

**UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI**  
**INSTITUTO DE CIÊNCIAS - ICI**  
**DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO – DMC**  
**PROJETO FINAL DE GRADUAÇÃO**

---

---

***DESENVOLVIMENTO DE JOGOS DE  
COMPUTADOR***

**MAIRLO HIDEYOSHI GUIBO CARNEIRO DA LUZ**

---

***Itajubá – MG***  
**2004**

**UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI**  
**INSTITUTO DE CIÊNCIAS - ICI**  
**DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO – DMC**  
**PROJETO FINAL DE GRADUAÇÃO**

***DESENVOLVIMENTO DE JOGOS DE COMPUTADOR***

**Trabalho organizado por MAIRLO HIDEYOSHI GUIBO  
CARNEIRO DA LUZ**

**Monografia apresentada ao  
Departamento de Matemática e  
Computação da Universidade Federal  
de Itajubá como parte dos requisitos  
para obtenção do Grau de Bacharel em  
Ciência da Computação.**

**ORIENTADOR: Siles Paulino de Toledo**  
**CO-ORIENTADOR: Alexandre Carlos Brandão Ramos**

**UNIFEI**  
**11/2004**

Luz, Mairlo Hideyoshi Guibo Carneiro da

“Desenvolvimento de Jogos de Computador”

Monografia apresentada ao Departamento de Matemática e Computação da Universidade Federal de Itajubá, para conclusão do Curso de Bacharelado em Ciência da Computação.

Orientador: Prof. Dr. Siles Paulino de Toledo  
Co-Orientador: Prof. Dr. Alexandre Carlos Brandão Ramos

1.Jogos 2.DirectX 3.Inteligência Artificial 4.Game Design  
5.Flocking 6.QuadTree 7.Programação C++

## **AGRADECIMENTOS**

A Deus.

Aos meus pais Amauri e Júlia.

A meu irmão Marlon e meu tio Hiromassa.

Aos Professores Siles e Alexandre, pelas lições e orientações.

Aos Professores do Curso de Ciência da Computação, pelas lições e pela amizade, e com quem compartilhei uma maravilhosa jornada acadêmica.

## SUMÁRIO

<b>CAPÍTULO 1 – INTRODUÇÃO</b>	<b>1</b>
<b>1.1 O QUE É UM JOGO?</b>	<b>1</b>
<b>1.2 TIPOS DE JOGOS</b>	<b>2</b>
<b>1.3 HISTÓRIA DOS JOGOS</b>	<b>6</b>
<b>1.4 MERCADO DE JOGOS</b>	<b>7</b>
<b>1.4.1 MERCADO MUNDIAL</b>	<b>7</b>
<b>1.4.2 MERCADO BRASILEIRO</b>	<b>8</b>
<b>1.5 TRABALHOS RELACIONADOS</b>	<b>9</b>
<b>1.6 OBJETIVO DO TRABALHO</b>	<b>9</b>
<b>1.7 APRESENTAÇÃO DA TESE</b>	<b>9</b>
<b>1.8 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 1</b>	<b>10</b>
<b>CAPÍTULO 2 – ESTRUTURA DO DESENVOLVIMENTO</b>	<b>12</b>
<b>2.1 ESQUELETO DE DESENVOLVIMENTO</b>	<b>12</b>
<b>2.2 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 2</b>	<b>13</b>
<b>CAPÍTULO 3 – GAME DESIGN</b>	<b>14</b>
<b>3.1 O QUE É GAME DESIGN?</b>	<b>14</b>
<b>3.2 QUAL A IMPORTÂNCIA DO GAME DESIGN?</b>	<b>15</b>
<b>3.3 CONCEITUAÇÃO DO JOGO</b>	<b>16</b>
<b>3.3.1 CRIANDO A IDÉIA DO JOGO</b>	<b>17</b>
<b>3.3.2 OS ELEMENTOS DE UM JOGO</b>	<b>18</b>
<b>3.4 O MUNDO E A FANTASIA DO JOGO</b>	<b>21</b>
<b>3.4.1 IMERSÃO E A SUSPENSÃO DA DESCONVICÇÃO</b>	<b>22</b>
<b>3.4.2 A IMPORTÂNCIA DA HARMONIA</b>	<b>22</b>
<b>3.4.3 A DIMENSÃO FÍSICA</b>	<b>22</b>
<b>3.4.4 A DIMENSÃO TEMPORAL</b>	<b>23</b>
<b>3.4.5 A DIMENSÃO ÉTICA</b>	<b>24</b>
<b>3.5 ESTÓRIA</b>	<b>25</b>
<b>3.6 DESENVOLVIMENTO DO PERSONAGEM</b>	<b>28</b>
<b>3.6.1 DESIGN DE PERSONAGENS DIRIGIDOS À ARTE</b>	<b>28</b>
<b>3.6.2 DESIGN DE PERSONAGENS DIRIGIDOS À ESTÓRIA</b>	<b>30</b>
<b>3.6.3 ESTERÉÓTIPOS DE PERSONAGENS</b>	<b>31</b>
<b>3.7 CRIANDO A EXPERIÊNCIA DO JOGADOR</b>	<b>32</b>
<b>3.7.1 CRIANDO A INTERFACE HOMEM-COMPUTADOR</b>	<b>33</b>
<b>3.7.1.1 Disposição da interface</b>	<b>33</b>
<b>3.7.1.2 Elementos para uma interface do usuário simples</b>	<b>34</b>
<b>3.8 BALANCEAMENTO DO JOGO</b>	<b>37</b>
<b>3.8.1 BALANCEAMENTO ESTÁTICO</b>	<b>37</b>
<b>3.8.2 BALANCEAMENTO DINÂMICO</b>	<b>38</b>
<b>3.8.3 CRIANDO SISTEMAS BALANCEADOS</b>	<b>39</b>
<b>3.9 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 3</b>	<b>40</b>

<b>CAPÍTULO 4 – INTELIGÊNCIA ARTIFICIAL</b>	<b>41</b>
<b>4.2 TÉCNICAS MAIS COMUNS DE IA UTILIZADAS EM JOGOS</b>	<b>41</b>
<b>4.3 IA DO DEMO</b>	<b>44</b>
<b>4.3.1 MÁQUINAS DE ESTADOS</b>	<b>44</b>
<b>4.3.2 ALGORITMO GENÉTICO</b>	<b>46</b>
<b>4.3.3 SCRIPTING</b>	<b>47</b>
<b>4.3.4 ALGORITMO DE FUGA E ATAQUE DO PÁSSARO MUTANTE</b>	<b>48</b>
<b>4.4 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 4</b>	<b>49</b>
<b>CAPÍTULO 5 – PROGRAMAÇÃO</b>	<b>50</b>
<b>5.1 APIS PARA A PROGRAMAÇÃO</b>	<b>50</b>
<b>5.2 ESTRUTURA DOS JOGOS</b>	<b>52</b>
<b>5.2.1 CÓDIGO-FONTE DE EXEMPLO</b>	<b>54</b>
<b>5.3 TÉCNICAS UTILIZADAS NO DEMO</b>	<b>56</b>
<b>5.3.1 MAPAS DE ALTITUDE</b>	<b>56</b>
<b>5.3.1.1 Gerando a geometria do terreno</b>	<b>57</b>
<b>5.3.1.2 Acessando o vetor terreno</b>	<b>59</b>
<b>5.3.2 BILLBOARD</b>	<b>60</b>
<b>5.3.2.1 Criação das texturas</b>	<b>60</b>
<b>5.3.2.2 Algoritmo do billboard</b>	<b>62</b>
<b>5.4 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 5</b>	<b>64</b>
<b>CAPÍTULO 6 – CONCLUSÕES</b>	<b>65</b>
<b>6.1 GAME DESIGN</b>	<b>65</b>
<b>6.2 INTELIGÊNCIA ARTIFICIAL</b>	<b>66</b>
<b>6.3 PROGRAMAÇÃO</b>	<b>67</b>
<b>6.4 PROPOSTAS FUTURAS</b>	<b>68</b>
<b>6.4.1 FLOCKING</b>	<b>68</b>
<b>6.4.2 IMPLEMENTAÇÃO DA REDE</b>	<b>69</b>
<b>6.4.3 ALGORITMO DE QUADTREE</b>	<b>70</b>
<b>6.5 BIBLIOGRAFIA REFERENTE AO CAPÍTULO 6</b>	<b>70</b>
<b>ANEXOS</b>	<b>71</b>
<b>ANEXO A: DOCUMENTAÇÃO DA IDÉIA DO DEMO</b>	<b>72</b>
<b>ANEXO B: IMAGENS EM TONS DE CINZA E O FORMATO RAW</b>	<b>76</b>
<b>ANEXO C: HARDWARE E SOFTWARES NECESSÁRIOS</b>	<b>78</b>
<b>ANEXO D: DOCUMENTAÇÃO DO CÓDIGO-FONTE DO DEMO</b>	<b>82</b>

## LISTA DAS IMAGENS

Figura 1 – Street Fighter.....	2
Figura 2 – Warcraft 3.....	3
Figura 3 – Winning Eleven.....	4
Figura 4 – Erinia.....	5
Figura 5 – SpaceWar!.....	6
Figura 6 – Estrutura do Desenvolvimento de Jogos.....	12
Figura 7 – Jogo de Damas.....	14
Figura 8 – McDonalds.....	15
Figura 9 – O Senhor dos Anéis e Spider-Man.....	17
Figura 10 – Winning Eleven.....	19
Figura 11 – Pac-Man.....	19
Figura 12 – Super Mario Bros.....	20
Figura 13 – The Need For Speed Underground 2.....	21
Figura 14 – Tetris.....	23
Figura 15 – GTA.....	25
Figura 16 – Profundidade nas histórias da série Final Fantasy.....	26
Figura 17 – Pac-Man.....	29
Figura 18 – Lara Croft.....	29
Figura 19 – Personagem Engraçadinha.....	30
Figura 20 – Personagens da série Final Fantasy.....	31
Figura 21 - Outlive.....	33
Figura 22 – Áreas mais Utilizadas para Interfaceamento.....	34
Figura 23 – Doom 3.....	42
Figura 24 – The Sims.....	44
Figura 25 – Máquina de Estados da IA do Pássaro Mutante.....	45
Figura 26 – Estrutura do Código de um Jogo.....	52
Figura 27 – Mapa de Altitude.....	57
Figura 28 – Conceitos para a Criação de Terrenos 3D.....	58
Figura 29 – Textura de uma Árvore.....	61
Figura 30 – Canal Alfa de uma Textura.....	62
Figura 31 – Shigeru Miyamoto, Will Wright e Sid Meier.....	65
Figura 32 – Bird’s Nightmare.....	69
Figura 33 – Variação dos Tons de Cinza.....	76

## RESUMO

Trata-se de uma introdução ao desenvolvimento de jogos para computador, seus vários gêneros e breve discussão sobre o estágio atual de seu mercado no Brasil e no Mundo.

São abordadas a Estrutura de Desenvolvimento de Jogos, a desmistificação de alguns conceitos sobre o Game Design e a sua importância no mundo virtual.

Fala da Inteligência Artificial aplicada aos jogos, juntamente com a descrição das diversas técnicas utilizadas e um desbravamento da Inteligência Artificial implementada no demo, que acompanha a obra.

São introduzidos os conceitos necessários para a implementação do esqueleto de um jogo e esmiuçadas duas técnicas utilizadas no demo.

Em seguida, são apresentadas as conclusões do autor referentes aos diversos tópicos abordados e sugestões para trabalhos futuros, com uma breve introdução sobre os temas sugeridos. Ao final, há o detalhamento do processo de desenvolvimento do demo, que acompanha a monografia, bem como uma pequena explicação sobre imagens em tons de cinza e o formato RAW, a especificação do hardware e dos softwares necessários para o desenvolvimento de jogos e a documentação do código-fonte do demo.

## **ABSTRACT**

It's an introduction to the computer's game development, its genres and a brief discussion about its actual market's stage in Brazil and in the World.

The Structure of Game's Development, the demystification of some concepts on the Game Design and its importance in the virtual world are boarded.

It speaks about the Artificial Intelligence applied to games, together with a description of the diverse used techniques and a taming of the Artificial Intelligence implemented in the demo, that folloies this workmanship.

The concepts necessary for the implementation of the game's skeleton are introduced and two techniques used in the demo are minced.

Soon after, the author's conclusions about the diverse boarded topics and suggestions for future works are presented, with a brief introduction on the suggested subjects. To end, there is the detailing of the demo's development process, that folloies this monograph, as well a little explanation about grayscale images and the RAW format, the hardware's and software's specification needed to the game's development, and the documentation of demo's source code.

## **CAPÍTULO 1 – INTRODUÇÃO**

### **1.1 O que é um Jogo?**

Para o desejado entendimento de jogos e da etapa de design dos jogos, é necessário o estabelecimento de certos conceitos. Um dos conceitos mais primordiais nesta fase inicial seria a definição do que seria um jogo.

“Primeiro, um jogo é um sistema formal, fechado, que subjetivamente representa um subconjunto da realidade.”

“Por fechado, quer dizer que o jogo é completo e auto-suficiente quanto a sua estrutura. O modelo do mundo criado pelo jogo é internamente completo; não é necessária a referência a agentes de fora do jogo. Alguns jogos com um mau design falham em encontrar esse requisito. Alguns jogos produzem conflitos entre as regras, por permitirem o desenvolvimento de situações onde as regras não se aplicam. O jogador precisa então estender as regras para cobrir a situação na qual ele se encontra. Essa situação sempre produz argumentos. Um design de um jogo correto impede essa possibilidade; ele é fechado porque as regras cobrem todas as contingências encontradas no jogo”.

“Por formal, quer dizer apenas que o jogo tem regras explícitas. Há jogos informais na qual as regras são declaradas de maneira livre ou deliberadamente vagas. Tais jogos são rapidamente removidos dos que tem mais popularidade quanto ao game play”.

“O termo sistema é frequentemente usado em abuso, mas neste caso sua aplicação é muito apropriada. Um jogo é uma coleção de partes, que interagem entre si, geralmente de maneiras complexas. Isso é um sistema”. (CRAWFORD, 1982, P. 7).

## 1.2 Tipos de Jogos

Para um bom projeto de um jogo, é desejável um conhecimento quanto aos diversos gêneros de jogos disponíveis. Dentre eles, podemos destacar os seguintes:

- (a) **Jogos de Ação:** representam uma grande faixa de estilos de jogos, podendo estes tanto serem encontrados com gráficos 3D, quanto com gráficos 2D. Geralmente eles são divididos em dois subgêneros: aqueles em que se atira e aqueles em que não se atira. As principais habilidades que este tipo de jogo explora no jogador são o tempo de reação e a coordenação entre a visão e as mãos do jogador em situações de pressão. Fazem parte desse gênero jogos de luta, como Street Fighter, que pode ser visto na *Figura 1*, jogos de tiro, como Doom III, jogos tipo labirinto, como Pac-Man, Mario, dentre outros tipos de jogos;



**Figura 1 – Street Fighter**

- (b) **Jogos de Estratégia:** jogos baseados nos jogos de tabuleiro. A grande maioria dos jogos deste gênero são lançados para PC, por sua grande capacidade de iteração entre usuário e jogo. Dividem-se em duas formas principais: baseado em turnos e em tempo real. Dentro desse

gênero destacam-se títulos como Civilization, Warcraft, Outlive, dentre outros. Na *Figura 2*, pode ser vista uma imagem do jogo Warcraft 3;



**Figura 2 – Warcraft 3**

(c) **Jogos de RPG:** assim como os jogos de estratégia, os RPGs formam um gênero de jogo que também foi derivado dos jogos de papel e caneta. Geralmente os RPGs possuem duas características que os distinguem dos demais gêneros:

- Personagens de jogo configuráveis, que melhoram no decorrer do jogo, com o aumento da experiência;
- Estórias muito bem definidas e que desenvolvem no decorrer do jogo.

A parte da estória é uma parte muito singular desse gênero. Nela, o jogador interage, de maneira que ele acaba se tornando uma parte muito real da mesma. Fazem parte desse gênero jogos do estilo Final Fantasy, Dragon Quest, Diablo, Erinia, dentre outros;

(d) **Jogos de Esportes:** ao contrário dos outros gêneros, onde o mundo em que a estória se desenvolve é pouco conhecida pelos jogadores, nos jogos esportivos o mundo é bastante conhecido. Por exemplo, muitas pessoas conhecem como é o futebol profissional e as suas regras. Entretanto, isso não quer dizer que os jogos desse gênero são extremamente realistas.

Exemplos de jogos desse gênero incluem Winning Eleven, Fifa Soccer, NHL, FutSim etc. Na *Figura 3*, tem-se um screenshot do jogo Winning Eleven;



**Figura 3 – Winning Eleven**

(e) **Jogos de Simulação de Veículos:** simuladores de veículos tentam criar a sensação de dirigir ou pilotar um veículo, real ou imaginário. No caso de veículos reais, o principal ponto a ser levado em consideração é a aproximação da realidade. Os jogadores que irão jogar este tipo de jogo querem ter a experiência mais próxima possível de qual seria a sensação na realidade. Jogos como The Need For Speed Underground, Nascar, Flight Simulator, dentre outros, fazem parte desse gênero.

(f) **Jogos de Construções e Gerenciamento de Simulações:** neste gênero de jogo o jogador não tem que enfrentar um inimigo, mas construir algo dentro de um determinado contexto e fazer o seu gerenciamento. Quanto melhor o jogador entender e controlar o processo de gerenciamento, maior será o seu sucesso na empreitada desse gênero. Sim City, Capitalism são os jogos mais famosos desse gênero;

- (g) **Jogos de Aventura:** ao contrário dos demais gêneros, um jogo de aventura não é uma simulação ou uma competição. Ele é uma história interativa sobre um determinado personagem que é controlado pelo jogador. Apesar do gênero ter se alterado muito no decorrer dos anos, existem certas qualidades que caracterizam esse gênero: exploração, coleção ou manipulação de objetos, solução de quebra-cabeças e a ênfase reduzida quanto ao combate ou aos elementos de ação. Podemos citar jogos como The Legend of Zelda, Grim Fandango, Gabriel Knight etc. como do gênero;
- (h) **Jogos de Vida Artificial:** estes jogos envolvem uma modelagem do processo biológico, geralmente para simular os ciclos de vida de seres vivos, podendo simular a vida de pessoas, animais, monstros etc. The Sims é um jogo desse gênero;



**Figura 4 – Erinia**

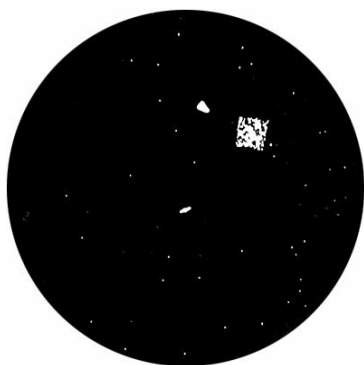
- (i) **Jogos On-line:** nos últimos anos, os jogos on-line deixaram de ser apenas uma minúscula fração do comércio de entretenimento interativo, para se tornar a maior fração desse mercado. Atualmente esse gênero tem crescido bastante para o lado dos MMOs, Multiplayer Massive Online, que são os jogos que se podem jogar somente via rede, acessando o servidor central do jogo e interagindo com outros

jogadores. Neste gênero, as trapaças no jogo devem ser evitadas ao máximo.

Futsim e Erinia são dois MMOs desenvolvidos no Brasil. Na *Figura 4*, vê-se uma imagem do jogo Erinia.

### 1.3 História dos Jogos

Em 1958, Willy Higinbothan criou um protótipo de um jogo de tênis que fazia uso de um osciloscópio como display. Embora seja considerada uma das primeiras tentativas de desenvolver um jogo eletrônico, é pouco lembrado pelos amantes de clássicos. O jogo mais lembrado pelos amantes é o SpaceWar!, da *Figura 5*, criado por Steve Russel durante a sua graduação em engenharia no MIT (Massachussets Institute of Technology).



**Figura 5 – SpaceWar!**

Depois, no decorrer dos anos, foram criados vários aparelhos, que utilizavam a televisão como um display para tornar os jogos mais interativos. Dentre os inúmeros, podemos citar o Odyssey, que foi o primeiro, o Atari 2600, o Master System, o Nintendo, o Mega Drive, o Famicom, o Neo Geo, o Sega CD, o PlayStation 1 e 2, o Nintendo 64, o Dreamcast, o Xbox e o Gamecube.

Nesse intervalo, também aparecem vários videogames portáteis, como o GameBoy, o Super GameBoy, GameBoy Color, o Neo Geo Pocket, o GameBoy Advanced, o N-Gage.

E no final de 2004, a briga pelo mercado de portáteis começou a pegar fogo, com a Sony anunciando que irá entrar nesse mercado, com o PSP, mercado esse dominado pelo Nintendo, com o GameBoy. Nisso a Nintendo contra-ataca, com o lançamento do Nintendo DS, que tem duas telas, sendo uma delas sensível ao toque.

Já no mercado de videogames, o PlayStation 2 detém a liderança absoluta, seguido por Xbox e GameCube.

No decorrer da história, verifica-se que o que faz um determinado videogame líder do mercado é a qualidade dos jogos que se encontram a disposição dos jogadores, bem como a variedade dos mesmos.

## **1.4 Mercado de Jogos**

### **1.4.1 Mercado mundial**

Atualmente o mercado de jogos no mundo se encontra em franca expansão, sendo que, segundo (BUENO, 2004), a previsão de crescimento do mercado para os próximos anos é a seguinte:

- **Mercado de jogos para Celular:** média de crescimento anual de 34%;
- **Mercado de jogos para PC:** ficará estabilizado até o final de 2005, começo de 2006, quando deverá começar a haver recessão;

- **Mercado de jogos para Consoles:** deverá ficar estável até o final de 2005, quando deverá chegar a nova geração de videogames, resultando em um aumento anual na ordem de 11%.

#### 1.4.2 Mercado brasileiro

Já no mercado brasileiro, a situação é a seguinte:

- **Mercado de jogos para Celular:** deve crescer bastante, seguindo o ritmo mundial de crescimento acelerado. Tem surgido bastante empresas, principalmente na região do Paraná, em Campinas e em Pernambuco para o desenvolvimento de jogos para celulares nacionais;
- **Mercado de jogos para PC:** a cada ano que se passa o capital arrecadado com as vendas está caindo, sendo que nos últimos anos a receita anual gira em torno de R\$ 60 milhões, pouco para um mercado que já chegou a girar R\$ 500 milhões por ano. No entanto, empresas brasileiras têm surgido para vender jogos ao mercado externo;
- **Mercado de jogos para Consoles:** comércio legal de consoles é inexistente no Brasil e, devido ao alto índice de pirataria, não há previsão de vendas legais no país. Quanto ao desenvolvimento de jogos, já existem duas empresas que dispõem de licenças para o desenvolvimento de jogos para o Xbox.

## **1.5 Trabalhos Relacionados**

**(SANTOS, 2002)**

SANTOS, Giliard Lopes dos, Inteligência Artificial em Jogos 3D. 2002. Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense em agosto de 2002.

**(DINÍZIO, 1999)**

DINÍZIO, Clailson Soares. Inteligência Artificial em Jogos de Tiro em Primeira Pessoa. 1999.

## **1.6 Objetivo do Trabalho**

Este trabalho tem por objetivo fazer um levantamento do processo de desenvolvimento de jogos de computador, desde a etapa de criação até a etapa de implementação. Para tal, além do embasamento teórico, também é criado um demo de um jogo de tiro, que acompanha a monografia.

## **1.7 Apresentação da Tese**

No capítulo 2 (dois) apresenta-se a estrutura para o desenvolvimento de um jogo, bem como uma explicação sucinta sobre cada etapa.

No capítulo 3 (três) há uma breve introdução ao que seja o game design, etapa na qual o jogo é desenvolvido, passando por esquemas que ajudam na criação de histórias e personagens.

Já no capítulo 4 (quatro) fala-se um pouco sobre as diversas técnicas utilizadas na inteligência artificial, bem como é feita a explicação da IA utilizada no demo.

Quanto ao capítulo 5 (cinco), são apresentadas as diversas opções para a implementação de jogos, além de serem explicadas algumas das técnicas utilizadas na implementação do demo.

E no capítulo 6 (seis) temos as conclusões referentes ao trabalho desenvolvido, além de sugestões para trabalhos futuros.

No anexo A, a idéia do demo é expressa. No anexo B é explicado o formato RAW, incluindo imagens em tons de cinza. Finalizando, o anexo C lista o hardware e os softwares necessários para o desenvolvimento de jogos de computador, enquanto que no anexo D explica-se a funcionalidade dos métodos das classes do demo.

## **1.8 Bibliografia Referente ao Capítulo 1**

### **(BANANAGAMES, 2004)**

BANANAGAMES, capturado no dia 25/08/2004, Diário, On-line, Disponível na Internet [www.bananagames.com.br](http://www.bananagames.com.br)

### **(BUENO, 2004)**

BUENO, Glauco D'Alessandro. Das salas de criação às prateleiras do varejo. Campinas, Unicamp, 11 set. 2004. Palestra Ministrada aos Congressistas da GameTech 2004, pelo Gerente Geral da América do Sul da ATARI Brasil Ltda.

**(CRAWFORD, 1982)**

CRAWFORD, Cris; The Art of Computer Game Design. 1982. Capturado em dezembro 2003. Online. Disponível na Internet <http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html>.

**(ROLLINGS; ADAMS, 2003)**

ROLLINGS, Andrew; ADAMS, Ernest. Andrew Rollings and Ernest Adams on Game Design. 1ª edição. Nova Iorque: New Riders, Maio de 2003. 622 páginas.

**(UOL, 2004)**

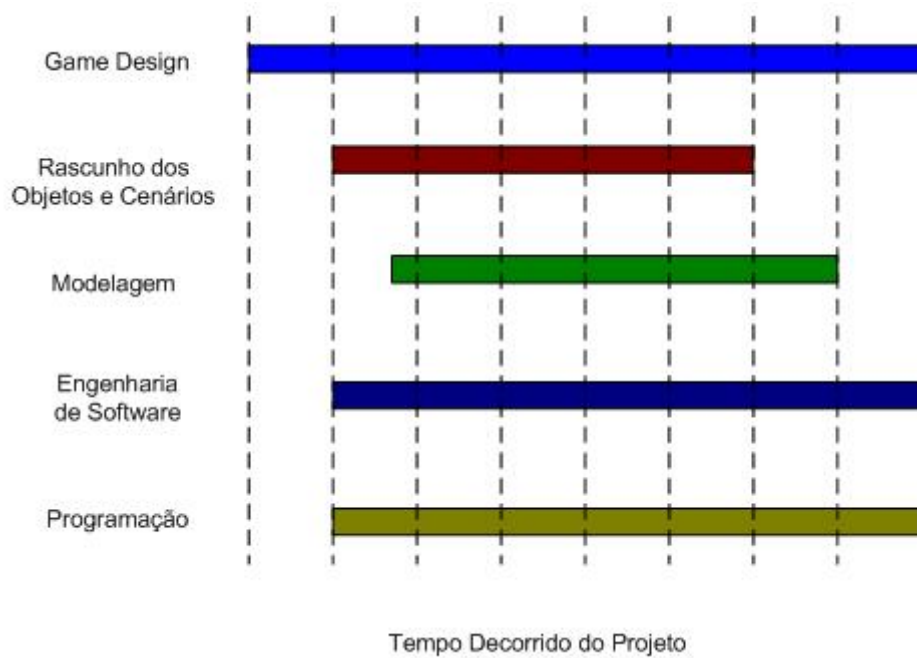
UOL, capturado no dia 25/08/2004, Diário, On-line, Disponível na Internet [www.uol.com.br/jogos](http://www.uol.com.br/jogos)

## CAPÍTULO 2 – ESTRUTURA DO DESENVOLVIMENTO

### 2.1 Esqueleto de Desenvolvimento

O desenvolvimento de jogos segue várias etapas, conforme pode ser visto na

*Figura 6.*



**Figura 6 – Estrutura do Desenvolvimento de Jogos**

Onde o **Game Design** seria a etapa onde o jogo seria criado, juntamente com a sua história, os personagens, o mundo que esses personagens interagiriam entre si etc.

Já no **Rascunho dos Objetos e Cenários** seriam feitos rascunhos dos objetos, dos cenários, dos personagens, das moradias etc., para auxiliar os modeladores na hora da modelagem.

Na **Modelagem**, os modeladores e as pessoas responsáveis pelas texturas criam os modelos e as texturas que serão utilizadas no jogo.

Enquanto isso na **Engenharia de Software** é feita a modelagem do sistema, geralmente por partes, ao contrário dos sistemas de banco de dados, onde a modelagem é feita por completo antes do início da sua implementação.

E na **Programação** o jogo é desenvolvido. Aqui é feita a programação da lógica do jogo, bem como a inteligência artificial, a inserção do áudio etc.

Vale lembrar que tanto **os Rascunhos e a Modelagem** são feitos em paralelo à **Engenharia de Software e a Programação**.

No decorrer dessa monografia, será dada uma maior ênfase sobre as etapas de **Game Design e Programação**.

## **2.2 Bibliografia Referente ao Capítulo 2**

**(CLUA, 2004)**

CLUA, Esteban Walter Gonzalez. Ferramentas necessárias para desenvolvimento de jogos. Campinas, Unicamp, 11 set. 2004. Palestra Ministrada aos Congressistas da GameTech 2004, pelo Representante do Departamento de Informática da Puc-Rio.

## CAPÍTULO 3 – GAME DESIGN

Quando as crianças brincam de jogos de tabuleiros, muitas vezes elas negociam as alterações das regras desses jogos de modo a torná-los mais atraentes. Um bom exemplo é o jogo de damas. Essa participação natural de jogar um jogo nos encoraja a pensar sobre e, às vezes, até alterar as regras do mesmo, isto é, seu design.



**Figura 7 – Jogo de Damas**

Mas algumas pessoas querem alterar mais coisas, além das regras. Eles querem criar o jogo inteiro. Eles têm idéias de como serão as regras, os personagens, o mundo, os desafios etc. destes jogos. É disso que será falado nesse capítulo.

### 3.1 O que é Game Design?

Game Design é um processo de:

- Imaginar um jogo;
- Definir o modo como ele funcionará;
- Descrever os elementos que criam um jogo (conceitual, funcional, artístico e outros);

- Transmitir essas informações para o time que construirá o jogo.

### 3.2 Qual a Importância do Game Design?

O Game Design, pelo menos para computadores e consoles, é um campo muito recente ainda, e há ainda muito a ser descoberto. A indústria fonográfica e a publicitária sabem muito mais sobre como invocar uma atmosfera interessante que qualquer game designer e, mais importante, eles sabem como aplicar as técnicas de uma maneira bem efetiva.

Um exemplo disso pode ser visto no McDonalds, onde as cores que eles utilizam são predominantemente o amarelo e o vermelho. Psicólogos dizem que a cor amarela influencia na percepção de fome, enquanto que o vermelho aumenta a ansiedade e a necessidade de correr. O resultado, segundo os psicólogos, é que as pessoas pedem mais comida e as comem mais rápido. As razões são que o vermelho lembra sangue, significando perigo, e o amarelo é uma cor predominante nas comidas. (*MORRIS apud ROLLINGS, ADAMS, P. 06*)



Figura 8 – McDonalds

Praticamente todas as pessoas que jogam algum jogo acreditam que podem ser um bom game designer. A habilidade de fazer o design de um jogo é, para se dizer, imperceptível. Muitas vezes parece que qualquer pessoa está apta a fazer isso. O Game Design não pode ser reduzido a apenas escrever uma história simples, criar 1 ou 2 documentos e dizer aos programadores o que fazer. É a mesma coisa que dizer que qualquer pessoa que tiver um martelo pode criar um violão. Um bom design de um jogo é consequência de muita habilidade do game designer.

Pode-se verificar em revistas de jogos o quanto um jogo perde pontos por um mau design. Jogos podem parecer tecnicamente soberbos e terem gráficos maravilhosos, mas se a jogabilidade não agrada, ele perde muitos pontos e vendas frente aos jornalistas e, principalmente, aos jogadores.

Apesar do design de um jogo ser um processo criativo que requer a habilidade de sonhar e imaginar maravilhosos mundos populados pelas mais diversas e estranhas criaturas, existem muitos princípios práticos que devem ser analisados. Quando se conhece bem as técnicas do game design, o game designer pode, então, usar a sua imaginação e o seu intelecto para trabalhar naquilo que é bastante importante para um jogo, a jogabilidade.

### **3.3 Conceituação do Jogo**

É a primeira parte do game design. Nesta etapa é onde se cria e refina a idéia do jogo. Ao final desta etapa não será necessário que se saiba todos os detalhes de como o jogo funcionará, mas é necessário um claro entendimento sobre como será o jogo.

### 3.3.1 Criando a idéia do jogo

Idéias de jogos vêm de quase todos os lugares. Estimulando-se a criatividade consegue-se bastantes resultados práticos.. Das coisas mais simples do dia-a-dia podem surgir idéias para grandes jogos.

Muitas idéias de jogos vêm dos sonhos. Não dos sonhos que acontecem quando se está dormindo, mas sim daqueles que acontecem em momentos de divagação. A mente começa a se remoer, imaginando mundos, personagens e situações que fazem parte de seu desejo. Mundos que se deseja viver. Personagens que se almeja conhecer e, até mesmo, viver. Situações que podem ser que nunca sejam vivenciadas porque são impossíveis, como voar ou lutar contra monstros mutantes, ou aquelas que vivem no seu subconsciente e que se almeja vivenciar.

Idéias também podem surgir de livros, filmes, programas de televisão ou de outros meios de entretenimento. Jogos como Spider-Man, O Senhor Dos Anéis, dentre outros, que se encontram na *Figura 9*, seguem essa linha.



**Figura 9 – O Senhor dos Anéis e Spider-Man**

Também podem surgir idéias de outros jogos. Muitas vezes as pessoas se perguntam, no decorrer de uma partida de um jogo, se determinada ação não poderia ser feita de um modo diferente ou se alguma coisa poderia ter uma dificuldade diferente. Neste caso,

não basta apenas jogar o jogo. É necessário que se preste atenção enquanto joga-se. E, principalmente, que se façam anotações sobre todas as observações feitas, desde coisas que poderiam mudar a até coisas que estão funcionando de uma maneira que lhe agrada.

No processo de escolha da idéia do jogo, deve-se levar em conta também que, se for um jogo comercial, a idéia dele não deve agradar apenas um jogador. No caso, o game designer. Ela deve agradar a maior faixa de jogadores possíveis. O principal propósito de um jogo é prover entretenimento para outras pessoas. Algumas das primeiras perguntas que um publisher fará ao game designer para saber se deve ou não promover o jogo serão: "Por que alguém vai querer jogar esse jogo?" e "O que fará com que alguém compre esse jogo ao invés de outro?".

### **3.3.2 Os elementos de um jogo**

Enquanto filmes, peças, livros e televisão são todos formas de entretenimento passivo, onde a pessoa não espera participar da interação dos fatos, os jogos são exatamente o contrário. As pessoas que participam de um jogo esperam serem entretidas através da participação ativa no mesmo. Por isso, jogos são muito mais complicados de serem feitos. As pessoas amam o sentimento de envolvimento e de poder que os jogos provêem.

Um jogo acontece num universo artificial que é governado por regras. São essas regras que definem as ações e movimentos que os jogadores podem e o que eles não podem fazer no decorrer do jogo. Definir as regras do jogo é uma parte chave do game design. Existem jogos onde as regras já estão definidas, como é o caso do futebol. Nestes casos as regras não precisam ser definidas, apenas aplicadas. Um bom exemplo é o jogo *Winning Eleven*, da *Figura 10*.



**Figura 10 – Winning Eleven**

As regras também definirão os desafios que os jogadores terão que enfrentam para vencer o jogo e então deverá ser definida também uma regra que definirá a *condição de vitória* do jogador. Por condição de vitória entende-se o estado na qual um ou mais jogadores podem se encontram e que os definem como ganhadores.

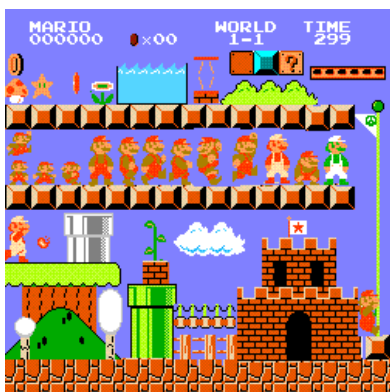
Outra coisa que deve ser definida é o mundo que o jogador interagirá. Por exemplo, um jogo de futebol tem um “mundo” definido, que é o campo de futebol. Será discutido um pouco mais sobre isso num tópico adiante.

Deve-se ter uma idéia também do modelo de interação entre o jogador para com o mundo. Aqui é feita a escolha para saber se o jogador poderá influenciar apenas sobre uma área local, como eu algum jogo no estilo aventura, ou poderá influenciar várias partes do mundo, como é o caso do Pac-Man, visto na *Figura 11*.



**Figura 11 – Pac-Man**

Depois vem a perspectiva a ser utilizada no jogo. Existem várias. Dentre elas, podemos destacar: top-down, isométrica, primeira pessoa, terceira pessoa e side-scrolling. A perspectiva em side-scrolling pode ser encontrada na *Figura 12*, no jogo Super Mario Bros.



**Figura 12 – Super Mario Bros.**

E, depois, deve-se ter em mente o quão real deverá ser o jogo. Por exemplo, no Flight Simulator, da Microsoft, quanto maior o realismo, melhor. O objetivo desse simulador de voo é ser o mais realista possível. Já no caso de jogos do estilo Mario, da Nintendo, o jogador pode cair de alturas inimagináveis e ainda assim continuar vivo. Essa é uma das escolhas que o game designer deve ter em mente na hora de criar o jogo.

Finalizando, no momento da concepção, é importante ter uma noção e saber sintetizar em uma frase qual é a história do jogador. Por exemplo: “Rodrigo Guedes, atirador da tropa de elite do exército brasileiro, tentará salvar a terra da manifestação de uma nova variação de um vírus letal para a raça humana, exterminando pássaros mutantes que hospedam esse vírus e que atacam os seres humanos” é uma síntese da história do demo que acompanha esta monografia.

### 3.4 O Mundo e a Fantasia do Jogo

O mundo de um jogo é um universo artificial, um lugar imaginário. Todo jogo, não importando o seu tamanho, acontece em um mundo. Muitos jogos têm um mundo físico ou, pelo menos, uma manifestação visível do mundo: um conjunto de cartas, um tabuleiro etc. Mesmo o jogo da velha contém um mundo: um pequeno entrelaçado de retas que é governado por regras e uma condição de vitória. Na *Figura 13* vemos o mundo do jogo *The Need for Speed Underground 2*.



**Figura 13 – The Need For Speed Underground 2**

A fantasia é um componente fictício dos jogos. Por exemplo, no xadrez, apesar do tabuleiro e do movimento das peças serem apenas uma idéia abstrata, as peças possuem nomes que remontam aos tempos medievais, recriando um mundo bélico em eras remotas.

Existem jogos cuja fantasia é desnecessária, como o próprio xadrez, mas há outros em que, sem a fantasia, não teriam razão de existir ou seriam jogos diferentes.

Uma regra geral que pode ser levada em consideração é que, quanto mais um jogo depende dos seus mecanismos centrais para entreter, menos a fantasia importa.

### 3.4.1 Imersão e a suspensão da desconvicção

Uma palavra chave que pode resumir a experiência de se ler um livro ou ver um filme é a suspensão da desconvicção. Suspensão da desconvicção é um estado mental na qual a pessoa pode se encontrar, por um período de tempo, onde ela acredita que tudo que está acontecendo faz parte da realidade. E isso também se aplica a jogos. Quanto melhor a ilusão que transcende um jogo, mais imerso no jogo o jogador se encontrará. E é esse um dos principais objetivos de um bom game design.

### 3.4.2 A importância da harmonia

Bons jogos e mundos passam a sensação de harmonia, isto é, um sentimento de que todas as partes do jogo pertencem a um único contexto.

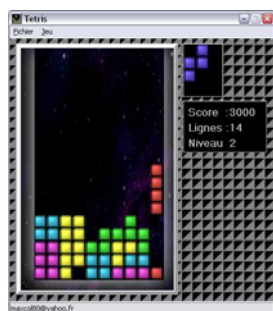
A harmonia é essencial num mundo de um bom jogo. A cada decisão tomada referente ao game design, deve ser perguntado se aquela idéia “pertence” ao contexto do mundo, se a harmonia não será quebrada. Não é recomendada a inserção de idéias simplesmente porque elas pareçam ser legais ou porque estejam na moda.

### 3.4.3 A dimensão física

A maioria dos jogos são implementados em algum tipo de espaço físico, onde o jogador pode interagir. Algumas decisões devem ser tomadas com relação a dimensão física de um jogo:

- **Dimensões:** uma das primeiras perguntas que deve ser feita refere-se a quantas dimensões deve ter o jogo. Se ele deve ser 2D, 3D ou até

mesmo 4D, sendo que, neste caso, existiria um mundo paralelo, onde as regras que regulariam esse mundo seriam diferentes das do outro mundo. Na *Figura 14* temos o Tetris, que é um jogo 2D;



**Figura 14 – Tetris**

- **Escala:** por escala entende-se o tamanho do espaço físico, assim como o tamanho relativo dos objetos. Em jogos com um estilo mais infantil, a escala nem sempre se aproxima da realidade, enquanto que, em jogos adultos, é recomendado que a escala fique o mais próximo possível da realidade;
- **Limites:** como computadores têm um espaço finito, o mundo físico também deve ter limites. Deve-se tomar cuidado ao escolher os limites do mundo para não quebrar a suspensão da desconvicção do jogador.

#### 3.4.4 A dimensão temporal

Ela define o modo com o tempo é tratado no mundo e os modos como ele difere em relação ao mundo real.

Em muitos jogos, o conceito da passagem do tempo não é incluído. Tudo acontece como se tudo estivesse acontecendo num mesmo momento. O cenário não escurece, não ocorrem trocas de estações etc.

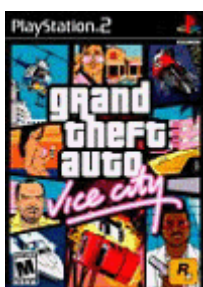
Para se fazer a inclusão da dimensão temporal, pode-se implementar:

- **Tempo Variável:** o tempo passa de uma maneira mais rápida que a realidade e, quando os eventos não são importantes como, por exemplo, quando o personagem vai dormir, o tempo real que decorre é insignificante;
- **Tempo Anômalo:** ocorre quando o tempo necessário para o crescimento de uma árvore, por exemplo, é o mesmo da poda da mesma, ou seja, o tempo transcorre de maneiras diferentes para as diversas atividades do jogo;
- **Deixando o jogador ajustar o tempo:** em jogos de esportes, por exemplo, um jogo de futebol, dificilmente um jogador gostaria de jogar 45 minutos cada tempo. Então, cabe ao jogo permitir que o jogador escolha o tempo real que deverá transcorrer na passagem de um tempo de 45 minutos.

### 3.4.5 A dimensão ética

A dimensão ética define o que é certo e o que é errado no contexto do jogo. Em um primeiro momento, essa dimensão pode parecer meio inapropriada para o contexto, mas em jogos que trazem uma fantasia, também há um sistema ético que define como deve se comportar o jogador.

No jogo deve haver uma análise sobre sua dimensão ética. Por exemplo, no jogo será permitido que se matem cidadãos inocentes ou somente terroristas? Haverá penalizações caso o jogador desrespeite alguma dessas regras? O objetivo do jogo fere alguma regra ética da sociedade? No demo que acompanha essa monografia, por exemplo, a idéia inicial era de se matar pássaros, mas como essa era uma idéia conflitante com a ética reinante na sociedade, os pássaros foram transformados em pássaros mutantes. Matar pássaros mutantes não é uma idéia tão conflitante perante a sociedade. Um outro bom exemplo é o jogo GTA, da *Figura 15*. Em sua primeira versão, o jogador ganhava pontos ao atropelar pessoas idosas, grávidas etc.



**Figura 15 – GTA**

### **3.5 Estória**

Jogos de computador geralmente têm algum tipo de estória anexada a eles. O aprofundamento e a importância da estória no jogo dependem muito do tipo de jogo a ser desenvolvido. Por exemplo, em jogos ao estilo Space Invaders a estória requer apenas uma linha: "ETs estão invadindo a terra e só existe uma pessoa capaz de acabar com eles: o jogador", enquanto que, em jogos ao estilo Grim Fandango, Final Fantasy, a estória é bem detalhada e envolvente, tão elaboradas quanto livros e/ou novelas.



**Figura 16 – Profundidade nas estórias da série Final Fantasy**

Segundo o livro “A Practical Guide to the Hero Journey”, de Christopher Vogler, a jornada de um herói segue os seguintes passos (*VOGLER apud ROLLINGS, ADAMS, P. 95*):

- **O Mundo:** o jogador é apresentado ao herói e ao mundo físico. A sua apresentação pode ocorrer de duas maneiras, sendo que na primeira os eventos que acontecerão ao herói são explicados ao jogador e, na segunda, eventos do passado que colidem com eventos presentes são apresentados ao jogador;
- **O Chamado:** ocorre algum evento convocando o herói a sair de seu mundo para entrar em um mundo especial, onde as aventuras o esperam;
- **A Recusa da Chamada:** aqui o herói luta contra a vontade de continuar em seu mundo seguro e a enfrentar os perigos e as aventuras do mundo externo, colocando em risco a sua vida. Em jogos, geralmente o herói não recusa a chamada;
- **O Encontro com o Mentor:** geralmente o mentor aparece na forma de uma pessoa velha e sua função é situar o herói quanto a sua localização e situação, provendo também dicas e sugestões para que o jogador se familiarize mais rápido ao jogo;

- **Atravessando o Primeiro Obstáculo:** é onde ocorre a mudança do mundo do personagem para o mundo das aventuras. Para entrar no mundo das aventuras, o herói deve estar pronto psicologicamente e cheio de coragem para enfrentar o desconhecido. Depois que o herói enfrentou seus medos, ele entra definitivamente no mundo do jogo;
- **Testes, Aliados e Inimigos:** a parte mais longa de uma estória, onde o herói fará testes, aprendendo sobre o chamado “bem e o mal” do mundo e também conhecerá aliados e inimigos que farão parte do desenvolvimento da estória;
- **A Aproximação à Caverna mais Profunda:** após vários testes, o herói se aproxima da parte mais esperada da estória. É utilizado para conduzir o herói à aprovação, que ocorrerá mais a frente;
- **A Aprovação:** é o último teste: lutar contra o chefe final. No decorrer da estória o herói deve ter passado por vários testes difíceis, mas este é o maior de todos. É nele que será definido o “futuro da humanidade”.
- **A Recompensa:** depois que o chefe final é derrotado, o herói recebe a recompensa. Geralmente é nessa parte que a estória termina, e o desfecho da estória é contado ao jogador;
- **A Estrada da Volta:** em alguns jogos, após o recebimento da recompensa, o herói se prepara para voltar da aventura a seu mundo. Mas como o jogador mudou muito no decorrer da estória, pode ser difícil, se não impossível, o jogador poder voltar a reintegrar o seu mundo;

- **A Ressurreição:** com todas (ou quase todas) as respostas das questões que surgiram na estória sendo respondidas, é aqui onde as dúvidas restantes são respondidas;
- **O Retorno com a Recompensa:** agora que a estória acabou, o herói volta para o seu mundo e retoma a sua vida normal. O jogador verá o herói aproveitar os benefícios de sua recompensa. Uma forma muito comum de se terminar é com um novo começo. Dessa forma, o jogador fica se perguntando “O que será que acontece depois?”, deixando uma brecha para uma continuação.

### 3.6 Desenvolvimento do Personagem

Foi-se o tempo em que os Personagens dos jogos eram meros blocos de pixels. Com a melhoria dos hardwares disponíveis, a questão da personalização dos personagens se tornou muito importante.

Atualmente o desenvolvimento de Personagens se dá de duas maneiras: o personagem é criado baseado na arte ou baseando na estória.

#### 3.6.1 Design de Personagens dirigidos à Arte

Muitos personagens são criados basicamente na aparência visual. Por exemplo, o Pac-Man, visto na *Figura 17*, foi inspirado numa pizza sem um pedaço. A heroína Lara Croft, da *Figura 18*, foi criada baseando-se na garota dos sonhos do artista.

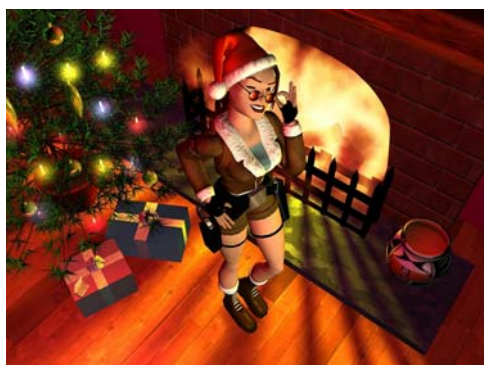


**Figura 17 – Pac-Man**

Personagens que foram desenvolvidos de fontes puramente artísticas tendem a ser mais superficiais do que aqueles baseados nas histórias. E em muitos jogos não há a necessidade de personagens bem desenvolvidos. Neste caso é muito melhor criar um personagem sem personalidade e deixar a cargo do jogador criar essa personalidade.

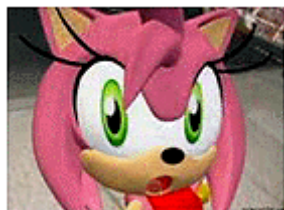
Muitos desses personagens são criados com propósitos bem definidos, apelando-se para aparências:

- **Mais sexys:** a forma do corpo é exagerada, aumentando-se os seios e os quadris das personagens femininas, diminuindo a barriga, de modo a abrigar apenas os órgãos essenciais, dando a essas personagens um maior apelo sexual, conforme pode ser visto na *Figura 18*, da Lara Croft;



**Figura 18 – Lara Croft**

- **Mais engraçadinhos:** a cabeça não é proporcional ao corpo, assim como os olhos, mais ou menos ao estilo dos animes japoneses. Na *Figura 19* temos um exemplo de uma personagem engraçadinha.



**Figura 19 – Personagem Engraçadinha**

### 3.6.2 Design de Personagens dirigidos à Estória

O melhor jeito de se criar um personagem bem definido é criar uma estória e, com base na estória, desenvolver as características e a personalidade do personagem antes de se considerar a sua aparência. Geralmente os artistas preferem trabalhar com descrições detalhadas, o que permite a eles entender e visualizar melhor o personagem.

Para se criar personagens de grande apelo, três regras de ouro devem ser seguidas:

- O personagem precisa intrigar o jogador;
- O personagem precisa cativar o jogador, fazendo com que o jogador goste dele;
- O personagem precisa mudar e crescer de acordo com a sua experiência no decorrer do jogo.

Pode-se verificar que essas regras são seguidas à risca nos jogos da série Final Fantasy. Alguns desses personagens podem ser vistos na *Figura 20*.



**Figura 20** – Personagens da série Final Fantasy

### 3.6.3 Esteriótipos de Personagens

Existem vários esteriótipos de personagens nas estórias. Alguns dos mais utilizados são:

- **O Herói:** tradicionalmente, é ele o centro da estória. Na literatura, o herói é o personagem com um ou mais problemas, que busca resolver esses problemas no decorrer da estória. O mais importante quanto ao herói é garantir que os jogadores se identifiquem com ele. O herói precisa ter qualidades que o jogador simpatize e aprecie;
- **Mentor:** é o personagem que guia o jogador na jornada. Nem sempre ele guia para o melhor caminho;
- **Alto Suficiente:** é o herói como ele aspira ser. É a forma ideal do herói. Em muitos jogos o objetivo do jogo é transformar o herói na sua forma alto suficiente;
- **Aliados:** são personagens colocados no jogo para ajudar o herói. Eles podem ajudar o herói tanto a desvendar mistérios, como a completar tarefas;

- **Guardiões:** sua função é evitar que o herói progrida;
- **Malandros:** são personagens neutros, cuja maior felicidade é causar danos ao herói. Eles podem tanto ajudar o herói, quanto o inimigo;
- **Sombra:** é, provavelmente, o segundo personagem mais importante da estória, depois do herói. É o último demônio, o grande adversário do herói.

### 3.7 Criando a Experiência do Jogador

Um jogo é muito mais do que apenas a soma de suas regras. Ele deve interagir com o jogador de modo a imergi-lo no mundo do jogo. A partir do primeiro momento que o jogador entra no jogo, tudo aquilo que ele vê, ouve, e sente deve convencê-lo de que a única coisa que existe é o jogo.

E é isso que é a experiência do usuário: uma combinação das três áreas distintas do design – o elemento visual, o elemento auditivo e o elemento interativo.

- **O Elemento Interativo:** é o modo como o jogador interage com o jogo. É mais relacionado com os aspectos funcionais da interface do usuário;
- **O Elemento Visual:** é todo o impacto da arte e como ela se combina para apresentar o mundo ao jogador;
- **O Elemento Auditivo:** apesar da parte do áudio não parecer tão importante quanto os outros 2 elementos, ela é tão importante quanto. Vários jogos ficam inviáveis de se jogar sem o som.

### 3.7.1 Criando a Interface Homem-Computador

Desenvolvendo uma boa interface para o usuário não é apenas deixá-la bonita e com um som legal quando se interage com os botões. O mais importante é disponibilizar ao jogador a melhor forma de interação para com o mundo do jogo. Quando se estiver desenvolvendo uma interface, deve-se pensar sempre em tornar a interface rápida e mais eficiente quanto ao acesso aos recursos, como foi feito no jogo Outlive, na *Figura 21*.



**Figura 21 - Outlive**

A funcionalidade da interface do usuário é a consideração mais importante a ser levada em conta. Afinal, que jogador utilizará seu tempo para jogar um jogo bonito se ele não puder reconhecer qual botão é responsável pelo começo do jogo?

Antes de se implementar uma interface em computador, recomenda-se fazer um esboço no papel e verificar se a interface funciona.

#### 3.7.1.1 Disposição da interface

A disposição da interface varia muito de jogo para jogo, mas as partes mais utilizadas para dispor a interface de um jogo são: a lateral esquerda do cenário, uma tira bem

finda na parte superior e a parte inferior da tela, como pode ser visto na *Figura 22*.  
(ROLLINGS; ADAMS, 2003 – pág. 169)



**Figura 22 – Áreas mais Utilizadas para Interfaceamento**

### **3.7.1.2 Elementos para uma interface do usuário simples**

A função primária de uma interface do usuário é interagir com o usuário. Função essa que pode ser dividida em duas tarefas principais: aceitar os comandos do jogador e informar ao jogador seu estado atual, bem como mostrar as opções disponíveis no formato gráfico mais simples possível.

O que a interface não deve fazer é distrair o jogador do jogo, desordenar a tela, confundir o jogador com informações estranhas ou desnecessárias ou ofuscar informações vitais.

No uso de textos, não é aconselhado o uso de TEXTOS GRAFADOS COM TODAS AS LETRAS EM MAIÚSCULAS, nem com todas as letras em minúsculo. Aconselha-se utilizar uma mistura, utilizando-se letras maiúsculas no início das frases e letras minúsculas no restante das palavras.

Já no caso dos ícones, não se deve utilizar ícones que tenham uma conotação mais local. O melhor é utilizar um ícone reconhecido mundialmente para o uso na interface.

Recomenda-se utilizar bastantes ícones, pois estes facilitam o entendimento do contexto da situação.

Simplifique ao máximo as opções disponíveis ao jogador. Quando for disponibilizar opções ao jogador na forma de ícones, crie botões com esses ícones e disponha-os na tela, de modo a criar um menu com largura igual a profundidade, onde a largura é a quantidade de opções de nível superior disponível e a profundidade corresponde a quantidade de opções abaixo do menu de maior nível.

A seguir algumas dicas de como desenvolver uma boa interface:

- **Seja Consistente:** o jogador apenas executará seqüências consistentes de ações em situações similares. Portanto a terminologia utilizada nos menus e nas telas do jogo devem ser idênticas. O uso de cores, fontes e a disposição na tela devem ser consistentes e logicamente conectados nas diversas seções do jogo;
- **Permita aos jogadores experientes o uso de atalhos:** disponibilize teclas especiais, comandos secretos e outras funcionalidades que permitam que se jogue mais rapidamente;
- **Dê bons feedbacks:** Para todas as ações do usuário, o jogo deve responder de alguma maneira imediatamente;
- **Crie a interface para oferecer tarefas definidas:** as seqüências das ações que o jogador pode executar devem estar organizadas em grupos conceituais de pequenas sub-tarefas, cada uma bem-definida, com começo, meio e fim. Ao fim de cada tarefa, deve-se informar ao jogador que a tarefa foi findada;

- **Não permitir que o jogador faça erros bobos e permitir que o jogador recupere-se de erros pequenos:** o jogador não deve ser capaz de fazer erros sérios simplesmente por ter clicado ou digitado algo errado. Todas as entradas do jogador devem ser verificadas. Se o jogador criar um erro de entrada que não possa ser tratado pelo jogo, deve-se permitir ao jogador a recuperação ao estado anterior;
- **Permitir fácil reversão das ações:** caso o jogador faça um erro bobo, permitir ao jogador rever a ação, a menos que isso afete o balanço do jogo;
- **Lembrar que o jogador é quem está no controle:** o jogador quer sentir que comanda o jogo, pelo menos com relação ao herói. Não crie eventos randômicos e incontroláveis, ou seqüências tediosas ou difíceis de serem realizadas.

E sempre que possível, deve-se utilizar representações visuais ao invés de números ou frases, quando for dar o feedback do estado atual do jogo para o jogador.

É recomendada a utilização de sons que correspondam a ações ou eventos que ocorram no mundo do jogo – por exemplo, o tiro de uma arma quando se dispara, o grito de um monstro quando ele vai atacar o herói etc. Com a melhoria das capacidades de reprodução de sons pelo computador, o som também se tornou um grande responsável pelo feedback ao jogador.

E não pode ser esquecido que o uso de músicas de fundo aumenta a imersão do jogador. São elas que dão o tom de tensão na disputa da última volta da corrida, ou na luta contra o chefe final do jogo.

### **3.8 Balanceamento do Jogo**

Um jogo balanceado é aquele onde o principal fator determinante do sucesso ou não do jogador é o nível de habilidade do jogador no jogo, ou seja, um jogador bom deve ter maior sucesso que um iniciante, a menos que aconteça algo de anormal.

Tradicionalmente, o balanceamento do jogo tem sido mais um processo de tentativa e erro. Joga-se e depois conserta.

Há diversas maneiras de implementar o balanceamento do jogo. Em particular, existem duas classes de balanceamento que serão discutidas: balanceamento estático e dinâmico.

#### **3.8.1 Balanceamento estático**

Balanceamento estático se refere ao balanceamento das regras do jogo e como elas interagirão entre si. Um exemplo concreto seria os diferentes valores das forças das unidades num jogo de estratégia.

No balanceamento estático, dever-se evitar o surgimento de estratégias dominantes, que são estratégias que são as melhores de serem escolhidas sob quaisquer circunstâncias. Estratégias dominantes fortes são aquelas que garantem a vitória, enquanto as estratégias dominantes fracas garantem que o jogador não perca.

Uma estratégia dominante forte nunca é desejada, mas deve-se tomar cuidado ao retirar algumas estratégias dominantes fracas – às vezes elas podem ter seu valor.

Dando aos jogadores, incluindo ao computador, as mesmas condições de início e habilidades disponíveis, é uma das maneiras mais simples de se balancear um jogo.

Essa tática se chama simetria. O Desafio Sebrae utiliza da simetria ao final de cada etapa para balancear o jogo.

Outra coisa que pode ser feita é, ao mesmo tempo que se dá um maior poder de fogo ao usuário, aplicar o mesmo raciocínio ao seu adversário. Por exemplo, se o jogador conseguir uma arma que tire mais dano dos monstros, um pouco depois se deve utilizar monstros mais fortes para atacar o jogador.

### 3.8.2 Balanceamento dinâmico

O game designer deve também se preocupar em manter o balanceamento no decorrer do desenvolvimento do jogo. O balanceamento dinâmico considera como o balanceamento muda no decorrer do tempo e com a interação do jogador.

Um jogo deve, inicialmente, estar balanceado estaticamente. Depois, o balanceamento dinâmico deve ser mantido. O sucesso ou a falha do game designer no gerenciamento do balanceamento do jogo define como será o jogo.

Há vários modos de o jogador interagir com o balanceamento dinâmico. Os 3 (três) modelos seguintes de interação estão disponíveis:

- **Restaurar o Balanço:** a tarefa do jogador é restaurar o balanço do jogo para um ponto de equilíbrio. Um exemplo é o quebra-cabeça, onde as várias peças começam desorganizadas e a condição de vitória é o jogador trazer o jogo de volta ao seu estado balanceado, ou seja, montar o quebra-cabeça;
- **Mantendo o Balanço:** o objetivo do jogo é manter o balanceamento do jogo, cujo oponente tenta desbalancear o jogo. Neste caso, não

existe uma condição de vitória. Uma hora o jogador irá perder. Esse é o caso do Tetris;

- **Destruindo o Balanço:** é o oposto do primeiro modelo. Aqui o jogador deve destruir o balanço. Um jogo que exemplifica bastante esse modelo é o jogo Outlive, da Continuum.

### 3.8.3 Criando sistemas balanceados

As seguintes regras podem ajudar no desenvolvimento de jogos balanceados:

- **Prover um Desafio Consistente:** a dificuldade do jogo deve escalar de acordo com o progresso do jogador. O fim deve ter sempre uma dificuldade maior que o começo e o meio do jogo;
- **Prover uma Percepção de uma Experiência Justa:** o jogador não deve sentir que está competindo com um jogador (no caso o computador) desleal, que trapaceia. Ele deve achar que as condições de luta devem ser iguais;
- **Evitar a Estagnação:** a estagnação acontece quando um jogador está num determinado ponto do jogo e não sabe o que fazer. Isso acontece bastante em jogos no estilo Doom;
- **Evitar as Trivialidades:** o jogador não deseja que ele tenha que tomar decisões triviais quando ele se encontra em níveis mais altos. Ele deseja ter que tomar decisões que estimulem o seu cérebro;
- **Fixar o Nível de Dificuldade:** tradicionalmente não são todos os jogos que disponibilizam vários níveis de dificuldade, mas é essa

disposição que atrairá um maior contingente de pessoas a jogarem o jogo, já que ele atenderá aos diversos níveis de habilidade dos jogadores.

### **3.9 Bibliografia Referente ao Capítulo 3**

#### **(RAMOS, 2004)**

RAMOS, Alexandre Carlos Brandão. Notas de aula da matéria CCO-27. 2004. Itajubá.

Disponível em [www.ici.unifei.edu.br/ramos](http://www.ici.unifei.edu.br/ramos)

#### **(ROLLINGS; ADAMS, 2003)**

ROLLINGS, Andrew; ADAMS, Ernest. Andrew Rollings and Ernest Adams on Game Design. 1ª edição. Nova Iorque: New Riders, Maio de 2003. 622 páginas.

## **CAPÍTULO 4 – INTELIGÊNCIA ARTIFICIAL**

Esta parte é dedicada ao tópico de Inteligência Artificial em jogos, uma área do game design que está sendo levada cada vez mais a sério pelos desenvolvedores e produtores de jogos.

Houve um grande crescimento de interesses nos últimos anos tanto por parte dos desenvolvedores de jogos, quanto dos jogadores, a fim de usar melhores técnicas de IA nos jogos. Por isso, neste capítulo será explicado um pouco sobre essas técnicas de IA.

### **4.1 O que é?**

“Inteligência Artificial é um ramo da ciência que se destina a ajudar máquinas a achar soluções para problemas complexos de uma maneira mais humana possível. Isto geralmente implica no empréstimo de características da inteligência humana e as aplicando na forma de algoritmos em um computador de uma maneira amigável. Uma aproximação mais ou menos flexível ou eficiente pode ser conseguida dependendo dos requisitos estabelecidos, que influenciarão do quão artificial deverá parecer o comportamento da inteligência.” *(IA DEPOT, 2004)*

### **4.2 Técnicas Mais Comuns de IA utilizadas em Jogos**

Em jogos, utilizam-se várias técnicas de IA. Dentre as mais utilizadas, podemos citar:

- **Máquinas de Estado Finito:** geralmente se referem a um modelo simples da IA usado nos jogos. MEF consiste de  $n$  estados, cada um com suas próprias características. A IA então decide para qual estado ir para governar o comportamento a ser adotado. Um exemplo de MEF são os engines da IA de jogos do estilo First Person Shooter (FPS) – se um jogador estiver visível, ir para o estado de ataque, se o jogador não estiver no campo de visão, trocar para o estado de procura do jogador ou para o estado de ficar parado. MEFs são muito fáceis de serem implementadas, e provêm Inteligência Artificial funcionáveis para jogos simples (*GENERATION5, 2004*). Na *Figura 23*, temos um exemplo de um jogo de FPS, o Doom 3, que utiliza essa técnica de IA;

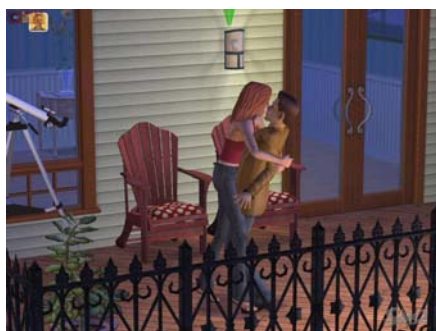


**Figura 23 – Doom 3**

- **Algoritmos de Path-Finding:** são algoritmos utilizados no cálculo do menor caminho entre dois pontos, fazendo os cálculos para desviar de possíveis obstáculos, inimigos etc. encontrados no decorrer do caminho. Existem vários algoritmos representantes dessa categoria, mas os algoritmos mais famosos são o BFS (Best First

Search) e os algoritmos que manipulam grafos, tais como o de Dijkstra e o A\* (A Estrela);

- **Scripting:** é uma técnica bastante utilizada em jogos. Nela podem ser feitas programações das mais variadas áreas referentes ao jogo, à exceção da renderização 3D. É bastante utilizada na área de IA. Scripting nada mais é do que criar um “pseudo-compilador”, onde os comandos escritos em um arquivo texto são interpretados em tempo de execução, diminuindo o tempo de desenvolvimento causado pela compilação de um jogo. As linguagens script Lua e Python são as mais utilizadas pelas empresas comerciais;
- **Algoritmos Genéticos:** são algoritmos que “pegam emprestado” a característica da genética dos seres vivos, baseando-se nos genes, que carregam características dos seres vivos, para que estes, quando praticam a reprodução, desenvolvam seres mais aptos a enfrentar o meio ambiente. O mesmo acontece com esses algoritmos, onde as características dos seres controlados pela IA do jogo possam ser combinados, de modo a formar seres mais eficazes de oferecerem maiores desafios aos jogadores;
- **A-Life:** é a parte referente aos algoritmos que simulam o desenvolvimento da vida dos seres vivos, tanto dos animais, quanto dos homens da maneira mais natural possível. Um bom exemplo na área de jogos é o jogo chamado The Sims, que pode ser visto na *Figura 24*, onde são feitas simulações da vida de seres vivos que, neste caso, são seres humanos;



**Figura 24 – The Sims**

- **Redes Neurais:** elas se estruturam baseadas no modelo neural do cérebro do ser humano. São utilizadas para o reconhecimento de padrões e podem sempre estar adquirindo novos conhecimentos. Em jogos ainda são pouco utilizadas.

### **4.3 IA do Demo**

No demo desenvolvido juntamente com essa monografia, a IA foi criada programando-se a utilização das técnicas explicadas abaixo.

#### **4.3.1 Máquinas de Estados**

No caso da IA do pássaro mutante, utilizou-se a máquina de estados finitos por sua fácil implementação e por ela se encaixar devidamente no resultado pretendido para o demo desenvolvido juntamente com essa monografia.

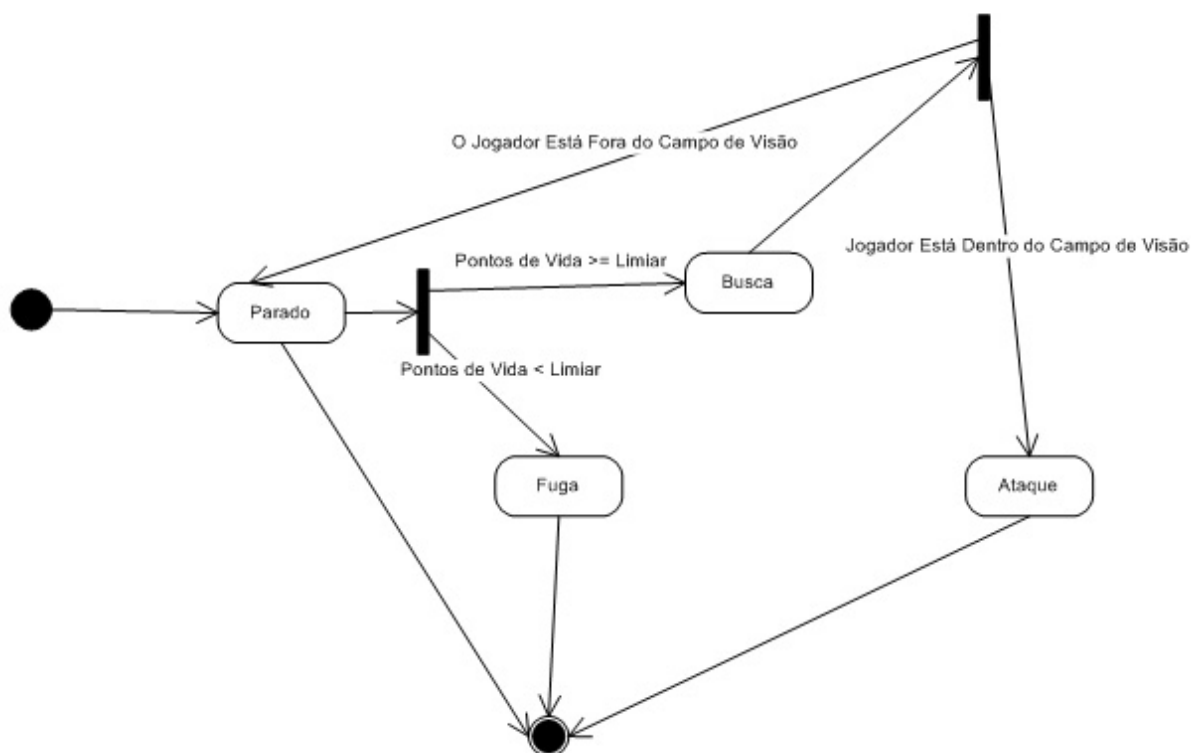


Figura 25 – Máquina de Estados da IA do Pássaro Mutante

Como pode ser visto na *Figura 25*, o pássaro mutante pode se encontrar em

4 estados:

- **PARADO:** o pássaro mutante fica voando, mas ele não muda de lugar. Ele se mantém neste estado mesmo que leve um tiro do jogador e o jogador se encontre fora de seu campo de visão;
- **BUSCA:** o pássaro mutante busca o jogador dentro de seu campo de visão, a fim de atacá-lo;
- **ATAQUE:** o pássaro mutante ataca o jogador, indo na direção que o jogador se encontra;
- **FUGA:** o pássaro mutante foge do jogador, indo na direção exatamente contrária a posição que se encontra o jogador.

No jogo, o pássaro mutante verificará a quantidade de pontos de vida que ele possui. Caso ele tenha menos pontos de vida que o limiar definido pelo nível que o usuário se encontra, o pássaro mutante fugirá do usuário até que ele encontre no limite do cenário. Chegando a esse limite, são adicionados 100 pontos de vida ao pássaro mutante que, a partir desse momento está novamente apto a atacar novamente o jogador.

No caso do pássaro mutante ter mais pontos que o limiar, ele verificará se o jogador se encontra dentro de seu campo de visão. Se ele se encontrar, o pássaro mutante o atacará. Caso contrário, ele voltará para o estado de parado.

Ao final desse ciclo, o pássaro mutante volta ao estado de parado novamente e refaz todo esse caminho pela máquina de estados. Vale ressaltar que essa máquina de estados somente será válida caso o pássaro mutante se encontre vivo, ou seja, seus pontos de vida sejam maiores que 0 (zero).

#### 4.3.2 Algoritmo genético

A IA do pássaro mutante é fundamentada em 3 (três) atributos membros de sua classe:

- **Limiar:** é o limiar que definirá se o pássaro mutante atacará o jogador. Caso o número de pontos de vida do pássaro mutante for menor que esse limiar, ele fugirá do jogador, caso contrário, ele pode tanto ficar voando no mesmo lugar ou atacar o jogador;
- **Velocidade:** define a velocidade do pássaro durante o voo. Quanto maior o nível do jogo, maior será a velocidade do pássaro. Tomou-se o cuidado para que o pássaro mutante nunca tenha uma

velocidade maior que a do jogador, que é de 40 (quarenta), nos diversos níveis do jogo;

- **Distância:** define o campo de visão que o pássaro mutante terá para atacar o jogador. Os valores deste atributo foram verificados para que a distância nunca ultrapasse o tamanho do cenário.

Todos esses atributos para os diversos níveis do jogo são definidos no código-fonte do jogo.

Já para a utilização de algoritmos genéticos, colocou-se um atributo em cada pássaro com o tempo de vida de cada um, sendo possível assim utilizar esse dado para que seja feita a escolha dos pássaros que mais viveram e criar novos pássaros dinamicamente com base nos dados dos atributos desses pássaros, que foram os mais preparados. Lógico que, no decorrer da criação desses novos pássaros mutantes, pode-se criar mutações desses dados, acrescentando-se valores aleatórios para os atributos dos pássaros explicados acima.

### 4.3.3 Scripting

Ao contrário do que muitos acreditam, para se fazer scripting não necessariamente é obrigatório o uso de uma linguagem script, como Lua ou Python. Um arquivo INI do Windows, por exemplo, é um arquivo script. Na programação de jogos, os programadores podem criar suas próprias linguagens de scripting, assim como utilizar um arquivo com uma estrutura parecida com o do arquivo INI do Windows para a implementação de scripting.

Por isso, no código-fonte do jogo, retira-se os parâmetros dos atributos dos pássaros mutantes de cada nível e transfere-se para um arquivo texto, onde os dados poderão ser alterados livremente, de maneira que não seria mais necessária a recompilação do código-

fonte toda vez que os valores dos atributos de um determinado nível fossem alterados. Dessa maneira, muito tempo será economizado na fase de testes do demo, quando seriam escolhidos os melhores valores para cada atributo nos níveis utilizados pelo demo.

#### 4.3.4 Algoritmo de fuga e ataque do pássaro mutante

Para que o pássaro mutante pudesse interagir com o jogador, foram criados métodos que criavam o vetor direção que o pássaro deveria seguir para atacar e fugir do jogador.

O pseudo-código dos métodos referentes ao ataque e à fuga dos pássaros mutantes com relação ao jogador são mostrados a seguir:

*Método perseguir ( posicaoDoJogador, altitudeDoTerreno )*

```
{
    Se o pássaro está vivo Faça
        destinoDoPassaro = posicaoDoJogador – posicaoDoPassaro;
        Normalizar(destinoDoPassaro);
        posicaoDoPassaro += destinoDoPassaro * velocidadeDoPassaro;
    fim do Se
}
```

*Método fugir ( posicaoDoJogador, altitudeDoTerreno )*

```
{
    Se o pássaro está vivo Faça
        destinoDoPassaro = posicaoDoJogador – posicaoDoPassaro;
        Normalizar(destinoDoPassaro);
        posicaoDoPassaro -= destinoDoPassaro * velocidadeDoPassaro;
    fim do Se
}
```

#### 4.4 Bibliografia Referente ao Capítulo 4

**(AMIT, 2004)**

AMIT'S A\* PAGES, capturado no dia 01/11/2004, On-line, Disponível na Internet

<http://theory.stanford.edu/~amitp/>

**(GENERATION5, 2004)**

GENERATION5, capturado no dia 01/11/2004, On-line, Disponível na Internet

<http://www.generation5.org>

**(IA DEPOT, 2004)**

IA DEPOT, capturado no dia 01/11/2004, On-line, Disponível na Internet <http://ai-depot.com>

**(TOLEDO, 2003)**

TOLEDO, Siles Paulino de. Apostila de Introdução à Inteligência Artificial da Matéria de

IAA-01. 2003. Itajubá. Disponível em [www.siles.unifei.edu.br](http://www.siles.unifei.edu.br)

## **CAPÍTULO 5 – PROGRAMAÇÃO**

### **5.1 APIs para a Programação**

Para se programar um jogo, pode-se utilizar praticamente todas as linguagens de programação, mas a linguagem mais utilizada no desenvolvimento de jogos comerciais é a linguagem C/C++.

Juntamente com o C/C++, utiliza-se uma API gráfica. As mais utilizadas no momento são o OpenGL e o Direct3D.

O OpenGL é uma API gratuita e bastante leve, disponibilizada pela Silicon Graphics. Ela é mais utilizada no ensinamento de conceitos de Computação Gráfica, por ser mais simples e de implementação mais fácil.

Já o Direct3D faz parte do pacote DirectX, sendo também uma API gratuita e desenvolvida pela Microsoft. É a API mais utilizada para o desenvolvimento de jogos por ter atualizações mais constantes, bem como fazer parte do pacote DirectX, que oferece suporte à redes, à exibição de vídeos, à execução de áudio, dentre outras coisas.

Mesclava-se também o uso do OpenGL com as demais bibliotecas do DirectX, para o desenvolvimento de jogos, mas atualmente essa é uma opção muito rara. Atualmente é utilizada o Direct3D juntamente com as demais bibliotecas do DirectX para o desenvolvimento de jogos.

Entre os jogos feitos utilizando-se OpenGL com as bibliotecas do DirectX, podemos citar Worms 3D e Final Fantasy 7. Já utilizando-se DirectX, temos The Need For Speed Underground, Max Payne 1 e 2, Outlive etc.

Já no caso de projetos de pesquisa sobre jogos 3D no ambiente acadêmico, utiliza-se bastante Java com a biblioteca JOGL, que nada mais é do que o uso do OpenGL para a linguagem Java. A biblioteca Java3D também é bastante utilizada para esse fim.

Apesar dessa escolha acadêmica, excetuando-se os jogos para celular, o aconselhável é a utilização da linguagem C/C++ para jogos comerciais, devido a sua maior velocidade de execução e pelo fato das bibliotecas da linguagem Java ainda não explorarem todos os recursos das placas 3D atuais.

No caso de celulares, para a programação de jogos, utiliza-se mais a linguagem Java, por conta de sua portabilidade, facilitando o desenvolvimento de um mesmo jogo para os diversos modelos de celulares. Neste caso é utilizado o J2ME, Java 2 Micro Edition, que é uma API Java voltada para micro-aplicativos que rodam em micro-processadores. Atualmente utiliza-se o J2ME MIDP 2.0 para a programação de jogos para celular.

Alguns jogos para celular são feitos utilizando-se C/C++, mas esses jogos são aqueles jogos mais trabalhados, que demoram mais para serem feitos e são feitos exclusivamente para um modelo de celular, como é o caso dos jogos do N-Gage, da Nokia.

Para o desenvolvimento do jogo que acompanha essa monografia foi feita a escolha da API DirectX, bem como o uso da linguagem de programação C/C++, por sua maior atualização com relação aos recursos disponibilizados e pela maior velocidade de execução do arquivo executável gerado.

## 5.2 Estrutura dos Jogos

A estrutura dos jogos é extremamente complexa. Na verdade, muitas vezes os jogos podem ser muito mais complexos que grandes programas comerciais.

Para se programar um jogo, deve-se aprender um novo modo de programação voltada para aplicações de tempo real e simulações, em detrimento da programação seqüencial, dirigida a eventos, mais comuns em programas comerciais.

O algoritmo de um jogo é, basicamente, um loop contínuo, em que são feitos os cálculos referentes a IA, a renderização das imagens na tela, a uma taxa de 30 ou mais quadros por segundo.

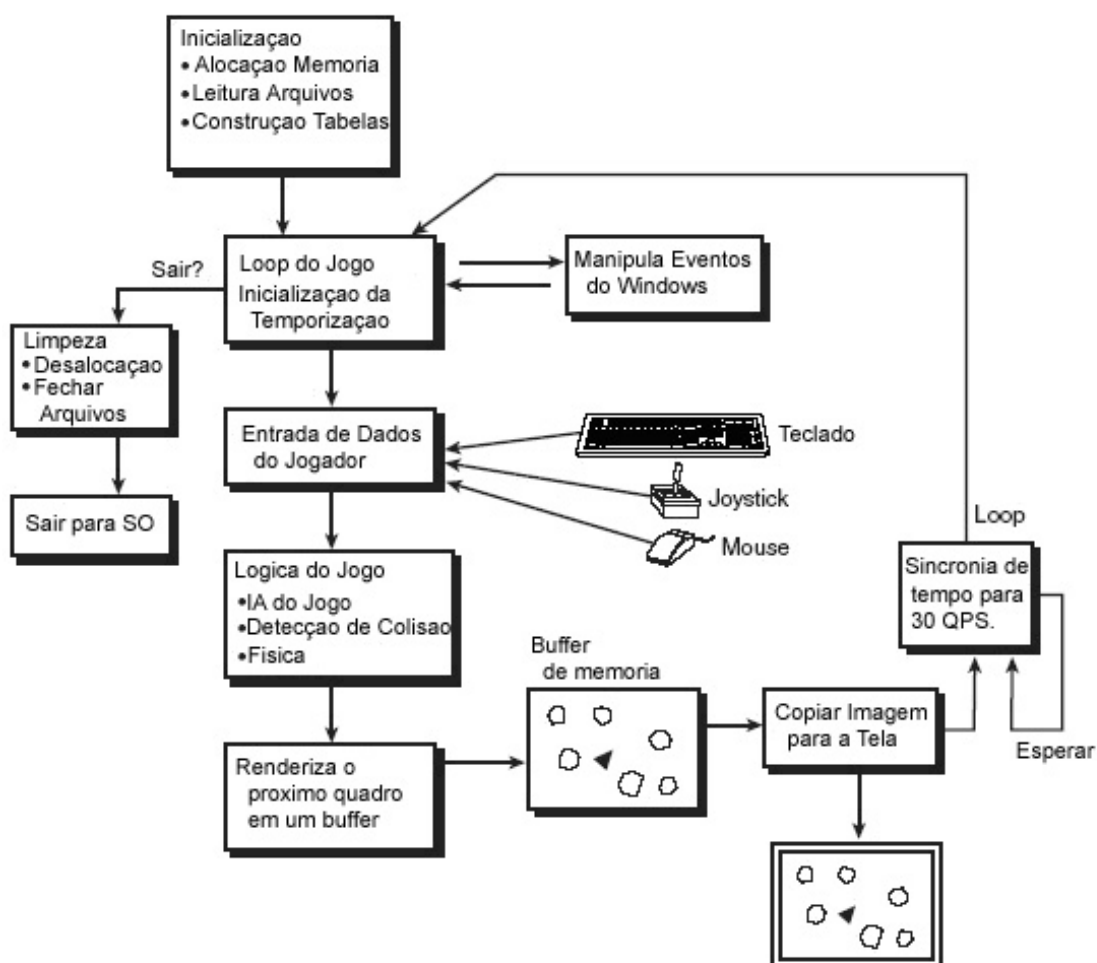


Figura 26 – Estrutura do Código de um Jogo

Conforme pode ser visto na *Figura 26*, um jogo possui as seguintes partes:

**(a) Inicialização**

Nesta parte, são efetuadas as operações padrão de qualquer programa, como alocação de memória, leitura de dados do disco, inicialização das variáveis etc.

**(b) Loop do jogo**

Aqui é o loop contínuo onde o motor do jogo é executado. É aqui que as ações do usuário são processadas até que o mesmo saia do jogo.

**(c) Entrada de dados do jogador**

Já nesta fase, as entradas do jogador são processadas ou guardadas em algum buffer para depois serem utilizadas pela IA ou pela lógica do jogo.

**(d) Lógica do Jogo**

É nesta seção que se encontra a maior parte do código do jogo. Pode ser encontrado aqui o processamento da lógica do jogo, da inteligência artificial e da simulação da física, sendo os resultados utilizados para a renderização do próximo quadro que será exibido na tela.

**(e) Renderização do Próximo Quadro**

Nesta parte é feita a coleta das entradas do jogador e dos resultados provenientes da execução da IA e da lógica do jogo para ser criado o próximo quadro a ser exibido. Geralmente, essa imagem é desenhada em um buffer que não é o mesmo da tela. Depois, ela é copiada rapidamente para a área de memória que representa o que está sendo mostrado na tela.

**(f) Sincronia da Exibição dos Quadros**

Dependendo do computador e do quadro que deverá ser renderizado na tela, o tempo necessário para o processamento dos dados para a realização do mesmo pode variar muito. Por isso, o número de quadros do jogo será muito variável. Neste caso, existem dois

caminhos a serem seguidos: criar algum método de espera ou fazer com que as entradas do usuário sejam aplicadas de maneira proporcional ao tempo necessário para a renderização do quadro.

### (g) **Limpeza**

Aqui é o fim do jogo. Isso significa que o usuário quer sair e voltar para o sistema operacional. Mas, antes disso, é necessária a desalocação dos dados e do fechamento dos arquivos que estavam sendo utilizados.

#### 5.2.1 Código-fonte de exemplo

```
#define GAME_INIT 0
#define GAME_MENU 1
#define GAME_COMECANDO 2
#define GAME_EXECUTANDO 3
#define GAME_SAINDO 4

// variáveis globais
int estado_jogo = GAME_INIT; //controla o estado do jogo
int erro = 0; //usada para reconhecimento de erros

// função main
void main()
{
    while (estado_jogo!=GAME_SAINDO)
    {
        //faz a verificação do estado em que o loop do jogo se encontra
        switch(estado_jogo)
        {
            case GAME_INIT: // o jogo está sendo inicializado

                // é feita a alocação de memória e dos recursos
                Inicializacao();

                // move para o estado de menu
                estado_jogo = GAME_MENU;
                break;

            case GAME_MENU: // o jogo está no modo de menu
```

```
    // recebe a escolha do usuário, depois da interação com o menu
    estado_jogo = Menu();
    break;

case GAME_COMECANDO: // o jogo vai ser executado

    // é um estado opcional, utilizado para preparar o jogo
    // para ser rodado
    PreparacaoParaExecutar();

    // muda para o estado de execução
    estado_jogo = GAME_EXECUTANDO;
    break;

case GAME_EXECUTANDO: // o jogo está rodando

    // esta seção contém o loop da lógica do jogo
    LimpaTela();

    //se o usuário decidir sair do jogo, é aqui que a variável
    //estado_jogo é alterada
    RecebeEntradaUsuario();

    //executa a lógica do jogo
    ExecutaLogicaIA();
    ExecutaLogicaFisica();

    //renderiza o próximo frame
    RenderizaProximoFrame();

    //espera o tempo necessário para a execução de um determinado
    //número de frames
    Espera();

    break;

case GAME_SAINDO: // o usuário está saindo do jogo

    //faz a desalocação da memória alocada
    //e dos recursos utilizados
    DesalocaMemoria();
    LiberaRecursos();

    break;
default:
    break;
} // fim do switch
} // fim do while
```

```
    // retorna o erro
    return(erro);
} // fim do main
```

### 5.3 Técnicas Utilizadas no Demo

No decorrer do desenvolvimento do demo, foram utilizadas várias técnicas utilizadas na produção de jogos comerciais. Dentre essas, neste tópico serão explicadas duas técnicas utilizadas. São elas: mapas de altitude e billboard.

#### 5.3.1 Mapas de altitude

Para se criar terrenos 3D, são criadas malhas de pontos, que formam triângulos, onde estes formarão os terrenos 3D.

Para criar estes terrenos, utiliza-se mapas de altitude, que descrevem as colinas e os vales do terreno. Um mapa de altitude é um vetor onde cada elemento especifica a altitude de um vértice do terreno.

Para se armazenar esses mapas de altitude em disco, geralmente, são criados arquivos de imagens em tons de cinza, onde os locais cujos valores são mais escuros representam as altitudes mais baixas, enquanto que os valores mais claros representam as porções mais altas do terreno, como pode ser visto na *Figura 27*.



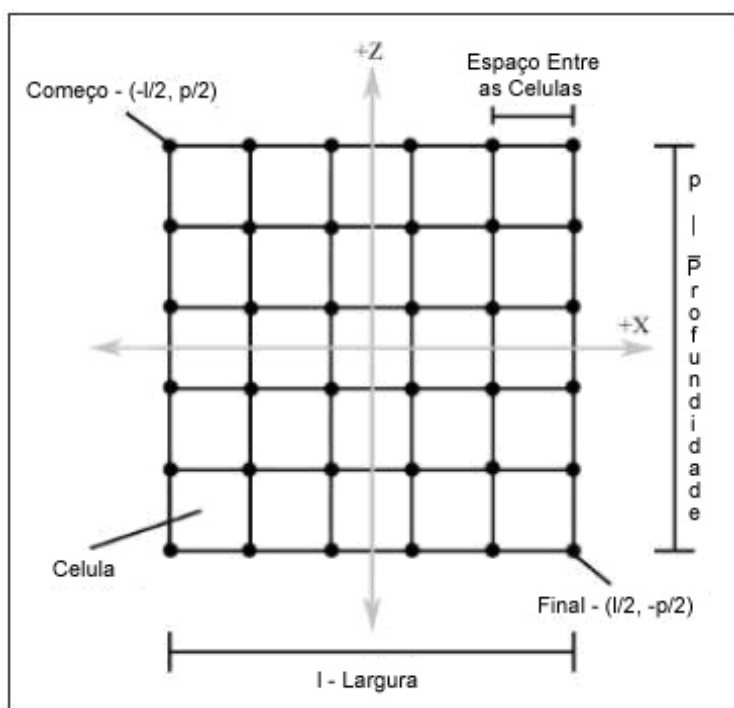
**Figura 27 – Mapa de Altitude**

No caso do demo que foi desenvolvido com o jogo, utilizou-se o formato RAW, devido a sua estrutura, onde os valores dos pixels são gravados um após o outro e sem cabeçalho.

Depois esses dados são lidos e gravados num vetor, para utilização posterior.

### **5.3.1.1 Gerando a geometria do terreno**

Para se gerar a geometria do terreno, após já ter sido feita a leitura do mapa de altitude, alguns conceitos devem ser adquiridos. A *Figura 28* nos ajuda na compreensão deles:



**Figura 28 – Conceitos para a Criação de Terrenos 3D**

A *largura* do terreno é o tamanho dele com relação ao eixo X do espaço, enquanto que a *profundidade* é o tamanho do mesmo com relação ao eixo Z. Já o *espaço entre as células* é a distância espacial entre cada ponto que forma o terreno. Como pode ser visto também, a posição inicial do espaço cartesiano se encontra no centro do terreno. Por isso, o limite superior esquerdo se encontra na posição (  $-largura/2$ ,  $profundidade/2$  ) e o limite inferior direito está em (  $largura/2$ ,  $-profundidade/2$  ). O *número de células por linha* é igual ao *número de vértices por linha* menos 1 (um). O mesmo vale para o *número de células por coluna*. Já o *número de vértices do terreno* é igual ao *número de vértices por linha* \* *número de vértices por coluna*.

Com base nesses conceitos, cria-se a geometria do terreno. O algoritmo abaixo exemplifica isso:

```
Estrutura VerticeTerreno
{   X, Y, Z;   };
```

```

Método GeraGeometriaTerreno()
{
    Criar vetor Terreno com tamanho igual ao número de vértices do terreno usando a
    estrutura VerticeTerreno;

    //Calcular os limites superior esquerdo e inferior direito
    comecoX = - largura / 2;
    comecoZ = profundidade / 2;

    fimX = largura / 2;
    fimZ = profundidade / 2;

    i = 0;
    Para ( z = comecoZ; z >= fimZ; z+= espacoEntreCelulas ) Faça

        j = 0;
        Para ( x = comecoX; x <= fimX; x+= espacoEntreCelulas ) Faça

            índice = i * numVerticesPorLinha + j;

            VerticeTerreno VerticeTemp;

            VerticeTemp.X = x;
            VerticeTemp.Y = mapaDeAltitude[ índice ];
            VerticeTemp.Z = z;

            terreno[ índice ] = VerticeTemp;

            j = j + 1;

        Fim Para

        i = i + 1;

    Fim Para
}

```

### 5.3.1.2 Acessando o vetor terreno

Depois, para acessar o mapa do terreno, basta utilizar o algoritmo abaixo:

```

Método AcessaTerreno( linha, coluna ): retorna Altitude
{
    Retorna terreno[ linha * numeroVerticesPorLinha + coluna ];
}

```

E de posse da estrutura *VerticeTerreno* retornada pelo método, pode-se verificar o altitude daquela posição no terreno, bem como renderizar o terreno 3D.

### 5.3.2 Billboard

Quando se cria cenários 3D, um jogo pode ganhar um pouco em performance, renderizando objetos 2D no lugar de objetos 3D. Esta é a idéia básica por trás da técnica chamada billboard.

Aplicações podem criar e renderizar este tipo de billboard definindo-se um retângulo sólido e aplicando uma textura sobre ele. A idéia base é criar texturas 2D que tenham a aparência de serem 3D e rotacioná-las de acordo com o campo de visão do jogador. Não importa se a imagem dos objetos não é retangular. Além disso, porções do billboard podem ser transparentes.

A técnica de billboard é bastante utilizada em jogos para renderizar imagens de árvores, arbustos e nuvens.

O billboard trabalha melhor com objetos simétricos, especialmente objetos que são simétricos ao longo do seu eixo vertical. Além do mais, é bom que a altura do jogador não seja muito grande, já que, se o jogador vir o billboard de cima, lhe parecerá que o objeto é 2D ao invés de 3D.

#### 5.3.2.1 Criação das texturas

Durante o desenvolvimento das texturas a serem utilizadas no billboard, deve-se ter em mente o conceito de canal alfa. Canal alfa é um valor entre 0 (zero) a 1 (um) inclusive, com precisão de número flutuante, associado ao valor da coloração de um pixel,

que define o nível de transparência daquele pixel. No caso do canal alfa ser 1 (um), a cor do pixel será totalmente opaca, enquanto que, no caso dele ser 0 (zero), a cor será totalmente transparente. Valores intermediários resultarão em cores não totalmente opacas, nem totalmente transparentes.

De posse deste conceito, cria-se uma textura, como pode ser visto na *Figura 29*.



**Figura 29 – Textura de uma Árvore**

Findada a textura, cria-se uma nova textura representando o canal alfa da textura anterior, como pode ser verificado na *Figura 30*, onde a cor preta representa o valor 0 (zero), a cor branca o valor 1 (um) e as cores intermediárias valores entre 0 (zero) e 1 (um).



**Figura 30 – Canal Alfa de uma Textura**

E, com apoio da ferramenta *DirectX Texture Tool*, que acompanha o DirectX SDK, mescla-se as duas figuras, formando um arquivo com uma extensão DDS, que conterà a textura e, aliado a cada pixel da textura, o seu respectivo canal alfa. Deste modo, quando a textura for utilizada, nos pontos onde o seu canal alfa diferir de 1 (um), ocorrerá a transparência, aumentando o realismo do cenário.

### **5.3.2.2 Algoritmo do billboard**

Inicialmente, habilita-se os estados de renderização a serem utilizados. Dentre esses estados, os necessários para o uso desta técnica são o blending e o teste alpha, de modo que, com esses estados habilitados, o canal alpha poderá ser utilizado para recriar a transparência, dando maior veracidade ao cenário.

Depois, o vetor que contém as posições das árvores deve ser ordenado, de modo a colocar as árvores mais próximas do campo de visão do jogador nas posições finais do vetor. Isso deve ser feito porque, quando as texturas das árvores forem desenhadas, para a

renderização da transparência das árvores, a API do Direct3D verifica o que está desenhado atrás daquela árvore e o redesenha, caso o teste alpha dê negativo.

Então, renderiza-se as árvores e desabilita-se os estados de renderização utilizados.

Portanto, para a implementação do billboard, deve-se seguir o algoritmo abaixo:

```
Método RenderizaArvores( direcaoVisaoJogador, posicaoJogador )
{
    SetaEstadosRenderizacao();

    Se o VetorDasArvores foi criado Faça

        OrdenaVetor( VetorDasArvores, posicaoDoJogador );
        Se a direcaoVisaoJogador.X > 0 Faça
            RotacioneY( -arcotangente(
                direcaoVisaoJogador.Z/ direcaoVisaoJogador.X) + PI/2 );
        Senão
            RotacioneY( -arcotangente(
                direcaoVisaoJogador.Z/ direcaoVisaoJogador.X) - PI/2 );
        Fim Se

        RenderizaAsArvores( VetorDasArvores );

    Fim Se

    DesabilitaEstadosRenderizacao();
}
```

Não se esquecendo que as funções destacadas fazem parte da API utilizada, devendo estas rotacionar o campo de visão do jogador com relação ao eixo Y.

## **5.4 Bibliografia Referente ao Capítulo 5**

### **(CARNIEL; TEIXEIRA, 2003)**

CARNIEL, Juliano; TEIXEIRA, Clóvis. Apostila de J2ME, versão 1.0. Capturado no dia 01/11/2004. On-line. Disponível em [http://portaljava.com/home/tutoriais/Tutorial\\_J2ME.pdf](http://portaljava.com/home/tutoriais/Tutorial_J2ME.pdf).

### **(LAMOTHE, 1999)**

LAMOTHE, André. Tricks of the Windows Game Programming Guru, Fundamentals of 2D and 3D Game Programming. 1ª edição. Indianópolis: SAM, outubro de 1999. 1005 páginas.

### **(LOURENÇO, 2004)**

LOURENÇO, Dalísio César Abreu. Criação de Áudio para Jogos com DirectX Áudio. Itajubá: Departamento de Matemática e Computação, 2004. 42 páginas.

### **(LUNA, 2003)**

LUNA, Frank D. Introduction to 3D Game Programming With DirectX 9.0. 1ª edição. Plano: Wordware Publishing Inc, junho de 2003. 388 páginas.

### **(MICROSOFT, 2002)**

MICROSOFT. DirectX Documentation for C++. Arquivo de Ajuda do SDK do DirectX.

### **(VALE, José Luciano Viana do, 2004)**

VALE, José Luciano Viana do. BREW, o caminho para o desenvolvedor. Campinas, Unicamp, 11 set. 2004. Palestra Ministrada aos Congressistas da GameTech 2004, pela QUALCOMM Serviços de Telecomunicações.

## CAPÍTULO 6 – CONCLUSÕES

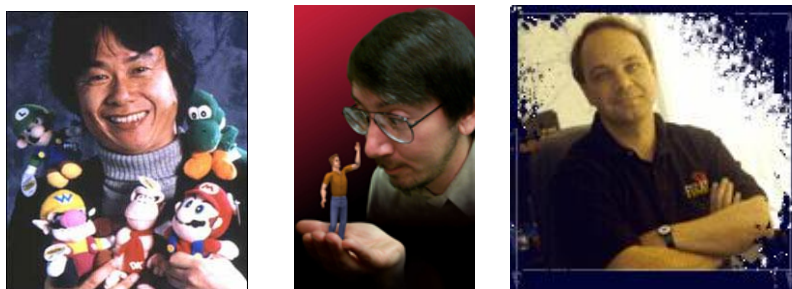
### 6.1 Game Design

O processo de game design é uma das áreas que mais são responsáveis pelo sucesso ou não de um determinado jogo. Jogos cujo design não tiveram a devida atenção, geralmente, estão destinados ao fracasso.

Os games designers que mais se dedicarem ao estudo das diversas técnicas de criação de personagens e de histórias terão maiores chances de criar jogos de sucesso.

Boas interfaces e bom balanceamento dos jogos também são capazes de diferenciar um jogo revolucionário de um que fracassará. Mesmo que o jogo seja o mesmo, mas se a interface e o balanceamento forem diferentes, aquele que tiver mais pontos nestes dois quesitos está destinado a ter um maior sucesso.

Nada substitui a experiência de um bom game designer. Não é à toa que, o simples fato de nomes de profissionais como Shigeru Miyamoto, Will Wright, Brian Reynolds, Sid Meier e Dani Berry figurarem no “Staff” de desenvolvimento de um jogo, é suficiente para criar uma expectativa muito grande com relação ao lançamento do mesmo. Na *Figura 31* podemos ver alguns deles.



**Figura 31 – Shigeru Miyamoto, Will Wright e Sid Meier**

Uma boa sugestão a aqueles que querem ser bons game designer é jogar bastante. Mas não só jogar para se divertir, mas sim jogar prestando atenção na funcionalidade da interface, na eficácia da IA etc.

## **6.2 Inteligência Artificial**

A IA é uma das áreas mais complicadas no desenvolvimento de jogos. Primeiramente porque existem muitos poucos livros na área e, depois, porque a teoria encontrada na internet nem sempre é muito boa. Por isso, os códigos-fontes encontrados não são facilmente entendidos.

Difícilmente se encontra engines de IA, já que a IA de cada jogo deve reagir das mais diversas maneiras aos acontecimentos. Portanto, na área de IA, deve-se implementar tudo.

Também é uma área muito perigosa para o programador de IA, já que, por mais que ele deseje criar a IA mais perfeita do mundo, ele deve levar em conta que o jogador também tem o sentimento de poder vencer essa IA. Por isso, se o programador criar um IA muito difícil, ele estará desestimulando os jogadores a continuarem a jogar o jogo, perdendo-se assim bastantes jogadores, que deixam de comprar o jogo por ele ser muito difícil. E, ao mesmo tempo em que o jogo não deve ser muito difícil, ele também não pode ser muito fácil, pelo mesmo motivo, que é o desestímulo o jogador. Por isso é muito complicado fazer esse balanceamento na IA.

### 6.3 Programação

O uso da Orientação ao Objeto é primordial no desenvolvimento de jogos. A orientação ao objeto garante muito a legibilidade do código, assim como a sua reutilização, o que é muito útil em um ambiente onde vários programadores participarão do processo de criação do jogo.

Na hora de fazer a modelagem do jogo, é desaconselhável criar uma modelagem completa do jogo antes do início de sua implementação. Como o jogo é um produto que não tem uma saída padrão, como é o caso dos relatórios dos sistemas de banco de dados, essa modelagem mudará bastante no decorrer da implementação, atrasando bastante o desenvolvimento do produto, devido as constantes alterações nos documentos da modelagem. Neste caso, o aconselhável é que a modelagem seja feita juntamente com o desenvolvimento do jogo, sempre um passo a frente do desenvolvimento.

Os programadores devem sempre modelar ou ter acesso à modelagem daquilo que será criado, antes da sua implementação. Desse modo, a implementação sai mais rápido e o programador não vai ficar com a sensação de que o código-fonte está uma bagunça. Ele vai ter toda a estrutura daquilo que será implementado em sua cabeça, agilizando assim o desenvolvimento.

Quanto maior a fragmentação do código-fonte para com os dados do jogo melhor, já que isso poupa bastante tempo de compilação. Se possíveis, textos explicativos, dados referentes a IA, mapas de cenários etc. devem ficar em arquivos fora do código-fonte, para facilitar a portabilidade do jogo para outras línguas, a adição de novos cenários, melhorias na IA do jogo etc.

## 6.4 Propostas Futuras

Para trabalhos futuros, visando dar continuidade a este projeto final de graduação, sugere-se as melhorias explicadas nos tópicos a seguir, onde cada tópico conta também com uma pequena explicação sobre o que significa cada melhoria.

### 6.4.1 Flocking

Para se aumentar a imersão do jogador, deve-se melhorar a IA do jogo, fazendo com que não seja passada a impressão ao jogador que tudo aquilo que ele está vendo é artificial. Para isso, um algoritmo que pode ser implementado na melhoria comportamental dos pássaros seria o Flocking.

Flocking é um algoritmo que faz a gerência do comportamento de grupos. Ele é muito utilizado para controlar o comportamento de peixes, de aves e até de bots em jogos do tipo FPS. Muitos jogos comerciais o utilizam, dentre os quais podemos citar Unreal (Epic) e Half-Life (Sierra).

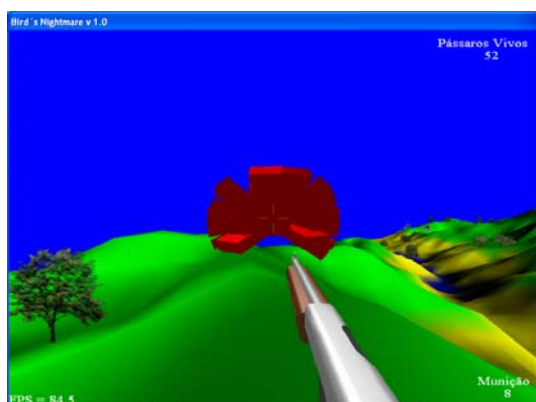
O algoritmo de Flocking está baseado em 4 (quatro) regras que devem ser seguidas:

- **Separação:** deve haver a tentativa de evitar colisão com os outros membros do grupo;
- **Alinhamento:** deve-se alinhar com relação à direção média dos seus companheiros de grupo;
- **Coesão:** o movimento deve ocorrer de acordo com o posicionamento dos demais membros do grupo;

- **Desvios:** deve-se andar evitando a colisão com obstáculos locais e inimigos.

O algoritmo de flocking é um algoritmo que não necessita do estado anterior para que seja feita a próxima decisão. Por isso ele é muito utilizado. Ele não necessita de muita memória.

E o algoritmo de flocking também estará evitando que aconteçam cenas como as da *Figura 32*, onde vários pássaros se encontram no mesmo local físico.



**Figura 32 – Bird's Nightmare**

## 6.4.2 Implementação da rede

Atualmente a opção de jogar no modo multiplayer é uma das áreas que tem dado um maior vigor aos jogos, já que esta opção permite que haja uma competição entre amigos e também se jogue contra pessoas desconhecidas, sempre alimentando o sentimento de competição entre os jogadores, que dá a graça ao jogo após o jogador já ter terminado o modo single player.

Por isso, sugere-se a implementação de um modo multiplayer, onde 2 jogadores podem jogar via rede, utilizando-se 2 computadores.

Para se programar o suporte à rede, conhecimentos em DirectPlay e programação multithreading são mais do que necessários, além de conhecimentos sobre o funcionamento de redes.

### **6.4.3 Algoritmo de quadtree**

Fica aqui a sugestão de se implementar um algoritmo de QuadTree ou outro existente para o gerenciamento do cenário, a fim de prover ao jogador um mundo maior de interação.

A idéia do algoritmo de QuadTree é a de dividir o cenário em 4 (quatro) retângulos de dimensões o mais semelhantes possíveis, de modo a verificar se aqueles retângulos se encontram no campo de visão do jogador. Se eles não se encontrarem, eles não são renderizados. Já no caso de algum dos retângulos se encontrar no campo de visão do jogador, ele é novamente dividido em outros 4 (quatro) retângulos e o teste é feito novamente sobre esses retângulos. E esse teste acontecerá recursivamente até que o tamanho do retângulo chegue a um tamanho mínimo definido pelo programador. Ao final desses passos, se o retângulo se encontrar no campo de visão do jogador, então ele é renderizado.

## **6.5 Bibliografia Referente ao Capítulo 6**

**(DeLoura, Mark. 2001)**

DeLoura, Mark. GAME PROGRAMMING GEMS 1. 1ª edição. Agosto de 2000. CHARLES RIVER MEDIA. 614 páginas.

## **ANEXOS**

## ANEXO A: DOCUMENTAÇÃO DA IDÉIA DO DEMO

Para a realização do Game Design do demo que acompanha esta monografia, o autor criou um grupo de estudos formado por integrantes do curso de Ciência da Computação da Universidade Federal de Itajubá, juntamente com pessoas de outras localizações que estavam interessadas em aprender mais sobre o desenvolvimento de jogos 3D.

Após a criação do grupo de estudos, começou-se a discutir como deveria ser o enredo do jogo. Após muita discussão, chegou-se a seguinte idéia:

- O jogador interagiria com um mundo campestre, com bosques, rios, montanhas etc.;
- O objetivo do jogo seria matar pássaros mutantes;
- Seria exibido apenas a arma carregada pelo jogador;
- O interfaceamento do jogador para com o jogo seria feito pelo teclado e o mouse, onde o mouse seria responsável pela alteração do campo visual do jogador, além de servir para dar o comando de atirar, e o teclado serviria para os demais comandos, como andar pelo cenário e recarregar a arma;
- Os pássaros mutantes mortos poderiam ser trocados em postos de troca por dinheiro, que poderia ser utilizado na compra de itens e novas armas.

Também existiria um modo multiplayer, onde até quatro jogadores poderiam jogar no mesmo cenário, baseado nas seguintes regras:

- Se um jogador atirar no outro jogador, nada aconteceria;
- O jogador poderia encontrar pedras no caminho, sendo que essas pedras poderiam ser jogadas contra outros jogadores, de modo a atordoá-los e ganhar uma pequena vantagem na caça de algum pássaro mutante;
- No caso de levar uma pedrada, o jogador que foi atingido ouviria alguém rindo da cara dele e, ao olhar para a pessoa que atirou a pedra nele, avistaria balões sobre a pessoa com desenhos provocantes.

No entanto, apesar do estilo do jogo parecer bem atrativo aos membros do grupo, ainda faltava algo: por que o caçador iria matar tantos pássaros mutantes assim? Qual seria a sua motivação para tal chacina? Depois de muito pensar e de um brainstorm entre o autor e seu pai, surgiu a seguinte estória, que daria a motivação necessária ao caçador:

"Num futuro próximo, numa época em que todos os povos vivem em paz, os homens clamam cada vez por mais conhecimento. Eles querem saber mais sobre a vida e o seu funcionamento. E para tal, realizam vários experimentos em animais. Até que, em um desses experimentos, foi injetada em um grupo de pássaros uma substância de testes. Essa substância sofreu uma mutação dentro dos pássaros, se transformando num vírus letal para os seres humanos, e que, ao mesmo tempo, deixava os pássaros bem agressivos e poderia ser passado para os seres humanos caso estes fossem atacados".

"Infelizmente, esses pássaros são soltos na natureza antes que a existência desse vírus seja descoberta. Eles se reproduzem e se firmam em certas regiões do mundo. Pouco tempo depois, esse vírus começa a se manifestar e faz com que os pássaros fiquem agressivos e ataquem os seres humanos. Num primeiro momento nada acontece a esses

humanos e, por isso, nada é descoberto. Até o momento em que começam a morrer várias pessoas, pessoas estas que haviam sido atacadas pelos pássaros. Com base na coincidência das mortes, é feita a caça e o estudo de alguns desses espécimes e descobre-se o vírus, mais tarde chamado de ITABRANOVBUENEV pelos cientistas. Ele tem um efeito letal sobre os seres humanos, mas se manifesta de forma curiosa. Inicialmente, a pessoa se sente mais forte e seus sentidos melhoram, para depois ir perdendo os sentidos gradativamente, até que a pessoa morre. Também se descobriu que esse vírus não mata os pássaros e que ele pode ser transmitido entre os seres humanos pela respiração. Com o passar do tempo as aves transformam-se em pássaros mutantes e atacavam os humanos".

"Os países do G-9, entre eles o Brasil, se reúnem e não querem que essa doença se alastre pelo mundo. Por isso, decidem contratar caçadores pelo mundo todo e lhes oferecem uma recompensa pela eliminação dos pássaros. E é aí, que começa a jornada. O caçador deve eliminar todos os focos desses pássaros mutantes pelo mundo. Ao todo são 5 lugares, onde o caçador deverá matar os pássaros mutantes e salvar a humanidade."

Após a apresentação dessa estória, o grupo acreditou que havia sido criada a motivação necessária aos jogadores do modo single player. No caso do modo multiplayer, haveria a seguinte adição a estória:

"O G-9 pagará uma grande recompensa (podendo ser um item) para o caçador que matar mais pássaros, obrigando os caçadores a brigarem entre si também para conseguirem ser o "HUNTER ONE", título que será dado pelo G-9 ao caçador mais eficiente".

E, após o término da elaboração da estória, o grupo também achou boa a inclusão das seguintes novidades na jogabilidade do jogo:

- E se o caçador não matar o pássaro mutante, depois que pássaro mutante saísse da fase, ele recuperaria 100 (cem) pontos de energia. Caso o caçador tivesse matado pássaros mutantes em uma determinada região e ferido outros, que tivessem se recuperado, outros poderiam aparecer;
- O nome das 5 regiões(fases) seriam: MALACACHETA, EAGLE'S NEST, SOLITUDE'S NIGHTMARE, BURADIRODIM E MAKIAEVE;
- O nome do jogo seria “Bird’s Nightmare”.

Depois de finalizada essa pequena fase de Game Design, decidiu-se pela implementação de um demo jogável, onde coube ao autor a codificação em C/C++ e DirectX da lógica, do gráfico e da IA do demo, e a outros membros do grupo a modelagem de objetos 3D e a programação e edição do áudio.

Para o desenvolvimento do demo, foi utilizada a prototipagem.

## ANEXO B: IMAGENS EM TONS DE CINZA E O FORMATO RAW

Imagens em tons de cinza são imagens cuja coloração varia entre os vários tons de cinza. Nos arquivos da imagem, as cores são representadas por valores que variam entre 0 e 255, sendo que o valor 0 representa o preto, o valor 255 representa o branco e os valores intermediários representam valores em tons de cinza proporcionais, como pode ser visto na *Figura 33*.



**Figura 33** – Variação dos Tons de Cinza

Os arquivos RAW guardam imagens em tons de cinza, sem a inclusão de um cabeçalho. Isso facilita bastante a sua leitura. E nesse formato, os valores da coloração dos pixels são gravados de maneira binária.

Como foi explicado na *Seção 5.3.1*, na construção do terreno do jogo, o mapa de altitude foi gravado em um arquivo *.raw*. Esse arquivo se chama *altura.raw*. Para fazer a alteração desse arquivo, basta utilizar qualquer editor de imagem que ofereça suporte ao formato *.raw*.

Juntamente com o arquivo *altura.raw*, compõe também o terreno os arquivos *acesso.raw* e *objetos.raw*. O arquivo *acesso.raw* possui os locais do terreno onde o jogador, no modo terrestre, terá acesso e o arquivo *objetos.raw* possui o objeto que deverá existir em determinada posição no terreno.

No caso dos objetos, os valores aceitos são: 1, 2, 3 e 4 para árvores utilizando Billboard, 11 para pedra, 12 para uma lata de coca-cola e 21 para um modelo 3D de uma árvore.

Para facilitar na criação dos arquivos de acesso e de objetos, foram criados 3 utilitários, que se encontram na pasta ***Programas***. A seguir, encontra-se uma breve explicação sobre esses utilitários:

- **ConverteObjetoRawTxt:** converte um arquivo chamado *objeto.raw*, que deverá estar no diretório local, para um arquivo texto chamado *obj.txt*, que conterà todos os valores das cores do arquivo *objeto.raw*, de modo a facilitar na edição desse arquivo de objetos;
- **CriaObjetoRaw:** converte o arquivo *obj.txt* no arquivo *objeto.raw*, para este último ser utilizado no terreno;
- **CriaAcessoRaw:** a partir dos arquivos *altura.raw* e *objeto.raw*, cria de maneira automática o arquivo *acesso.raw*, não permitindo que o jogador sai do cenário, nem que ele possa andar por locais impróprios, como sobre a água, além de não permitir que se atravesse o modelo 3D da árvore.

## **ANEXO C: HARDWARE E SOFTWARES NECESSÁRIOS**

Antes de definir qual o hardware necessário para o desenvolvimento de um jogo, é necessário fazer um estudo sobre qual será o público alvo do jogo. Se o público alvo for o chamado CASUAL, que jogará de vez em quando e que possui um computador de baixo desempenho, com uma placa 3D da primeira geração, o desenvolvedor deverá dispor de uma máquina de baixo desempenho para testes, além de um computador de médio desempenho para o desenvolvimento do jogo.

Já no caso do público alvo for o chamado HARDCORE, aquele que adora ficar jogando e que possui uma máquina de último tipo ou próxima disso, com uma placa 3D da terceira geração e uma placa de som com 5 ou 6 canais de saída, o desenvolvedor deverá ter computadores de alto e médio desempenho para o desenvolvimento, além de computadores de baixo desempenho, com placa 3D da 1ª e 2ª geração, para se processar os testes tanto nestes computadores de baixo desempenho, como nos de médio e alto desempenho. Isto se faz necessário para conseguir atingir a maior quantidade de compradores para o jogo, já que permite que sejam efetuados testes para a disponibilização de recursos extras para aqueles que tiverem melhores computadores e, ao mesmo tempo, garante que os jogadores que tiverem um hardware menos poderoso possam jogar também, mas com menos recursos.

E no caso do hardware dos modeladores e criadores de texturas, recomenda-se um computador de médio/alto desempenho em todos os casos, já que o processo de se criar modelos 3D requer grande poder de processamento.

Segue abaixo as configurações recomendadas para cada público alvo:

**Público Alvo:** CASUAL;

**Micro Hipotético do jogador:** Pentium 200MHz, 32 MB de RAM, HD 2GB, Placa de Vídeo 3D da primeira geração off-board (GeForce, Voodoo etc.), placa de som com saída para 2 canais, placa de rede 10/100 Mbits/s, leitor de CD-ROM, joystick com 4 botões, teclado e mouse;

**Micro de Desenvolvimento:** Pentium III 800 MHz ou Athlon 800Mhz, 128 MB de RAM, HD 20GB, Placa de Vídeo 3D da segunda geração off-board (GeForce 3, ATI Radeon 8500 etc.), placa de som com saída para 2 canais, placa de rede 10/100 Mbits/s, gravador de CD-ROM, joystick com 4 botões, teclado e mouse;

**Micros para Testes:** tanto o micro hipotético do jogador, quanto o micro de desenvolvimento;

**Micro para Modelagem 3D:** Pentium 4 2.600 MHz ou Athlon XP 2.600MHz, 512 MB de RAM (recomendado 768 MB), HD 80GB, Placa de Vídeo 3D da terceira geração off-board (GeForce 4, ATI Radeon 9500 etc.), gravador de CD-ROM, teclado e mouse.

---

**Público Alvo:** HARDCORE;

**Micro Hipotético do jogador:** Pentium 4 1.800 MHz ou Athlon XP 1.800MHz, 256 MB de RAM, HD 20GB, Placa de Vídeo 3D da terceira geração off-board (GeForce 4, ATI Radeon

9500 etc.), placa de som com saída para 6 canais, placa de rede 10/100 Mbits/s, gravador de CD-ROM, joystick com 8 botões, teclado e mouse;

**Micro de Desenvolvimento:** Pentium 4 2.800 MHz ou Athlon XP 2.800MHz, 512 MB de RAM, HD 80GB, Placa de Vídeo 3D da terceira geração off-board (GeForce 4, ATI Radeon 9500 etc.), placa de som com saída para 6 canais, placa de rede 10/100 Mbits/s, gravador de CD-ROM, joystick com 8 botões, teclado e mouse;

**Micros para Testes:** tanto o micro hipotético do jogador, quanto o micro de desenvolvimento, além de micros inferiores, com clock de processamento em torno de 600 MHz ou equivalente, 64 MB de RAM, HD 10GB, Placa de Vídeo da segunda geração off-board (GeForce 3, ATI Radeon 8500 etc.), placa de som com saída para 2 canais e demais periféricos;

**Micro para Modelagem 3D:** Pentium 4 2.600 MHz ou Athlon XP 2.600MHz, 512 MB de RAM (recomendado 768 MB), HD 80GB, Placa de Vídeo 3D da terceira geração off-board (GeForce 4, ATI Radeon 9500 etc.), gravador de CD-ROM, teclado e mouse.

Já na parte do software, temos que:

- Para a fase de programação, necessita-se de um compilador C/C++, juntamente com o SDK do DirectX para C/C++, que pode ser baixado gratuitamente pelo site da Microsoft;
- Para a fase de modelagem 3D e criação de texturas, faz-se necessário um editor de imagens e um modelador 3D.

Tomando-se como base os hardwares listados anteriormente, aconselha-se o uso dos seguintes softwares:

**Programação:** Visual C++ .NET ou DevCPP, com o SDK da versão 9.0c do DirectX;

**Modelagem e Texturização:** 3DStudio Max 5.0 ou superior e Photoshop 6.0 ou superior.

## ANEXO D: DOCUMENTAÇÃO DO CÓDIGO-FONTE DO DEMO

Neste anexo será feita uma breve documentação das classes do demo do jogo Bird's Nightmare, para que estudos futuros possam ser realizados de uma maneira mais simples.

### ➤ **ARMA**

**Arma( IDirect3DDevice9 \*device, Mesh \*meshe, D3DXMATRIX mundo, CSomPtr somTiro, CSomPtr somRecarga, CSomPtr somSemBala, int numTiros, int dano, int tirosConsecutivos );**

Construtor da classe, onde:

- *device* é o dispositivo de renderização do Direct3D;
- *meshe* é um objeto do tipo Mesh que contém o modelo 3D da arma;
- *mundo* é a matriz de translação da arma com relação ao campo de visão do jogador;
- *somTiro* é o ponteiro CSomPtr do som do tiro da arma;
- *somRecarga* é o ponteiro CSomPtr do som do tiro de recarga da arma;
- *somSemBala* é o ponteiro CSomPtr do som do tiro da arma, quando a mesma se encontra sem munição;
- *numTiros* é a quantidade máxima de munição que a arma pode carregar;
- *dano* é o dano de cada tiro daquela arma;
- *tirosConsecutivos* é a quantidade de projéteis disparados a cada tiro.

**Arma( IDirect3DDevice9 \*device, char \*diretorio, char \*nomeDoArquivo, D3DXMATRIX mundo, CSomPtr somTiro, CSomPtr somRecarga, CSomPtr somSemBala, int numTiros, int dano, int tirosConsecutivos );**

Construtor da classe, onde:

- *device* é o dispositivo de renderização do Direct3D;
- *diretorio* é o diretório onde o meshe da arma se encontra;
- *nomeDoArquivo* é o nome do arquivo com o meshe a ser importado;
- *mundo* é a matriz de translação da arma com relação ao campo de visão do

jogador;

- *somTiro* é o ponteiro CSomPtr do som do tiro da arma;
- *somRecarga* é o ponteiro CSomPtr do som do tiro de recarga da arma;
- *somSemBala* é o ponteiro CSomPtr do som do tiro da arma, quando a

mesma se encontra sem munição;

- *numTiros* é a quantidade máxima de munição que a arma pode carregar;
- *dano* é o dano de cada tiro daquela arma;
- *tirosConsecutivos* é a quantidade de projéteis disparados a cada tiro.

**~Arma();**

Destrutor da classe.

**void desenha( D3DXVECTOR3 pos, D3DXVECTOR3 dir );**

Desenha a arma, onde:

- *pos* é a posição do jogador;
- *dir* é a direção do campo de visão do jogador.

**int recarrega( int numTiros, bool som );**

Recarrega a arma, onde:

- *numTiros* é a quantidade de tiros disponíveis;

- *som* é uma variável booleana para decidir se o som da recarga deverá ser tocado ou não.

**bool atira( bool espera );**

Dá um tiro com a arma, onde:

- *espera* define se deve-se esperar pelo término da execução do som do tiro ou não para atirar novamente.

**bool atirou();**

Verifica se a arma foi disparada.

**int retornaDano();**

Retorna o dano daquela arma.

## ➤ **ARVORE**

**Arvore( IDirect3DDevice9 \*device, char \*nomeTextura );**

Contrutor da classe, onde:

- *device* é o dispositivo de renderização do Direct3D;

- *nomeTextura* é o nome da textura a ser utilizada como árvore.

**~Arvore();**  
Destrutor da classe.

**void desenha();**  
Método para imprimir a árvore.

## ➤ **CÂMERA**

**Camera();**  
Construtor da classe;

**~Camera();**  
Destrutor da classe;

**void retornaPos( D3DXVECTOR3 \*pos );**  
Retorna o vetor posição da câmera, onde:  
- *pos* é o vetor de 3 posições onde será retornado a posição.

**void retornaUp( D3DXVECTOR3 \*up );**  
Retorna o vetor up da câmera, onde:  
- *up* é o vetor de 3 posições onde será retornado o vetor up.

**void retornaLook( D3DXVECTOR3 \*look );**  
Retorna o vetor look da câmera, onde:  
- *look* é o vetor de 3 posições onde será retornado o vetor look.

**void retornaRight( D3DXVECTOR3 \*right );**

Retorna o vetor right da câmera, onde:

- *right* é o vetor de 3 posições onde será retornado o vetor right.

**void retornaView( D3DXMATRIX \*V );**

Retorna a matriz View da câmera, onde:

- *V* é a matriz 3x3 onde será retornada a matriz View.

**void setaPos( D3DXVECTOR3 pos );**

Seta a posição da câmera, onde:

- *pos* é o vetor de 3 posições onde será enviado o vetor posição.

**void andaLateral( float unidades );**

Faz a camera andar lateralmente, onde:

- *unidades* é a quantidade de unidades a serem andadas.

**void andaFrontal( float unidades );**

Faz a camera andar de maneira frontal, onde:

- *unidades* é a quantidade de unidades a serem andadas.

**void olhaLado(float angle);**

Faz a camera se mover lateralmente, onde:

- *angle* é o ângulo de rotação.

**bool olhaCima(float angle);**

Faz a camera se mover para cima e baixo, onde:

- *angle* é o ângulo de rotação.

**void voa( float unidades );**

Faz a camera “voar”, onde:

- *unidades* é a quantidade de unidades a serem “voadas”.

## ➤ **ESCRITA**

**Escrita( IDirect3DDevice9 \*device, int tamanho );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D;

- *tamanho* é o tamanho da letra.

**~Escrita();**

Destrutor.

**void escrevexy( int x, int y, char \*texto );**

Escreve um texto 2D na tela, onde:

- *x* é a coordenada X da tela;

- *y* é a coordenada Y da tela;

- *texto* é o texto a ser escrito na tela.

**void escreve( char \*texto );**

Escreve um texto 2D na tela, onde:

- *texto* é o texto a ser escrito na tela.

**void escrevexyz3D( float x, float y, float z );**

Escreve um texto 3D do mundo, onde:

- *x* é a coordenada X do mundo;

- *y* é a coordenada Y do mundo;

- *z* é a coordenada Z do mundo;

**void escreve3D();**

Escreve um texto 3D do mundo.

**void setaPosicao( float x, float y, float z );**

Seta a posição do texto a ser escrito, onde:

- *x* é a coordenada X do mundo ou da tela;

- *y* é a coordenada Y do mundo ou da tela;

- *z* é a coordenada Z do mundo;

**void setaCor( int r, int g, int b );**

Seta a cor do Texto, onde:

- *r* é o fator de vermelho;

- *g* é o fator de verde;

- *b* é o fator de azul;

**void inicializaTexto3D( char \*texto );**

Cria o meshe do texto 3D, onde:

- *texto* é o texto que se deseja criar o meshe.

## ➤ *FPS*

**FPS( IDirect3DDevice9 \*device, int largura, int altura );**

Construtor, onde:

- *device* é o dispositivo de renderização do Direct3D;

- *largura* é a largura da caixa na qual a taxa de FPS será mostrada;

- *altura* é a altura da caixa na qual a taxa de FPS será mostrada.

**~FPS();**

Destrutor.

**void imprime();**

Imprime a taxa de frames por segundo.

**void addTempo( float tempo );**

Adiciona o tempo decorrido para se fazer o cálculo da taxa de FPS, onde:

- *tempo* é o tempo a ser utilizada para o cálculo da taxa de FPS.

## ➤ **GERENCIAARVORETEXTURA**

**GerenciaArvoreTex( IDirect3DDevice9 \*device );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D.

**~GerenciaArvoreTex();**

Destrutor.

**bool insereArvore( char \*caminhoDoArquivo );**

Inserir uma árvore no gerenciador de árvores, onde:

- *caminhoDoArquivo* é o caminho onde se encontra a textura da árvore a ser inserida.

**void deletaArvore( int index );**

Deleta a textura da árvore, onde:

- *index* é a posição da textura da árvore que será deletada.

**void criaVetorPosicaoArvore( int x, int y, int largura, int profundidade, int espaco,**

**vector<int> altitudes, vector<int> objetos );**

Cria o vetor com as posições das árvores, onde:

- *x* é o número de vértices por coluna do cenário;

- *y* é o número de vértices por linha do cenário;

- *largura* é a largura do cenário;

- *profundidade* é a profundidade do cenário;

- *altitudes* é o mapa de altitude;

- *objetos* é o mapa que contém os objetos do cenário.

**void desenhaArvores( D3DXVECTOR3 dir, D3DXVECTOR3 pos );**

Desenha as árvores do gerenciador, onde:

- *dir* é a direção para a qual elas devem estar apontadas;

- *pos* é a posição do jogador.

**void preRenderizacao();**

Seta os estados de renderização.

**void posRenderizacao();**

Desabilita os estados de renderização utilizados no processo.

## ➤ **GERENCIALUZ**

**GerenciaLuz( IDirect3DDevice9 \*device );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D.

**~GerenciaLuz();**

Destrutor.

**int insereLuz( int tipo, D3DXCOLOR cor, D3DXVECTOR3 dir, D3DXVECTOR3 pos );**

Inserir uma luz no gerenciador, onde:

- *tipo* é o tipo de luz. Neste caso só aceita o valor LUZ\_SPOT.
- *cor* é a cor da luz;
- *dir* é a direção que ela deve apontar;
- *pos* é a posição que ele deve ser colocada.

**int insereLuz( int tipo, D3DXCOLOR cor, D3DXVECTOR3 vet );**

Inserir uma luz no gerenciador, onde:

- *tipo* é o tipo de luz. Neste caso, aceita os valores LUZ\_DIRECIONAL E LUZ\_PONTO;
- *cor* é a cor da luz;
- *vet* é o vetor que, no caso de ser luz direcional, é um vetor que representa a direção, enquanto que, se for luz pontual, indica a posição da luz.

**void deletaLuz( int index );**

Apaga a luz do gerenciador, onde:

- *index* é o índice da luz no vetor.

**void ligaLuz( int index );**

Liga a luz, onde:

- *index* é a posição da luz no vetor.

**void apagaLuz( int index );**

Apaga a luz do cenário, onde:

- *index* é o índice da luz no vetor.

**void interruptor( int index );**

Liga/Desliga a luz do cenário, onde:

- *index* é o índice da luz no vetor.

**void ligarLuzes();**

Liga todas as luzes do cenário.

**void desligarLuzes();**

Desliga todas as luzes do cenário.

## ➤ **GERENCIAMESHES**

**GerenciaMeshes( IDirect3DDevice9 \*device );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D.

**~GerenciaMeshes();**

Destrutor.

**int insereMesh( int tipo, char \*diretorio, char \*nomeDoArquivo );**

Insere um meshe no gerenciador, onde:

- *tipo* é o tipo do meshe, aceitando os valores MESH\_ARVORE, MESH\_ARMA e MESH\_OBJETOS;

- *diretório* é o diretório dentro do diretório Meshes que se encontra o meshe;

- *nomeDoArquivo* é o nome do arquivo a ser importado.

**void deletaMesh( int tipo, int index );**

Deleta um meshe do gerenciador, onde:

- *tipo* é o tipo do meshe, aceitando os valores MESH\_ARVORE, MESH\_ARMA e MESH\_OBJETOS;

- *index* é o índice do meshe no gerenciador.

**void desenha( int tipo, int index, D3DXMATRIX mundo );**

Desenha o meshe, onde:

- *tipo* é o tipo do meshe, aceitando os valores MESH\_ARVORE, MESH\_ARMA e MESH\_OBJETOS;

- *index* é o índice do meshe no gerenciador;

- *mundo* é a matriz que posiciona o objeto no cenário.

## ➤ **GERENCIAPASSAROS**

**GerenciaPassaro( IDirect3DDevice9 \*device, Terreno \*terreno, RECT limites, int numPassaros, int numPassarosLista );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D;

- *terreno* é o terreno do cenário;

- *limites* delimita os limites do cenário;
- *numPassaros* é o número de pássaros para serem gerenciados;
- *numPassarosLista* é o número de pássaros que devem atacar ao mesmo tempo, mas não implementado.

**~GerenciaPassaro();**

Destrutor.

**void setaNivel( int nivel );**

Seta o nível dos pássaros, onde:

- *nível* é o nível a ser setado.

**void atirou( d3d :: Raio \*raio, int pontos );**

Verifica se o tiro acertou os pássaros, onde:

- *raio* é o raio que representa o tiro dado;

- *pontos* é a quantidade de pontos que os pássaros perderão se forem acertados.

**void desenhaPassaros( D3DXVECTOR3 destino, float tempo );**

Desenha os pássaros, onde:

- *destino* é a posição destino dos pássaros;

- *tempo* é o tempo decorrido desde a última vez que os pássaros foram desenhados.

**void inserePassaro();**

Inserir um pássaro no gerenciador.

**float distancia( D3DXVECTOR3 pos, D3DXVECTOR3 posPassaro );**

Calcula a distância do pássaro ao jogador, onde:

- *pos* é a posição do jogador;

- *posPassaro* é a posição do pássaro.

**bool estaDentro( D3DXVECTOR3 pos );**

Verifica se o pássaro se encontra dentro do cenário, onde:

- *pos* é a posição do pássaro no cenário.

**void insereNovoPassaro();**

Inserir um novo pássaro na lista.

**void limpaPassaros();**

Apaga da lista todos os pássaros.

**void limpaPassarosMortos();**

Apaga da lista somente os pássaros mortos.

**int retornaNumPassaros();**

Retorna o número de pássaros vivos.

**void CriaPassaros( int numPassaros );**

Cria a lista com o número de pássaros requisitados, onde:

- *numPassaros* é o número de pássaros requisitados.

**void leIA( char \*localizacaoDoArquivo );**

Le a IA de um arquivo externo, onde:

- *localizacaoDoArquivo* é a localização do arquivo de IA, incluindo caminho e nome do arquivo.

## ➤ **HELICE**

**Helice( IDirect3DDevice9 \*device, char \*diretorio, char \*nomeDoArquivo, CSomPtr som );**

Construtor da classe, onde:

- *device* é o dispositivo de renderização do Direct3D;

- *diretorio* é o diretório onde se encontra o meshe da hélice;

- *nomeDoArquivo* é o nome do arquivo do meshe da hélice;

- *som* é o ponteiro CSomPtr do som da hélice.

**~Helice();**

Destrutor da classe.

**void desenha( D3DXVECTOR3 pos, D3DXVECTOR3 dir, float tempo );**

Desenha a Hélice na tela, onde:

- *pos* é a posição do jogador;

- *tempo* é o tempo decorrido desde a última renderização.

**void inicializa();**

Inicializa e toca o som da Hélice.

**void desinicializa();**

Para de tocar o som da Hélice.

## ➤ **LUZ**

**Luz();**

Construtor.

**~Luz();**

Destrutor.

**void criaDirecional ( D3DXVECTOR3 direcao, D3DXCOLOR cor );**

Cria uma luz direcional, onde:

- *direção* é a direção da luz;

- *cor* é a cor da luz.

**void criaPonto ( D3DXVECTOR3 posição, D3DXCOLOR cor );**

Cria uma luz pontual, onde:

- *posição* é a posição da luz;

- *cor* é a cor da luz.

**void criaSpot ( D3DXVECTOR3 posição, D3DXVECTOR3 direcao, D3DXCOLOR cor );**

Cria uma luz spot, onde:

- *posição* é a posição da luz;

- *direção* é a direção da luz;

- *cor* é a cor da luz.

**D3DLIGHT9 \*retornaLight();**

Retorna a estrutura que descreve a luz.

**void habilita();**

Habilita a luz.

**void desabilita();**

Desabilita a luz.

**bool estaAtiva();**

Verifica se a luz está ativa.

**int ehTipo();**

Retorna o tipo da luz.

## ➤ **MESH**

**Mesh( IDirect3DDevice9 \*device, char \*diretorio, char \*nomeDoArquivo );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D;

- *diretório* é o diretório onde o meshe se encontra;

- *nomeDoArquivo* é o nome do arquivo com o meshe a ser importado.

**~Mesh();**

Destrutor.

**void desenha( D3DXMATRIX mundo );**

Desenha o meshe, onde:

- *mundo* é a matriz que posiciona o meshe no cenário.

## ➤ **MIRA**

**Mira( IDirect3DDevice9 \*device );**

Construtor da classe, onde:

- *device* é o dispositivo de renderização do Direct3D.

**~Mira();**

Destrutor da classe.

**void desenha();**

Desenha a mira na tela.

## ➤ **PÁSSARO**

**Passaro( IDirect3DDevice9 \*device, D3DXVECTOR3 pos );**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D;

- *pos* é a posição do pássaro no cenário.

**~Passaro();**

Destrutor.

**void desenha( float tempo );**

Desenha o pássaro, onde:

- *tempo* é o tempo decorrido desde o último desenho.

**void persegue( D3DXVECTOR3 destino, float tempo, float altitude );**

Faz o pássaro ir à direção quista, onde:

- *destino* é a posição destino do pássaro;

- *tempo* é o tempo decorrido desde o último desenho;

- *altitude* é a altitude do cenário na posição que o pássaro se encontra.

**void foge( D3DXVECTOR3 destino, float tempo, float altitude );**

Faz o pássaro fugir da direção quista, onde:

- *destino* é a posição destino do pássaro;
- *tempo* é o tempo decorrido desde o último desenho;
- *altitude* é a altitude do cenário na posição que o pássaro se encontra.

**void ficaParado();**

Faz o pássaro ficar parado.

**void rotacionarPerseguir( float tempo );**

Rotaciona o pássaro para perseguir o destino, onde:

- *tempo* é o tempo decorrido desde o último desenho.

**void rotacionarFugir( float tempo );**

Rotaciona o pássaro para fugir do destino, onde:

- *tempo* é o tempo decorrido desde o último desenho.

**void setaVelocidade( float velocidade );**

Seta a velocidade do pássaro, onde:

- *velocidade* é a velocidade quista.

**float retornaPosX();**

Retorna a posição X do pássaro.

**float retornaPosY();**

Retorna a posição Y do pássaro.

**float retornaPosZ();**

Retorna a posição Z do pássaro.

**D3DXVECTOR3 retornaPos();**

Retorna o vetor posição do pássaro.

**bool acertou( d3d::Raio\* raio );**

Verifica se o pássaro foi acertado, onde:

- *raio* é a direção do tiro.

**bool testeColisao( d3d::Raio\* raio, d3d::EsferaLimite\* esfera );**

Verifica se houve colisão entre o raio e a esfera, onde:

- *raio* é a direção do tiro;

- *esfera* é a esfera que indica onde pode ocorrer colisão com o pássaro.

**void retiraPontos( int pontos );**

Retira pontos de vida do pássaro, onde:

- *pontos* é a quantidade de pontos.

**void adicionaPontos( int pontos );**

Adiciona pontos de vida ao pássaro, onde:

- *pontos* é a quantidade de pontos.

**int retornaVida();**

Retorna a quantidade de pontos de vida do pássaro.

**bool estaVivo();**

Retorna o estado de vida do pássaro.

**void troca( int &t1, int &t2 );**

Faz a troca de valores entre os dois valores inteiros.

## ➤ ***PERIFÉRICOS***

**Perifericos();**

Construtor.

**~Perifericos();**

Destrutor.

**bool inicializaMouse( HWND hwnd );**

Faz a inicialização do mouse, onde:

- *hwnd* é o handler da janela.

**void leMouse();**

Lê os dados de entrada do mouse.

**void retornaMouse( MOUSE \*entrada );**

Retorna o estado do mouse, onde:

- *entrada* é a estrutura que contém os dados de entrada do mouse.

**void limpaMouse();**

Limpa os dados lidos do mouse que estavam no objeto.

**bool botaoApertado( int index );**

Verifica se o botão do mouse foi pressionado, onde:

- *index* é o índice do botão do mouse, sendo 0 o botão esquerdo.

## ➤ **TERRENO**

**Terreno( IDirect3DDevice9\* device, GerenciaMeshes \*gMeshes, GerenciaArvoreTex  
\*gArvore, char \*diretorio, int numVertsPorLinha, int  
numVertsPorColuna, int espacoEntreCelulas, float escalaAltitude);**

Construtor, onde;

- *device* é o dispositivo de renderização do Direct3D;

- *gMeshes* é o gerenciador dos meshes que será utilizado no cenário;

- *gArvore* é o gerenciador das texturas das árvores;

- *diretorio* é o diretório dentro da pasta Terrenos, onde se encontra os

arquivos do terreno;

- *numVertsPorLinha* é o número de vértices por linha do terreno;

- *numVertsPorColuna* é o número de vértices por coluna do terreno;

- *espacoEntreCelulas* é o tamanho de cada célula do terreno;

- *escalaAltitude* é o fator pela qual deve ser multiplicado o mapa de altitude.

**~Terreno();**

Destrutor.

**int retornaEntradaMapaAltitude( int linha, int coluna );**

Retorna a entrada do mapa de Altitude, onde:

- *linha* é a linha da matriz imaginária do mapa de altitude;
- *coluna* é a coluna da matriz imaginária do mapa de altitude.

**void setaEntradaMapaAltitude( int linha, int coluna, int valor );**

Seta a entrada do mapa de Altitude, onde:

- *linha* é a linha da matriz imaginária do mapa de altitude;
- *coluna* é a coluna da matriz imaginária do mapa de altitude;
- *valor* é o valor a ser usado.

**bool estaInacessivel( float x, float z );**

Verifica se a posição está inacessível, onde:

- *x* é a posição sobre o eixo X do cenário.
- *z* é a posição sobre o eixo Z do cenário.

**float retornaAltitude( float x, float z );**

Retorna a altitude da posição, onde:

- *x* é a posição sobre o eixo X do cenário.
- *z* é a posição sobre o eixo Z do cenário.

**bool leTextura( char \*nomeDoArquivo );**

Lê uma textura para ser aplicada ao cenário, onde:

- *nomeDoArquivo* é o nome do arquivo da textura a ser importada.

**bool geraTextura( D3DXVECTOR3\* direcaoDaLuz );**

Gera uma textura automática, onde:

- *direcaoDaLuz* é a direção da luz aplicada sobre o terreno.

**bool desenha( D3DXMATRIX\* mundo, bool desenhaTriangulos, D3DXVECTOR3 dir, D3DXVECTOR3 pos );**

Desenha o terreno, onde:

- *mundo* é a matriz que posiciona o terreno no plano cartesiano.

- *desenhaTriangulos* é setado para true quando se desenha também

os triângulos do cenário;

- *dir* é a direção da visão do jogador para se desenha as árvores;

- *pos* é a posição do jogador para se desenha as árvores.

**void desenhaArvores( D3DXVECTOR3 dir, D3DXVECTOR3 pos );**

Desenha as árvores, onde:

- *dir* é a direção da visão do jogador;

- *pos* é a posição do jogador.

**void desenhaObjetos();**

Desenha os objetos do cenário.

**bool leArquivosRaw(int tipo, char \*nomeDoArquivo);**

Lê o arquivo RAW, onde:

- *tipo* é o tipo do arquivo, aceitando os valores TERRENO\_ALTITUDE, TERRENO\_OBJETOS e TERRENO\_ACESSO;

- *nomeDoArquivo* é o nome do arquivo a ser lido.

**bool computarVertices();**

Computa os vértices para criar o terreno.

**bool computarIndices();**

Computa os índices para criar o terreno.

**bool iluminarTerreno(D3DXVECTOR3\* direcaoDaLuz);**

Calcula a iluminação do terreno, onde:

- *direcaoDaLuz* é a direção da luz incidente no terreno.

**float computarSombra(int celulaDaLinha, int celulaDaColuna, D3DXVECTOR3\* direcaoDaLuz);**

Computa a sombra do terreno, onde:

- *celulaDaLinha* é o índice da célula da linha a ser computada a sombra;

- *celulaDaColuna* é o índice da célula da coluna a ser computada a sombra;

- *direcaoDaLuz* é o vetor da direção da luz incidente sobre o terreno.