrados são preenchidos. O número em cada nó representa o valor de utilidade do estado terminal para o jogador, valores altos são bons para MAX e valores baixos são bons para MIN, razão para os nomes dos jogadores, então cabe a MAX usar a árvore de busca com base nos estados terminais gerados para escolher a melhor jogada. (RUSSEL & NORVIG, 2004: 158)

O algoritmo MiniMax calcula a decisão MiniMax a partir do estado corrente, utilizando computação recursiva simples dos valores MiniMax de cada estado sucessor, implementando diretamente equações necessárias. O método recursivo percorre um caminho descendente até chegar às folhas da árvore e depois os valores encontrados são retornados até o nó raiz do algoritmo de busca. (*Idem*, 160)

A figura abaixo é de uma árvore tendo a aplicação do MiniMax.



**Figura 18: Exemplo da Aplicação de *MiniMax***

FONTE: Adaptado de RUSSEL & NORVIG, 2004:159

Primeiro o algoritmo desce até o nível de folha que se quer obter a melhor resposta e utiliza a função UTILIDADE para descobrir os valores de cada folha sendo que no galho B temos os valores 3, 12, e 8 tendo escolhido o mínimo entre eles o 3 e manda para B com um valor, o mesmo processo acontece nos galhos C e D que forneceram o valor de 2 cada um. Por fim obtemos o valor máximo entre 3, 2 e 2 que é 3 e enviamos para o nó raiz A. (*Ibidem*, 2004:160)

O algoritmo MiniMax executa uma exploração completa em profundidade de toda a árvore do jogo. Se a profundidade é  $m$  e existem  $b$  movimentos o tempo que o algoritmo

demorará em executar é de  $O(b^m)$ . A complexidade do espaço é definida por  $O(bm)$  para um algoritmo que gera todos os sucessores de uma vez ou  $O(m)$  para um algoritmo que gera um sucessor por vez. É claro que em jogos, o custo de tempo é totalmente impraticável, mas o algoritmo serve como base para análise matemática de jogos e algoritmos práticos. (RUSSEL & NORVIG, 2004:160)

Abaixo o algoritmo MiniMax.

Função DECISÃO-MINIMAX

função DECISÃO-MINIMAX(estado) retorna uma ação

entradas: estado, estado corrente do jogo

$v = \text{MAX}(\text{estado})$

retornar a ação em SUCESSORES(estado) com valor  $v$

fim

Essa função retorna uma ação de acordo com o valor MAX obtido do estado corrente do jogo.

Função MAX

função MAX (estado) retorna um valor de utilidade

SE(TESTE-TERMINAL(estado))

retornar UTILIDADE(estado)

FIM-SE

$v = -\infty$

PARA a, s em SUCESSORES(estado) FAÇA

$v = \text{MAX}(v, \text{MIN}(s))$

FIM-PARA

retornar  $v$

fim

Essa função retorna o valor de utilidade do estado atual do jogo se este for um estado terminal ou retorna o valor MAX entre  $-\infty$  e todos os nós daquele estado.

Função MIN

função MAX (estado) retorna um valor de utilidade

SE(TESTE-TERMINAL(estado))

retornar UTILIDADE(estado)

FIM-SE

$v = +\infty$

PARA a, s em SUCESSORES(estado) FAÇA

$$v = \text{MIN}(v, \text{MAX}(s))$$

FIM-PARA

retornar v

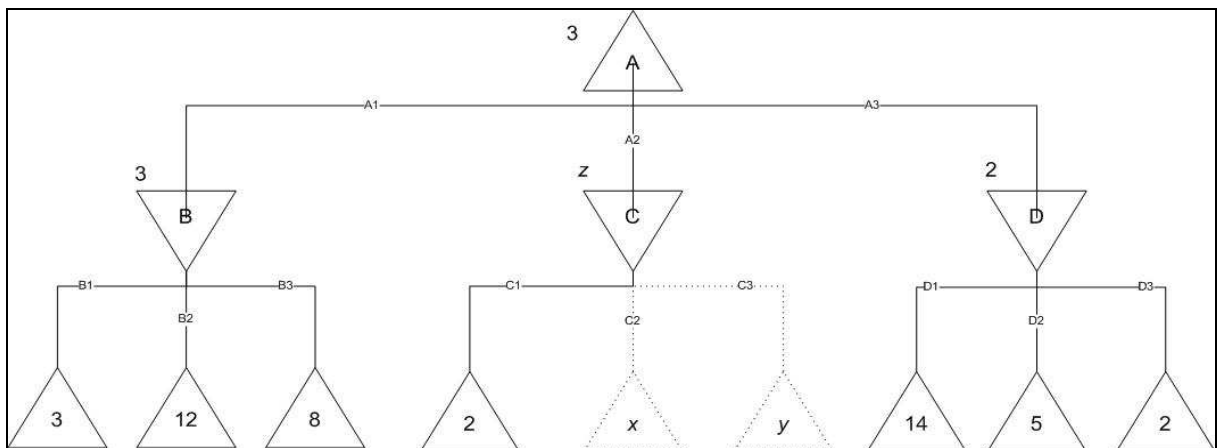
fim

Essa função retorna o valor de utilidade do estado atual do jogo se este for um estado terminal ou retorna o valor MIN entre  $\infty$  e todos os nós daquele estado.

### 5.5.4.1.1 Poda alfa-beta

Para RUSSEL & NORVIG (2004:162), o grande problema do algoritmo MiniMax é o número de estados do jogo examinados é exponencial em relação ao número de movimentos. Infelizmente não se pode eliminar o expoente, mas pode-se reduzi-lo pela metade. O método é poder calcular a decisão MiniMax sem examinar todos os nós do jogo, a fim de deixar de considerar grandes partes da árvore, utiliza-se a técnica de poda alfa-beta. Quando aplicada em uma árvore MiniMax ela retornará o mesmo resultado que MiniMax retornaria, mas poda os nós que não teriam influência a tomada de decisão da jogada.

A figura abaixo é a árvore da Figura 18, mas o algoritmo MiniMax está com a função de Poda alfa-beta.



**Figura 19: Algoritmo MiniMax com poda alfa-beta**

FONTE: Adaptado de RUSSEL & NORVIG, 2004:162

Primeiro o algoritmo busca os valores de B 3, 12 e 8, escolhendo o valor MIN como 3, após isso move-se para o nó C o primeiro valor obtido é o 2, sabendo que B tem valor 3, MAX nunca escolheria C, pois o único valor que poderia sair de toda a análise de C seria 2 ou menor que 2 e MAX escolheria o maior valor entre 3 (obtido de B) e os valores obtidos de C

& D, então não há razão para continuar a explorar o nó C. Esse é um exemplo de poda alfa-beta. Movendo agora para o nó D o primeiro valor obtido é 14, esse valor é maior que 3 que seria a melhor escolha de MAX, por isso é preciso continuar a análise de D, continuando a análise em D encontra-se o valor 5 que agora é definido como o valor de D esse valor ainda é maior que 3, assim ainda precisamos continuar a avaliar D, o terceiro sucessor é 2, agora D vale 2. A decisão MAX na raiz é entre 3 e 2, MAX decide que a melhor jogada a ser realizada é o nó B que dá valor 3. (RUSSEL & NORVIG, 23004:162)

Esse algoritmo pode ser visto com uma simplificação da fórmula Valor-MiniMax. Sejam  $x$  e  $y$  o valor dos dois sucessores não avaliados em C, e seja  $z$  o nó mínimo entre  $x$  e  $y$ . o valor nó da raiz é dado por:

$$\text{VALOR-MINIMAX}(\text{raiz}) = \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))$$

$$\text{VALOR-MINIMAX}(\text{raiz}) = \max(3, \min(2,x,y), 2)$$

$$\text{VALOR-MINIMAX}(\text{raiz}) = \max(3,z,2) \quad \text{onde } z \leq 2$$

$$\text{VALOR-MINIMAX}(\text{raiz}) = 3$$

Ou seja, o valor da raiz e a decisão MiniMax são independentes das folhas podadas  $x$  e  $y$ .

A poda alfa-beta é aplicada em árvores de qualquer profundidade, sendo possível podar subárvores inteiras no lugar de apenas folhas. O princípio geral da função é esse: considerando um nó  $n$  em algum lugar da árvore, tal que o jogador tenha a escolha até esse nó, se o jogador tiver uma escolha melhor no nó pai de  $n$  ou em qualquer outro nó acima dele então  $n$  nunca será alcançado em um jogo real. Sendo uma vez que descobriremos o suficiente de  $n$  para chegar a conclusão de podá-lo. (*Idem*, 163)

Lembrando que o algoritmo MiniMax é em profundidade, então a qualquer momento temos somente ter de considerar nós ao longo de um único caminho na árvore. A poda alfa-beta recebe esse nome a partir de dois parâmetros que descrevem os limites de valores propagados de volta que aparecem em qualquer lugar ao longo do caminho: (*Ibidem*, 163)

$\alpha$  = o valor da melhor escolha ao longo do caminho MAX

$\beta$  = o valor da melhor escolha ao longo do caminho MIN

A poda alfa-beta atualiza os valores de  $\alpha$  e  $\beta$  à medida que se prossegue e as ramificações são podadas, logo sabe-se que o valor do nó corrente é pior que os valores de  $\alpha$  e  $\beta$  para MAX ou MIN respectivamente. (RUSSEL & NORVIG, 2004:163)

Abaixo está o algoritmo MiniMax com a poda alfa-beta inclusa

Função BUSCA-ALFA-BETA

função BUSCA-ALFA-BETA(estados) retorna uma ação

entradas: estado, estado corrente em jogo

$v = \text{MAX}(\text{estado}, -\infty, +\infty)$

retornar a ação em  $\text{SUCESSORES}(\text{estado})$  com valor  $v$

fim

Essa função retorna uma ação com base no valor MAX entre o estado atual e  $-\infty, \infty$ .

Função MAX

função MAX (estado, $\alpha$ , $\beta$ ) retorna um valor de utilidade

entradas: estado, estado corrente em jogo

$\alpha$ , o valor da melhor escolha para MAX até o estado atual

$\beta$ , o valor da melhor escolha para MIN até o estado atual

SE( $\text{TESTE-TERMINAL}(\text{estado})$ )

retornar UTILIDADE(estados)

FIM-SE

$v = -\infty$

PARA  $a, s$  em  $\text{SUCESSORES}(\text{estado})$  FAÇA

$v = \text{MAX}(v, \text{MIN}(s, \alpha, \beta))$

SE( $v \geq \beta$ )

retornar  $v$

FIM-SE

$\alpha = \text{MAX}(\alpha, v)$

FIM-PARA

retornar  $v$

fim

Essa função retorna o valor de utilidade do estado atual do jogo se este for um estado terminal ou retorna o valor MAX entre  $-\infty$  e todos os valores MIN de  $s, \alpha, \beta$  daquele estado.

Função MIN

função MIN (estado, $\alpha$ , $\beta$ ) retorna um valor de utilidade

entradas: estado, estado corrente em jogo

$\alpha$ , o valor da melhor escolha para MAX até o estado atual

$\beta$ , o valor da melhor escolha para MIN até o estado atual

SE( $\text{TESTE-TERMINAL}(\text{estado})$ )

retornar UTILIDADE(estados)

FIM-SE

$v = +\infty$

```

PARA a, s em SUCESSORES(estado) FAÇA
    v = MIN(v, MAX(s,α,β))
    SE(v ≤ α)
        retornar v
    FIM-SE
    β = MIN(β,v)
FIM-PARA
retornar v
fim

```

Essa função retorna o valor de utilidade do estado atual do jogo se este for um estado terminal ou retorna o valor MIN entre  $\infty$  e todos os valores MAX de  $s, \alpha, \beta$  daquele estado.

A poda alfa-beta tem seu efeito dependente da ordem que é feita a análise dos sucessores, não se pode podar quaisquer sucessores de D, porque os primeiros sucessores gerados foram os piores do ponto de vista de MIN. Se o terceiro sucessor fosse analisado primeiro, D poderia ser podado, Isso da idéia que pode valer a pena avaliar os melhores sucessores primeiro. (RUSSEL & NORVIG, 2004:164)

Supondo que isso seja feito, então alfa-beta precisará examinar apenas  $O(b^{d/2})$  nós para escolher entre o melhor, em vez de  $O(b^d)$  para MiniMax, significando um fator de ramificação efetivo igual a  $\sqrt{b}$  em vez de  $b$ , no caso do xadrez 6 em vez de 36, ou seja a poda alfa-beta poderá examinar antecipadamente uma distância duas vezes maior que a do MiniMax no mesmo período de tempo. E os sucessores forem examinados em ordem aleatória, ao invés de seguir uma ordem lógica o número de nós examinados seria de  $O(b^{3d/4})$ . No caso do xadrez com uma ordenação simples de experimentar primeiros as capturas, depois ameaças, depois movimentos para frente, e em seguida os movimentos para trás levará a uma profundidade duas vezes maior que  $O(b^{d/2})$ . (*Idem*, 164-165)

O algoritmo de MiniMax foi um dos primeiros algoritmos de IA a chegar nos jogos, ele é muito utilizado em jogos de tabuleiro como xadrez, damas, gamão, jogo-da-velha e etc.

#### 5.5.4.2 Algoritmo A\* (A-Star) *pathfinding*

Dos algoritmos de busca disponíveis o A-Star é o mais utilizado em jogos eletrônicos, sendo que os próprios desenvolvedores fazem suas próprias versões do algoritmo. O A\* é um algoritmo onde utiliza uma função heurística que determina a qualidade de cada um dos

estados possíveis, por meio de um custo para a melhor rota até o destino passando pelo nó atual. Esse custo determina a qualidade do caminho, quanto menor o custo melhor é o caminho. (PERÚCIA *et. al.*, 2005:156)

Pelo seu funcionamento o A\* é um algoritmo completo e ótimo, ou seja, dada uma função heurística, se o problema tiver uma resposta, ele vai ser encontrado e sempre será o de menor custo. Essa técnica é amplamente utilizada para jogos de estratégia, onde os agentes devem procurar a melhor rota para se deslocar no cenário essa busca é conhecida como *path finding*, pode-se utilizar a análise do terreno para aumentar ou diminuir o custo do caminho, onde pontos do mapa são identificados como pontes, travessias ou rios. (*Idem*, 56)

A busca A\* é a forma de busca pela melhor escolha mais conhecida. Ela avalia nós combinando  $g(n)$ , o custo para alcançar cada nó, e  $h(n)$ , o custo para ir do nó até o objetivo: (RUSSEL & NORVIG, 2004:97)

$$f(n) = g(n) + h(n)$$

Conforme RUSSEL & NORVIG (2004:97),  $g(n)$  é o custo do caminho do nó inicial até o nó atual, e  $h(n)$  é o custo do caminho entre o nó atual e o nó de destino, temos:

$$f(n) = \text{custo estimado da solução de custo mais baixo passando pelo nó atual.}$$

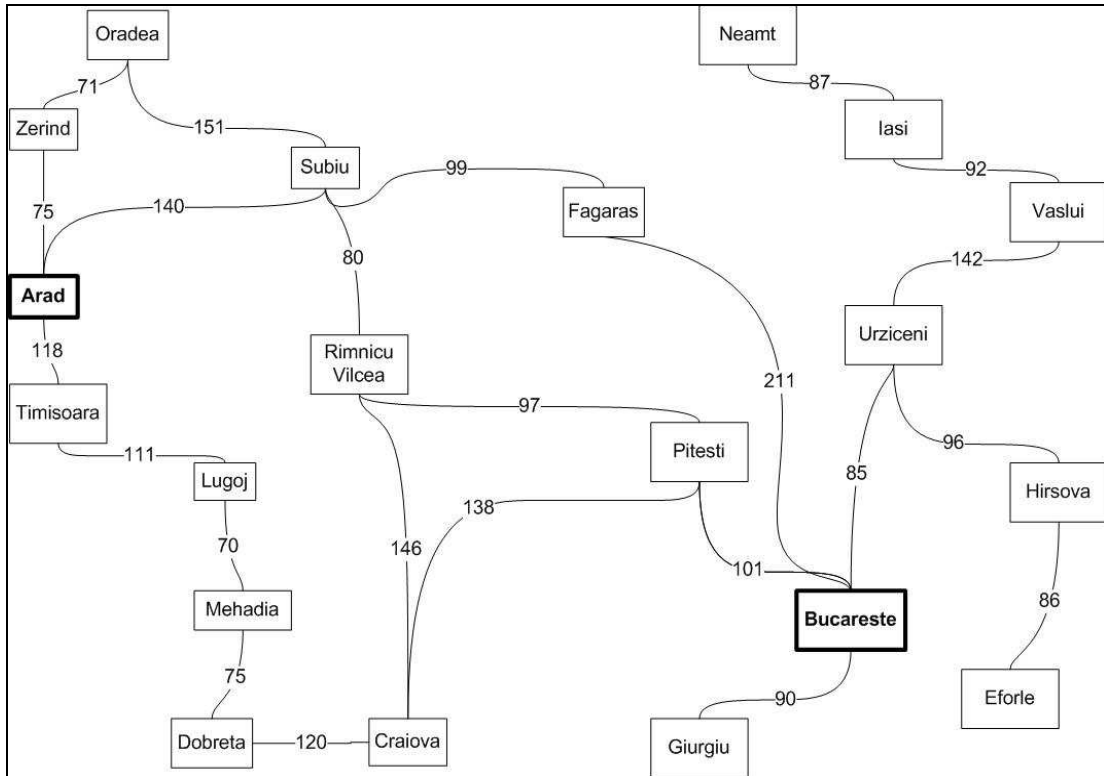
Desse modo, se tenta encontrar a solução de custo mais baixo, a opção razoável seria experimentar o menor valor de  $g(n) + h(n)$ . Realmente essa estratégia é mais do que razoável. Desde que a função heurística  $h(n)$  satisfaça certas condições, a busca A\* será ao mesmo tempo completa e ótima. (*Idem*, 97)

A análise do caráter ótimo de A\* se usada com uma BUSCA-EM-ÁRVORE. Nesse caso A\* será ótimo se  $h(n)$  seja uma heurística admissível, ou seja, desde que  $h(n)$  supere o seu custo para alcançar o objetivo. Heurísticas são sempre otimistas por natureza, pois sempre estimam um custo menor de solução do que é na realidade. Com isso como  $g(n)$  é o custo verdadeiro de se chegar ao nó atual, tem como consequência  $f(n)$  não superestimar o custo de uma solução passando por aquele nó. (*Ibidem*, 97)

Imagine alguém na cidade de Arad, na Romênia, ele aproveita as suas férias, aproveitando os pontos turísticos da cidade e tudo mais, porém ele tem uma passagem não-reembonsável para sair de Bucareste no dia seguinte, então ele toma como um objetivo chegar a Bucareste, porém ele não sabe como chegar a Bucareste o mais rápido possível. (RUSSEL & NORVIG, 2004:62)

De Arad saem três estradas uma para Sibiu, uma para Timisoara e uma para Zerind, nenhuma delas leva diretamente para Bucareste, para isso ele precisa saber qual cidade dentre essas três cidades fica mais próxima de Bucareste para escolher o seu caminho. (*Idem*, 62)

A figura abaixo demonstra um mapa rodoviário da Romênia de modo simplificado demonstrando o ponto de partida Arad e o ponto de destino Bucareste em destaque, além das distâncias entre cada cidade.



**Figura 20: Mapa rodoviário simplificado da Romênia**

FONTE: Adaptado de RUSSEL & NORVIG, 2004:65

Como exemplo de heurística será utilizado a distância em linha reta  $h_{DLR}$ , pois a menor distância entre dois pontos é uma reta, sendo assim a reta não pode ser superestimada. (RUSSEL & NORVIG, 2004:97)

A tabela abaixo demonstra os valores das distancias em linha reta  $h_{DLR}$ .

**TABELA 4: Distâncias  $h_{DLR}$  das cidades da Romênia**

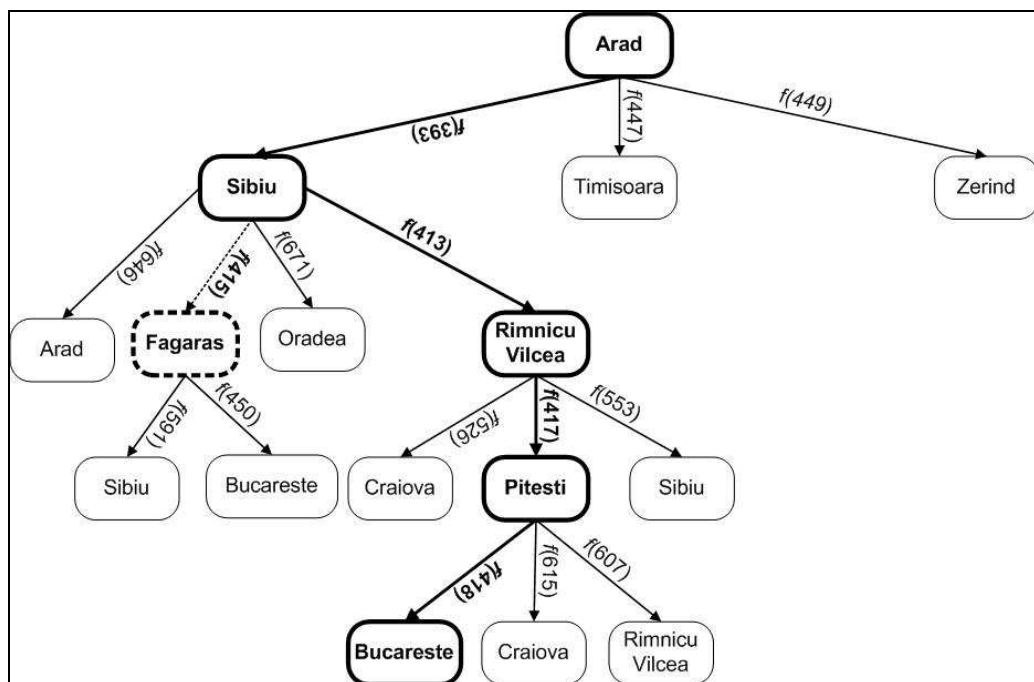
Cidade	Distância	Cidade	Distância
Arad	366	Mehadia	241
Bucareste	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urzecini	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

FONTE: Adaptado de RUSSEL & NORVIG, 2004:96

O algoritmo A\* para Bucareste é executado ele obtém os valores de  $g(n)$  com base nos dados da Tabela 4 e da Figura 20.

O algoritmo expande de Arad exibindo Sibiu  $f(393)$ , Timisoara  $f(447)$  e Zerind  $f(449)$ , Sibiu é a cidade escolhida pois é a que possui o menor  $f(n)$ . Agora Sibiu é expandida obtendo as seguintes cidades com seus respectivos  $f(n)$ : Arad  $f(646)$ , Fagaras  $f(415)$ , Oradea  $f(671)$ , Rimnicu Vilcea  $f(413)$ , essas cidades se juntam a Timisoara e Zerind, para a próxima escolha, sendo escolhida a cidade de Rimnicu Vilcea  $f(413)$ , com a expansão de Rimnicu Vilcea as cidades de Craiova  $f(526)$ , Pitesti  $f(417)$  são adicionadas e Sibiu  $f(553)$  aparece novamente, porém com outro valor, nessa expansão dentre as cidades foi escolhida a cidade de Fagaras por apresentar o menor  $f(n)$ , nessa iteração  $f(415)$ , já obtida na expansão anterior. Com a expansão de Fagaras a cidade Sibiu aparece novamente com o valor de  $f(591)$  e a cidade destino Bucareste aparece com o valor de  $f(450)$ , porém dentre as cidades que possam ser escolhidas a que possui o menor valor de  $f(n)$  é Pitesti com o valor de  $f(417)$ . Expandindo Pitesti aparecem novamente as seguintes cidades com os valores de  $f(n)$  Bucareste  $f(418)$ , Craiova  $f(615)$  e Rimnicu Vilcea  $f(607)$  dentre todas as cidades presentes agora a próxima escolhida será Bucareste cujo  $f(418)$  é a menor dentre todas, chegando assim ao destino. (RUSSEL & NORFIVG, 2004:98)

A figura abaixo demonstra o último processo do A\*, mostrando os valores de  $f(n)$  de cada cidade e indicando o melhor caminho a ser tomado.



**Figura 21: Algoritmo A\* (A-Star) em ação**  
 FONTE: Adaptado de RUSSEL & NORVIG, 2004:98

Percebe-se que Bucareste a cidade de destino aparece na expansão de Fagaras, porém não foi escolhida, pois seu valor de  $f(450)$  é mais alto do que de Pitesti  $f(417)$ , isso significa que talvez tenha uma melhor solução para chegar a Bucareste passando por Pitesti. Isso prova que  $A^*$  em uma BUSCA-EM-ÁRVORE é ótima se  $h(n)$  é admissível. Suponha que um nó objetivo não-ótimo  $G_2$  apareça na borda e tenha um custo  $C^*$  da solução ótima. Então com o  $G_2$  não é ótimo de  $h(G_2) = 0$ , sabemos que (RUSSEL & NORVIG, 2004:99):

$$f(G_2) = g(G_2) + h(G_2) = G(G_2) > C^*$$

Considerando agora, um nó de borda  $n$  que está no caminho de uma solução ótima, por exemplo, Pitesti. Sempre deve haver tal nó se existir uma solução. Se  $h(n)$  não superestimar o valor do caminho de solução, sabemos que (*Idem*, 99):

$$f(n) = g(n) + h(n) \leq C^*$$

Isso demonstra que  $f(n) \leq C^* < f(G_2)$ , e assim o nó  $G_2$  não será expandido e  $A^*$  deve retornar a solução ótima.

Se ao invés de utilizar como base a BUSCA-EM-ÁRVORE e utilizar a BUSCA-EM-GRAFO, essa prova será derrubada, pois o algoritmo de BUSCA-EM-GRAFO pode descartar uma solução ótima, pelo seguinte motivo, em BUSCA-EM-GRAFO descarta-se um estado repetido, se o estado da solução ótima não for gerado primeiro ele será descartado. (*Ibidem*, 99)

Para corrigir este problema tem-se duas maneiras, as primeira solução é estender a BUSCA-EM-GRAFO para que ele consiga analisar os caminhos gerados pelo mesmo nó conseguindo dispensar o caminho de maior custo, a segunda solução é garantir que o primeiro caminho encontrado de qualquer nó repetido seja o melhor. Essa propriedade será válida se for imposta o requisito de consistência ou de monotonicidade<sup>53</sup>. Uma heurística  $h(n)$  é consistente se para todo nó  $n$  e todo seu sucessor  $n'$  gerado por qualquer ação, o custo do objetivo gerado por  $n$  não seja maior que  $n'$  somado ao custo de alcançar o objetivo a partir de  $n'$ : (RUSSEL & NORVIG, 2004:99)

$$h(n) \leq c(n,a,n') + h(n')$$

Essa é uma forma de desigualdade de triângulos geral que diz que cada lado de um triângulo não pode ser maior que a soma dos outros dois lados, nesse caso o triângulo é formado por  $n$ ,  $n'$  e o objetivo mais próximo de  $n$ . É fácil mostrar que toda heurística consistente também é admissível. A consequência mais importante de consistência é:  $A^*$  usando BUSCA-EM-GRAFO é ótimo se  $h(n)$  é consistente. (*Idem*, 2004:99)

<sup>53</sup> Similaridade de dois objetos crescendo com o aumento de correspondência e a redução da diferenças.

Embora a consistência seja um requisito mais rígido que a admissibilidade, precisa-se de muito trabalho para desenvolver uma heurística admissível, porém não consistente, por exemplo,  $h_{DLR}$ , sabemos que a desigualdade de triângulos geral é satisfeita quando cada lado é medida pela distância em linha reta, e a distância em linha reta de  $n$ ,  $n'$  não seja maior que  $c(n,a,n')$ . em conseqüência  $h_{DLR}$  é uma heurística consistente. (RUSSEL & NORVIG, 2004:99)

Abaixo está um exemplo do algoritmo de A\* em Java

```
package aimasearch.informed;
import java.util.Comparator;
import aimasearch.framework.Metrics;
import aimasearch.framework.Node;
import aimasearch.framework.PrioritySearch;
import aimasearch.framework.Problem;
import aimasearch.framework.QueueSearch;
public class AStarSearch extends PrioritySearch {
    public AStarSearch(QueueSearch search) {
        this.search = search;
    }
    class NodeComparator implements Comparator {
        private Problem problem;
        NodeComparator(Problem problem) {
            this.problem = problem;
        }
        public int compare(Object aNode, Object anotherNode) {
            Node one = (Node) aNode;
            Node two = (Node) anotherNode;
            int h1 = problem.getHeuristicFunction().getHeuristicValue(
                one.getState());
            double g1 = one.getPathCost();
            int h2 = problem.getHeuristicFunction().getHeuristicValue(
                two.getState());
            double g2 = two.getPathCost();
            double s1 = g1 + h1;
            double s2 = g2 + h2;
```

```

        if (s1 == s2) { return 0; }
        else if (s1 < s2) { return -1;}
        else { return 1; }
    }
}
public Metrics getMetrics() {
    return search.getMetrics();
}
protected Comparator getComparator(Problem p) {
    return new NodeComparator(p);
}
}

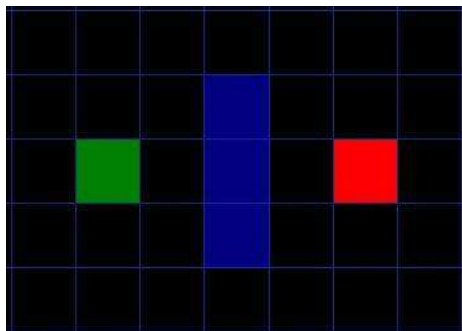
```

Neste exemplo a classe AStarSearch compara o custo de dois nós retornando 0 se forem iguais, -1 se o primeiro nó for menor que o segundo e 1 se o primeiro nó for maior que o segundo, o nó definido menor novamente passará por essa classe, porém com um outro nó para verificar que tem o custo menor, para ser adicionado ao caminho da solução ótima.

#### 5.5.4.2.1 Algoritmo A\* (A-Star) *pathfinding* nos jogos

O algoritmo de A\* em jogos é um pouco diferente do normal, pois ele precisa ser tratado de uma forma diferente, por exemplo, tem-se um mapa e um personagem esse personagem está num ponto A e precisa ir para um ponto B entre esses dois pontos há uma parede. A primeira coisa a se fazer é escanear o mapa dividindo ele em pedaços iguais. (LESTER, POLICYALMANAC:2004)

A figura abaixo mostra o mapa sendo dividido.



**Figura 22: Mapa escaneado em quadrados**  
 FONTE: LESTER, POLICYALMANAC:2004

Foi utilizado quadrados para dividir o mapa, porém pode-se utilizar qualquer forma geométrica para dividir o mapa triângulo, trapézio, hexágonos entre outros. Esses quadrados serão os nós para a escolha do caminho. (LESTER, POLICYALMANAC:2004)

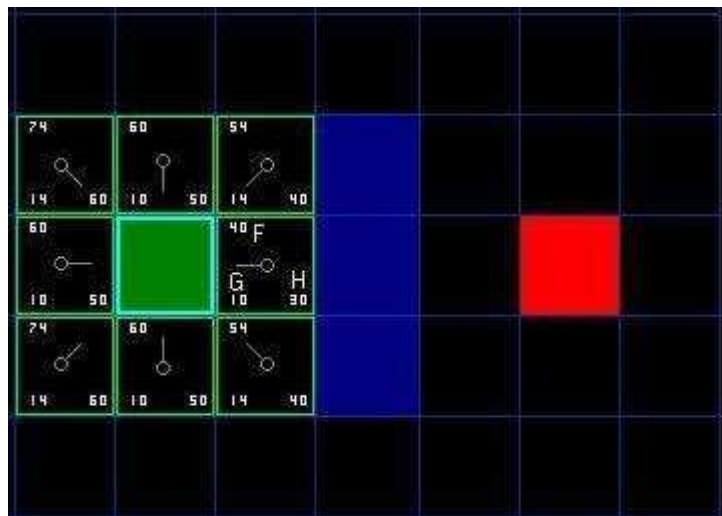
O próximo passo é definir os valores de  $f(n)$ ,  $g(n)$  e  $h(n)$  para cada  $n$ .  $f(n)$  continua o mesmo:

$$f(n) = g(n) + h(n)$$

Para  $g(n)$ , que é a distância do quadrado corrente para o próximo quadrado sendo definido com dois valores 10 e 14, 10 para quadrados que estejam na horizontal ou vertical e 14 para os quadrados que estão pelas diagonais. (Idem, POLICYALMANAC:2004)

Para  $h(n)$ , que é a distância entre o quadrado corrente e o quadrado de destino é definido como 10 vezes o número de quadrados que se leva para ir do quadrado atual até o quadrado de destino ignorando obstáculos e qualquer movimento diagonal, esse método é conhecido como método *Manhatan*. (Ibidem, POLICYALMANAC:2004)

É calculado os  $f(n)$ ,  $g(n)$ ,  $h(n)$ , de todos os quadrados adjacentes (filhos) do quadrado inicial (pai), dando o resultado demonstrado na figura abaixo.



**Figura 22: Obtenção dos primeiros valores  $f(n)$ ,  $g(n)$ ,  $h(n)$**

FONTE: LESTER, POLICYALMANAC:2004

É encontrado como melhor opção o quadrado da direita com o  $f(14)$ , agora ele será o quadrado principal (pai), nesse momento o quadrado inicial anterior e o quadrado atual são adicionados em uma lista. (LESTER, POLICYALMANAC:2004)

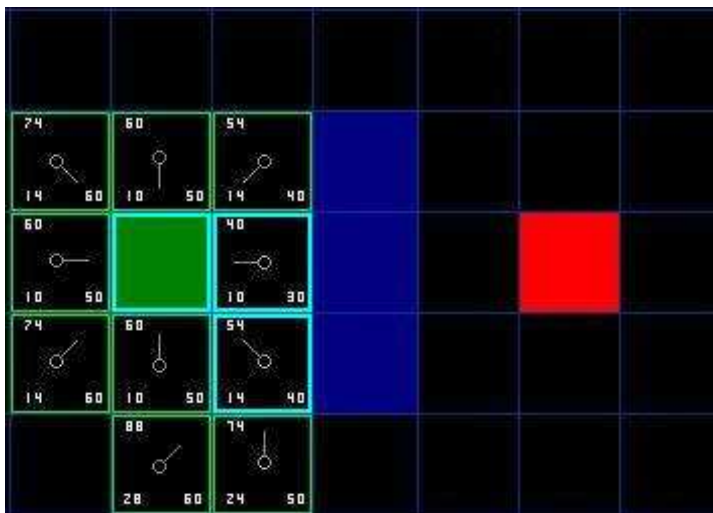
Executa os passos anteriores, porém ignoram-se os quadrados que representam as paredes, nesse momento encontram-se dois quadrados com valores de  $f(n)$  iguais, para a tomada de decisão pode-se escolher qualquer quadrado, por um motivo de rapidez pode

escolher o último a ter feito os cálculos da rodada anterior. (LESTER, POLICYALMANAC:2004)

É escolhido então o quadrado abaixo ao quadrado atual, esse quadrado também é adicionado na lista juntamente com os outros dois quadrados atuais antigos, passando o título de quadrado atual para esse novo. (*Idem*, POLICYALMANAC:2004)

Nesse momento encontra-se um outro caso especial, pois um dos quadrados adjacentes a este novo quadrado está abaixo da parede no mapa é ignorado, pois para chegar até ele não podemos andar pela diagonal apesar de termos o valor de  $g(n)$  para a diagonal 14, para chegar lá primeiro deve-se descer e depois ir para a direita dando a volta na parede, essa regra é opcional ela depende de como os nós foram definidos os nós no primeiro momento. (*Ibidem*, POLICYALMANAC:2004)

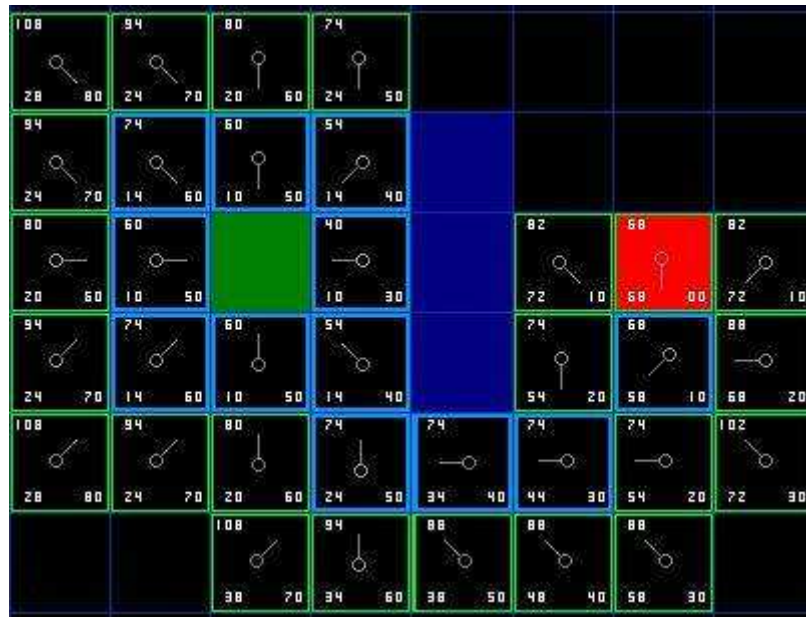
A figura abaixo apresenta esse cenário.



**Figura 23: Quadrado abaixo da parede sendo ignorado**  
 FONTE: LESTER, POLICYALMANAC:2004

Retirando os quadrados já escolhidos, agora temos três opções o quadrado da esquerda o quadrado abaixo e o quadrado abaixo e a direita, agora é tomada a decisão do próximo quadrado para ser adicionado a lista sendo formada pelos quadrados atuais anteriores. Esses passos são repetidos até ser acrescentado na lista o quadrado de destino. (LESTER, POLICYALMANAC:2004)

A figura abaixo mostra o quadro final da execução do A\*



**Figura 24: Caminho completo**

FONTE: LESTER, POLICYALMANAC:2004

A figura 24 demonstra ainda que durante a execução do algoritmo houve uma troca da melhor solução demonstrada pelo segundo quadrado abaixo do quadrado inicial em que o quadrado pai dele foi mudado da última vez que ele foi demonstrado na figura 23. (LESTER, POLICYALMANAC:2004)

Na figura 23 o quadrado pai dele estava acima a direita, e agora o quadrado pai dele está acima dele.

Para encontrar o caminho agora basta seguir a lista formada pelos quadrados adicionados lá iniciando do quadrado de destino até chegar ao quadrado de origem. (*Op. Cit.*, POLICYALMANAC:2004)

O algoritmo de A\* é utilizado em jogos pra realmente encontrar o melhor caminho a ser tomado, sendo mais utilizado em jogos de estratégia como *Age of Empires* ou *WarCraft*, onde deve-se mover unidades de exército ou unidades normais.

O algoritmo de A\* pode ter várias implementações, porém o que foi demonstrado aqui é o básico que o algoritmo deve fazer.

### 5.5.4.3 Algoritmos Genéticos

Algoritmos genéticos são utilizados em qualquer campo como *a-life*, computação revolucionária e redes neurais o bom entendimento do processo de um algoritmo genético é importante para a compreensão da IA de um jogo.

Algoritmos genéticos são métodos usados para resolução de problemas de busca e otimização inspirados no princípio da evolução. (FERNANDES, 2005:115)

Ao longo das gerações, as populações evoluíram na terra utilizando os princípios definidos por Darwin (1859), seleção natural e sobrevivência do mais forte. Os Algoritmos Genéticos (AG) imita esse processo para encontrar soluções para os problemas do mundo real. Essa evolução que as soluções sofrem, dependem, de uma codificação entre elas. (*Idem*, 115)

Na natureza, o cruzamento entre diferentes indivíduos ancestrais pode gerar “superindivíduos”, tendo uma adaptação bem maior que seus pais. Desta forma que as espécies evoluem, tendo características adaptadas ao ambiente em que vive. (*Ibidem*, 115)

Os AGs trabalha com uma analogia direta com a natureza. Trabalhe-se com uma população de indivíduos, onde cada um representa a solução do problema. (FERNANDES, 2005:115)

Cada indivíduo se associa um grau de aptidão, determinando a capacidade de competir com os outros indivíduos da população, quanto maior a sua aptidão mais chances ele tem de ser escolhido pra cruzar com outro individuo escolhido da mesma forma, desse cruzamento será gerado descendentes com as características dos indivíduos que se cruzaram. Quanto menor for a probabilidade do individuo, menor será a possibilidade de propagação das características desse individuo para as gerações futuras. Com isso se produz uma nova população que substitui a antiga atendendo melhor a solução. (*Idem*, 115-116)

Assim ao longo das gerações, as melhores características se propagam, facilitando assim a exploração de áreas do espaço de busca. Quando os AGs são bem projetados é grande a possibilidade de encontrar a solução ótima. (*Ibidem*, 116)

Apesar que AG's não encontram a melhor solução sempre uma mais próxima, com bases empíricas ele irá retornar sempre a solução mais próxima da melhor. (FERNANDES, 2005:116)

Caso seja utilizado outra técnica para solucionar o problema os AG's com certeza seriam passados para trás. (*Idem*, 116)

Basicamente o, os AG's tratam os problemas de otimização como o processo iterativo da busca pela melhor solução para o problema. Iniciando com um conjunto de seleções naturais, constituindo a população inicial. Com essa população é gerada uma nova população, sendo que a cada repetição a população é trocada, chegando até a solução do computador. (FERNANDES, 2005:116)

#### **5.5.4.3.1 Codificação**

Para FERNANDES (2005:117), os indivíduos participantes de uma população são identificados como genes, que juntos formam uma coleção de valores a representação deles geralmente é em cadeias de 1 e 0. A adaptação de um indivíduo irá depender somente da análise de seu genótipo a partir de seu fenótipo, ou seja, é calculada em conta com um cromossomo escolhido por uma função de avaliação ou de adaptação. A função de adaptação é projetada para cada problema. Dado um cromossomo, a função associa um número real, que se supõem refletir o nível da adaptação do indivíduo representado pelo cromossomo ao problema.

Durante a fase reprodutiva são selecionados os indivíduos da população para o cruzamento e produção de descendentes, que formarão a próxima geração após sofrerem a mutação. (*Idem*, 118)

O sorteio dos pais deve ser feito randomicamente, por um sistema que favoreça os indivíduos melhores adaptados, uma vez que cada indivíduo se associa a uma probabilidade de ser selecionado que é proporcional a função de avaliação. Esse procedimento é conhecido como roleta. Isso faz com que os indivíduos que são mais adaptados possuam a menor probabilidade de ser escolhido. (*Ibidem*, 2005:118)

Após, selecionados os cromossomos dos indivíduos são combinados, utilizando operadores de cruzamento e mutação.

#### **5.5.4.3.1.1 Cruzamento**

O operador de cruzamento toma dois pais selecionados e divide seus cromossomos em uma forma aleatória, após os pedaços entre os dois serem separados, é preciso intercalar os pedaços geando assim uma família inteira, os descendentes herdaram características de cada um dos pais. (FERNANDES, 2005:118)

Habitualmente o operador de cruzamento é executado em indivíduos que tenham uma probabilidade entre 0,5 e 1,0. Se o operador não se aplicar ao gene é feita cópia dos pais muito bem simplesmente. (FERNANDES, 2005:118)

#### **5.5.4.3.1.2 Mutação**

O operador de mutação se aplica em cada filho de maneira individual, alterando de forma aleatória toda a estrutura de cada gene gerado. (FERNANDES, 2005:119)

Pensa-se que o operador de cruzamento seja mais importante que o de mutação, porém o operador de mutação serve para garantir que nenhum individuo tenha a probabilidade de escolha igual a zero, e é de fundamental importância para assegurar a convergência dos AGs. (*Idem*, 119)

Para critérios práticos é de grande importância a convergência introduzida por De Jong(1975). Se o AG foi implementado corretamente, a população evoluirá de tal forma que a média da adaptação geral, assim como a adaptação do melhor individuo será incrementada a chegar a ser uma solução ótima. (*Ibidem*, 119)

Este conceito está relacionado com a idéia de progressão para a uniformidade, ou seja um gene será convertido quando ao menos 95% da população possuir o mesmo gene. A população converge quando todos os genes são convergidos. Essa definição pode ser generalizada quando pouco dos genes tiverem sido convergidos. (FERNANDES, 2005:119)

Cada vez que a população aumenta, a probabilidade de encontrar a solução ótima aumenta também.

#### **5.5.4.3.2 Função Objetivo**

Um dos aspectos que mais influenciam um AG é a determinação da sua função de adaptação ou objetivo  $f(o)$ , bem como a codificação utilizada para tal. (FERNANDES, 2005:123)

O ideal é implementar uma função de tal forma que, seja permitido verificar dois indivíduos próximos em uma população serem válidos. (*Idem*, 123)

Para essa função, existem diversas propostas, porém a mais comum é onde os indivíduos não verificam as restrições, são desconsiderados e o AG segue gerando novos indivíduos para a obtenção de indivíduos válidos. Outra forma é utilizar um reparador que

refará a construção dos genes que foram utilizados para a obtenção desse indivíduo. (FERNANDES, 2005:123)

Outro enfoque se baseia penalizando a função de adaptação. Isso seria feito dividindo o resultado da função pela quantidade de infrações que o gene cometeu no momento de sua criação isso é denominado de “custo de reconstrução”, que consiste realmente em dar um custo à conversão do gene de tal forma que não viole mais nenhuma restrição. (*Idem*, 123)

Outra técnica utilizada quando a função objetivo é muito complexa é denominada de “avaliação aproximada”, em certos casos é melhor ter como base o resultado de várias funções objetivo do que se ater a uma. (*Ibidem*, 123)

Os AG's têm um problema nas suas execuções. As vezes o algoritmo é tão rápido que a convergência dá uma falha no momento da convergência, esse erro é denominado de “convergência prematura”, em que o algoritmo converge para o local ótimo. Em outros casos acontece realmente o trabalho, é feito uma convergência lenta do algoritmo, a solução para isso seria as transformações na função. (*Ibidem*, 123)

O problema de convergência prematura ocorre quando a seleção de um indivíduo demora o tempo proporcional do teu trabalho. Neste caso podem existir indivíduos com uma adaptação ao problema bem superior aos outros, ou seja, a medida que o algoritmo cresce esses indivíduos dominam todos os outros. Costuma-se evitar esses “superindivíduos”, por meio da transformação. Esse problema é lento e é resolvido de maneira bem análoga, expandindo a área de influencia do algoritmo genético. (FERNANDES, 2005:123)

### 5.5.4.3.3 Algoritmo Genético Simples

Abaixo está um exemplo do Algoritmo Genético bem simples.

Função ALGORITMO-GENETICO

função ALGORITMO-GENETICO(população, FN-FITNESS) retorna um indivíduo

entradas: população um conjunto de indivíduos

FN-FITNESS, função que mede a adaptação do indivíduo

REPITA

nova população = conjunto vazio

PARA  $i = 1$  até TAMANHO(população) FAÇA

$x = \text{SELECÇÃO-ALEATÓRIA}(\text{POPULASÇÃO}, \text{FN-FITNESS})$

$y = \text{SELECÇÃO-ALEATÓRIA}(\text{POPULASÇÃO}, \text{FN-FITNESS})$

```

filho = REPRODUZ(x, y)
SE (pequena probabilidade aleatória)
    filho = MUTAÇÃO(filho)
FIM-SE
Adicionar filho a nova população
população = nova população
ATÉ algum individuo estar adaptado ou até ter decorrido o tempo suficiente
RETORNAR o melhor individuo em população de acordo com FN-FITNESS
fim

```

Essa função mostra de que forma foi feita a escolha dos genes pais e a avaliação do gene gerado do cruzamento para a criação da nova população

Função REPRODUZ()

função REPRODUZ(x,y) retorna um individuo

entradas: x, y, indivíduos pais

n = COMPRIMENTO(x)

c = número aleatório de 1 a n

retornar CONCATENA(SUBCADEIA(x,1,c), SUBCADEIA(y,c+1,n))

fim

Essa função demonstra a forma que é feita o cruzamento entres os pais gerando assim um único filho.

#### 5.5.4.3.4 Algoritmos Genéticos em Jogos

Alguns jogos de corrida possuem diversos carros como *Gran Turismo* cerca de 500 carros cada um requer uma adaptação evolução de suas habilidades relacionadas em performance e direção, então algumas companhias utilizam dos algoritmos genéticos para automatizar a atividade de *tuning*, gerando assim todas as possíveis combinações de carros, modificando os parâmetros de performance dos carros chegando a obtenção dos resultados ótimos. Esses resultados serão guardados para serem utilizados durante o jogo. (FILHO, 2005:25)

Em jogos de estratégia ou de simulação como *SimCity* utiliza-se dos Algoritmos Genéticos para obter a população da cidade em que está administrando.

### 5.5.5 IA Hierárquico

Em jogos de estratégia há objetivos que devem ser cumpridos, porém que entram em conflito, por exemplo o computador deve mover um exército do ponto A para o ponto B, porém durante o caminho o exercito é atacado pelo inimigo, agora o computador deve decidir o que fazer, se as unidades atacadas contra-atacam ou se o exercito inteiro para e só continua quando tudo estiver bem, ou simplesmente ignorar o ataque e continuar até o seu principal objetivo. Para que o computador tenha uma decisão a IA do jogo deve estar preparada para tal situação, tratando a situação como um objetivo individual ou um objetivo do grupo como um todo. (POLIS, UNIDEV:2003)

Para a implementação de uma IA em jogos de estratégia como Age of Empires, Command & Conquer entre outros é preciso que o IA do oponente seja referente a uma estratégia militar, para isso basta manter um subsistema de programação em que possui entradas para uma situação geral. Esse subsistema pode ser uma simples tabela de pesquisas, ou uma rede neural, coleção e regras, configuração de lógica e etc, com a característica de ser simples e obediente, ou deve aceitar entradas e gerar saídas. (*Idem*, UNIDEV:2003)

Este simples subsistema deverá ter informações relacionadas a todas as suas unidades como posição e status, sendo muito ineficiente, isso seria igual a apresentação de dados a um general que com essas informações daria ordens a seus soldados. (*Ibidem*, UNIDEV:2003)

Porém o general não cria a decisão e nem busca as informações de seus soldados, ao invés disso ele deverá receber informações de estratégia dos comandantes, desenvolver uma estratégia e mostrar aos seus comandantes, para que os comandantes tenham uma decisão, e isso vai acontecendo até chegar ao soldado de mais baixo nível. (POLIS, UNIDEV:2003)

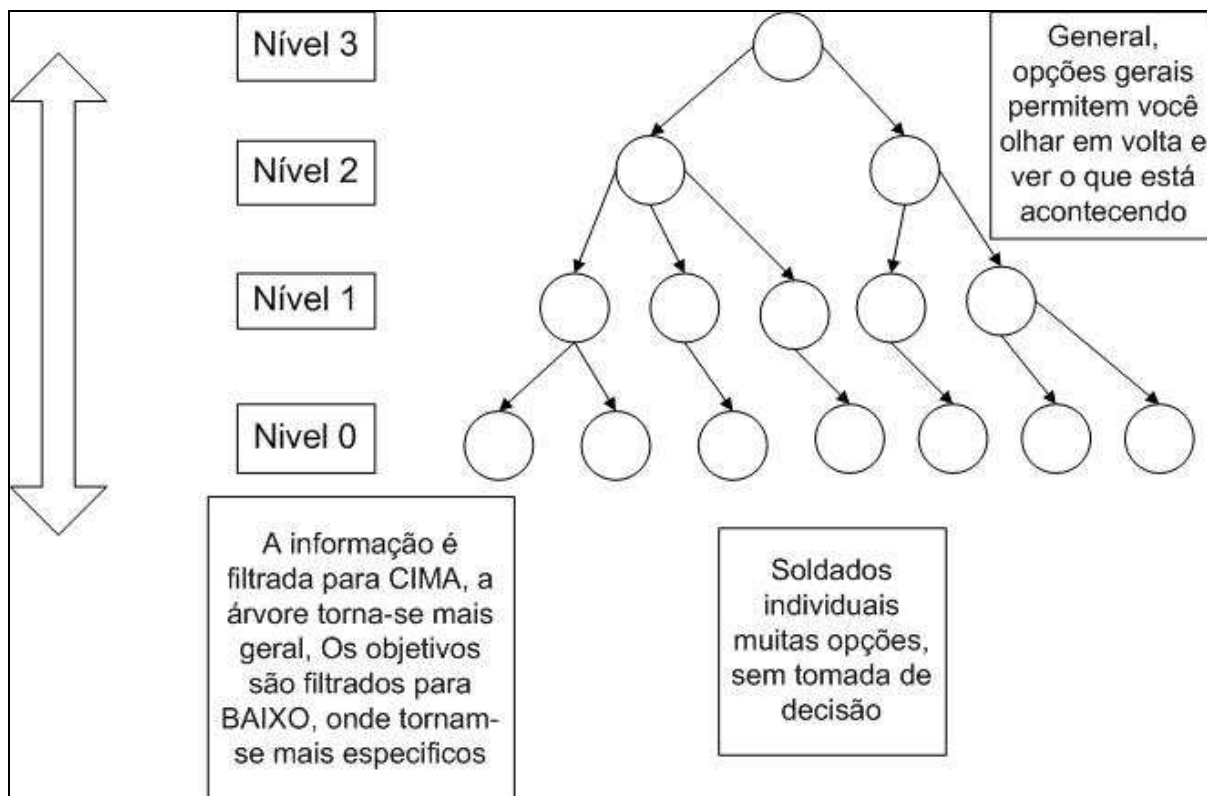
Esse processo pode ser dividido em diversos níveis, onde cada nível recebe informações do nível imediatamente abaixo, resumos ou generalizações apresentando o resultado para o nível acima, já esse nível ainda receberá configurações dos objetivos, ele utiliza as duas informações para ter o objetivo da forma mais clara o possível, após isso o objetivo é mandado para o nível inferior. (*Idem*, UNIDEV:2003)

Resumindo a informação através dos níveis é filtrada para cima, sendo mais progressivo conforme realiza a subida, enquanto os comandos se tornam objetivos e são filtrados para baixo tornando-se mais detalhado e progressivo. (*Ibidem*, UNIDEV:2003)

O modelo para a implementação será uma árvore de DMS (*decision making system*), sendo utilizada para demonstrar o encadeamento do nível militar, cada hierarquia militar será

representada por um nível. As entradas para esse sistema seria a entrada do nível e a informação obtida seria resumida e enviada para um nível abaixo. As saídas desses níveis serviriam para mais um nível. Saídas para o nível 0 é igual a mandar atualizar o campo. (POLIS, UNIDEV:2003).

Abaixo uma figura mostrando essa abstração



**FIGURA 25: IA Hierárquico**

Fonte: Adaptado de POLIS

A vantagem desse esquema é que permite vários níveis. No nível mais alto, as decisões poderiam ser essas:

- Direcionar todas as atividades de militares em territórios em X;
- Conduzir as guerras de atrito em territórios X,Y e Z;
- Evitar compromisso militar direto – concentrar na interrupção do inimigo.

Nestas circunstâncias é, mais provável que só o computador tente uma quantia de olhares digitais, ou considere o que está no campo.

### 5.5.6 Linguagens de Script

A grande maioria dos jogos são desenvolvidos com linguagens de *script*,

Sem uma definição muito precisa, as linguagens de Script apresentam algumas características em comum. As linguagens de Script são interpretadas, possuem gerencia de memória automática, tipagem dinâmica, fornecendo suporte a criação de estruturas de dados e manipulação de dados. Essas linguagens normalmente funcionam em programas implementados em linguagens compiladas como C ou C++, outra vantagem é que elas são seguras, não podendo acessar serviços sem a permissão do programa “hospedeiro”. A união dessas características torna as linguagens de script uma ótima ferramenta para o desenvolvimento de um jogo. (CELES *et. al.*, 2004:1)

Utilizar uma linguagem de script no desenvolvimento de jogos traz diversos benefícios, elas podem ser usadas para o desenvolvimento de um jogo inteiro, para definição completa dos objetos do jogo, gerenciamento de algoritmos de IA, controlar personagens, tratar eventos de entrada e montar a interface com o usuário. Ela também é usada nas etapas de prototipação, teste, depuração e análise do jogo. A utilização de linguagens de script ainda permite que roteiristas e designer possam “programar” o jogo, onde eles podem testar novas idéias. Esses profissionais em grande parte do tempo estão ligados ao desenvolvimento de um jogo, porém não são grandes profissionais especializados em programação. (*Idem*, 1)

### 5.5.6.1 Classificação

As linguagens de script podem ser divididas pelo seu nível de complexidade:

- Linguagens de configuração;
- Linguagens de Macro;
- Linguagens embutidas.

Linguagens de configuração: servem para selecionar preferências são tipicamente arquivos com valores referenciados para diversas variáveis, por exemplo arquivos .ini do Windows. (CELES *et. al.*, 2004:2)

Linguagens de macros: servem para automação de tarefas, normalmente são arquivos sem nenhum ou pouco controle com uma lista de ações a ser executadas, por exemplo arquivos para configuração automática de conexões com a internet via modem. (*Idem*, 2)

Linguagens embutidas: permitem o acesso programável a serviços referentes a aplicação, com um completo fluxo de controle além de funções escritas pelo usuário. Essas linguagens são completas e geralmente são variantes simples de linguagens tradicionais como Lisp ou C. (*Ibidem*, 2)

Fora do contexto da complexidade da linguagem, a utilização de uma linguagem de script é uma ferramenta muito poderosa, permitindo que muitos aspectos da aplicação sejam controlados a partir de arquivos textos que podem ser editados facilmente pelo programador ou pelo usuário, sem precisar reconstruir todo o software. Tornando o desenvolvimento mais ágil, pois partes não importante podem ser desenvolvidas por membros da equipe que não são programadores profissionais como no desenvolvimento dos jogos. (CELES *et. al.*, 2004:2)

Além de poder ser utilizada para várias tarefas permitindo um aumento na produtividade global, pois não é preciso documentar mais de uma vez o sistema para padrões diferentes. Para o usuário isso define o reaproveitamento automático dos conceitos aprendidos para a realização de varias tarefas. (*Idem*, 2)

### 5.5.6.2 Scripts em Jogos

Em jogos de RPG é utilizados scripts, pois a maioria dos RPG's possuem uma história especifica lineares ou com bifurcações lineares, ou seja, o jogador não tem liberdade, deixando ele seguir somente um caminho pré-definido, isso facilita o uso de scripts. Os scripta são utilizados para a construção de flags, diálogos, inimigos específicos e comportamento de um NPC's. (FILHO, 2005:7)

Em Adventures utilizam muitos diálogos, e são exibidas seqüências de quebra-cabeças em algum lugar do jogo, o script permite aos programadores e designers a incorporar essas características ao jogo. (*Idem*, 10)

Em alguns jogos de FTPS é utilizado scripts em todos os objetos de ambientes como inimigos, diálogos interação com o jogador e com o mundo. A utilização de Scripts em geral ajuda a contar a história do jogo, nos demais jogos de FTPS os scripts são utilizados para inserção de cust scenes e movimentar a câmera. (*Ibidem*, 15)

O uso de scripts em jogos de plataforma permite um fino controle sobre o fluxo do jogo, como o encontro com o chefe da fase com o personagem principal. É muito comum ainda existir algum personagem que ajuda o jogador em relação aos comandos e movimentos especiais, o script para esse personagem pode ser usado para adicionar uma ação a esse personagem ou aprender movimentos que o jogador ensine para ele. (FILHO, 2005:17-18)

O uso de scripts em jogos de corrida é utilizado para combinar os sistemas de tráfego de trânsito e de pedestres, onde padrões de movimentação são scripts e se interagem entre si, por exemplo se um carro derrapar da rua e entrar na calçada os pedestres terão uma reação

para aquela situação, eles devem ter os comportamentos alterados excedendo o script para escapar do atropelamento. (FILHO, 2004:25)

Scripts são usados em jogos de luta para que os designers definem tudo o que deve acontecer para todo movimento do personagem, além de ser utilizado para introdução de cut scenes, como quando os dois lutadores entram na arena ou depois de haver algum vencedor esse vencedor executar uma dança festejando sua vitória. (*Idem*, 27)

### 5.5.6.3 Linguagem Lua

Grande parte dos jogos que utilizam linguagens de script tem como principal linguagem utilizada a linguagem Lua, devido ao seu tamanho, bom desempenho, fácil integração e portabilidade. Lua é utilizada por diversas empresas da indústria de jogos como *LucasArts, BioWare, Microsoft, Relic Entertainment, Absolute Studios e Monkeystone Games*. (CELES *et. al.*, 2004:1)

A Linguagem Lua é uma linguagem de programação poderosa e leve, desenvolvida para estender aplicações, ou seja, para ser acopladas em sistemas maiores que necessitem ler arquivos de configurações escritos pelos usuários, ela possui uma sintaxe semelhante a do Pascal, porém com construções modernas, como funções anônimas, inspiradas no paradigma funcional, e poderosos construtores de dados, fazendo com que Lua seja uma linguagem de grande expressão e uma sintaxe familiar. (*Idem*, 2)

Ela foi desenvolvida para oferecer meta-mecanismos, possibilitando mecanismos mais específicos. Sendo fácil adequar a linguagem Lua às necessidades da aplicação, sem comprometer as características básicas da própria linguagem. Geralmente os programadores de jogos fornecem abstrações adequadas para facilitar a tarefa dos roteiristas e artistas. (*Idem*, 1)

A Linguagem Lua foi projetada e implementada no Tecgraf, grupo de computação gráfica da PUC-Rio. A primeira versão de Lua (1.0) foi lançada em julho de 1993. A versão atual dela (5.0.2) é de março de 2004, lançada com correção das falhas da versão 5.0. (*Ibidem*, 2)

Ela é uma linguagem projetada para dar suporte a linguagem procedural, oferecendo facilidade para descrição dos dados. Na programação dos jogos, isso significa que a Lua possibilita combinar os objetos e sua descrição com a programação dos comportamentos de cada objeto somente em um contexto. Ela é uma biblioteca padrão desenvolvida em C,

podendo ser compilada por qualquer compilador de C ou C++. Sendo uma linguagem de script ela trabalha acoplada em uma aplicação. Com essa aplicação pode-se criar e ler valores guardados em Lua, executar funções de Lua registrar funções de C, sendo que essas funções podem ser utilizadas por um programa feito em Lua, podendo conciliar as facilidades da linguagem Lua com a eficiência de linguagens como C. A distribuição Lua vem com um programa hospedeiro lua.c, podendo ser usado para a execução de scripts e Lua. (CELES *et. al.*, 2005:)

## CONCLUSÃO

Ao terminar este trabalho perceberam-se as vantagens de mesclar entretenimento com tecnologia.

A inteligência artificial é um campo que está sendo pesquisado e aprimorado em grande escala nos últimos anos trazendo benefícios para a humanidade, uma das áreas que irá impulsionar o desenvolvimento da inteligência artificial é a área de jogos.

Para se desenvolver um jogo é preciso ter conceitos de análise de sistemas, e uma grande base em computação gráfica. A área de desenvolvimento de jogos está em expansão crescente.

A integração da inteligência artificial nos jogos ainda é precária tendo agora uma grande ênfase, pois os jogos já chegaram a patamares bem grandes em gráficos e sons. Sendo agora o desafio de tornar o jogo mais competitivo em relação ao jogador.

O trabalho demonstrou o conceito básico de cada algoritmo de inteligência artificial utilizado em jogos e também demonstrou a lógica de cada um.

Para uma continuação seria desenvolvido um jogo contendo o máximo de algoritmos de Inteligência Artificial para a demonstração de suas utilizações

## **ANEXOS**

## REFERÊNCIAS BIBLIOGRÁFICAS

CARVALHO, Luís Alfredo Vidal de, *Datamining – Medicina, economia, engenharia e administração*. São Paulo: Érica, 2001.

CELES, Waldemar; FIGUEREDO, Luiz Henrique de; IERUSALIMSKY, Roberto. *A Linguagem Lua e suas Aplicações em Jogos*. Rio de Janeiro, 2004.

DUARTE, Marcelo. *Guia dos Curiosos*. São Paulo: Cia. Das Letras, 1998.

FERNADES, Anita Maria da Rocha. *Inteligência artificial*. Florianópolis: Visual Books, 2005.

FERNANDES, Leandro D. *Sistemas de controle microprocessados para motores elétricos para motores elétricos utilizando lógica fuzzy*. Faculdade de Ciência da computação Dissertação (Graduação) – Faculdade Luterana do Brasil. Gravataí, 2004

FILHO, Vicente Vieira. *Revolution ai engine - desenvolvimento de um motor de inteligência artificial para a criação de jogos eletrônicos*. Faculdade de ciência da computação - Dissertação (graduação) Universidade Federal de Pernambuco. Recife, 2005.

KISHIMOTO, André. *Inteligência Artificial em Jogos Eletrônicos*. São Paulo, 2004.

MARTIN, Natalia. <http://br.news.yahoo.com/18112006/40/saude-noticias-sony-nintendo-abrem-novo-capitulo-da-guerra-dos.html> -- Acesso em: 20/11/2006

MERKEL, Daniel Santa Catarina. *DESENVOLVIMENTO DE UM PROTÓTIPO DE JOGO 3D UTILIZANDO ENGINE*. Faculdade de informática. Dissertação (graduação) – Universidade de Passo Fundo, Passo Fundo, 2005.

MESSA, Marcelo; PAIM Marcos,. <http://oea.psico.ufrgs.br/roboticando/aibo.htm> - Acesso em: 17/10/2006

NORMAND. Reinaldo. <http://outerspace.ig.com.br/retrospace/> - Acesso em: 10/09/2006

PANCHIERI, Fabio.; MORATO. Gabriel. <http://jogos.uol.com.br/reportagens/historia/> - Acesso em: 10/09/2006

PERUCIA, Alexandre Souza; BERTHÊM, Antonio Córdova; BERTSCHINGER, Guilherme Lage; MENEZES, Roberto Robeiro Castro. *Desenvolvimento de jogos eletrônicos*. São Paulo: Novatec, 2005.

RUSSELL Stuart; NORVIG Peter. *Inteligência artificial*. Rio de Janeiro: Editora Campus, 2004.

SANTEE, André. *Programação de jogos com C++ e DirectX*. São Paulo: Novatec, 2005.

SILVA, Flávio Soares Corrêa da. *MAC5701 - Tópicos em Ciência da Computação - Agentes Inteligentes em Jogos de Computador*. Faculdade de informática. Dissertação (graduação) – Universidade de São Paulo, 2005.

STAIR , Ralph M.; REYNOLDS, George W.. *Princípios de sistemas de informação*. São Paulo: Thomson, 2006.

TATAI, Vitor Kazuo. *Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador*. Faculdade de Engenharia Elétrica e de Computação. Dissertação (Mestrado) – Universidade estadual de campinas, Campinas, 2003.

TATIBANA, Cássia Yuri; KAETSU Deisi Yuki,. <http://www.din.uem.br/ia/neurais/> - Acesso em: 16/10/2006

VILIEGAS, Renato. Tudo sobre o Playstation 2: conheça o console lançado no Japão que promete revolucionar o conceito de entretenimento doméstico no mundo. *Ação games*. São Paulo, n.149, p.14-19, mar. 2000

“FERRARI”, Alexandre da Costa Nascimento. <http://www.n-planet.com.br/?view=article&article=3983>