



CENTRO UNIVERSITÁRIO DO TRIÂNGULO
PRÓ-REITORIA DE GRADUAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Aplicação de Técnicas de Inteligência Artificial na Implementação de Jogos Eletrônicos

Natal Vieira de Souza Neto

Uberlândia, Dezembro/2011.

Aplicação de Técnicas de Inteligência Artificial na Implementação de Jogos Eletrônicos

Natal Vieira de Souza Neto

Monografia apresentada ao Curso de
Ciência da Computação do Centro
Universitário do Triângulo - Unitri, como
requisito básico à obtenção do grau de
Bacharel em Ciência da Computação, sob a
orientação do Prof. Dr. Marcos Alberto
Lopes da Silva.

Uberlândia, Dezembro/2011.

Aplicação de Técnicas de Inteligência Artificial na Implementação de Jogos Eletrônicos

Natal Vieira de Souza Neto

Monografia apresentada ao Curso de Ciência da Computação do Centro
Universitário do Triângulo - Unitri, como requisito básico à obtenção do grau de
Bacharel em Ciência da Computação.

Marcos Alberto Lopes da Silva, Dr.
(Orientador)

Ângela Abreu Rosa de Sá, Dra..
(Co-Orientadora)

Carlos Eduardo de Carvalho Dantas, Esp.
(Avaliador 1)

Jony Teixeira de Melo, Msc.
(Avaliador 2)

Clayder Cristiam Coêlho, Msc.
(Coordenador de Curso)

Uberlândia, Dezembro/2011.

*Agradecimentos à
minha família, amigos, orientadores e professores da Unitri, que
estiveram ao meu lado ao longo de quatro anos de graduação e um ano de
dedicação a este trabalho.*

RESUMO

Na indústria do entretenimento, o ramo dos jogos eletrônicos constitui um campo bilionário (TATAI, 2003). Para a criação de um jogo, vários componentes são utilizados (*engine*, som, rede, Inteligência Artificial, etc.), com grande destaque para a IA (Inteligência Artificial), responsável por implementar e controlar oponentes e aliados do jogador. Este destaque deve-se ao fato de que, cada dia que passa, os jogadores buscam não só jogos de qualidade, mas personagens de jogos que atuem de forma inteligente (BORGES & BARREIRA & SOUZA, 2009).

Os personagens, chamados NPC (Personagem Não Jogável) é que garantem a dificuldade do jogo, e, portanto, o nível de diversão que será garantido a um jogador. Para a criação dos NPC's, várias técnicas foram utilizadas ao longo dos anos, como Máquina de Estados Finitos, Padrões de Movimento, dentre outras. Técnicas clássicas de IA, como Redes Neurais, Algoritmos Genéticos e Lógica *Fuzzy* foram menos utilizadas, mas tendem, no futuro, a serem as mais estudadas pelos desenvolvedores de jogos.

Este trabalho tem como objetivo analisar como a IA foi utilizada em jogos eletrônicos ao longo da história, e como e em qual parte do jogo ela pode ser explorada, além das principais características das técnicas clássicas.

SUMÁRIO

RESUMO	v
LISTA DE FIGURAS	ix
LISTA DE ABREVIATURAS E SÍMBOLOS.....	xi
1. INTRODUÇÃO.....	12
2. HISTÓRICO DE JOGOS ELETRÔNICOS.....	15
2.1. O que é um jogo eletrônico	15
2.2. Os jogos antes de 1970	17
2.3. A década de 1970	18
2.4. A década de 1980	24
2.5. A década de 1990	31
2.6. Os jogos após o ano 2000	36
2.7. Conclusão	40
3. TÉCNICAS DE DESENVOLVIMENTO DE JOGOS.....	42
3.1. Game Design	42
3.2. Roteiro.....	45
3.3. Motor (<i>engine</i>).....	47
3.4. Rede.....	48
3.5. Áudio	50
3.6. Arquitetura	51
3.7. IA.....	52
3.8. Física.....	55
3.9. Criação e desenvolvimento de um oponente inteligente	56
3.9.1. Oponente inteligente	56
3.9.2. Inteligência Artificial e os NPC's.....	57

3.9.3. Agentes Autônomos	57
3.9.4. Operações de um NPC	58
3.10. Inteligência artificial nos jogos eletrônicos	59
3.11. Conclusão	64
4. TÉCNICAS DE IA PARA CRIAÇÃO DE JOGOS.....	66
4.1. Redes Neurais Artificiais	66
4.1.1. Cérebro e Neurônios.....	67
4.1.2. Modelo de Camadas em RNA	68
4.1.3. Algoritmo.....	69
4.1.3.1. Perceptron	70
4.1.3.2. Backpropagation	70
4.2. Algoritmos Genéticos.....	74
4.3. Lógica <i>Fuzzy</i>.....	81
4.4. Máquina de Estados Finitos	86
4.5. Conclusão	87
5. ESTUDO DE CASO - JOGO DE RPG COM ALGORITMO GENÉTICO E MÁQUINA DE ESTADOS FINITOS	89
5.1. Roteiro do Jogo	90
5.1.1. Ambiente	90
5.1.2. Personagens	91
5.1.3. Sequência dos acontecimentos no jogo	92
5.2. Linguagem de Programação e Engine	94
5.3. Implementação.....	95
5.3.1. A classe Ambiente	96
5.3.1.1. O método gameLoop	98

5.3.1.2. O método initGame.....	100
5.3.1.3. Algoritmo Genético – métodos geraFilhos e roleta.....	101
5.3.1.4. O método drawFrame	105
5.3.2. A classe Inimigo	106
5.3.2.1. O método move	107
5.3.2.2. O método desenhaInimigo	109
5.3.3. A classe Jogador	110
5.3.3.1. O método fogo.....	110
5.3.3.2. O método desenhaTiros	111
5.3.3.3. O método desenhaJogador	112
5.3.3.4. O método move	112
5.3.4. A classe Tiro	113
5.3.4.1. O método desenhaTiro.....	113
5.3.4.2. O método move	114
5.4. Utilização de IA x não utilização de IA	115
5.5. Conclusão	117
6. CONCLUSÃO.....	119
REFERÊNCIAS BIBLIOGRÁFICAS.....	122
BIBLIOGRAFIA	127

LISTA DE FIGURAS

Figura 2.1 - Jogo <i>Tennis For Two!</i> (VELASQUEZ, 2009)	16
Figura 2.2 - Tela de <i>SpaceWar!</i> (LUZ, 2004)	17
Figura 2.3 - <i>Pong</i> , da Atari (NEWSGAMER, 2011)	19
Figura 2.4 - Tela de <i>Super Breakout</i> (PEABODY, 1997)	22
Figura 2.5 - Jogo <i>Asteroids</i> (PEABODY, 1997)	23
Figura 2.6 - Jogo <i>Pac-Man</i> (LUZ, 2004)	25
Figura 2.7 - Jogo <i>Battlezone</i> (PEABODY, 1997)	25
Figura 2.8 - Jogo <i>Missile Command</i> (PEABODY, 1997)	26
Figura 2.9 - Jogo <i>Tempest</i> (PEABODY, 1997)	27
Figura 2.10 - Jogo <i>Centipede</i> (NEWSGAMER, 2011)	27
Figura 2.11 - Tela reconstituída do jogo Tetris (LUZ, 2004)	29
Figura 2.12 - Jogo SimCity (BALDANCE & REIS, 2006)	31
Figura 2.13 - Tela do Jogo Street Fighter (BALDANCE & REIS, 2006)	32
Figura 2.14 - Tela de Doom3 (LUZ, 2004)	33
Figura 2.15 - Jogo <i>The Sims</i> (PEABODY, 1997)	37
Figura 2.16 - Imagem de <i>FIFA 2005</i> (ENE, 2011)	39
Figura 2.17 - Jogo FIFA 2011 (SPORTS, 2011)	40
Figura 2.18 - Cena de explosão em GTA IV (GAMES, 2011)	40
Figura 3.1 - Modelo de IA em um jogo (MILLINGTON & FUNGE, 2009)	54
Figura 4.1 - Modelo básico de camadas em RNA	68
Figura 4.2 - Fluxo do Backpropagation (TONSIG, 2000)	71
Figura 4.3 - Fluxograma Backpropagation (DAZZI, 1999)	72
Figura 4.4 - Fluxograma de um AG (FILITTO, 2008)	77
Figura 4.5 - A roleta em um AG (SOARES, 1997)	78
Figura 4.7 - Lógica Contemporânea (da SILVA, 2008)	82
Figura 4.8 - Velocidade X Pertinência em lógica booleana (da SILVA, 2008)	83
Figura 4.9 - Velocidade X Pertinência em lógica Fuzzy (da SILVA, 2008)	84
Figura 4.10 - Diagrama Lógica Fuzzy (da SILVA, 2008)	85
Figura 4.11 - Exemplo estados em um jogo (GALDINO, 2007)	87
Figura 5.1 - Tela da Fase 1 do Estudo de Caso	92
Figura 5.2 - Inimigo destruindo o personagem do jogador	94
Figura 5.3 - Diagrama de classes	96
Figura 5.4 - Construtor da classe Ambiente	97
Figura 5.5 - Método gameLoop - Parte 1	98
Figura 5.6 - Método gameLoop - Parte 2	99
Figura 5.7 - Método initGame	101
Figura 5.8 - Método geraFilhos()	103
Figura 5.9 - Algoritmo da roleta	104
Figura 5.10 - Método drawFrame	105
Figura 5.11 - Construtor da classe Inimigo	106
Figura 5.12 - Método move - Parte 1	107
Figura 5.13 - Método move - Parte 2	108

Figura 5.14 - Método move - Parte 3.....	109
Figura 5.15 - Método desenhaInimigo.....	110
Figura 5.16 - Construtor da classe Jogador.....	110
Figura 5.17 - O método fogo.....	111
Figura 5.18 - O método desenhaTiros.....	111
Figura 5.19 - Método desenhaJogador.....	112
Figura 5.20 - O método move.....	112
Figura 5.21 - Construtor da classe Tiro.....	113
Figura 5.22 - Método desenhaTiro	114
Figura 5.23 - Método move.....	114

LISTA DE ABREVIATURAS E SÍMBOLOS

2D - Bidimensional

3D - Três Dimensões

AG - Algoritmo Genético

CD-ROM - *Compact Disc Read-only Memory*

FFSM - *Fuzzy Finite-State Machine*

FSM - *Finite-State Machine*

GTA - *Gran Theft Auto*

IA - Inteligência Artificial

MMO - *Massively Multiplayer Online*

MMORPG - *Massively Multiplayer Online Role Playing Game*

NES - *Nintendo Entertainment System*

NPC - Personagem Não Jogável

PC - *Personal Computer*

RNA - Rede Neural Artificial

RNN - Rede Neural Natural

ROM - *Read-only Memory*

RPG - *Role Playing Game*

SNES - *Super Nintendo Entertainment System*

1. INTRODUÇÃO

Jogos eletrônicos são comercializados em vários países, e novas versões são criadas todos os anos, sejam por empresas especializadas, sejam por simples fanáticos ou estudantes das áreas de computação e design. O ponto forte a ser ressaltado é que cada dia que passa essa forma de entretenimento busca melhorar. Inicialmente, buscava-se a criação de jogos com um poder gráfico cada vez melhor, pois os próprios desenvolvedores tinham isso em mente, criar jogos que pudessem se tornar cada vez mais reais. A questão é que na indústria do entretenimento, o ramo dos jogos eletrônicos constitui um campo bilionário (TATAI, 2003), portanto a opinião do público alvo, ou seja, dos jogadores, torna-se muito valiosa.

Atualmente, os jogadores não querem apenas jogos bonitos e às vezes não querem apenas jogos muito reais, e sim jogos que tragam muita interatividade e jogabilidade. A interatividade é garantida por componentes que estão em alta e vem sendo cada dia mais pesquisados, voltados, por exemplo, para as redes de computadores. Com a crescente utilização do modo *multiplayer*, onde várias pessoas podem jogar o mesmo jogo de lugares diferentes. Já a jogabilidade é garantida pela movimentação e desenvolvimento dos personagens que estão no jogo. Normalmente essa movimentação e desenvolvimento são criadas através de técnicas de IA (Inteligência Artificial).

Para a criação de um jogo, vários componentes são utilizados, como por exemplo, a *engine*, considerada o motor responsável por fazer a renderização gráfica do jogo. Outro exemplo é o áudio, pelo fato de que a trilha sonora e os próprios sons criados ao longo de um jogo agradam muito os jogadores. Porém os

componentes que estão cada dia recebendo mais atenção são os já citados IA e Redes de Computadores.

A IA, responsável por implementar e controlar oponentes e aliados do jogador, destaca-se um pouco mais, pois cada dia que passa, os jogadores buscam não só jogos de qualidade, mas personagens de jogos que atuem de forma inteligente (BORGES & BARREIRA & SOUZA, 2009). Estes personagens, chamados NPC (Personagem Não Jogável) é que garantem a dificuldade do jogo, e, portanto, o nível de diversão que será garantido a um jogador.

A inteligência artificial é uma importante disciplina da ciência da computação, que com o uso de técnicas e dispositivos computacionais, busca simular a inteligência humana. A inteligência possui diversas definições, mas ao se tratar de inteligência artificial, está principalmente relacionada com a definição de ação racional. Para os filósofos, a mente humana pode ser bastante parecida com uma máquina, pois opera sobre o conhecimento codificado em alguma linguagem interna. O próprio uso da linguagem se ajusta ao sistema de processamento de informações (RUSSEL & NORVIG, 2009).

É claro que os jogos eletrônicos não poderiam deixar isso de lado, e já a alguns anos utilizam fortemente de inteligência artificial. Isso se deve ao fato de que cada vez mais os usuários buscam uma jogabilidade melhor. Normalmente, quem dita a jogabilidade são os personagens do jogo, pois eles são os seres animados, que se movimentam, e de certa forma interagem com o jogador. A inteligência artificial em jogos foi provada com o Deep Blue, da IBM, que foi um computador que derrotou um campeão mundial em uma partida de xadrez (RUSSEL & NORVIG, 2009). Primeiramente, deve-se definir que a IA (Inteligência Artificial), quando citada no contexto de jogos, trata-se das técnicas de IA (redes neurais, algoritmos genéticos, lógica *fuzzy*, etc.) que foram utilizadas para a criação do jogo.

Para a criação dos NPC's, várias técnicas foram utilizadas ao longo dos anos, como Máquina de Estados Finitos, Padrões de Movimento, dentre outras. Técnicas clássicas de IA, como Redes Neurais, Algoritmos Genéticos e Lógica Fuzzy foram menos utilizadas, mas tendem, no futuro, a serem as mais estudadas

pelos desenvolvedores de jogos.

Este trabalho tem como objetivo analisar como a IA foi utilizada em jogos eletrônicos ao longo da história, e como e em qual parte do jogo ela pode ser explorada, além das principais características das técnicas clássicas. O capítulo 2 trata basicamente da história dos jogos. O capítulo 3 apresenta os principais componentes utilizados em um jogo e o capítulo 4 analisa as técnicas de IA propriamente ditas. No capítulo 5, há a documentação de um estudo de caso: um jogo desenvolvido utilizando algumas das técnicas citadas no capítulo 4.

2. HISTÓRICO DE JOGOS ELETRÔNICOS

Os jogos eletrônicos ou jogos de computador evoluíram ao longo da história. Essa evolução se deu principalmente na computação gráfica, mas outras áreas começaram a se desenvolver nos últimos anos, como IA ou redes. Este capítulo apresenta o conceito de jogos, além de uma análise dos principais jogos da história, ou seja, aqueles que marcaram e aqueles que influenciaram diversos outros.

2.1. O que é um jogo eletrônico

Segundo Huizinha (2000), o jogo surgiu antes da cultura, já que para existir cultura é necessária a existência de sociedade. O jogo é algo que já está presente nas relações entre animais, antes do surgimento da sociedade, como por exemplo, cachorros brincando. É algo do instinto animal, no que se define como passatempo.

Um jogo possui cinco características fundamentais. Primeiro, deve ser livre, ou seja, não se pode obrigar ninguém a participar. Deve ser evasivo, isto é, quando alguém joga, deve estar convicto de que aquilo está além da realidade (apesar de alguns jogos simularem a realidade). Deve ter início, meio e fim. Um jogo também cria uma ordem e regras próprias e se desenvolve a partir delas. Por fim, deve haver tensão, ou seja, o jogo fica mais emocionante com o passar do tempo (SILVA, 2008).

Os jogos eletrônicos surgem quando surge o *videogame*, que é a união do jogo com o vídeo (uma mídia muito popularizada há bastante tempo). Vale aqui

apresentar a diferença entre o jogo de computador e o *videogame*. O primeiro necessita de um microprocessador, por isso é chamado de jogo eletrônico. Já o segundo refere-se à parte visual (WOLF, 2008). Muitas vezes os dois são confundidos, mas sempre colocados juntos, pois não há muita lógica em um jogo se não houver processamento de informações e, claro, se não houver a parte visual.

A definição de jogo eletrônico, então, pode ser dada como um programa de computador (coleção de instruções que descrevem uma tarefa a ser realizada por um computador), que seja interessante e interativo, além de envolver o usuário de maneira atrativa.

Não existe diferença entre o jogo e o jogo eletrônico. Para melhor entendimento, durante o resto deste capítulo, a palavra jogo deve ser interpretada como jogo eletrônico, ou seja, a ação de efetuar o jogo através de um computador.

Este capítulo faz uma revisão bibliográfica sobre a história dos jogos eletrônicos, desde 1958 até os dias atuais. Uma abordagem completa pode ser encontrada em Wolf (2008). Neste trabalho são relatados os jogos que mais venderam, e também os jogos mais importantes, isto é, jogos que por algum motivo influenciaram diversos outros jogos, e aqueles jogos que marcaram época, que são lembrados por toda uma geração.

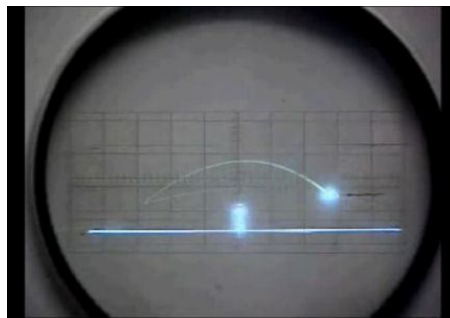


Figura 2.1 - Jogo *Tennis For Two!* (VELASQUEZ, 2009)

Existe muita divergência entre escritores e historiadores quando se pergunta qual foi o primeiro jogo eletrônico da história. Wolf (2008) relata *Tennis*

for Two (Figura 2.1) como o primeiro experimento considerado um jogo eletrônico. Esse jogo foi criado por William Higinbotham no *Brookhaven Laboratory*, em 1958, e demonstra uma bolinha que cai gravitacionalmente de um lado para outro. Muitos autores discordam que *Tennis for Two* possa ser considerado um jogo eletrônico, pois ele apenas demonstrava o poder gráfico da época.

2.2. Os jogos antes de 1970

Segundo Schwab (2004), o primeiro jogo eletrônico é o *SpaceWar!* (Figura 2.2), lançado em 1962. Este trabalho também irá considerá-lo como o primeiro jogo, pois ele foi criado para ser jogado, e não apenas demonstrar poder gráfico. O *SpaceWar!* trata de um programa de computador que na época deveria demonstrar todo o potencial do computador, ser interessante e interativo, além de envolver o usuário de maneira atrativa. Essas qualidades o caracterizam como um jogo eletrônico. Desejava-se com ele transpor a ficção científica da literatura para outra mídia (o computador). Esse jogo inspirou o *Computer Space*, de 1971 (KISHIMOTO, 2004).

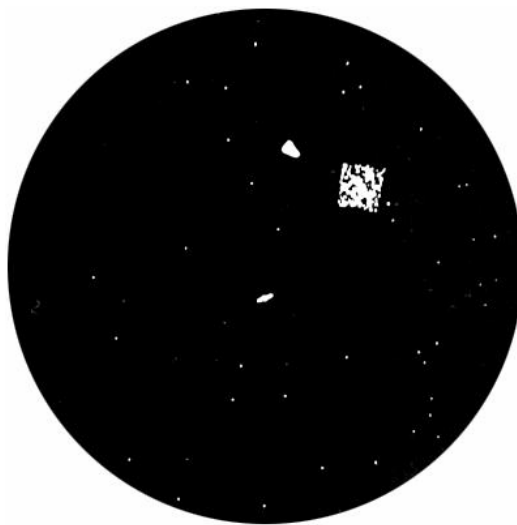


Figura 2.2 - Tela de *SpaceWar!* (LUZ, 2004)

2.3. A década de 1970

Após a década de 1970 a história dos jogos eletrônicos teve vários avanços.

Em 1971, foi lançado o primeiro fliperama, baseado no *SpaceWar!*, batizado de *Computer Space*. Para jogar, o jogador deveria inserir moedas na máquina. Começa então a movimentação de dinheiro no mundo interativo dos jogos eletrônicos. O *Computer Space* possuía um *score* que somava os pontos do jogador, e era bem parecido com o *SpaceWar!*, o jogador controlava uma nave espacial e tinha que destruir as naves espaciais inimigas.

No ano seguinte, 1972, a empresa de Ralph Baer, *Magnavox*, lança o *Odyssey*, considerado o primeiro *vide game* vendido para ser usado em casa (WOLF, 2008).

Na década de 1970 outras empresas surgiram no cenário, como *Taito*, *Midway* e *Capcom* (KISHIMOTO, 2004). Mas uma destaca-se bastante: criada por Nolan Bushnell (que foi o inventor do *Computer Space*), a empresa levou o nome de *Atari*, e ficou famosa pela criação do jogo *Pong* (Figura 2.3). Aqui relata-se uma importância significativa na história. Não que *Pong* fosse mais extraordinário que os outros jogos da época, mas foi o *videogame* que realmente lucrou até então, e iniciou a ideia de ganhar dinheiro com jogos eletrônicos e *videogames*. *Pong* foi, também inicialmente, movido à moedas, e era um jogo onde o jogador devia acertar uma bola (esfera) com uma raquete, que era representada por uma barra vertical, lançando a bola ao campo adversário. A bola passava ao campo do adversário e o jogador marcava ponto.

Logo com o sucesso da *Atari*, surgiram, em 1973, outras empresas do ramo dos jogos eletrônicos, dentre outras, as que tiveram maior e considerável sucesso foram as já citadas *Midway* e *Taito*, além de *Chicago Coin*, *Ramtek*, *Allied Leisure* e a *Kee Games*, que na verdade era uma subdivisão da própria *Atari*.

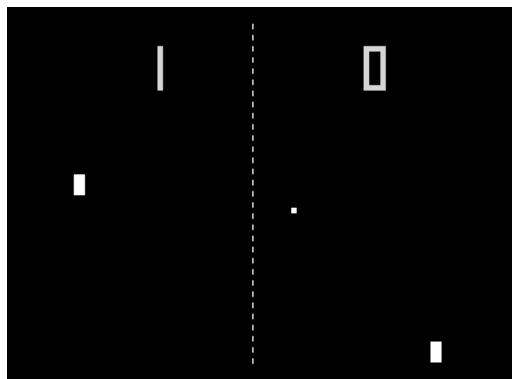


Figura 2.3 - *Pong*, da *Atari* (NEWSGAMER, 2011)

Em 1974, a *Kee Games* lança o *Tank*, que segundo Wolf (2008) foi o primeiro jogo onde o armazenamento de dados gráficos ficava em um chip ROM (*Read-only memory*). Uma memória ROM é uma mídia onde dados são armazenados somente para leitura, não podendo o usuário alterá-los. Esse tipo de tecnologia viria a ser então usado em muitos jogos eletrônicos. O *Tank* é um jogo para dois jogadores, onde cada jogador move-se desviando de minas e atirando no outro, como se estivesse em um tanque de guerra. Ainda, como outros jogos já citados, o *Tank* representava seus gráficos com pontos de *pixel* na tela, usando um plano de fundo na cor preta e os gráficos eram representados com pontos ou linhas verticais e horizontais na cor branca.

O ano de 1974 não é importante somente pelo *Tank*, mas também pelo *TV Basketball*, da empresa *Midway*, que foi o primeiro jogo que usava figuras humanas como avatar, ou seja, buscava desenhar na tela a figura de homens. Os desenhos ainda eram brancos e a cor de fundo preta, mas desenhos mais reais (dos jogadores de basquete e da cesta) já eram mostrados. O *TV Basketball* podia ser jogado por dois ou quatro jogadores ao mesmo tempo, onde esses jogadores controlavam os personagens (seres humanos na tela do *videogame*), e assim como no basquete jogavam a bola em cestas e marcavam dois pontos a cada arremesso certo.

Ainda em 1974, a *Atari* fabrica o *Pursuit* e o *Qwak* (SCHWAB, 2004). Em *Pursuit*, que na verdade foi lançado pela subdivisão da *Atari*, a *Kee Games*, o

jogador controla seu avião para cima, para baixo, direita e esquerda e ataca seus inimigos. *Qwak* é composto por um pato que fica no cenário fugindo do jogador, que deve acertá-lo com um de três tiros a cada cenário diferente.

O ano de 1975 ficou marcado por alguns jogos e *videogames* lançados. Primeiramente, destaca-se o *GunFight*, da *Midway*. Segundo Battaiola (2000), foi o primeiro jogo do mundo a usar um microprocessador. *GunFight* foi provavelmente também o primeiro jogo a tratar de violência, mostrando armas na tela. O jogo, para um ou dois jogadores, trata de um combate entre seres humanos (representados como *cowboy*), onde cada um deve atirar no outro (inimigo). O ambiente era o velho oeste, tanto retratado em diversos filmes americanos.

A *Atari* não fica atrás em 1975, e lança *Steeplechase*, o primeiro jogo *arcade* para seis jogadores ao mesmo tempo (WOLF, 2008). O *Steeplechase* é um jogo que simula uma corrida de cavalos com obstáculos, onde os jogadores controlam cavalos, pulando alguns obstáculos e no final quem conseguir o percurso com menor tempo vence.

A *Kee Games* também inova em 1975, lançando o *Indy 800*, primeiro jogo para até oito jogadores. Nesse jogo foram usadas cores (WOLF, 2008). É um jogo de corrida de carros em que os carros são apresentados de forma colorida.

Seguindo um pouco essa linha anual, em 1976 foi lançado o *AY-3-8500*, pela *General Instruments*. *AY-3-8500* era um chip que continha todos os circuitos necessários para um *videogame*. Vários vertentes do jogo *Pong* usaram esse chip posteriormente (WOLF, 2008).

Mas 1976 ficou marcado mesmo pelo *The Fairchild / Zircon Channel F*, da *Fairchild*. Tratava-se do primeiro *videogame* programável, pois usava cartuchos. Isso permitia que os jogos fossem vendidos separadamente do *videogame*, o que não ocorria até então. Para esse *videogame* foram lançados vários jogos, alguns bem simples, voltados ao público infantil. O jogo *Pong* era um dos que já vinham no chip. Vários outros jogos foram criados e disponibilizados através de cartuchos (BATTAIOLA, 2000).

A importância histórica de 1976 no ramo dos jogos não parou por aí, pois também deve-se relatar o lançamento de *Night Driver*, da *Atari*. Era simplesmente

o primeiro jogo em primeira pessoa (WOLF, 2008). Até então, o jogador tinha uma visão em terceira pessoa, ou seja, a perspectiva gráfica vinha a partir de uma distância fixa, numa visão atrás e acima do personagem do jogador. Em um jogo em primeira pessoa, a perspectiva gráfica vem do ponto de vista do próprio personagem que o jogador controla. Para exemplificar, suponha um jogo de corrida de carros. Em terceira pessoa, o jogador terá uma visão da pista por cima, vendo seu carro totalmente, na lateral e acima. Já em primeira pessoa, terá uma visão de como se estivesse realmente dentro do carro, com a pista e outros carros à sua frente. *Night Driver* era justamente um jogo de corrida, em que a corrida se passava a noite. O jogador tinha a visão do seu carro logo à sua frente, e devia correr pela pista (demarcada com luzes), e desviar de outros carros que estavam no percurso.

A *Atari* também lançou, em 1976, o *Breakout*, que ficou famoso e baseou vários outros jogos (BATTAIOLA, 2000). Era um jogo que iniciava com alguns tijolos, e uma bola devia acertar esses tijolos. O jogador controlava uma barra vertical onde essa bola batia e voltava contra os tijolos. Dessa maneira todos os tijolos deviam ser destruídos. O *Super Breakout* (Figura 2.4) segue o mesmo princípio de jogabilidade e é bem parecido com *Breakout*.

Nessa cronologia tudo corria muito bem, com vários jogos e várias formas de inovação surgindo rapidamente, como primeira pessoa, e jogos em cartucho, por exemplo. Porém, em 1977, a indústria dos jogos eletrônicos sofre o que seria seu primeiro crash (WOLF, 2008). Com isso, diversas companhias fecham ou saem desse ramo.

A *Atari* não sofre tanto com isso, pois lança, em 1977, o console de videogame *VCS*, que logo após foi renomeado para *Atari 2600* (WOLF, 2008). O *2600* era baseado em microprocessador e usava a técnica já citada dos cartuchos. No Brasil ele viria a ser lançado apenas em meados da década de 1980, e foi um fenômeno de vendas, permanecendo na memória dos jovens por muito tempo.

Além do *2600*, outro console importante lançado em 1977 foi o *Color TV Game 6*, da empresa japonesa *Nintendo*, que se tornaria então uma das maiores empresas de videogames de todos os tempos. O *Color TV Game 6* era também um

videogame para ser usado em casa, e possuía como jogos várias variações do *Pong*.

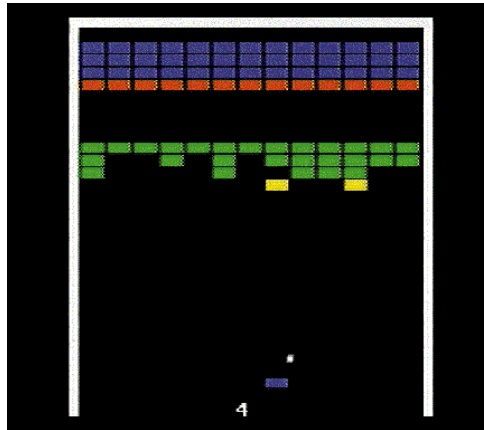


Figura 2.4 - Tela de *Super Breakout* (PEABODY, 1997)

Para terminar o relato do ano de 1977, vale destacar a criação do *Super Bug*, pela *Kee Games*. Era um jogo do tipo *arcade* em preto e branco que simulava condução de veículos. Sua importância se deve ao fato de ter inspirado vários outros jogos posteriormente.

No ano de 1978 a *Taito* lança o *Space Invaders*, que torna-se inspiração para diversos outros jogos, gera muito dinheiro para os desenvolvedores e até mesmo para outras empresas que criaram jogos similares a ele. No *Space Invaders*, o jogador controla os movimentos de uma arma (para a direita e para a esquerda). Essa arma fica na parte inferior, e da parte superior vão caindo *aliens*. O objetivo então é atirar com a arma nesses *aliens* antes que eles cheguem totalmente à parte inferior. O *Space Invaders* foi um dos primeiros jogos de tiro 2D (bidimensional).

Vários jogos já citados acima utilizam projeção 2D, portanto é importante defini-lo. Trata-se da representação de objetos e entidades em duas dimensões (largura e altura). Vários jogos das décadas de 1960, 1970, 1980 e 1990 foram feitos dessa maneira, pela simplicidade e pelo pequeno tamanho que ocupam.

Ainda em 1978 a *Atari* lançou o *Football*, introduzindo ao mundo dos jogos eletrônicos o conceito de rolagem em duas direções. *Football* era um jogo que simulava partidas de futebol americano, e podia ser jogado por dois jogadores. Dessa forma, os personagens do jogo podiam movimentar-se de um lado para o outro (WOLF, 2008). Jogos simulando jogos esportivos viriam a ser muito populares e até hoje movimentam bastante o mercado todo ano.

Por fim, 1978 fica marcado pelo lançamento de *Fire Truck*, pela *Atari*. *Fire Truck* compartilhava funções entre dois jogadores (BATTAIOLA, 2000). Um desses jogadores dirigia um carro (de corpo de bombeiros) e o outro manipulava a escada na parte traseira do carro.

Em 1979 a *Vectorbeam* lançou o *Warrior*, que foi o primeiro jogo de luta um personagem contra o outro. (WOLF, 2008). Esse jogo inspirou diversos outros jogos durante muitos anos, pois jogos de luta tornaram-se muito famosos algum tempo depois. Nesse jogo, os jogadores controlam não só os movimentos do personagem, como também controla a espada que o mesmo possui.

Também em 1979 a *Atari* lançou um dos jogos mais conhecidos de todos os tempos: *Asteroids* (Figura 2.5). Tratava-se de um jogo com gráficos vetoriais bidimensionais. O jogador controlava uma nave espacial, e tinha que atirar em asteróides e discos voadores que ficavam voando na tela, não podendo deixar esses asteróides e discos encostarem na nave. A cada asteróide ou disco destruído o *score* do jogo somava pontos (PEABODY, 1997).



Figura 2.5 - Jogo *Asteroids* (PEABODY, 1997)

A *Atari* também lançou o *Lunar Lander* em 1979. O objetivo do jogo era controlar um módulo espacial sobre a lua. Segundo Battaiola (2000), foi o primeiro jogo a incorporar a tecnologia dos monitores vetoriais. Aqui observa-se que a ideia inicial do primeiro jogo eletrônico (*SpaceWar!*) de transpor a ficção científica da literatura para jogos eletrônicos permanecia, e essa ideia ainda inspirou muitos outros jogos e inspira até os dias atuais.

O ano de 1979 também foi o ano em que a *Namco* lançou o *Galaxian*, segundo Wolf (2008), o primeiro jogo a ter 100% de uso de gráficos no padrão de cores RGB.

Para terminar o relato dos fatos importantes de 1979, ressalta-se o jogo *Puck-Man*, que viria a ser tornar no ano seguinte o *Pac-Man*. *Puck-Man* recebeu esse nome quando a *Namco* o liberou inicialmente no Japão.

Com o surgimento do até então *Puck-Man*, a década de 1970 termina. Entra em ação a década de 1980, que também teve um movimento significativo ano a ano no mundo dos jogos eletrônicos.

2.4. A década de 1980

No ano de 1980, o *Puck-Man* muda de nome para *Pac-Man* (Figura 2.6) e é lançado na América do Norte (SCHWAB, 2004). *Pac-Man*, produzido inicialmente para *arcade*, foi posteriormente versionado para diversos consoles diferentes. Trata-se de um jogo em que o jogador controla um personagem (o Pacman, uma cabeça redonda) que come pastilhas por um labirinto. Como inimigos, haviam alguns fantasmas que perseguiam o personagem do jogador. O objetivo do jogo era simples: comer todas as pastilhas do labirinto sem que os fantasmas o pegassem.

Em 1980 houve o lançamento de *Berzek*, da *Universal Research Laboratories/Stern Inc.* Esse jogo impressionou porque a máquina falava, e a

grande maioria da população não era acostumada com sintetizadores de voz (BATTAIOLA, 2000).



Figura 2.6 - Jogo *Pac-Man* (LUZ, 2004)

Também em 1980 a *Atari* lançou o *Battlezone* (Figura 2.7), e foi segundo Wolf (2008) o primeiro jogo a realmente utilizar um ambiente gráfico 3D (três dimensões). O *Battlezone* constituía-se de um tanque de guerra, e as imagens eram tracejados na tela que formavam a ilusão de profundidade, não possuindo cores dentro desses tracejados, como mostra a figura abaixo.

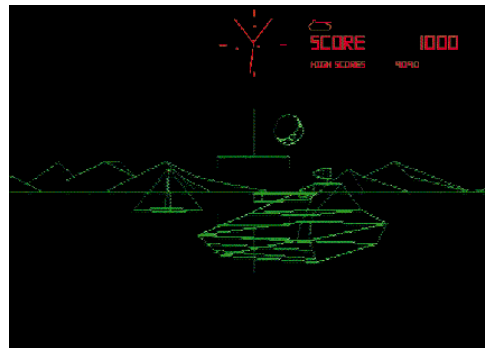


Figura 2.7 - Jogo *Battlezone* (PEABODY, 1997)

Em 1980 outro jogo, em 2 dimensões, lembrado por alguns autores é o *Defender*, que também era um *arcade videogame* (BATTAIOLA, 2000). O jogador controla uma nave e destrói *aliens*.

Outro fato interessante de 1980 é a série *Ultima*, que trouxe o conceito de rolagem em quatro direções (para cima, para baixo, para a direita e para a esquerda). Talvez seja a série de RPG (*Role-Playing Game*) mais antiga. O RPG (aqui considerado o RPG eletrônico) é o tipo de jogo em que os personagens ou ambiente vão sendo desenvolvidos com o tempo, ou seja, o jogador começa em um ambiente com poucos recursos, e esse ambiente vai ganhando mais recursos durante o jogo, assim como seu(s) personagem(ns) e NPC's (Personagens Não Jogáveis) também se desenvolvem durante o jogo. Por fim, 1980 ficou marcado pelo *Star Fire*, que foi o primeiro jogo onde o jogador podia, ao final, colocar seu nome na tabela de melhores *score*, outro recurso utilizado até hoje em muitos jogos. No *Star Fire* o jogador controla um navio, o qual possui movimentos lentos, desviando-se de lasers e atirando eventualmente (WOLF, 2008).

Missile Command (Figura 2.8) também é datado de 1980, e buscava mostrar o medo de um conflito nuclear (BATTAIOLA, 2000).

Em 1981 a *Nintendo* lança o *Donkey Kong*, que segundo Crawford (2003) mudou um pouco o conceito de jogos eletrônicos até então, pois trouxe o conceito de plataformas, isto é, o personagem tinha acesso à pisos diferentes através de rampas e escadas. Em *Donkey Kong*, o personagem *Jumpman* tinha que fugir de certos obstáculos e inimigos através de escadas passando de um andar (pisos) a outro.

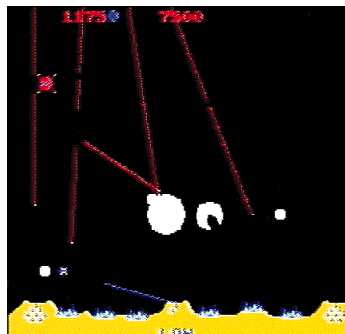


Figura 2.8 - Jogo *Missile Command* (PEABODY, 1997)

A Atari lança *Tempest* (Figura 2.9), que possuía o conceito de escolha de dificuldade, em 1981. Serviu de inspiração para o famoso *Tempest 2000*. (BATTAIOLA, 2000).

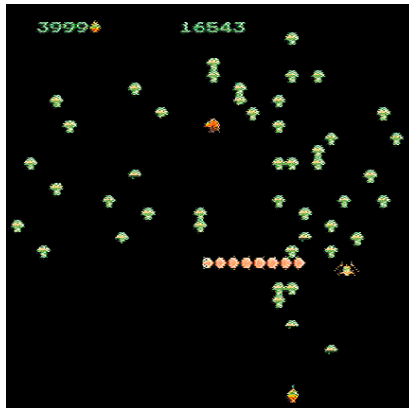


Figura 2.9 - Jogo *Tempest* (PEABODY, 1997)

Ainda em 1981 a Atari também lançou *Centipede* (Figura 2.10), com gráficos bem coloridos, o que atraiu muitos fãs femininos (BATTAIOLA, 2000). Talvez por ter sido o primeiro jogo projetado por uma mulher.



Figura 2.10 - Jogo *Centipede* (NEWSGAMER, 2011)

Em 1982 a Gottlieb lança *Q*bert*, um acarde *videogame*, que usava um conceito de plataforma também. Em *Q*bert*, há um tipo de pirâmide formada por

cubos. O personagem controlado pelo jogador inicia no topo dessa pirâmide e deve então pular para os cubos na direção para baixo. Conforme esses cubos mudam de cor ele pode ou não parar neles (SCHWAB, 2004).

Também em 1982 uma empresa chamada *Sega* lança o *Zaxxon*, o primeiro *arcade* a ser anunciado na televisão (WOLF, 2008). A *Sega* deve ser estudada atentamente, pois se tornou uma empresa muito forte do ramo, algum tempo depois. O *Zaxxon* tenta também simular uma percepção de visão em 3D.

Segundo Wolf (2008), em 1982 a indústria dos *videogames* sofreu seu segundo *crash*, que foi tão ruim quanto o de 1977, e em 1983, a indústria dos *videogames* caseiros foi muito afetada.

Mas 1982 também é marcado pelo lançamento da *Williams Eletronics: Joust*. Era um jogo para vários jogadores, onde eles poderiam colaborar entre si. Segundo Battaiola (2000), há uma teoria em que jogos colaborativos aumentam a agressividade. Isso pode estar relacionado ao fato de que jogadores sempre irão querer vencer a máquina, talvez mais do que vencer outro jogador.

Em 1983 a *Nintendo* lança o *Famicom*, *videogame* que ficou famoso em todo o mundo. As vendas desse *videogame* certamente ajudaram muito a indústria dos jogos eletrônicos a se recuperar da crise de 1982.

Em 1983 há notícia do primeiro jogo de xadrez a vencer um ser humano no nível difícil (SCHWAB, 2004), dado importante do ponto de vista de inteligência artificial.

Também em 1983 a *Atari* lança o *I, Robot*, o primeiro *videogame* com polígonos gráficos tridimensionais. O objetivo do *I, Robot* em si é um robô que deve destruir um gigante, chamado de *Big Brother*. Ele influenciou diversos jogos simuladores de vôo e de corrida (BATTAIOLA, 2000).

Porém, em 1983, o jogo mais conhecido foi o *Star Wars*, da *Atari*. Era licenciado pela *Lucasfilm*, pois o jogo era baseado no filme de mesmo nome. O jogo era colorido e usava renderização 3D. O objetivo eram batalhas no espaço e perseguições nas estrelas. Outra característica interessante era o uso de vozes do filme no jogo (WOLF, 2008).

Em 1985 a *Nintendo* lança uma nova versão do *Famicom* na América chamado de *NES (Nintendo Entertainment System)*. Segundo Wolf (2008), a popularidade desse *videogame* deu fim ao *crash* que vinha à alguns anos. No mesmo ano, a própria *Nintendo* lança o *Super Mario Bros*, um dos jogos de maior êxito da história. No *Super Mario Bros*, o personagem Mario é controlado pelo jogador e movimenta-se para esquerda, direita e pula, acertando seus adversários.

Também em 1985, Alex Pajitnov projetou o *Tetris* (Figura 2.11), outro dos jogos de grande êxito de todos os tempos. O *Tetris*, aparentemente de funcionamento muito simples, é um jogo que utiliza da lógica para ser jogado. No jogo, blocos coloridos em diversos formatos vão caindo de cima para baixo, e o jogador muda o posicionamento desses blocos de forma que os blocos vão unindo-se, e quando uma linha é totalmente preenchida, ela some, e o jogador soma pontos.

Em 1986 surgem *The Legend of Zelda* (da *Nintendo*), *Arkanoid* e *Bubble Bobble* (da *Taito*), e *Sega Master System (SMS)*, da *Sega*. *The Legend of Zelda* foi o primeiro jogo de vários da série *Zelda*. Esse tipo de recurso (série de jogos) é utilizado bastante até hoje em dia, com vários títulos tendo ficado famosos. *Arkanoid* era similar ao *Breakout*, de 1976, já citado nessa seção. *Bubble Bobble* era também baseado em plataformas, sendo inicialmente um *arcade*, passando para *videogames* caseiros logo em seguida.



Figura 2.11 - Tela reconstituída do jogo Tetris (LUZ, 2004)

The Manhole é lançado em 1987, pela Cyan, o primeiro jogo eletrônico a ser lançado em CD-ROM (*Compact Disc Read-Only Memory*) (WOLF, 2008). O CD-ROM, disco compacto para armazenamento de informações, foi usado em muitos jogos eletrônicos, herdando certas características que foram criadas pelo *Tank* de 1974 (chip ROM) e também jogos em cartuchos. O *The Manhole* foi desenvolvido então para várias plataformas computacionais diferentes.

Ainda em 1987 é lançado o *arcade videogame Yokai Douchuuki*, o primeiro *arcade game* 16-bit. 16 bits, na arquitetura de computadores, diz respeito à geração de processadores de 16 bits, nos quais a palavra para representar inteiros, endereço de memória e outros tipos de dados são constituídos de 16 bits. Um processador 16 bits pode acessar uma memória de no máximo 64 KB.

Maniac Mansion é outro jogo lançado em 1987, pela LucasArt, o primeiro jogo que utilizava *point-and-click*. Tratava-se de um jogo onde o jogador controlava o personagem, guiando o mesmo através de um cursor. Com algum botão (seja do controle do *videogame* ou mouse no caso de computadores), a ação é acionada.

Ainda em 1987 uma empresa chamada *Incentive Software* lança o *Driller*, um jogo para computador com grandes avanços em gráficos 3D (WOLF, 2008). O jogo é em primeira pessoa e o jogador controla um tipo de sonda de escavação.

Também em 1987 a Taito lança o *Double Dragon*, jogo famoso por seus cenários. Trata-se de um jogo de luta onde os personagens enfrentam certos inimigos vindos de gangues. A história do jogo é bastante interessante e também ganhou certa fama.

Em 1988 a *Namco* lançou o *Assault*. É um jogo também *arcade*, em que o jogador controla um tanque em um ambiente de *alliens*.

Surgiu o *NARC*, ainda no ano de 1988, outro marco, pois foi o primeiro jogo a utilizar um processador 32-bit. Um processador 32 bits consegue acessar memórias de até 4 GB. Até hoje jogos são criados para 32 bits, apesar de que a pouco tempo jogos começaram a migrar para processadores 64 bits. *NARC* é conhecido por ser um jogo violento. Jogos violentos iriam fazer bastante sucesso anos mais tarde.

Vale ressaltar ainda em 1988 o lançamento do *Super Mario Bros. 2* (pela *Nintendo*).

O ano de 1989 ficou marcado pelo lançamento do *arcade game Hard Drivin*, que foi um jogo de carros, onde o jogador se sentia em uma perspectiva de primeira pessoa (a câmera mostrava o painel do carro).

Também em 1989 foram lançados dois *consoles* de mão: *Game Boy* (pela *Nintendo*) e *Lynx* (pela *Atari*). Até hoje *videogames* portáteis são famosos e bastante vendidos em todo o mundo.

2.5. A década de 1990

A década de 1990 também teve um movimento grande na indústria dos jogos eletrônicos. Observa-se aqui que desde a década de 1970 essa indústria foi bastante movimentada, entre crises aparecendo e desaparecendo, além de inúmeras novas tecnologias e componentes para jogos sendo criados.

Já em 1990, foi criado o *SimCity* (Figura 2.12), jogo que mostrava a criação de uma cidade. Nesse mesmo ano a *Nintendo* lança o *Super Mario Bros. 3*, e já é observado aqui que a franquia de *Super Mario Bros* foi bem rentável.

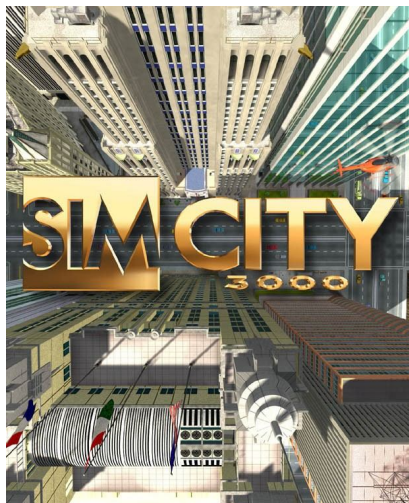


Figura 2.12 - Jogo SimCity (BALDANCE & REIS, 2006)

Ainda em 1990 a *SEGA* criou o videogame portátil *Sega Game Gear*, talvez em resposta ao *Game Boy*, da *Nintendo*. Concorrência é algo que existe no ramo dos jogos eletrônicos, assim como em quase todos os ramos da interatividade.

O ano de 1990 marca o início da série *Final Fantasy*, da *Squaresoft*. Jogo conhecido, de *RPG*, e que possuía enredo forte. A questão do enredo já era utilizado por outros jogos e hoje a maioria dos jogos preocupa-se efetivamente com isso.

Por fim, em 1990, é lançado *Herzog Wei*, considerado o primeiro jogo de estratégia em tempo real.

Em 1991, a *Nintendo* criou o *SNES* (*Super Nintendo Entertainment System*), sucessor do *NES*, de 1985. Vale ressaltar aqui, que a partir da década de 1990 um novo conceito seria o forte. Acontece que nas décadas 1970 e 1980 os *arcades* foram o forte da indústria, como relatado nessa seção. De 1990 pra frente, os videogames caseiros (muitas vezes chamados de *consoles*), além dos jogos para PC (*Personal Computer*). Jogos para PC são jogos desenvolvidos para rodarem em cima de Sistemas Operacionais em computadores pessoais.

Também em 1991 a *Capcom* lançou o *Street Fighter II*, jogo de luta que ajudou a popularizar o gênero em todo o mundo. *Street Fighter II*, assim como vários jogos de luta, tem o funcionamento bem simples: o jogador controla seu personagem e também os movimentos na tela, além de socos, chutes, etc. Foi continuação do *Street Fighter* (Figura 2.13), da década anterior.



Figura 2.13 - Tela do Jogo Street Fighter (BALDANCE & REIS, 2006)

O ano de 1991 foi também o marco para a *Sega*, com o lançamento de *Sonic the Hedgehog*, o jogo possuía *Sonic*, que viria a ser o mascote da *Sega* (WOLF, 2008). Personagens viraram marcos dos jogos eletrônicos, adorados por muitos jogadores durante a história.

Em 1992 a *Midway* lançou o *Mortal Kombat*. Também foi um jogo de luta, bastante popularizado, que viria a ter então diversas outras versões. Similar ao *Street Fighter II* em termos de jogabilidade, esse jogo imortalizou vários personagens, conhecidos por muitas pessoas.

Também em 1992 a *Sega* lançou *Virtua Racing*, que iniciou uma era de jogos de corrida em 3D, além do início da era dos poderosos simuladores para múltiplos jogadores (BATTAIOLA, 2000).

Alguns autores relatam à 1992 a importância de *The 7th Guest*, que teria se tornado um best-seller.

Em 1993 a *Cyan* lançou *Myst*, que foi outro best-seller (WOLF, 2008). O jogo foi marcado pelos gráficos excelentes, em uma época em que a parte gráfica ainda possuía grandes limitações.

A empresa *id Software* lançou em 1993 o *Doom*, jogo violento no estilo primeira pessoa. Controverso quanto à violência explícita, mas imensamente popularizado em todo o mundo. Vários jogos viriam a ter problemas com violência, e alguns jogos chegam a ser proibidos, atualmente, em alguns países. *Doom* teve algumas continuações, dentre eles *Doom3* (Figura 2.14), já na década de 2000.



Figura 2.14 - Tela de Doom3 (LUZ, 2004)

Ainda em 1993 a *Sega* lançou *Virtua Fighter*, jogo de luta. Como já mencionado, jogos de luta foram bastante populares e renderam muito às produtoras. Mas *Virtua Fighter* era um jogo em 3D, e excepcionalmente para jogos de luta, os jogadores costumam preferir jogos em 2D. Claro que jogos 3D tendem a ter gráficos mais reais, então o que acontece com jogos de luta é que a popularização de jogos como *Mortal Kombat*, por exemplo, fez com que os jogadores se prendessem a esse tipo de jogo, até mesmo pela facilidade de movimentação em ambientes apenas 2D.

Em 1994 a *Nintendo* lançou o jogo *Donkey Kong Country*, que possuía mais de um cenário, em um ambiente no estilo plataforma. É interessante o fato de que grandes empresas de décadas anteriores, como a *Nintendo*, continuaram fortes, passaram por crises, mas provaram que o negócio dos jogos eletrônicos é algo imensamente rentável.

Também em 1994, a *Sega* lançou o *Sega Saturn*, e a *Sony* lançou o *Sony PlayStation*, no Japão. Isso prova o fato de que os *consoles caseiros* dominariam a década. O *PlayStation* ficou muito famoso, e vários jogos foram criados para essa plataforma. Algo muito interessante desse *console* é o controle (*joystick*), que não sofreu muitas mudanças para versões futuras do *console*. Isso se deve ao fato de ser um controle bem confortável para as mãos do jogador.

Ainda em 1994 a *Blizzard* lançou *Warcraft*. *Warcraft* foi um jogo de estratégia em tempo real. Combates entre personagens humanos e os chamados *orcs* era o forte do jogo.

O ano de 1994 ainda teve outros marcos, um deles o jogo *Daytona USA*, do tipo *arcade*, da *Sega*, e talvez um dos *arcades* mais famosos de todos os tempos. Nele, o jogador fica sentado como se estivesse em um carro de verdade, o controle é um volante e o jogador participa de corridas. Esse jogo iniciou os jogos que buscavam se parecer com a realidade (BATTAIOLA, 2000).

Em 1995 o *Sony PlayStation* e o *Sega Saturn* chegam à América do Norte. O uso de *console* caseiro na década de 1990 não foi exclusivo no Japão, mas sim em todo o mundo.

Também em 1995 foram lançadas novas versões de *Donkey Kong Country* e *Warcraft: Donkey Kong Country 2* e *Warcraft II*. Isso reforça a fama desses jogos, pois não foi sem motivos que criariam novas versões dessas franquias.

O ano de 1996 foi marcado pela *Nintendo*, que lançou, tanto no Japão quanto nos Estados Unidos da América, o *Nintendo 64*, console que foi muito popularizado, e os jogos eram lançados em cartuchos.

Também em 1996 foi publicado *BattleCruiser: 3000AD*, um jogo eletrônico de ficção científica. Nesse jogo, o jogador controlava uma espécie de nave pelo espaço.

Sega Saturn, *Sony PlayStation* e *Nintendo 64* foram os grandes consoles concorrentes da mesma geração.

Em 1997 o *Nintendo 64* foi lançado na Europa e na Austrália. No mesmo ano, a *Sega* lançou o *Top Skater*, um jogo com interface de *skateboard*.

Ainda em 1997 a *Nintendo* lançou o *Mario Kart 64*, que era um jogo de corrida de carros. Os personagens corriam e trapaceavam de diversas formas.

Segundo Wolf (2008), também em 1997 começa o *Ultima Online*. *Ultima Online* é um tipo de MMORPG (*Massive Multiplayer Online Role-Playing Game*), e foi um dos mais jogados durante bastante tempo. Um MMORPG é um jogo em que vários jogadores criam personagens em um ambiente ao mesmo tempo.

O ano de 1998 ficou marcado por jogos em um tipo de estilo musical. Um dos bem famosos foi lançado pela *Konami: Dance Dance Revolution*, que inspirou diversos outros jogos, posteriormente. Em *Dance Dance Revolution*, o jogador dançava em cima de uma plataforma, que possuía setas para ele pisar. Conforme as setas apareciam na tela, ele devia pressioná-las, e ao acertar o tempo, acumulava pontos.

Com o sucesso do *Game Boy*, a *Nintendo* lançou em 1998 o *Game Boy Color*, que apresentava uma tela colorida.

Ainda em 1998 a *Sierra Studios* lançou *Half-Life*, que ficou eternizado como um dos melhores jogos de tiro em primeira pessoa. Podia-se jogar *Half-Life* em computadores pessoais.

Por fim, a *Rockstar Games* lançou o GTA (*Grand Theft Auto*), em 1998. *Grand Theft Auto* era um jogo no qual o jogador controlava seu personagem, movimentando-o pela cidade. Tinha como objetivos, então, realizar algumas missões, dentre as quais podia fazer ações ilegais, como por exemplo roubar carros. Esse jogo daria início a uma série de jogos com o título *Grand Theft Auto*, mais conhecidos como GTA. GTA então foi muito criticado, pelas cenas ilegais que o jogo proporciona, mas por outro lado, tornou-se um fenômeno de vendas, concorrendo com os jogos mais vendidos de todos os tempos.

Aqui observa-se, que no final da década de 1990 os *arcades* sofreriam uma certa extinção, e os fabricantes começariam a criar jogos para computadores pessoais e *consoles* caseiros, tais como o *PlayStation*, por exemplo. A importância de focar esforços no software tornou-se inevitável, pois o consumidor final comprava um *console* e podia então comprar vários jogos diferentes para esse *console*. Observa-se também que empresas de menos destaque anteriormente começavam a se destacar, como a *Rockstar Games*, por exemplo.

Em 1999 destaca-se o *Tony Hawk's Pro Skater*, jogo em que o jogador controla um skatista, que faz manobras em pistas de *skate*.

2.6. Os jogos após o ano 2000

A década de 2000 começa com a *Sony* lançando o *PlayStation 2*, *console* sucessor do *PlayStation*, e que também faria bastante sucesso.

Em 2000 a *Maxis* lançou *The Sims* (Figura 2.15), sucesso de vendas. Tratava-se de um jogo no qual o jogador controlava seres humanos, desde o seu nascimento até a morte. Os personagens interagiam entre si, construíam casas, estudavam e adquiriam conhecimento, namoravam, casavam, tinham filhos, etc. Tratava-se realmente de um simulador de vida. *The Sims* teve várias variações depois, extensões e até continuação da série.



Figura 2.15 - Jogo *The Sims* (PEABODY, 1997)

No ano seguinte, 2001, surgem os *consoles* Xbox (da *Microsoft*), e também o *GameCube* (da *Nintendo*). Com o *GameCube*, tinha-se por objetivo concorrer com os *consoles* da *Sony*. *Nintendo* e *Sony* travariam realmente batalhas de concorrência a partir de então, e até hoje produzem *consoles* de ponta, que concorrem entre si. O *Xbox* mostrou como a indústria dos jogos eletrônicos ficou rentável durante os anos, pois a *Microsoft*, empresa que ficou conhecida pela criação de Sistemas Operacionais e outros tipo de programas de computador, acabou entrando também no ramo do jogos eletrônicos (WOLF, 2008).

Também em 2001 vale ressaltar o aparecimento de *Halo: Combat Evolved*, da *Bungie Studios*. Era outro jogo do estilo tiro em primeira pessoa. Vale ressaltar o quanto esse tipo de jogo fez sucesso, ao ponto de tantos jogos terem sido criados para esse estilo. Em *Halo*, uma curiosidade interessante é que o rosto do personagem não apareceu, nem mesmo nos jogos de continuação (WOLF, 2008).

Ainda em 2001 a *Sega* anunciou o fim de seus trabalhos com *consoles*, ou seja, não desenvolveria mais *videogames* para casa, ficaria apenas na criação de software e hardware (específico) para jogos eletrônicos.

O grande marco de 2001 foi então *Black & White*. Foi um jogo onde o jogador assumia a posição de um deus e controlava o ambiente do jogo (KISHIMOTO, 2004).

Em 2002, *The Sims* ultrapassou *Myst* em vendas, tornando-se o best-seller de vendas de todos os tempos. No mesmo ano foi criado o *MMORPG Sims Online*, que é importante ressaltar não pelo sucesso (que não obteve), mas pelo

rumo que as fornecedoras de jogos eletrônicos poderiam tomar: o uso da internet e dos jogos eletrônicos combinados. A própria *Microsoft* anunciou no mesmo ano o *Xbox Live online gaming service*, que possuía, como ótimos serviços, funcionalidades como: download de bônus para jogos, criação de perfis, chat, torneios, etc (WOLF, 2008).

Em 2003 houve o início da MMORPG *Star Wars Galaxies*, que foi um MMORPG baseado nos filmes de título *Star Wars*. Nesse mesmo ano a *Nintendo* anunciou o fim da produção do *NES* e do *SNES*. Também em 2003 a *Atari* lançou *Enter the Matrix*, um jogo baseado no filme *Matrix*. *Enter the Matrix* era bastante baseado no filme, com os personagens passando por situações parecidas, além de ter tido um fluxo de vendas alto. Por fim, ao falar de 2003 deve-se destacar o *N-Gage*, da *Nokia*. Tratava-se de um smartphone concebido para jogos. Cada jogador jogava individualmente no seu, mas podia jogar com outros por *bluetooth*, por exemplo (WOLF, 2008).

Em 2004 a *Sony* lançou o *PlayStation Portable*, que foi outro sucesso de vendas (lançado no Japão, seria lançado na América do Norte em 2005). Vários jogos do *PlayStation* e *PlayStation 2* foram migrados para esse *videogame*. Tratava-se da versão portátil (de mão) do *PlayStation*.

Também em 2004 a *Nintendo* não ficou para trás e lançou o *Nintendo DS*, portátil que, como característica única, possuía dois visores. É utilizado e muito vendido até hoje (SANTANA, 2006).

Em 2005 a *Microsoft* lançou o *Xbox 360*. Era a nova versão do *videogame* de alta performance da *Microsoft*. É colocado atualmente como um dos três grandes *videogames* de última geração. Em 2006, a *Sony* lançou o *PlayStation 3* e a *Nintendo* lançou o *Nintendo Wii*. Tratam-se dos outros dois grandes *videogames* de última geração. Dentre características fortes do *PlayStation 3* destaca-se a grande variedade de jogos para esse *console*, e para o *Wii* destaca-se o seu controle primário, que seria uma revolução no modo de jogar. Trata-se de um controle conectado ao *console* via *bluetooth*, e os movimentos são captados e transmitidos quando o jogador move esse controle. É um controle interessante

para jogos do tipo Tênis, por exemplo, onde o controle simula a raquete (SANTANA, 2006).

Ainda a 2005, a *EA Sports* (sub divisão da empresa *EA Games*) lança o *FIFA 2005* (Figura 2.16). *FIFA* é uma série de jogos de futebol que existe desde a década de 1990, e é licenciada pela própria FIFA como jogo oficial da organização. Jogadores, times e campeonatos são baseados nos reais. A série FIFA lança um jogo a cada ano. *FIFA 2005* destaca-se aqui pelos ótimos gráficos que trazem. FIFA tornou-se então um fenômeno, com jogos cada vez mais realistas, como *FIFA 2008*, *FIFA 2009*, *FIFA 2010*.

O ano de 2007 detém uma estimativa interessante: o *World of Warcraft* teria nove milhões de jogadores em todo o mundo (WOLF, 2008). O *World of Warcraft* trata-se de um MMORPG.



Figura 2.16 - Imagem de *FIFA 2005* (ENE, 2011)

Com consoles suportando jogos cada vez melhores para suportar gráficos e IA (Inteligência Artificial) melhores, no final de 2010 foi lançado o *FIFA 2011* (Figura 2.17). Realmente impressionante, com gráficos e uma jogabilidade não vistos em outros jogos de futebol. Pode-se fazer uma comparação da melhoria gráfica dos jogadores, desde 2005 até 2011, observando as figuras 2.16 e 2.17.

Na década de 2000 inúmeros jogos surgiram, de todos os tipos imagináveis: jogos de ação, de luta, de futebol, de basquete, ficção científica, jogos baseados em filmes, etc.



Figura 2.17 - Jogo FIFA 2011 (SPORTS, 2011)

Mas para finalizar, vale ressaltar um fenômeno de vendas: a série GTA. Principalmente pelos jogos *GTA: Vice City*, *GTA San Andreas* e *GTA IV* (Figura 2.18). Todos seguem o mesmo princípio de GTA, do final da década de 1990. Para exemplificar, em *GTA: Vice City*, o personagem principal (controlado pelo jogador) está em uma cidade baseada em Miami, e faz diversas missões, enfrentando gangues e até mesmo a polícia. É um tipo de jogo violento, mas que agrada milhões de pessoas. *GTA IV* trouxe gráficos ainda mais impressionantes.



Figura 2.18 - Cena de explosão em GTA IV (GAMES, 2011)

2.7. Conclusão

Os jogos eletrônicos estão presentes na sociedade há poucas décadas, mas a história apresentada mostra a rápida evolução desse tipo de entretenimento. O

desenvolvimento desses jogos foi possível devido à evolução da computação gráfica nesse período. Os fornecedores de jogos mostraram muita inovação e criatividade, pois jogos dos mais variados tipos foram apresentados. O processo de desenvolvimento de um jogo (as técnicas utilizadas em sua criação) também deve ser analisado, além da história da inteligência nesses jogos. O capítulo 3 mostra os componentes mais utilizados na criação de jogos, além da IA utilizada por alguns jogos da história.

3. TÉCNICAS DE DESENVOLVIMENTO DE JOGOS

É importante identificar e explicar o processo de criação de um jogo eletrônico, no qual os oponentes inteligentes são inseridos. Em vários jogos, o processo de criação pode variar, como por exemplo, quando os jogos possuem gêneros diferentes. O desenvolvimento de um jogo de ação possui alguns componentes que um jogo de futebol não possui, e vice versa, por exemplo.

Existem algumas fases de desenvolvimento que praticamente quase todos os jogos possuem, dentre as quais pode-se citar roteiro, motor, áudio, IA, rede, entre outras. Aqui serão analisadas algumas das mais importantes e comuns. Não é analisado como esses elementos devem ser implementados, mas ao menos o que são e qual a sua importância é algo que deve ser relatado.

Ao ler as descrições de cada fase, pode-se observar que uma está totalmente ou parcialmente relacionada a outra(s). Isso obviamente ocorre, pois no final está criando-se um jogo só.

3.1. Game Design

O *Game Design* talvez seja o primeiro passo na criação de um jogo, pois o ato de imaginar um jogo já é um passo do *Game Design*. O *Game Design* é o processo de se criar um jogo, definir como ele será, o que ele será. Isso significa que o *Game Design* decidirá como o jogo irá funcionar, além de conceituá-lo, em termos de funcionamento e parte artística (LUZ, 2004).

Como o próprio nome sugere, essa fase tem por objetivo desenhar o jogo, rascunhá-lo. Ele pode ser pensado por uma pessoa, mas normalmente não é

totalmente feito “do zero”. Um jogo, muitas vezes, pega ideias de filmes, ou até mesmo é baseado em filmes. Outras fontes de ideias para jogos são os esportes em geral. Existem inúmeros jogos de luta, futebol, basquete, tênis, etc. Como exemplos estão a série *Fifa* e a série *Winning Eleven*. Na seção história podem ser observados vários outros jogos baseados em esportes. Jogos de tabuleiro, como damas ou xadrez, também influenciaram muitos jogos. Portanto, ideia para um jogo pode vir dos mais diferentes contextos.

Outra importante transposição que ocorre é a influência do lado contrário, ou seja, jogos eletrônicos que influenciam filmes, por exemplo. É o caso do jogo *Max Payne*, que influenciou um filme de sucesso, com o mesmo nome.

O *Game Design* é um processo importante, além de implicar em grande responsabilidade para o projetista. Isso acontece porque muitos que jogam se acham capazes de criar um jogo (LUZ, 2004). Quando uma pessoa está jogando o gênero ação, por exemplo, ela pode ter várias ideias de como as fases poderiam ser diferentes, de como certos cenários poderiam ter sido trabalhados de forma a ficarem melhores ou de como a dificuldade poderia ter sido aumentada. O projetista deve então conseguir agradar as outras pessoas, e isso realmente não é uma tarefa trivial.

O *Game Design* não deve ser confundido com o roteiro. As principais funções do *Game Design* estão na jogabilidade. Isso significa pensar em como o jogador irá atuar no jogo, suas escolhas, o que ele pode ou não controlar, como os elementos que ele não controla irão influenciá-lo. Então, é definir como o jogo funcionará (SANTANA, 2006).

A ideia, como já dito, é o primeiro passo para um jogo. Idealizar é transpor do pensamento humano para algo objetivo. Nesse caso, o algo seria o programa de computador, denominado jogo eletrônico. Desenvolver um jogo da maneira como o mesmo foi pensado não é fácil. O projetista imagina determinado jogo e deve considerar também o que seria necessário para criá-lo: os cenários imaginados são possíveis de serem criados, considerando o poder gráfico disponível? O controle dos movimentos e decisões dos NPC's serão possíveis de serem criados com as

técnicas de IA conhecidas? Será realmente possível transpor o que foi imaginado para o mundo virtual?

Até agora foi considerado bastante o fato de que um jogo é criado pelo pensamento de um ser humano. Outra vertente interessante de analisar é a que trata de jogos que não foram criados pelo pensamento de seres humanos, mas para transpor o mundo real para o mundo virtual. Trata-se de jogos como *The Sims* ou *Fifa 2011*. Esse tipo de jogo tenta colocar a realidade em um jogo eletrônico. Em *Fifa 2011*, um elemento da realidade (o jogo de futebol) é transposto para um jogo eletrônico. Nesse jogo em específico, foi mantido a realidade quase que totalmente. Os jogadores existem realmente, os clubes, etc. Os cenários são baseados em campos de futebol reais. Os jogadores são baseados em seres humanos e renderizados graficamente como tal.

Segundo Santana (2006), além de ter uma boa ideia, ainda no *Design* é necessário cuidar do público, pensar em séries, ter conteúdo, ser flexível, dentre outras regras básicas. A flexibilidade já foi citada, está relacionada ao fato de que talvez não seja possível criar o jogo da maneira como o mesmo foi imaginado.

Cuidar do público é imaginar qual o público alvo, e como esse público irá esperar o jogo. Uma análise de público pode-se fazer necessária, para observar qual o tipo de jogo esperado naquele momento, qual o diferencial que ele deverá ter para que as pessoas o comprem e onde lançá-lo (país ou região) primeiro.

Pensar em séries é imaginar o que poderia vir depois no jogo, ou seja, o jogo poderá tornar-se uma série, novos *plugins* (muito comumente chamados de expansão) poderão ser instalados, etc. Série é o caso de *Age of Empires*, *GTA* ou *Castlevania*. Exemplo de expansão pode ser *The Sims: Fazendo a Festa*, expansão do jogo *The Sims*.

Um jogo que possui conteúdo é aquele que pode ser considerado completo, ou seja, possua boa jogabilidade, cenários suficientes, personagens e história interessantes. Isso pode tornar um jogo divertido (SANTANA, 2006).

O *Game Design* não deve ser confundido com roteiro. O roteiro é um passo feito logo após, e formaliza o que foi imaginado no *Game Design*.

3.2. Roteiro

O roteiro é responsável pela história do jogo. Além da história dos personagens, época em que o jogo se passa e ambiente. Para Borges & Barreira & Souza (2009) é uma das partes fundamentais, pois é o roteiro que poderá convencer os investidores das potencialidades do jogo.

Em jogos baseados na realidade, as vezes o roteiro tende a se tornar mais simples, como no *Fifa 2011*: serão criadas partidas de futebol, que poderão ou não estar inseridas em campeonatos. Nesse tipo de jogo o roteiro é menos complexo pois não trata da história. Deve-se ressaltar que no roteiro estarão os clubes e campeonatos que serão disponibilizados no jogo, os estádios de futebol, dentre outras coisas.

Tratando inicialmente da história (história geral, ou história do jogo), é comum esta estar bem definida em jogos de ação, tiro, ficção científica, dentre outros. Muitas vezes é a história que fará com que o jogador fique preso à tela. Jogadores preferem uma boa jogabilidade, mas a história caminha com a jogabilidade. O que o jogador pode fazer está relacionado com o contexto no qual o jogo está inserido. A história também é o que faz a continuação do jogo, o desenrolar dos fatos, e por isso é que está diretamente relacionada com a jogabilidade. Para melhor entendimento, pode-se imaginar um cenário onde o jogador deve interagir com certos oponentes (destruí-los, vencê-los). Dependendo da história do jogo, o mesmo pode acabar ou continuar. Se continuar, é a história que irá dizer o que acontecerá: uma mudança de cenário, a chegada de mais oponentes, etc. Portanto, o que o jogador poderá fazer posteriormente (jogabilidade) será decidida já na história.

A história dos personagens está bem relacionada com a história geral, pois são os personagens que irão atuar nessa história. Pode-se fazer aqui uma analogia entre novelas de televisão ou filmes: um filme possui uma história, muito comumente chamada de trama, e personagens (vividos na trama por atores) que irão viver essa história, que irão desenvolver o filme. Em um jogo o mesmo ocorre: uma história geral, onde personagens (aqueles que o jogador controla) e

agentes eletrônicos (que dão vida aos NPC's) irão atuar. Além dessa relação entre história geral e a atuação do personagem, a história dos personagens torna-se específica em determinado ponto. Cada personagem possui sua história. Se o personagem é a parte viva do jogo, ele possui um passado, e no presente está atuando no jogo. Esse passado pode ou não influenciar o jogo e sua história.

Quanto ao personagem, é interessante que suas características também já estejam definidas no roteiro. Uma característica trata do que ele é: ser humano, outro animal qualquer, robô, uma forma de vida fictícia, etc. Se é o personagem quem dá vida ao jogo, ele deve ser algum ser que consiga viver ou simular vida (no caso de robôs ou similares). Outras características tratam dos aspectos físicos do personagem: mesmo que vários personagens sejam seres humanos, eles possuem físico diferente, alguns mais altos, outros mais baixos, cores diferentes, rostos diferentes, etc. Mais características a serem abordadas são as relacionadas à parte mental do personagem: o que ele consegue fazer, até onde ele consegue ir. Observa-se que portanto, a IA em um personagem já pode ser colocada no roteiro, ou seja, o personagem conseguirá ou não agir de forma diferente sobre diferentes escolhas do jogador.

A época em que o jogo se passa também é muito importante e deve ser colocada no roteiro. Ela diz respeito ao tempo (do calendário) em que o jogo acontece. Esse tempo pode ser épocas passadas, presente ou até no futuro. Como exemplo pode-se citar o *Age of Empires*, que se passa no tempo das antigas civilizações, como a Roma Antiga. Como *Age of Empires* tinha por objetivo mostrar a construção de civilizações daquela época, é natural o jogo ser projetado no passado. Analogamente, vários outros jogos se passam em épocas diferentes, e podem até mesmo avançar essas épocas, como é o caso do próprio *Age of Empires*, que avança nas Eras.

Por fim, é interessante que o ambiente também esteja descrito no roteiro. O ambiente é o local onde o jogo se passa, onde os personagens interagem (dialogam, lutam, duelam, etc.) e onde a história é totalmente criada. Para um jogo em que a época está no passado, é importante que seja retratado um ambiente no

passado, e assim por diante. Deve-se analisar a época e tomar cuidado para que se retrate essa época com o máximo de detalhes possíveis.

Logo após o fim do roteiro, que pode ser modificado posteriormente, há a escolha de um motor, que fará o trabalho de desenvolvimento do jogo, ou seja, irá transpor o que está definido no roteiro para um programa de computador.

3.3. Motor (*engine*)

O motor implementa o módulo de renderização gráfica do jogo, responsável por renderizar gráficos em duas ou três dimensões, além de simular a física do jogo (detecção de colisão, por exemplo), e suporte a sons e animações. Pode também implementar a IA do jogo. Muitas vezes a física e IA são tratadas fora do motor.

Além de renderizar gráficos, o motor é quem normalmente controla IA e física (quando não estão dentro do motor), rede, sons, animações, e os outros componentes que porventura possam ser utilizados. Isso acontece normalmente e não sempre porque em alguns casos o motor não é único. O que acontece é que pode-se dividir em vários motores: um para criação de IA, outro para a parte gráfica, outro para rede e assim analogamente, e no final há o trabalho apenas de juntar as partes (SANTANA, 2006). Motores para o desenvolvimento da IA não são encontrados facilmente, e portanto é aconselhável utilizar IA isolada de motor. Assim será feito neste trabalho, em que a implementação será focada na IA.

O motor que decide o que acontecerá após determinada ação do jogador, ou seja, o fluxo do jogo também é de sua responsabilidade. Outro fator que ele controla são as regras do jogo que foram estabelecidas ainda no *Game Design* (SILVA, 2008).

Os responsáveis pelo jogo são quem decidem qual motor irá ser utilizado. Não é obrigatório a utilização de um, vários jogos podem ser criados sem nenhum motor, onde tudo é feito separadamente e no final há a união dos componentes.

Na maior parte dos casos, utiliza-se um motor pronto, pois são robustos e foram utilizados por outros jogos, então há uma certeza no seu funcionamento, e

até mesmo poupa-se trabalho. Existem motores pagos e outros de iniciativa *open source*. Um exemplo de motor é o *Source Engine*, famoso por ter sido usado, segundo Santana (2006), para criação de jogos famosos, como *Half-Life* e *Counter-Strike*. Ainda segundo Santana (2006), outro bom exemplo de motor é o *Reality Engine*, que possui técnicas clássicas de IA, como o *pathfinding*. Exemplo de motores *open source* são o *OGRE*, *Delta3D* e *Blender*. O *OGRE* não é considerado um motor de jogo, mas pode e têm sido usado como tal. Foi criado para renderização 3D. Já o *Delta3D* é considerado um motor, e pode ser usado para motor de jogo, simulação ou outras aplicações gráficas. O *Blender* também é usado para gráficos em 3D, e possui um motor próprio. Alguns trabalhos costumam até mesmo usar esses tipos de motores em conjunto, como por exemplo, fazer o desenho de texturas e personagens 3D no *Blender* e posteriormente passá-lo para o *OGRE* para tratar do controle e ordem dos movimentos.

Além de utilizar motores prontos ou simplesmente não utilizar nenhum motor, há a possibilidade ainda da criação de um próprio motor. Dependendo do tamanho do projeto, isso pode-se tornar viável, pois pode acontecer de no mercado ou nas comunidades *open source* não existirem nenhum que atenda todos os requisitos, então, se o investimento for bem alto, pode-se criar um próprio. Para projetos não tão grandes não é aconselhável, pois gasta muito dinheiro e tempo, além do que um motor é uma peça fundamental.

O motor normalmente não faz o papel de rede, ou seja, comunicar o jogo através de máquinas diferentes. Isso é feito por um componente específico, a Rede.

3.4. Rede

A rede é responsável por comunicar dois ou mais jogadores que estão usando computadores (ou *console*) diferentes, para que possam atuar em um único jogo ao mesmo tempo, como, por exemplo duas pessoas jogarem de computadores diferentes uma única partida de futebol em *Fifa 2011*.

Essa opção de jogo em rede normalmente é chamada de *multiplayer*. Atualmente, quase todos os jogos lançados permitem o jogo via rede. Isso acontece porque essa forma de jogo traz uma interação muito forte, pois o jogador terá a oportunidade de jogar contra outros seres humanos. Essas outras pessoas podem estar no mesmo local, ou em casas diferentes, cidade e até mesmo países diferentes.

Para se jogar via rede existem muitas vertentes. O jogo um contra um é uma opção. Um jogador joga contra outro, apenas os dois. Isso é comumente utilizado em jogos de futebol. Muitos caracterizam *multiplayer* como jogo via rede com um número limitado de participantes. Outra opção é o MMO (*Massively Multiplayer Online Game*), onde vários jogadores de diversas partes do mundo jogam juntos. Esse tipo de jogo guarda informações em servidores próprios. Algumas vertentes desse tipo de jogo podem ser criados, como por exemplo rodar um jogo totalmente em um servidor único, e os jogadores competirem através de terminais burros. O MMORPG citado na seção História dos Jogos eletrônicos é um tipo de MMO.

Esses tipos de jogos chegam a criar até mesmo comunidades *online*. Para exemplificar, imagina-se um jogo de futebol. Pode ser criado então uma rede onde qualquer jogador de qualquer parte do mundo possa entrar para achar adversários para jogar esse jogo. Assim, classificação dos melhores colocados e pontuação do jogo podem ser criados. A série *Fifa* é um exemplo de jogo bem próximo disso, com a rede tendo várias outras funcionalidades. A rede citada poderia até mesmo gerar campeonatos de vez em quando.

Realmente o maior objetivo de se utilizar rede em jogos eletrônicos é o fato de jogadores diferentes poderem se conhecer, e até mesmo se desafiar. Mas, os NPC's não ficam fora disso. Isso pode ser observado no seguinte exemplo: imagine um jogo em rede, onde vários personagens podem ser controlados por diversas pessoas de todo o mundo. Um jogo inicia com duas pessoas querendo jogar, por exemplo. Além dos personagens delas, outros personagens vão continuar sendo controlados pelo jogo (NPC's). A responsabilidade desses NPC's então torna-se muito grande. Se uma pessoa joga contra outra, é responsabilidade

dos NPC's parecerem bastante com outros seres humanos, para que o jogo fique muito melhor.

Pensando em redes de computadores, e no crescimento dessa disciplina por todo o mundo, pode-se imaginar no futuro que jogos não serão mais vendidos em CD-ROM ou DVD-ROM, nem instalados em máquinas. Cada usuário poderá ter sua máquina conectando em um servidor, e portanto jogar a partir de lá. É o conceito de *cloud computing*, que não pode ser de maneira alguma descartado do ramo dos jogos eletrônicos.

A rede pode ser implementada em paralelo com o áudio, pois um não irá influenciar muito no outro. Os dois estarão no mesmo jogo, e um jogo em rede também possui áudio, mas isso não significa que não podem ser feitos separadamente.

3.5. Áudio

O áudio é o componente responsável por todos os sons emitidos durante o jogo. Desde os jogos mais antigos a utilização de áudio nos jogos é importante. Atualmente, é difícil de imaginar um jogo sem nenhum áudio.

Existem, basicamente, dois tipos de áudio em um jogo: trilha sonora e sons.

A trilha sonora, como a trilha sonora de filmes, é o conjunto de músicas ou trechos de músicas que podem ser tocados ao longo do jogo. Esse conjunto deve ser escolhido com bastante propriedade, analisando o gênero do jogo, o público alvo (que tipos de músicas o público gosta?), além das situações que um jogo pode apresentar. Quanto ao momento em que músicas podem tocar, há uma certa variação. No jogo *Fifa 2008*, por exemplo, enquanto o jogador navega pelos menus do jogo antes de iniciar uma partida, músicas de várias partes do mundo são tocadas. Quando o jogo inicia, não são mais tocadas músicas. Outro exemplo é o jogo *GTA: Vice City*, em que, quando o personagem principal entra em um veículo, ele pode acionar diferentes rádios, que então tocam músicas de diferentes ritmos musicais.

O outro tipo de áudio em um jogo, os sons, são até mais importantes que a trilha sonora, pois na maioria dos jogos estão presentes o tempo todo. O simples andar de um jogador emite som, como é o caso do *Counter Strike*. Além do andado do jogador e NPC's, qualquer outro tipo de interação de um personagem com o ambiente emite um som, seja quando ele quebra algum objeto, como uma cadeira, ou quando uma arma dispara em jogos de tiro. Como foi apresentado o *Fifa 2008* para exemplificar o uso de trilha sonora, o mesmo pode servir de exemplo para sons. Como já dito, quando uma partida inicia, as músicas param de tocar, e é nesse momento que entram os sons. Som do narrador no ato de narrar o jogo (em que as falas são pré-definidas, e aparentemente não usam IA), som de um chute forte na bola, e o som da torcida durante a partida.

Desde as fases anteriores, mais especificamente logo após o roteiro, já é definido como será a arquitetura do jogo.

3.6. Arquitetura

O termo arquitetura é muito amplo, e por isso pode ser usado para diversas definições em um jogo eletrônico. A arquitetura pode ser a organização dos componentes de software do jogo, ou seja, o desenho de como eles interagem entre si. Outra definição está relacionada à arquitetura de computadores, comumente chamada de plataformas. Por fim, pode-se dizer também da arquitetura de rede.

O certo é que a arquitetura de computadores deve ser muito bem analisada, pois trata da questão de plataformas. Um jogo normalmente é construído para rodar em determinada plataforma, ou mais de uma. Uma plataforma pode ser um computador ou até mesmo um *console*. Para as arquiteturas da família x86, os jogos são produzidos em grande escala, e essa plataforma é chamada comumente de PC. Quando se tratam de PC, outro assunto que pode ser revisado dentro da arquitetura é o sistema operacional. Vários jogos (e os mais famosos) são criados para sistemas operacionais da *Microsoft*, como o *Windows XP*, por exemplo. Para sistemas operacionais *Linux*, são poucos os jogos, ou pelo menos não fazem tanto

sucesso. O sistema operacional *Mac OS* também possui uma boa variedade de jogos.

Diferentes *consoles* também trazem diferentes jogos, com jogos exclusivos para um ou outro, ou jogos disponíveis para os dois. Alguns jogos, com o mesmo título, chegam a ter versões diferentes para diferentes personagens. Isso aconteceu com o *Mortal Kombat* de 2011, que possui um personagem de outra série de jogos. Como essa outra série de jogos é da *Sony*, o personagem está disponível apenas na versão para *PlayStation 3*. Para a versão *Xbox 360*, o mesmo jogo não possui esse personagem em particular.

Outro tipo de plataforma que possui vários jogos são os *videogames* portáteis. Muitos jogos são migrados de outras plataformas para eles, o que não impede que jogos sejam criados apenas para eles. Vale ressaltar também jogos para celular, mas nesse caso, normalmente os jogos são muito adaptados, pois esse tipo de dispositivo não possui muito poder de processamento como os computadores e *consoles*.

O que é realmente importante ressaltar é que a arquitetura deve ser algo muito bem tratado, pois ela definirá o esforço necessário para construção do jogo. Dependendo da arquitetura para a qual se deseja implementar um jogo, conhecimentos em linguagens de baixo nível são importantes (BALDANCE & REIS, 2006). A escolha da linguagem de programação a ser utilizada é então primordial nesse componente.

Quanto a outro tipo de arquitetura, a arquitetura de rede, trata-se da arquitetura e pilha de protocolos que serão necessários para que jogos possam se comunicar em rede.

A arquitetura não trata da IA em si, isso é feito de forma separada ou dentro do motor.

3.7. IA

Para Santana (2006), a inteligência artificial caminha junto com os jogos eletrônicos desde sempre, porém, os desenvolvedores não focaram nela durante

muito tempo. Isso é bem correto, pois desde os primeiros jogos eletrônicos, existiam oponentes para o jogador, e esses oponentes movimentavam-se na tela de diversas formas. Os padrões de movimento eram então considerados como a IA da época, mas não era realmente uma verdadeira IA.

Inteligência Artificial trata-se da ação racional por parte de um computador. Essa ação racional, que os seres humanos possuem, faz com que eles tomem decisões, por exemplo. Em um jogo eletrônico, espera-se que os personagens também sejam capazes de tomar decisões. Isto significa que um NPC irá fazer uma escolha baseado no que o jogador fizer momentos antes. Para qualquer ação do NPC (esperar, movimentar, etc.), ele deverá pensar de alguma forma para efetuá-la.

Muitos autores tratam a inteligência artificial, simplesmente chamada de IA, como as técnicas de inteligência artificial usadas na criação dos aliados e oponentes do jogador, ou seja, dos NPC. Essas técnicas são usadas exclusivamente em NPC, pois são os únicos elementos no jogo que agem nos cenários, portanto, os únicos que podem simular a inteligência humana.

A programação de IA é a parte mais difícil no desenvolvimento de um jogo, ao lado da renderização de imagens. Em muitos casos, são utilizados *scripts* para o desenvolvimento de IA, pois dessa maneira é possível alterar personagens de modo mais fácil (KISHIMOTO, 2004), e também não compromete o resto do desenvolvimento do jogo.

IA é um assunto tão poderoso que muitos dos jogos que vemos serem lançados até hoje não mostram NPC's com inteligência, fazendo movimentos pré-definidos.

A figura 3.1 mostra um modelo de IA em um jogo eletrônico. Os principais elementos contidos na figura, referentes a IA são Estratégia, Tomada de Decisão e Movimento. A estratégia é mais geral, de todos os NPC, é definida como o que eles irão fazer. Por exemplo, em um jogo de guerra entre civilizações, a estratégia seria como a tropa de uma civilização iria atacar a civilização do jogador. Ela está ligada diretamente aos personagens de IA, que são os NPC's. Esses, então, possuem dois elementos fundamentais: Tomada de Decisão e

Movimento. A tomada de decisão é uma escolha feita por um NPC, que pode ou não estar diretamente ligada a um movimento de um jogador. Por sua vez, um movimento é como o NPC irá se movimentar no ambiente do jogo. Fora da execução relacionada a IA, o NPC liga-se à física e animações. Isso acontece porque ele deve saber se um movimento é ou não possível, através da física (no que diz respeito ao tratamento de colisões) e de animações (ao fazer determinada escolha, o NPC pode ter que agir de forma específica naquele momento). Do lado esquerdo da figura há a Interface Geral, a qual se comunica com os elementos de IA diretamente, como apresentado pelas setas horizontais da figura. Normalmente isso é feito pelo motor do jogo. Por fim, do lado direito, há Criação de Conteúdo e *Scripts*, que está totalmente desligada dos outros componentes, pois são feitas separadamente e depois são adicionadas, como por exemplo, criar um determinado personagem posteriormente e depois apenas adicioná-lo ao restante.

Outro componente além da IA que pode ou não ser tratado dentro do motor é a física.

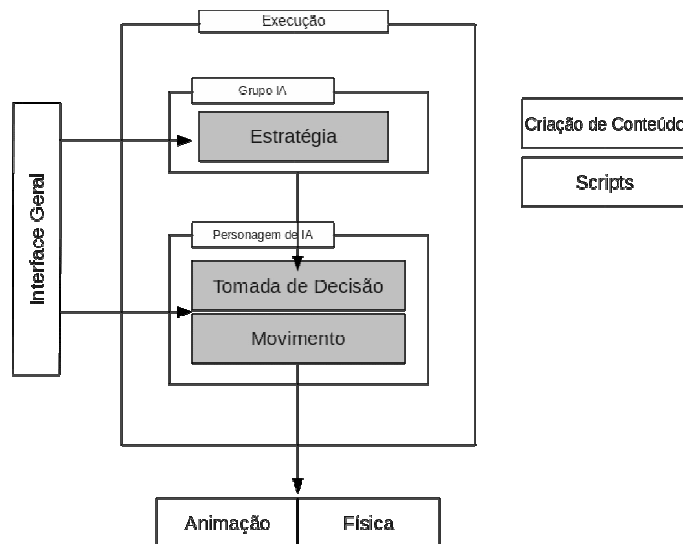


Figura 3.1 - Modelo de IA em um jogo (MILLINGTON & FUNGE, 2009)

3.8. Física

A física, e cálculos em geral, pode ser considerado um componente muito importante para diversos momentos da implementação de um jogo. Ela é tão importante que são encontrados até mesmo motores para cuidar exclusivamente da física.

Uma das aplicações de física em jogos está no tratamento de colisões. Isso é feito para o estabelecimento de limites. Por exemplo, um personagem não pode ser capaz de atravessar uma parede (a não ser que tenha poderes para isso). Do mesmo modo, não é tão trivial manter um personagem em cima do chão, e da mesma forma, o mesmo não pode afundar em um solo sólido.

Tabela 3.1 - Elementos mais comuns em um jogo eletrônico

Elemento	Breve descrição
<i>Game Design</i>	Ideia, desenho do jogo, definição de como ele será e como funcionará
Roteiro	Descrição da história do jogo, a história dos personagens, a época que o jogo se passa e o ambiente do mesmo
Motor (<i>engine</i>)	Renderização gráfica do jogo, e controle dos outros elementos
Rede	Componente para jogos em rede
Áudio	Todos os sons apresentados durante o jogo, além da trilha sonora
Arquitetura	Composição do software, plataformas e sistemas operacionais
IA	Técnicas de IA usadas para o desenvolvimento de NPC
Física	Cálculos físicos e matemáticos usados para tratamento de colisões, criação de motor, etc.

Outras aplicações de física e cálculos estão na própria renderização dos personagens e outros objetos. Caso não haja um motor, isso deve ser feito manualmente, e não é muito fácil. Também está presente quando deseja-se criar um motor próprio.

A tabela 3.1 apresenta uma visão geral dos elementos descritos anteriormente.

3.9. Criação e desenvolvimento de um oponente inteligente

Em jogos eletrônicos, os NPC (*Non-player Characters* ou Personagens Não Jogáveis) são aqueles personagens do jogo que não são controlados pelo jogador, isto é, a própria máquina é quem os controla. Basicamente, eles podem ser classificados em ajudantes ou oponentes. Os ajudantes são os NPC que ajudam o jogador (ou ajudam o personagem do jogador). Já os oponentes são aqueles que atrapalham os inimigos do jogador. Para exemplificar, pode-se imaginar um jogo de futebol, com apenas um jogador jogando. O jogador controla os seus jogadores, mas em determinadas situações, não controla o seu goleiro. Seu goleiro decide sozinho (isso quer dizer que o próprio programa decide) para qual lado irá pular, para defender a bola. Portanto, o goleiro é um tipo de NPC ajudante. De contrapartida, os jogadores do time adversário do jogador são seus adversários, e também NPC. A diferença é que eles são inimigos do jogador, aqui denominados então de oponentes.

3.9.1. Oponente inteligente

Um oponente inteligente em um jogo eletrônico, que será referenciado aqui apenas como oponente, é definido como um NPC (*Non-Player Character*) que irá atuar como inimigo do jogador (ser humano que está jogando). Como os jogadores atualmente exigem muito em jogabilidade, o estudo desses oponentes, e como eles são implementados em um programa de computador é importante, visto que são eles que ditam a dificuldade do jogo, e prendem o jogador à tela.

3.9.2. Inteligência Artificial e os NPC's

A IA usada em jogos eletrônicos está diretamente relacionada aos NPC's, pois são eles os únicos elementos de um jogo que possuem “vida”, ou seja, os únicos elementos que agem no ambiente de um jogo.

Diferente de outros campos da computação, a IA utilizada para implementar jogos eletrônicos difere-se no ponto de vista acadêmico e no ponto de vista comercial. Academicamente falando, o objetivo está em construir personagens que possam agir como seres humanos, simulando inteligência. Na área comercial, busca-se a jogabilidade, onde o usuário exige cada vez mais oponentes à altura. As frentes de pesquisa acadêmicas e comerciais diferem-se algumas vezes, sendo que especialistas colocam que as pesquisas comerciais estão bem mais avançadas do que as acadêmicas. (KISHIMOTO, 2004).

Vários cenários devem ser considerados ao criar-se oponentes inteligentes, como por exemplo, o quão eles devem ser inteligentes. Isso significa que um jogador quer um jogo difícil, mas que seja possível de ser ganho, afinal, ele não quer que a máquina derrote-o sempre. Para essa dificuldade ser alcançada, os NPC devem ser inteligentes, a ponto de criarem diversas situações diferentes ao jogador. Para essa inteligência ser alcançada, a IA deve ser trabalhada neles, de modo a equilibrar as habilidades do jogador com os NPC (GALDINO, 2007).

3.9.3. Agentes Autônomos

Outro quesito importante que um oponente inteligente deve ter é aquele que o define como agente autônomo. Isso significa que o oponente deve perceber o ambiente que o cerca e conseguir agir de forma autônoma para atingir seus objetivos (BORGES & BARREIRA & SOUZA, 2009). No começo do jogo, ele terá um conhecimento não muito grande, mas com o tempo ele deve aprender com os movimentos do jogador e dos outros agentes, sendo capaz de reconhecer uma situação que viveu antes e saber como agir de forma melhor. Observa-se que uma característica muito importante em um NPC é a movimentação, já que ele

aprenderá com os movimentos ao seu redor e efetuará seus movimentos da melhor forma possível. Atenta-se ao fato de que a movimentação está totalmente ligada ao controle de colisão, pois o ambiente que o NPC está possui diversos obstáculos, não podendo o mesmo escolher uma melhor tática, mas não conseguir efetuar-la, pois não conseguirá se mover da maneira de imaginou. Essa observação coloca a IA trabalhar diretamente com física.

Existem características diferentes que tornam um NPC um Agente Autônomo ou um Agente Emocional Hedonista (BORGES & BARREIRA & SOUZA, 2009).

Um NPC é considerado um Agente Autônomo se ele conseguir perceber o ambiente e operar de maneira autônoma, buscando sempre o melhor caminho para efetuar seus objetivos. Portanto, o ambiente é realmente um fator importante para esse tipo de NPC. Técnicas de IA como Redes Neurais são bastante interessantes para o uso desses agentes, pois eles têm um conhecimento limitado no início do jogo, e a partir do desenrolar do jogo e das operações feitas pelo jogador, ele vai adquirindo conhecimento. Pode-se então considerar o Agente Autônomo como um agente que busca suas decisões baseado na experiência, isto é, o que ele já passou e o que o ambiente/jogador fez e o influenciou.

Um NPC é considerado um Agente Emocional Hedonista se ele agir da seguinte forma: ele irá tomar decisões baseado em emoções, ou seja, agirá de forma que suas escolhas o tragam prazer e não dor. Dessa forma, um personagem que possua emoções consegue se parecer bastante com um jogador humano (BORGES, BARREIRA, SOUZA, 2009).

3.9.4. Operações de um NPC

Segundo Pozzer & Dreux & Feijó (2003), um NPC ganha vida através de um agente, e esse agente possui algumas operações, ou micro-operações. A tabela 3.2 apresenta um exemplo de um conjunto de micro-operações:

Tabela 3.2 - Conjunto de micro-operações de um agente (POZZER & DREUX & FELJÓ, 2003)

Micro-operação	Descrição	Especializações
Deslocar	Deslocar personagem de um local para outro	Voar, Caminhar Rastejar, Nadar
Lutar	Duelo entre personagens	Arma, Magia
Esperar	Ação inexistente	Descansar, dormir
Pegar/soltar	Capturar ou adquirir algo	Rapto, Roubo
Pedir/dar	Requisição de algo por meio de diálogo	Informação, Poderes, Objetos
Observar	Captação dos dados do ambiente próximos ao personagem	Procurar Ver Ouvir
Trabalhar	Ato de trabalhar	Doméstico Agrícola

3.10. Inteligência artificial nos jogos eletrônicos

Como definido na seção anterior, o *SpaceWar!* (1962) foi o primeiro jogo eletrônico da história. Segundo Schwab (2004), não havia nenhum tipo de IA real nele.

O segundo jogo importante relatado foi o *Computer Space* (1970), que era baseado no *SpaceWar!*, e também não possuía uma IA de verdade. O fato dos inimigos se movimentarem sozinhos na tela, sem outro jogador controlá-los, não quer dizer que são agentes inteligentes, pois foram apenas programados para movimentar-se em determinada direção. Quando tratam-se de agentes inteligentes, os personagens devem ser capazes de movimentar-se de acordo com o movimento do jogador, e até mesmo decidir qual a melhor opção naquele momento do jogo.

O jogo *Pong* (1972) foi provavelmente baseado no esporte *Ping Pong*, mas não possuía nenhum tipo de inteligência também, visto que era muito simples e talvez nem exigisse que a máquina fosse inteligente. Como se tratava apenas de conseguir acertar a bola e passá-la para o outro lado, não tinha realmente necessidade de movimentar de acordo com o que o jogador fazia.

Tank (1974) não trouxe nada de IA também, e nem mesmo *TV Basketball*. Mas esse último deve ser analisado, pois como se tratava de uma partida de basquete, influenciaria muitos outros jogos de esportes. Jogos de esporte atualmente realmente utilizam bastante IA, pois quando o jogador joga contra a máquina, esta deve fazer com que os personagens atuem de forma a dificultar a partida, de acordo com os movimentos do jogador. Em jogos de futebol, por exemplo, o jogador normalmente não controla o seu goleiro, ele atua de forma independente. A IA em cima desse tipo de personagem deve ser desenvolvida cuidadosamente.

Em *Qwak* (1974), o personagem pato deve fugir do jogador. Para isso seria interessante que o pato tivesse alguma forma de inteligência para cumprir sua tarefa. E tinha. Esse jogo utilizava padrões de movimento (GALDINO, 2007). Padrões de movimento é a técnica que faz com que um personagem se movimente através de um padrão pré-definido. Por exemplo, pode-se programar um personagem de forma que ele efetue movimentos circulares na tela. Muitos não consideram padrões de movimento como IA, mas pode-se considerar ao menos que é uma técnica que tenta simular algum tipo de inteligência.

GunFight (1975) foi um jogo em que os personagens possuíam movimentos aleatórios (SCHWAB, 2004). Movimentos aleatórios são fáceis de serem programados, basta obter um valor de forma randômica (aleatória) e incrementar a posição do personagem de acordo com esse valor. Alguns autores consideram movimentos aleatórios como um tipo de padrões de movimento também.

Alguns jogos não têm necessidade de possuírem IA, o que acontece até hoje, onde vários jogos não possuem nenhuma técnica de IA incorporada. O jogo *Breakout* (1976) era um que não tinha a mínima necessidade de implementação de

IA, pois o jogador era quem controlava a raquete que movimentava a bola. O que poderia ser considerado um “inimigo” do jogador eram os tijolos, e esses realmente não tinham nenhuma inteligência, afinal, não eram considerados agentes ou personagens.

Space Invaders (1978) usava também o recurso de padrões de movimento, e os inimigos atiravam contra o jogador. Isso é importante de ser ressaltado, pois em vários jogos os personagens inteligentes devem preocupar-se em movimentar de acordo com os movimentos do jogador, além de atacá-lo. Se o personagem está atacando o personagem do jogador, ele deve necessariamente saber onde o jogador está, e quando irá atacá-lo. É claro que a IA de *Space Invaders* não era tão forte assim, restrita à padrões de movimento.

Galaxian usa a fórmula do *Space Invaders*, mas usa vários movimentos de forma complexa. Os padrões de movimento, em nível de programação, podem ser interligados de diversas formas.

O *Pac-Man* também utilizava padrões de movimento, porém cada inimigo do jogador, no caso os fantasmas, tinham uma “personalidade” única, isto é, cada um usava movimentos e portanto caçava o jogador de maneira diferente. Segundo Kishimoto (2004), *Pac-Man* também utilizava uma Máquina de Estados para cada fantasma. Com essa técnica, um fantasma assumia diferentes estados: ele podia estar procurando pelo jogador, perseguindo ou fugindo. O estado era então alterado conforme o decorrer do jogo. Por exemplo, se o fantasma encontrava o jogador, ele decidia se devia persegui-lo ou fugir (determinados momentos do jogo era o jogador quem “comia” os fantasmas). Máquinas de estado, denominadas Máquinas de Estado Finitos seriam então bem utilizadas na década de 1990, e tratam prioritariamente do estado que os personagens assumem.

Berzek também não tinha indícios de IA, apesar da máquina falar, as vozes eram sintetizadas e pré-definidas.

Toda a década de 1980 ficou marcada pelos padrões de movimento. Em *Donkey Kong* não existia certo padrão de movimento, mas era bastante reduzido, já que o mesmo utilizava de plataformas, portanto cada inimigo movimentava-se limitadamente.

O jogo *Tempest* (1981) pode ser citado como tendo importância significativa, pois trouxe o conceito de dificuldade, e isso, em jogos mais atuais, pode estar diretamente relacionado com a programação de inteligência.

*Q*bert* (1982) não apresentava necessariamente nenhum tipo de IA, apesar de ser um jogo em que inteligência poderia ser bastante explorada. Dependendo de onde o jogador movesse seu personagem, a pirâmide poderia escolher quais cubos mudariam de cor, de forma a dificultar o jogo em momentos diferentes, por exemplo. Observa-se que alguns jogos da década de 1980 possuíam padrões de movimentos, mas outros não tinham nenhum tipo de IA, nem padrões de movimentos ou movimentos aleatórios.

Joust (1982), com seu conceito de colaboração, era um jogo que podia exigir bastante da máquina. Programação de IA gasta muito processamento, tanto que atualmente jogos com bons gráficos e IA boa são altamente dependentes de máquinas com bom processamento.

Super Mario Bros também não tinha a necessidade de personagens inteligentes, já que o movimento dos inimigos podia ser pré-definido.

A década de 1980 foi, portanto, forte pelos padrões de movimento, onde os movimentos seguem um padrão. Algoritmos de perseguição (tal como usado em *Pac-Man*) foram utilizados por alguns outros jogos, e funcionavam de maneira simples: buscavam o posicionamento do jogador e faziam com que o oponente se movimentasse até lá. Algoritmos de evasão detectavam onde estava o jogador, e faziam com que o oponente corresse dele, ou seja, o oponente distanciava o máximo possível do jogador (KISHIMOTO, 2004).

A década de 1990 ficou marcada pelo uso da Máquina de Estados Finitos e diversas técnicas consideradas realmente IA. Os padrões de movimento são colocados como um início de IA, mas não são necessariamente IA, pois os algoritmos realmente fazem com que os personagens se movam como seres humanos, mas não fazem com que eles pensem como tal, ou tomem decisões baseados em conhecimento adquirido. Dessa forma, apenas da década de 1990 para frente é que os jogos realmente usaram inteligência artificial.

Logo em 1990, o jogo *Herzog Wei* apresenta Máquina de Estados. Os jogadores depararam-se com um algoritmo de *pathfinding*, e esse algoritmo foi considerado muito ruim (GALDINO, 2007). O *pathfinding*, ou busca de caminhos, é um tipo de algoritmo em que é feita busca, ou seja, sair de um local e chegar a outro. Esse tipo de algoritmo, mesmo sendo básico na implementação de jogos, não é muito simples de se programar, dependendo da quantidade de inteligência que se deseje buscar. Se a busca não for muito boa, o personagem torna-se bastante artificial, ou seja, o jogador percebe facilmente que aquele personagem não é inteligente.

Jogos do início da década de 1990, como *Mortal Kombat* e *Street Fighter II* provavelmente possuíam algum tipo de inteligência, visto que o jogador podia jogar contra um NPC. Como foram jogos que se popularizaram bastante por um jogador contra outro, a IA (se foi usada neles) não será explorada aqui.

Virtua Racing (1992) ficou famoso pela simulação de múltiplos jogadores, também não trouxe nenhum avanço em IA. *Myst* (1993) impressionou pelo avanço gráfico, e não pela IA.

Em *Doom* (1993), há o uso de Máquina de Estados Finitos, usado com o mesmo intuito de outros jogos que usaram esse recurso (SCHWAB, 2004).

O *Sega Saturn* e o *Sony PlayStation* trouxeram capacidade dos *videogames* para renderizar gráficos melhores, além de processar algoritmos de IA, que como já dito, consomem muito em termos de CPU.

O *BattleCruiser: 3000AD* foi o primeiro jogo a utilizar Redes Neurais (SCHWAB, 2004), uma das técnicas da inteligência artificial. Redes Neurais é a técnica para se criar sistemas que simulam neurônios do cérebro humano. São estruturas capazes de adquirir, armazenar e utilizar conhecimento experimental. Redes neurais podem ser muito bem utilizada para jogos, apesar de que é uma técnica considerada isolada, e as mais utilizadas ainda são as consideradas tradicionais, como por exemplo a máquina de estados. Essas técnicas são mais exploradas no capítulo 4.

Half-Life, de 1998, foi analisado pelos críticos como o jogo com melhor IA em jogos até a época (KISHIMOTO, 2004). *Half-Life* usava *scripts* e máquina

de estados. Jogos com IA na forma de *scripts*, sugestivamente, são aqueles em que a IA é implementada com *scripts*, de uma forma isolado do jogo num todo. Mesmo não sendo uma técnica clássica dos estudos em inteligência artificial, a utilização de *scripts* é interessante, pois possibilita que novos NPC ou o aperfeiçoamento de um NPC já existente seja possível.

Observa-se que então a década de 2000 foi fortemente explorada em termos de IA. O jogo *The Sims* utiliza A-Life, que são aqueles algoritmos que simulam a vida. Antes do meio da década, o jogo *Doom 3* foi lançado, e usa a técnica de Máquina de Estados Finitos (LUZ, 2004). A IA usada em *Doom 3* certamente foi bem mais desenvolvida do que aquela utilizada em *Pac-Man* (1980).

Para a IA em geral, *Black & White* (2001) pode ser considerado um jogo onde a inteligência realmente foi implementada. Foi alvo da mídia no que diz respeito de como as criaturas do jogo aprendem com as decisões feitas pelo jogador. O jogo utilizava redes neurais, além de *reinforcement* e *observational learning* (GALDINO, 2007). *Reinforcement* pode ser definida como uma técnica onde as ações do agente no ambiente são influenciadas pelas consequências dessas ações. Já na técnica *observational learning*, o personagem toma decisões de acordo com o que ele observa ao seu redor.

The Sims e *Black & White* são exemplos de alguns dos jogos com as mais avançadas criaturas de IA (MILLINGTON & FUNGE, 2009).

No final da década 2000, observa-se claramente jogos com IA mais trabalhada. Em *FIFA 2008*, os jogadores do time adversário do jogador (oponentes) conseguem pensar melhor que em outros jogos de futebol. Por exemplo, em jogos mais antigos, se o jogador ficasse parado, o adversário não fazia nada. Em *FIFA 2008*, ao ficar parado, o time adversário faz o gol.

3.11. Conclusão

As diferentes técnicas utilizadas para o desenvolvimento de um jogo trabalham em conjunto, e para isso podem ou não serem desenvolvidas em

conjunto. Se forem feitas juntas, o motor será o responsável pelo controle do todo. Conclui-se que o desenvolvimento dos NPC's é importante e requer cuidados únicos. Para a criação desses NPC's, as técnicas de IA são muito usadas, e por isso há a necessidade de serem observadas separadamente.

4. TÉCNICAS DE IA PARA CRIAÇÃO DE JOGOS

Técnicas de IA das mais diversas foram utilizadas em jogos eletrônicos, como Máquina de Estados Finitos, Sistemas Baseados em Regras, Algoritmos de busca, etc. Atualmente, o mercado tem buscado técnicas mais clássicas para o desenvolvimento dos NPC's, como Redes Neurais, Algoritmos Genéticos e Lógica *Fuzzy*, e são essas técnicas que são enfatizadas neste capítulo. Além delas, um breve relato sobre Máquina de Estados Finitos é feito, pois foi uma técnica muito utilizada e que até hoje é recomendada para ser usada em qualquer jogo, pelo poder que possui e fácil entendimento e programação da mesma.

4.1. Redes Neurais Artificiais

Uma das técnicas mais impressionantes da IA é a RNA (Redes Neurais Artificiais), pois é uma técnica que busca simular o cérebro humano. Como IA trata-se da ação racional por parte de um computador (RUSSEL & NORVIG, 2003), conseguir fazer com que esse computador tenha um cérebro com a mesma topologia do cérebro humano torna-se uma implementação formidável.

A parte do cérebro que as RNA's simulam são os neurônios, que na maioria das implementações são divididos computacionalmente em camadas. Esses neurônios são ligados entre si, assim como os neurônios naturais. Particularmente aplicada a jogos eletrônicos, a RNA, mesmo não sendo muito usada até hoje, é uma técnica muito interessante, pois como trata de aprendizado, os NPC's podem adquirir novos conhecimentos ao longo do jogo, e até mesmo reconhecer padrões. Adquirir conhecimento é fundamental, pois dessa maneira um

NPC aprende alguma coisa que viveu no jogo, ou seja, alguma ação que o jogador efetuou. Quando esse jogador efetuar uma ação parecida, o NPC estará preparado, pois aprendeu com a experiência. Associado a isso, reconhecer padrões torna-se fundamental, como, por exemplo, padrões de movimento do jogador.

Diferente de outras técnicas, o NPC que usa RNA não é pré-programado com padrões de movimento. Ele espera a ação do jogador para somente assim decidir qual movimento fará.

Outra característica que é observada com o uso de RNA é a capacidade de desenvolvimento dos personagens ao longo do jogo, e essa técnica, portanto, pode ser muito bem trabalhada em conjunto com Algoritmos Genéticos ou outras técnicas que buscam simular sistemas biológicos. Para os jogos RPG (*Role Playing Game*), que são jogos nos quais os personagens e ambiente evoluem ao longo do tempo, essas técnicas em conjunto podem trazer um cenário muito divertido para ser jogado.

4.1.1. Cérebro e Neurônios

Para entender o funcionamento de uma RNA, é importante primeiro entender como seria uma RNN (Rede Neural Natural). Dessa forma, a formação do cérebro é algo que deve ser detalhado.

O cérebro é basicamente dividido em duas partes principais: direita e esquerda. A parte direita do cérebro controla a parte esquerda do corpo e a parte esquerda do cérebro controla a parte direita do corpo. Essas duas partes possuem algumas funções que são mais desenvolvidas em um lado, como o caso da linguagem, mais desenvolvida do lado esquerdo. As duas partes conversam entre si através do corpo caloso, que é um feixe de ligação entre elas. Já é de conhecimento que mesmo sem esse feixe as duas partes conseguem trabalhar independentes uma da outra (EDWARDS, 1984).

De forma mais detalhada, o cérebro é dividido entre as regiões: frontal, temporal, parietal e occipital, e seu funcionamento está diretamente ligado aos neurônios. Para a RNA, a simulação desses neurônios é o mais importante

(SCHNEIDER, 2001).

Um neurônio natural é capaz de conectar-se a dendritos e axônios. Os dendritos conseguem receber impulsos de células. No caso portanto das RNA's, os dendritos serão os neurônios responsáveis pela camada de entrada dos mesmo. Os axônios conseguem se comunicar com músculos, portanto serão encaixados no modelo de camadas como neurônios da camada de saída (SCHNEIDER, 2001). O modelo de camadas em RNA é explicado no tópico abaixo.

4.1.2. Modelo de Camadas em RNA

Para a implementação de um programa que simule uma rede neural, a melhor abordagem a ser utilizada é o modelo que pode ser chamado de modelo de camadas. Nesse modelo, os neurônios são dispostos em camadas diferentes, no mínimo três: camada de entrada, camada intermediária e camada de saída. São no mínimo 3 pois a camada intermediária pode ser dividida em várias camadas. Redes neurais podem ser implementadas até mesmo sem camadas intermediárias, ou com apenas uma ou duas, mas quanto maior o número resultados mais satisfatórios serão encontrados. Na camada intermediária pode-se dizer que ocorre a aplicação das regras de negócio do programa. A figura 4.1 apresenta como são as ligações entre os neurônios (SANTANA, 2006).

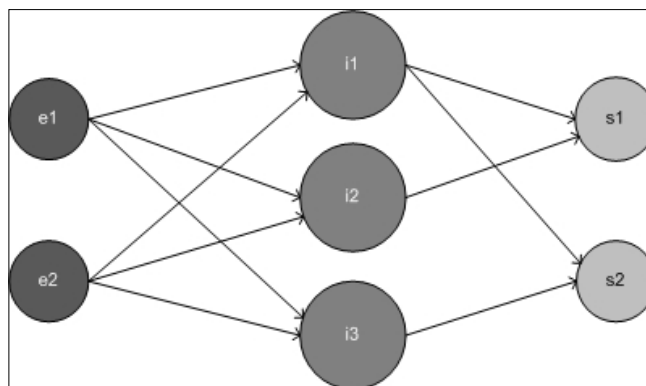


Figura 4.1 - Modelo básico de camadas em RNA.

Na Figura 4.1, os neurônios e1 e e2 são da camada de entrada, ou seja, recebem dados inicialmente. Posteriormente esses dados são passados à camada intermediária, onde como mostrado na figura 4.1, dados dos dois neurônios passam para todos os neurônios da segunda camada. Assim que a lógica do programa é efetuada, os resultados saem para a camada de saída (neurônio s1 e s2 na figura 4.1). Pode ser observado que a ligação dos neurônios é feita por camadas, ou seja, um neurônio da camada de entrada conversa com todos da camada intermediária, e todos desta conversam com todos da camada de saída.

Particularmente falando de NPC's, uma quantidade relevante de camadas intermediárias é interessante para ser utilizada, visto que quanto maior esse número, mais treinada a rede estará, e como busca-se a capacidade de aprendizado com experiência e de reconhecimento de padrões de movimento do jogador, o aprendizado que surge desse treinamento deve ser explorado o máximo possível.

O que ocorre antes dos dados passarem de uma camada para outra, ou seja, a lógica da técnica é explicada abaixo, no algoritmo.

4.1.3. Algoritmo

Diversos algoritmos para criação de RNA foram criados ao longo dos anos para serem utilizados nos mais diversos casos, destacando-se *Perceptron*, *Backpropagation* e *Madalaine*, dentre outros. Cada um possui diferenças dos outros, como por exemplo no *Backpropagation*, onde o erro (obtido na camada de saída) faz o percurso contrário, voltando para que a camada intermediária processe. É como se utilizasse a saída como nova entrada na rede. Para este trabalho, como o foco é a implementação de NPC's, que são inseridos em jogos eletrônicos, o *Backpropagation* é recomendado, pois treinar a rede exaustivamente até que ela assuma um estado perfeito é como estar treinando um NPC de forma que ele busque o estado mais perfeito possível que possa chegar no jogo. Além do *Backpropagation*, outro algoritmo que pode ser recomendado para o uso em jogos eletrônicos é o *Perceptron*, utilizado em um Jogo da Velha 3D (TRÉ, 2009).

4.1.3.1. Perceptron

Esse algoritmo tende a ser muito simples, pois é um exemplo clássico de redes neurais, além do que pode ser utilizado apenas com uma camada intermediária. Deve-se seguir os seguintes passos:

1. Inicializar os pesos;
2. Treinamento da rede;
3. Cálculo para saída de dados;
4. Jogar os dados dos neurônios de saída para os de entrada da rede.

Os pesos são valores respectivos a cada neurônio da camada de entrada, onde esse valor irá influenciar o dado a ser obtido na camada de saída. Para NPC's, esses pesos podem ser inicializados todos como zero, pois dessa maneira o NPC irá ser inserido em um ambiente totalmente desconhecido e o que o fará desenvolver é o treinamento. O treinamento em um *Perceptron* é passar cada neurônio pela função de ativação, que dará o valor da saída daquele neurônio. Logo após será efetuada uma comparação para ver se aquele é um valor esperado ou não, e se não for esses dados obtidos serão jogados na entrada novamente, até essa condição ser satisfeita ou os ciclos chegarem ao fim. Para cada programa diferente a quantidade máxima de ciclos é diferente, e cabe ao desenvolvedor avaliar qual será a quantidade máxima de vezes que a rede será treinada para chegar no melhor estado possível (SANTANA, 2006).

4.1.3.2. Backpropagation

Outro algoritmo clássico de RNA e um dos mais estudado atualmente é o *Backpropagation*, que como dito anteriormente, propaga o erro de volta para que a camada intermediária o processe, dessa forma sempre até uma condição satisfatória.

O fluxo desse algoritmo é apresentado pelas setas ERRO e Atividade na

figura 4.2. O fluxo segue a seta Atividade, e ao fim, o erro é propagado de volta para um novo fluxo.

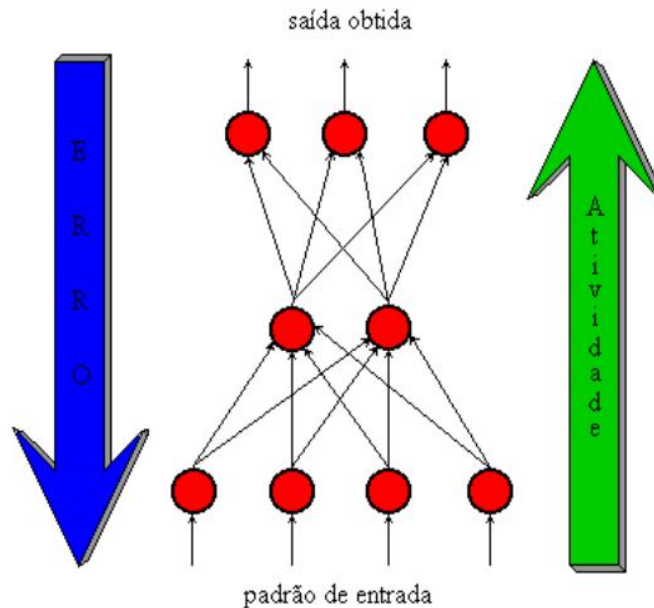


Figura 4.2 - Fluxo do Backpropagation (TONSIG, 2000)

O fluxograma da figura 4.3 apresenta como deve ser um algoritmo clássico de *Backpropagation*.

O primeiro passo no algoritmo é inicializar os pesos com valores randômicos, como mostrado no fluxograma da figura 4.3. Uma quantidade limitada de ciclos deve ser definida, ou épocas. Até os ciclos atingirem essa quantidade limite, deve ser feito o treinamento da rede, executar a rede para valores da camada de saída e fazer a retro propagação.

Treinar a rede é fazer um algoritmo que receba, de cada neurônio da camada de entrada, uma unidade x_i , chamada de sinal, passando esses sinais para a camada intermediária. Logo após, cada neurônio da camada intermediária deve fazer o somatório do sinal de entrada (considerando que cada unidade dessa possui um peso, que pode ser inicialmente pré determinado).

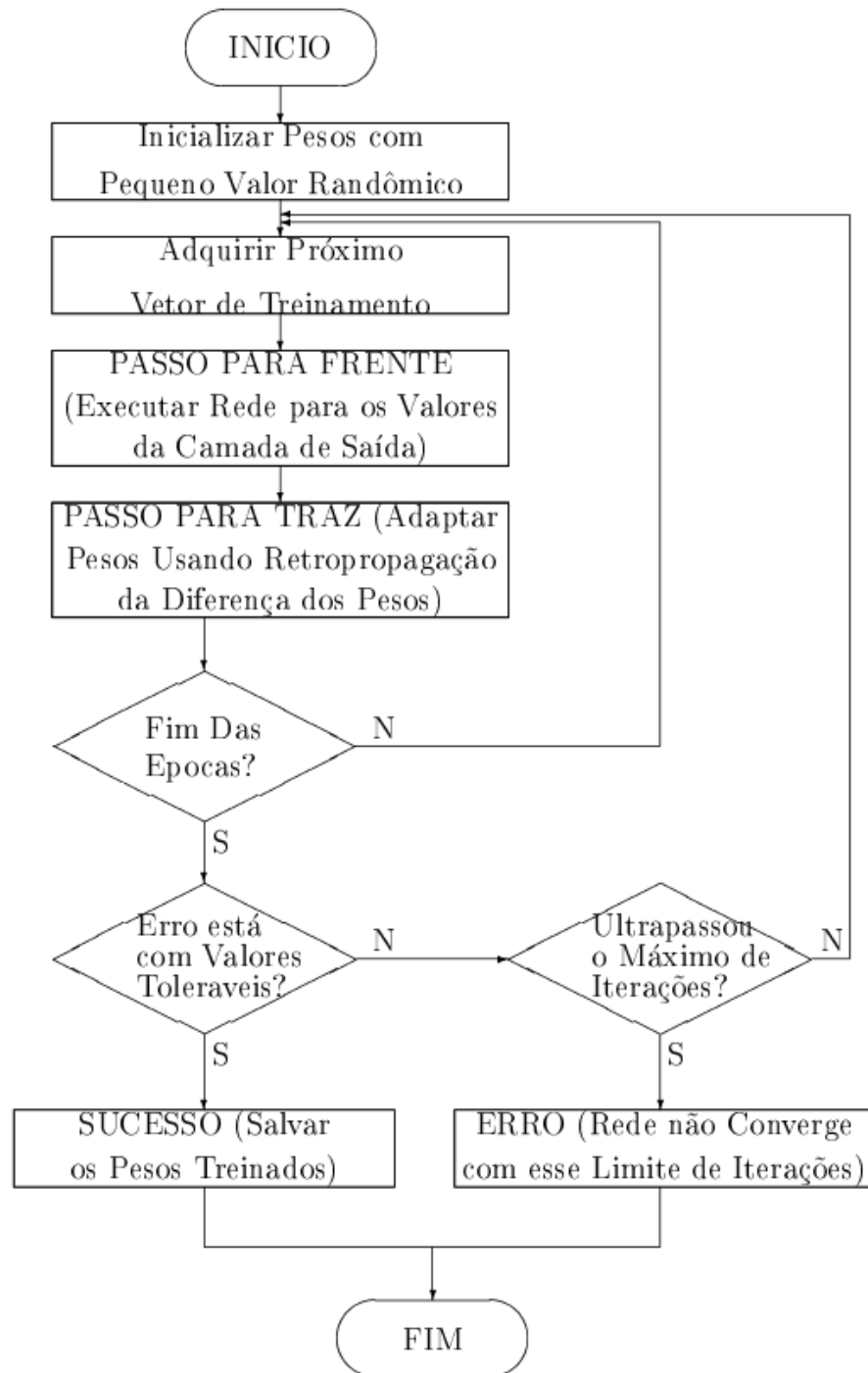


Figura 4.3 - Fluxograma Backpropagation (DAZZI, 1999)

Esse somatório pode ser dado pela fórmula abaixo (COSTA E SILVA,

2003):

$$\sum_{i=1}^n w_{ki}(n)x_i(n) + b_{kc}(n) \quad (\text{eq. 4.1})$$

Na fórmula acima, x é entrada da rede e w seu respectivo peso, e b é o bias associado ao neurônio k . Esses valores vêm dos neurônios da camada de entrada. Normalmente utiliza-se bias como 1, mas esse valor pode variar.

Após esse somatório deve ser aplicada a função de ativação. Essa função de ativação pode variar, dependendo da implementação. Duas que podem ser usadas são, da matemática, a função logística e a função tangente hiperbólica (CASTRO & CASTRO).

Logo após a ativação, o somatório acima e a ativação são repetidas, agora com a camada intermediária fazendo o papel de entrada e a camada de saída fazendo o processamento.

Após o processo acima, a rede terá uma saída (nota-se que foram passados os valores já pelas 3 camadas). Deve-se então calcular o erro, através da fórmula:

$$e_k(n) = d_k(n) - y_k(n) \quad (\text{eq. 4.2})$$

Na fórmula acima, e é o erro que será obtido, d o valor esperado, y o valor encontrado e k o neurônio atual ($k \leq n$).

O valor dos pesos então devem ser corrigidos, com a fórmula:

$$\Delta w_j = \alpha \delta z_j + \Delta w_j \quad (\text{eq. 4.3})$$

Na fórmula acima, Δw do lado esquerdo é o novo peso de cada neurônio (do lado direito é o peso atual), α é a constante de momento e δ a razão de aprendizado. Essa razão de aprendizado não deve ser demasiadamente grande, pois o algoritmo se torna instável, nem pequena demais, pois se não o algoritmo

não corrige os pesos de maneira satisfatória (CASTRO & CASTRO).

Além de corrigir todos os pesos, o peso do bias também deve ser alterado. Para isso, uma fórmula válida segue abaixo:

$$\Delta w_j = \alpha \delta + \mu \Delta w \quad (\text{eq. 4.4})$$

Neste momento, com os erros corrigidos, há efetivamente o que chamamos de *Backpropagation*, pois o erro definiu novos pesos que voltarão para novo treinamento, ou seja, retro propagação.

Agora a camada de saída serve como entrada para a camada intermediária, que faz a soma dos deltas dos pesos. Uma fórmula válida pode ser:

$$\delta_j = \sum_{k=1}^n \delta \Delta w_j \quad (\text{eq. 4.5})$$

Na fórmula, n é a quantidade de neurônios.

Cada valor obtido na fórmula acima é multiplicado por esse valor passado pela função de ativação.

Uma nova variação de erro deve ser calculada, e novos pesos para cada neurônio e para bias devem ser calculados. As funções anteriormente apresentadas podem ser reaproveitadas.

Agora, após todos esses passos, tem-se o momento Fim das Épocas no fluxograma da figura 4.3. Deve ser visto se os ciclos excederam o limite permitido (pré-definido).

Novos treinamentos podem ser feitos caso o erro não esteja com valores toleráveis, e iterações extras podem ser criadas, assim o algoritmo chega ao fim.

4.2. Algoritmos Genéticos

Para os jogos eletrônicos, uma técnica muito importante e talvez a que venha a ser mais utilizada, principalmente em jogos de RPG (*Role Playing*

Game), são os algoritmos genéticos, muito conhecidos também, ao menos o conceito, como computação evolucionária. Trata-se de uma técnica para simular o sistema genético, ou seja, da mesma maneira que um ser humano consegue evoluir, se adaptando aos diferentes meios que tem para sobreviver, o personagem no jogo irá evoluir e se adaptar ao ambiente do jogo e a sua interação com o jogador.

Analisando mais especificamente um jogo eletrônico, o jogador irá enfrentar NPC's, que são normalmente no início, mais fracos, e então é mais fácil do jogador conseguir derrotá-los. Dessa forma, novos NPC's vão surgindo assim que outros são derrotados, e devido aos genes conseguem evoluir, ou seja, ficarem mais fortes e mais difíceis para o jogador conseguir derrotá-los (LUZ, 2004). Na teoria da evolução ocorre o mesmo, onde os herdeiros de determinados pais pegam a carga genética desses pais, e se adaptam das diferentes maneiras, assim conseguindo o desenvolvimento, e portanto o algoritmo para simular isso muitas vezes é chamado de algoritmo de evolução

Algoritmos genéticos podem ser utilizados para a criação de populações inteiras em um jogo, ou até mesmo a mutação e evolução de personagens. Jogos de RPG usam, portanto, esse tipo de algoritmo pela própria definição do mesmo: um jogo onde o ambiente e os personagens nele contidos se desenvolvem ao longo do jogo. É notável como algoritmo genético é o mais recomendado para RPG, pois faz por definição o que RPG's precisam (KISHIMOTO, 2004).

O processo de evolução de um AG (Algoritmo Genético) ocorre quando há o cruzamento de indivíduos da mesma espécie, ou seja, são criados personagens com cromossomos pré-definidos e esses personagens são colocados no ambiente de maneira com que se cruzem com outros (BORGES & BARREIRA & SOUZA, 2009). Os cromossomos podem ser representados computacionalmente como um vetor de bits. Quando há um cruzamento, um novo personagem é criado, e a lei do mais forte prevalece. Personagens melhores sobrevivem, enquanto os mais fracos são retirados do ambiente. A cada iteração, esses personagens podem ou não receber mutação, que é a outra maneira dos cromossomos mudarem. Em jogos eletrônicos, os personagens mais fracos são

logo eliminados, pois o próprio jogador irá conseguir derrotá-lo.

Um AG não deve ser utilizado caso haja um método específico de se resolver um problema, mas no caso de jogos eletrônicos, sempre existirá um jogador, e os seres humanos sempre pensam de maneira diferente e jogam de maneira diferente. Dessa forma, é altamente recomendado o seu uso, já que o objetivo é criar cada NPC de forma que pareça muito com um ser humano, não apenas fisicamente (parte gráfica), mas também quanto ao comportamento, maneira com age no ambiente, ou seja, como se movimenta e como interage com o personagem que o jogador controla. Deve ser relatado também que esse tipo de técnica consome muito em termos computacionais, ou seja, necessita muito hardware (processador e memória) para ser realizado (TATAI, 2003).

Para implementar um AG, o fluxograma apresentado na figura 4.4 pode ser seguido.

A criação da população inicial normalmente é feita de forma aleatória, ou seja, criando um método no código fonte que gere randomicamente essa população. Para um jogo eletrônico, talvez a maneira como essa população seja iniciada deve mudar um pouco, onde para cada jogo diferente será criado populações diferentes. Na figura 4.4, é o passo inicial do fluxograma: gerar uma população. O nível de desenvolvimento dessa população deve ser avaliado antes, pois determinados jogos trazem personagens em uma época da história que eles não podem possuir tanta evolução, tanto inteligência. Para outros ocorre o inverso, personagens já devem ser criados totalmente desenvolvidos, ou até mesmo mais evoluídos que o presente, como em casos de jogos que se passam no futuro.

O cálculo da aptidão é feito individualmente, ou seja, cada indivíduo deverá ter uma aptidão. Essa aptidão é qual a capacidade que determinado indivíduo tem de ser selecionado para se reproduzir. Essa aptidão é calculada usando uma função específica no código fonte, e essa função varia de programa para programa. No caso de jogos eletrônicos, uma função que busque a capacidade de um NPC' conseguir adquirir algo novo talvez seja a ideal. Exemplificando, é interessante que em um jogo onde NPC's e personagens do jogador lutam, os NPC's recebem a aptidão vinculada a capacidade que cada um

deles tem de usar uma nova arma ou até mesmo determinado poder, em jogos fictícios.



Figura 4.4 - Fluxograma de um AG (FILITTO, 2008)

Após a determinação da aptidão de cada indivíduo, é criado um algoritmo clássico de AG: a roleta (Figura 4.5), que irá determinar quem serão os pais da população, ou seja, indivíduos que irão cruzar entre si. Esse é o segundo passo no fluxograma da figura 4.4, seleção. É usado um algoritmo que crie uma roleta, pois os pais deverão ser sorteados. Para isso cada indivíduo possui uma aptidão diferente, e a roleta deverá levar isso em conta, isto é, indivíduos com maior aptidão deverão ter maiores chances de serem escolhidos, e assim analogamente, indivíduos devem ter chances de serem escolhidos

proporcionalmente à sua aptidão.

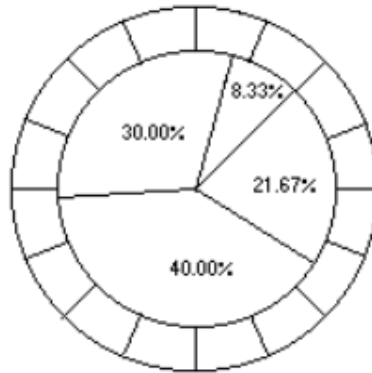


Figura 4.5 - A roleta em um AG (SOARES, 1997)

A figura 4.5 exemplifica bem como seria uma roleta. No caso, são 4 indivíduos, onde a aptidão de cada um deles determina que terão 40%, 30%, 21,67% e 8,33% de serem escolhidos. Como pode ser observado, a chance da roleta ser girada e parar em uma posição do indivíduo de 40% é bem maior do que parar no de 8,33%.

Um algoritmo para roleta é apresentado abaixo:

```
int roleta(int aux)
{
    int somatorio;
    somatorio = 0;
    for(i=0; i<N; i++)
    {
        somatorio = somatorio + apP[i];
        if(somatorio >= aux)
            return pais[i];
    }
}
```

Figura 4.6 - Algoritmo da roleta

No algoritmo acima, o método roleta recebe uma variável 'aux' que foi

gerada de modo aleatório e um somatório ('somatorio') é feito com as aptidões ('apP') de cada indivíduo da população. Desse modo, quando o somatório for maior que 'aux', o método retorna aquele indivíduo, ou seja, ele foi escolhido ou selecionado como reprodutor. O laço *for* percorrendo o vetor de aptidões dos indivíduos e parando quando for maior garante a roleta, isto é, quem tiver aptidão maior terá mais chances de ser retornado.

Quando a roleta decidir quem serão os pais, deve haver a geração dos filhos. Determina-se, então, quais pais irão cruzar entre si, gerando um ou mais filhos. Um filho é gerado de maneira que sua carga genética tenha similaridade com a carga genética de seus pais, tanto pai quanto mãe, e para isso poderão ser determinadas regras diferentes. Para cada programa, utilizar a mesma quantidade de cromossomos do pai e da mãe ou resolver como serão pego esses cromossomos pode variar. Esse é o passo Cruzamento apresentado no fluxograma da figura 4.4.

Para exemplificar, imagine dois cromossomos abaixo, de pai e mãe:

Pai: 110010010100

Mãe: 110001010010

Supondo que o algoritmo irá gerar dois filhos para cada casal de pais, poderia ser definido que um filho é a junção dos 6 primeiros bits do pai e dos 6 últimos da mãe, e o outro filho o contrário. Portanto, pai e mãe seriam divididos da seguinte maneira:

Pai: 110010|010100

Mãe: 110001|010010

E juntando os bits iniciais do pai e finais da mãe, seria gerado o filho 1, e juntando os bits iniciais da mãe e finais do pai, seria gerado o filho 2, como apresentado abaixo:

Filho 1: 110010010010

Filho 2: 110001010100

O interessante do cruzamento em AG's é a liberdade como isso pode ser feito. Pode ser seguido o exemplo acima, ou para cada caso determinar como isso será feito, quantos filhos serão gerados, quais bits do pai e quais da mãe serão utilizados, e em qual situação, e assim por diante.

Um filho gerado será incorporado à população já existente, e então toda a população irá sofrer mutação. Dependendo de uma função qualquer, um indivíduo pode ou não sofrer essa mutação. A mutação é o passo 5 no fluxograma da figura 4.4. É muito difícil um indivíduo receber mutação, portanto o algoritmo deverá garantir que essa mutação ocorra raramente, em poucos indivíduos.

Como cada cromossomo de um indivíduo é representado por um vetor de bits (0 ou 1), para a mutação imagina-se, por exemplo, um filho com a seguinte carga:

1101111000011110

O algoritmo que está sendo usado deve varrer esse vetor e de algum modo decidir se irá ou não haver mutação em cada posição. Esse algoritmo deve garantir que a mutação ocorra apenas pouquíssimas vezes na evolução.

Supondo que o bit da posição 3 foi escolhido para ser mutado, ele trocaria de bit, e portanto esse cromossomo ficaria da seguinte maneira:

1100111000011110

O zero em destaque é o bit que sofreu mutação.

Por fim, uma nova população deverá ser gerada, ou seja, pegar os melhores (geneticamente falando) e eliminar os piores. A condição do fluxograma da figura 4.4 busca isso. Fim da busca significa que a população chegou em um

estado final. Se ela não chegar, tudo volta à reprodução e o algoritmo segue o fluxo apresentado novamente, até encontrar uma população ideal. Em um jogo, isso não é muito difícil, pois quando um indivíduo fraco é colocado no ambiente do jogo, o próprio jogador trata de eliminá-lo. Mesmo assim requer alguns cuidados em casos específicos, como jogos de corrida que possam eventualmente usar AG para o chamado *tunning* dos carros. Alguns jogos de corrida usam AG para desenvolver carros que possam combinar com outros, gerando performance melhor (SANTANA, 2006). Nesse tipo de jogo, cabe ao próprio programa eliminar carros sem características boas para combinação, e não ao jogador.

Um AG não está apenas direcionado ao NPC em si, mas também ao que ele faz, por exemplo, trajetória que o mesmo possui. Um robô utiliza AG para identificar o trajeto que fará (PIRES, 2008). Da mesma forma, um NPC poderá identificar o melhor caminho através de AG, ele é considerado como se fosse um robô dentro do jogo, por exemplo. A trajetória não será pré-definida, e o NPC irá ser capaz de desenvolver trajetos diferentes e que evoluem quando um trajeto anterior for utilizado.

Algoritmos Genéticos podem ser utilizados para qualquer tipo de cenário, mas é interessante analisar algumas coisas antes de utilizá-lo, como por exemplo, se uma aptidão para determinado problema é possível de ser criada ou se as soluções possíveis estão em uma faixa (BRUN).

4.3. Lógica *Fuzzy*

A Lógica *Fuzzy* é outra técnica de IA usada em diversos ramos da computação, e que pode ser perfeitamente utilizada em jogos eletrônicos. Trata-se de uma técnica que consegue tomar decisões parecidas com a de seres humanos. Um ser humano toma uma decisão muitas vezes baseado no que ele acha, e não em fatos comprovados por experimento. Por exemplo, imagine um homem perguntando a uma mulher se vai chover amanhã. Ela pode responder que sim, mas não avaliar com os equipamentos necessários para ter certeza sobre isso. Portanto, ela acha que vai chover no dia de amanhã. Várias coisas influenciaram a

decisão da resposta dela, como a época do ano (propicia a chuvas, por exemplo) ou aquela semana ter feito muito calor. Esse exemplo mostra claramente como funciona um ser humano em termos de tomada de decisões. Não é sempre que uma decisão é tomada estando totalmente experimentada, testada e com embasamento teórico e prático suficientes. Na verdade, na maioria das vezes as pessoas respondem aquilo que elas acham, e o achar não é relacionado somente a fatos, mas opinião dentre outras coisas influencia.

Para entender onde se encontra a lógica *Fuzzy*, o diagrama da figura 4.6 mostra a ramificação das lógicas derivadas da lógica propriamente dita:

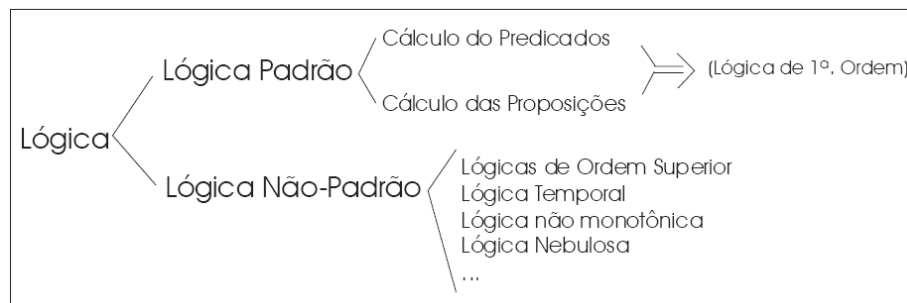


Figura 4.7 - Lógica Contemporânea (da SILVA, 2008)

Na figura 4.7, a Lógica Nebulosa é a lógica *Fuzzy*, ou seja, não está no que é chamado de lógica padrão, estudada na computação como lógica para ciência da computação.

Durante bastante tempo, técnicas para implementar IA em NPC foram baseadas em FSM (*Finite-State Machine*) e Sistemas Baseados em regras. Esse tipo de técnica necessita de respostas exatas, ou seja, um NPC irá decidir o que fazer pensando no que pode fazer e resolver sim ou não. Por exemplo, em um jogo de confronto com armas, se o NPC quiser saber se vai atirar ou não no personagem do jogador naquele momento ele irá analisar e decidirá como na lógica clássica, verdadeiro ou falso, ou seja, sim ou não, irá atirar naquele momento ou não. Sistemas Baseados em Regras será dessa forma, pois todas as regras já estão pré-definidas. FSM também, pois um NPC decidirá se irá ou não

mudar de estado, decisão sim ou não. Já na Lógica *Fuzzy* acontece diferente, uma decisão de um NPC não será necessariamente sim ou não, ele irá analisar outras coisas, o que ele acha que deve ou não fazer (BORGES, BARREIRA & SOUZA, 2009).

O fato de uma NPC achar algo já o torna inteligente, pois achar já é um ato de pensar, e se ele consegue pensar, ele possui certo nível de inteligência. O que irá influenciar a decisão de um NPC baseado no que ele acha são fatores do ambiente que ele está envolvido, e até mesmo no que o jogador pensa ou no que os outros NPC's estão fazendo naquele momento. É interessante ressaltar que um NPC trabalha em conjunto com outros NPC's, e juntos formam o que chamamos de estratégia.

Imagine um jogo de corrida, onde a velocidade de um carro guiado por um NPC deve ser sempre levada em consideração, para ele fazer curvas, por exemplo, ele deve reduzir se sua velocidade for alta. Considerando esse exemplo e a lógica booleana, teríamos o gráfico de Velocidade X Pertinência representado na figura 4.8.

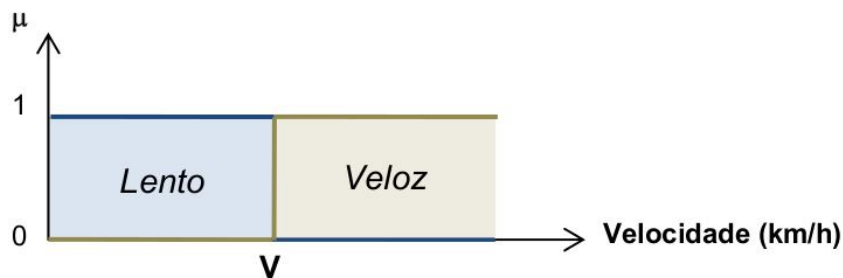


Figura 4.8 - Velocidade X Pertinência em lógica booleana (da SILVA, 2008)

Na figura 4.8, μ é a pertinência, ou seja, na lógica booleana, um carro de um NPC pode estar lento ou veloz, portanto a velocidade lenta pode ser apenas 1 até certo ponto do eixo velocidade, e a veloz pode ser apenas 1 de certo ponto pra frente.

Diferente da lógica booleana, a lógica *Fuzzy* irá tratar a velocidade no

caso acima de modo diferente. Uma pessoa pode considerar 100 km/h rápido, já outra pode considerar lento, mas é quase certo que elas consideram 10 km/h lento, e 140 km/h veloz. A figura 4.9 representa como o gráfico Velocidade X Pertinência seria representado na lógica *Fuzzy*.

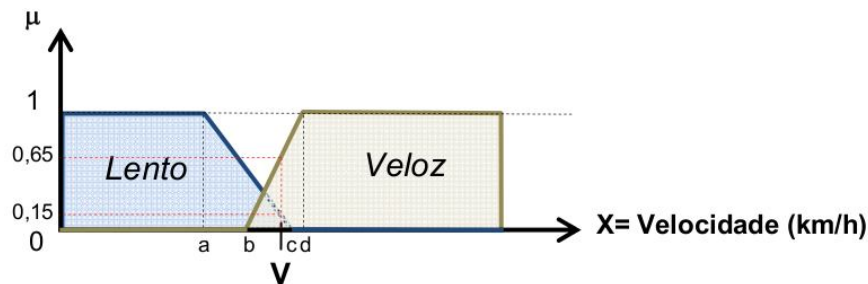


Figura 4.9 - Velocidade X Pertinência em lógica Fuzzy (da SILVA, 2008)

Na figura 4.9, a partir do ponto a no eixo velocidade, um carro está lento, mas já começando a ficar rápido, e muitos o já consideram rápido antes do ponto c. Do ponto b ao ponto c tem-se a interseção do conjunto, e é nesse ponto que a lógica *Fuzzy* trabalha com exatidão.

A Lógica *Fuzzy* devido a sua grande gama de aplicações é utilizada em grande escala para as mais diversas aplicações e nos mais variados ramos da indústria, tornando-se inclusive uma técnica padrão quando se fala em IA (SOARES, 2005).

Para o uso específico em jogos eletrônicos, o mais interessante é a chamada FFSM (*Fuzzy Finite-State Machine*), que é uma evolução da FSM (*Finite-State Machine*). Essa última técnica trata-se de uma implementação em que o NPC irá possuir estados diferentes. Por exemplo, em um jogo de batalhas, um NPC pode estar no estado de perseguir um personagem do jogador ou no estado de correr do jogador, dependendo da situação. Esse estado pode portanto variar. A FFSM é uma técnica que desenvolve a Máquina de Estados Finitos, acrescentando a mesma a lógica *Fuzzy*. A FFSM é interessante para ser usada, pois muitos jogos utilizam as FSM (TATAI, 2003).

Para a implementação da Lógica *Fuzzy* os passos abaixo devem ser seguidos:

- Fuzzificação;
- Inferência;
- Defuzzificação.

Exemplificando, a figura 4.10 mostra como os passos acima são utilizados:

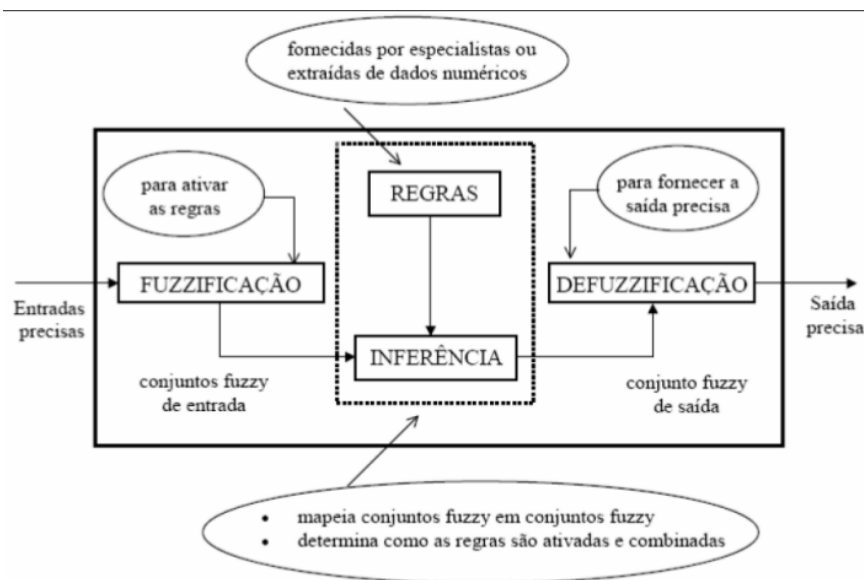


Figura 4.10 - Diagrama Lógica Fuzzy (da SILVA, 2008)

Na fuzzificação, o desenvolvedor deve entender o problema proposto e decidir as variáveis lingüísticas e as regiões. Variáveis lingüísticas são aquelas variáveis que podem mudar. Por exemplo, a variável “chover” pode possuir variáveis lingüísticas tais como “poderá chover”, “irá chover”, “talvez chova”, “certamente choverá”, dentre outras. Essas variáveis lingüísticas é que tornarão o NPC capaz de achar algo. As regiões são os extremos, ou seja, até onde uma variável poderá ou não variar.

Na inferência ocorrerá a lógica do código, ou seja, onde haverá a definição da região final, de que ponto a que ponto haverá variação, pois na fuzzificação ficaram vários pontos podendo ser início ou fim de um intervalo. Ao final da inferência todos os intervalos deverão estar especificados e sem dúvidas.

O processo final, de defuzzificação é onde o resultado obtido na fuzzificação é colocado na forma de dados que o NPC consiga entender.

4.4. Máquina de Estados Finitos

Uma técnica que muitas vezes não é colocada como IA e outras vezes sim, é a Máquina de Estados Finitos, ou FSM. As FSM's foram utilizadas em muitos jogos na história, e diferentes de técnicas como RNA, AG e *Fuzzy*, são bem simples de serem implementadas. Dentre jogos que utilizaram essa técnica pode-se destacar o *Doom3* (LUZ, 2004). O jogo Pac-Man como dito no capítulo 2 também possui traços fortes de utilização de FSM.

Por definição, essa técnica trata de um algoritmo onde o estado do NPC pode alterar. O estado é nada mais do que o que ele sente no momento. Por exemplo, em determinado momento do jogo o NPC está em um estado de ataque, em outro em estado de fuga e assim por diante.

Para Santana (2006), não existe muita dificuldade em programar um algoritmo de FSM. Apenas devem existir funções que validam o estado do personagem, ou seja, mudam seu estado de acordo com o que está acontecendo. Esse tipo de função trabalha através de flags, que podem ser variáveis que guardam o estado do NPC. Dependendo do parâmetro que essa função recebe e da flag no momento, essa flag poderá ser alterada.

A figura 4.11 exemplifica como é um diagrama de estados em um jogo eletrônico. Observa-se que o NPC está no estado parado inicialmente e pode passar para um estado de perseguição do inimigo. Deste voltar a ficar parado ou atacar o inimigo, ou até mesmo regenerar, e assim várias outras transições podem acontecer, como mostrado.

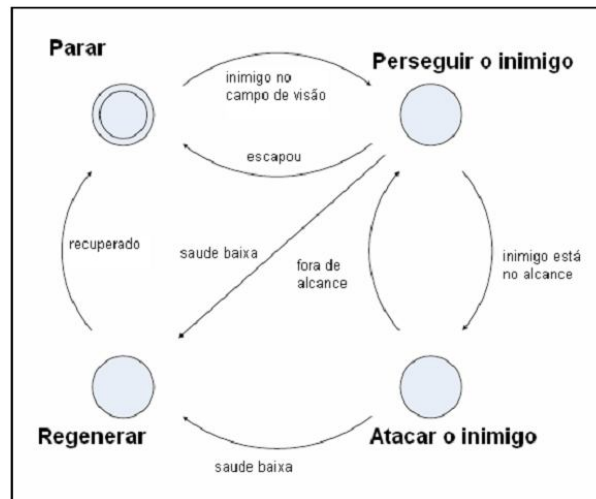


Figura 4.11 - Exemplo estados em um jogo (GALDINO, 2007)

4.5. Conclusão

A utilização de RNA, AG e Lógica *Fuzzy* pode (e é recomendado) serem utilizadas em conjunto, para que possa existir um jogo eletrônico mais aprimorado. Não foi discutido aqui o poder computacional (em termos de hardware) necessários para isso, dependendo do tamanho do jogo.

A melhor forma de garantir aprendizado é com uma RNA, e por isso é aconselhável utilizá-la para que os NPC's do jogo tenham um cérebro e possam agir de forma inteligente. Junto a esse cérebro, é interessante colocar algoritmos de lógica *Fuzzy*, pois são fundamentais na hora de tomada de decisão, e foram criados justamente para tomar decisões, para que o NPC possa pensar. Para pensar nada mais aceitável que ele tenha um cérebro e seja capaz de aprender e de interpretar certos fatos com base no que ele acha que acontecerá ao ter determinada escolha.

Não fora do escopo como um todo, o AG é fundamental para a estratégia, e praticamente todo jogo possui uma estratégia, tanto de combate, corrida, esporte, etc. Essa estratégia é garantida com o trabalho conjunto dos NPC's. Como vários NPC's trabalham em equipe consideramos que são uma população, e AG é

realmente feito para o desenvolvimento e evolução de populações. Combinar AG com RNA e lógica *Fuzzy* é uma forma de fazer com que os indivíduos sejam inteligentes do ponto de vista cerebral, ou seja, pensar e guardar informações de experiências que viveram, e conseguir interagir da melhor forma possível com o ambiente à sua volta. A FSM pode entrar em qualquer momento do jogo, pois irá apenas praticamente garantir o estado do NPC naquele momento.

O próximo capítulo irá mostrar a implementação de um jogo eletrônico usando em diversos momentos as técnicas aqui descritas, dando ênfase para uma ou outra dependendo do progresso no desenvolvimento.

5. ESTUDO DE CASO - JOGO DE RPG COM ALGORITMO GENÉTICO E MÁQUINA DE ESTADOS FINITOS

O código completo para esse estudo de caso pode ser encontrado em <http://code.google.com/p/natalvsn-monografia-cap5>.

Um estudo de caso que pode mostrar bem algumas das técnicas de IA é a implementação de jogos de RPG. Como já explicado neste trabalho, em um RPG os personagens e o próprio ambiente se desenvolvem ao longo do tempo de jogo. Os personagens começam fracos, e vão ganhando mais poder ao longo do jogo. O ambiente é desenvolvido através dos cenários que podem ir alterando para ficarem mais difíceis para o jogador ou até mesmo a população de inimigos, que cresce consideravelmente ao longo das fases de um jogo e torna-se mais poderosa, e portanto mais difícil de ser derrotada.

Para um RPG o algoritmo mais recomendado é o Algoritmo Genético, pela própria definição do mesmo. Um algoritmo que pode fazer com que uma população de indivíduos evolua ao longo do tempo pode perfeitamente ser transposto a um jogo onde os inimigos do jogador farão parte dessa população. Cada fase do jogo pode ser considerada um ou vários novos ciclos, onde os indivíduos são desenvolvidos e ganham uma aptidão maior.

Outra técnica que foi muito utilizada em jogos e não pode ser descartada é a Máquina de Estados Finitos. Em jogos de RPG normalmente os inimigos vão atacar o personagem do jogador ou fugir do jogador. A primeira ação seria um estado de perseguição e a segunda de fuga. Mesmo em jogos bem simples é

importante o uso de estados nos NPC's.

Esse estudo de caso tem como objetivo apresentar como foi desenvolvido o estudo de caso para essa monografia, um jogo eletrônico do tipo RPG que usa as técnicas Algoritmo Genético e Máquina de Estados Finitos, apresentadas no capítulo 4. O jogo é graficamente construído em 2D, numa tela de 800x600 pixels. Esse tamanho de tela é utilizado para que o jogo execute perfeitamente até mesmo em monitores um pouco mais antigos.

5.1. Roteiro do Jogo

Esse estudo de caso trata de um jogo que demonstra o uso de Algoritmo Genético com Máquina de Estados Finitos, portanto o roteiro não é tão extenso como em um jogo de RPG que fosse lançado no mercado, por exemplo, que deveria ter uma história por trás para agradar os jogadores, além de cenários e personagens bem detalhados.

O roteiro nesse estudo de caso apenas especifica em texto como é o jogo, quem são inimigos, o que o jogador controla como personagem, etc.

5.1.1. Ambiente

Para o cenário do jogo é utilizado um terreno plano, como um tabuleiro de jogos clássicos do tipo dama ou xadrez. Nesse tabuleiro, os personagens, que podem ser imaginados como personagem do jogador e inimigos, ficarão transitando pelo terreno.

O terreno possuirá tamanho de 800 pixels de largura por 600 pixels de altura, podendo assim ser visto em toda a sua extensão pelos monitores mais usados no mercado.

A quantidade de personagens que podem ser vistos no terreno ao mesmo tempo pode variar, não sendo descrito nesse roteiro um limite para isso. Mais especificamente na parte da implementação o número é especificado, mas para o roteiro isso não é importante, pois não foi utilizado como algo que atrapalhasse o

desenvolvimento do jogo.

A cor do terreno é preta, e os personagens que ficarão sobre o mesmo possuirão as cores azul (para o personagem controlado pelo jogador) e vermelha (para os inimigos do jogador), além de branca (para os tiros que o jogador pode efetuar).

5.1.2. Personagens

Nesse estudo de caso existem apenas dois povoados, o do jogador e o que é controlado pelo computador. A equipe do jogador possui apenas um personagem, o qual é dotado de tiros, ou seja, pode o jogador pode atirar contra seus inimigos. A equipe dos inimigos é dotada de n personagens, onde n pode variar dependendo da fase. A utilização de apenas duas equipes é suficiente para mostrar o rendimento do AG e também para utilizar a FSM.

Cada inimigo do jogador será mostrado na tela em forma de círculo, preenchido na cor vermelha. O tamanho do círculo poderá variar, e esse valor é calculado pelo AG. O personagem do jogador será mostrado na tela em forma de quadrado, preenchido na cor azul. O tamanho do quadrado é de 10 pixels de altura por 10 pixels de largura. Além do quadrado do personagem do jogador e dos círculos dos inimigos, outros objetos que poderão ser vistos na tela são os tiros que o jogador efetua contra seus inimigos. Elas são apresentadas na cor branca, sem preenchimento.

A definição de mostrar os guerreiros na tela apenas em forma de círculos e quadrado se dá pela facilidade de desenhar esses objetos, e como o foco do trabalho não envolve computação gráfica, a forma como eles serão mostrados não interfere no uso de IA. O único item necessário é que o jogador consiga diferenciar sua equipe da adversária, e isso se dá com as cores diferentes e o próprio formato dos personagens dos inimigos e do personagem do jogador.

A figura 5.1 mostra uma tela do jogo, na fase 1. A fase é mostrada do lado esquerdo superior da tela, em cor verde. A palavra fase aparece em todas as fases, seguidas do número referente àquela fase.

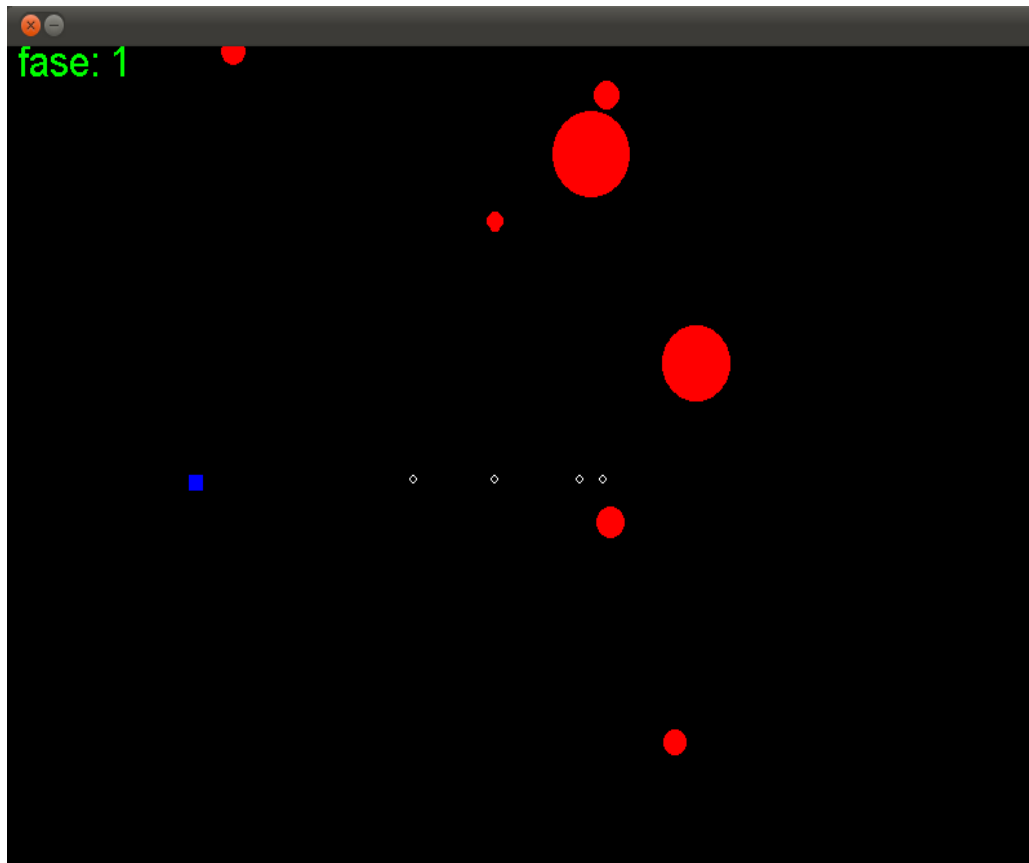


Figura 5.1 - Tela da Fase 1 do Estudo de Caso

Como especificado nesse roteiro, a figura 5.1 apresenta uma tela do jogo implementado. O quadrado azul é o personagem que o jogador está controlando, os círculos vermelhos são os inimigos do jogador e os círculos desenhados em branco sem preenchimento são os tiros que o jogador efetuou contra seus inimigos.

5.1.3. Sequência dos acontecimentos no jogo

Quando o jogo inicia, o personagem do jogador está no centro do terreno, ou seja, na posição 400 x 300 pixels. O jogo sempre inicia na fase 1. Os inimigos surgem ao mesmo tempo do início do jogo, e são sete. A cada fase que passa são acrescentados sete inimigos, ou seja, na fase 2 são 14 inimigos, na fase 3 são 21

inimigos e assim analogamente.

A movimentação dos inimigos se dá de forma aleatória, isto baseado na Máquina de Estados Finitos. De vez em quando alguns inimigos perseguem o jogador, mas quando o jogador atira contra eles, os mesmos têm a tendência de fugir dos tiros. Isso acontece quando um tiro está na direção do inimigo.

O jogador pode movimentar seu personagem nas direções horizontal sentido direito e esquerdo e vertical sentido acima e abaixo, ou mais detalhadamente, para cima, para baixo, para a esquerda e para a direita. Essa movimentação é feita através das teclas de movimentação do teclado, as setas. Para atirar é usada a tecla barra de espaço. Se o jogador segura uma tecla o funcionamento é o mesmo que apertar várias vezes repetidamente a mesma tecla. Por exemplo, se o jogador segura a tecla barra de espaço o seu personagem irá atirar sem parar. Os tiros do jogador saem na direção da última movimentação que ele fez. Por exemplo, se ele fez seu personagem caminhar para baixo na tela e apertou logo em seguida a barra de espaço, o tiro sairá para baixo. Os inimigos não podem atirar. Eles apenas ficam movimentando na tela. O personagem do jogador é destruído quando um inimigo encosta nele. Já para destruir um inimigo a única forma é através de tiros. Cada inimigo possui um tamanho em pixels, que pode variar de 10 a 63. Esse valor máximo de 63 é devido ao AG, e é explicado na parte da implementação. O valor mínimo de 5 é devido ao fato de que menos de 10 pixels o inimigo fica muito pequeno, e, portanto é muito difícil para o jogador conseguir acertá-lo com um tiro. Quando um tiro acerta um inimigo esse inimigo diminui em 5 pixels seu tamanho. Por exemplo, se o inimigo possui tamanho de 40 pixels, ao receber um tiro ele irá diminuir para o tamanho de 35 pixels.

O jogo termina quando um inimigo consegue encostar no jogador, e a mensagem de Game Over é mostrada na tela. Essa situação é mostrada na figura 5.2. Observa-se a mensagem na janela pequena ao centro.

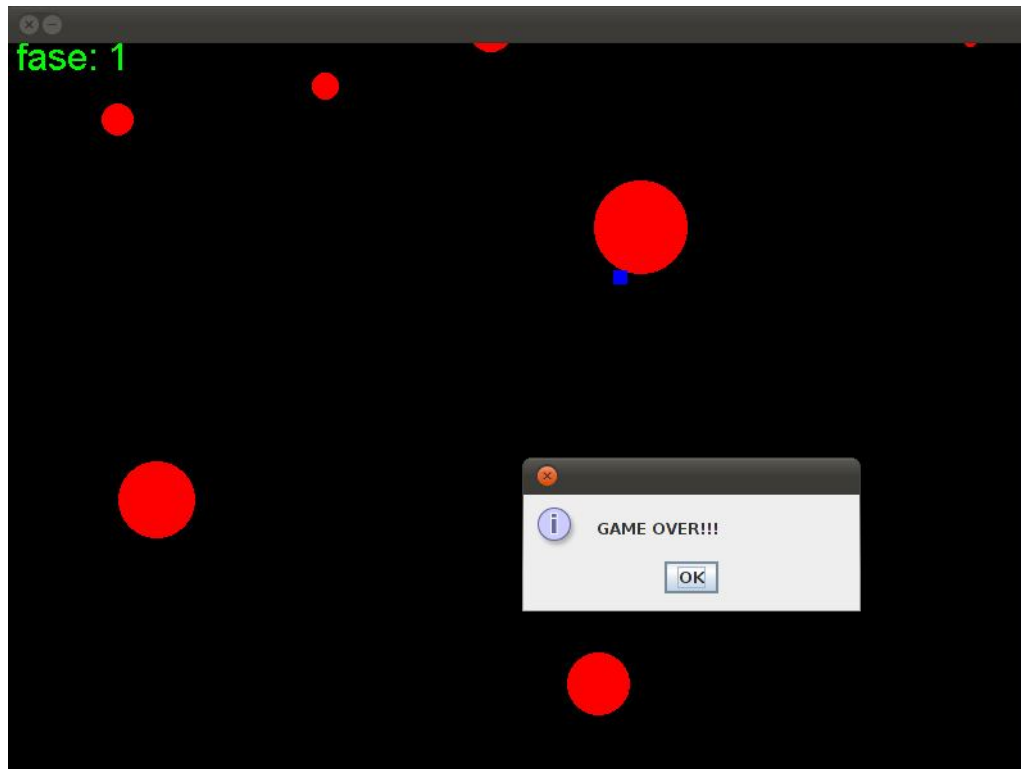


Figura 5.2 - Inimigo destruindo o personagem do jogador

Na figura 5.2 pode-se observar que o inimigo (um dos círculos em vermelho) conseguiu encostar no personagem do jogador (quadrado azul). Dessa forma o jogador perde o jogo. A mensagem “GAME OVER!!!” foi apresentada, e após o jogador clicar no botão OK o aplicativo é encerrado.

O jogo não termina caso o jogador não seja destruído. Cada vez mais inimigos são criados quando uma nova fase é iniciada. O jogo não termina pois a intenção é mostrar as técnicas de IA.

5.2. Linguagem de Programação e Engine

Várias linguagens de programação podem ser utilizadas em jogos eletrônicos, e as linguagem C e C++ são muito utilizadas. A explicação da grande utilização dessas linguagens é o fato de existirem ao longo da história muitas bibliotecas próprias para a criação de jogos, além da velocidade que as linguagens

proporcionam. Deve-se lembrar que um jogo exige muito de processamento na parte gráfica, e jogos que utilizam IA normalmente colocam técnicas que também exigem muito de processamento.

Para esse estudo de caso não será utilizada nenhuma dessas linguagens, mas sim a linguagem Java. É uma linguagem utilizada em larga escala atualmente, orientada a objetos e fácil de ser utilizada. Como o jogo do estudo de caso não possui uma parte gráfica muito boa, apenas a necessária para demonstrar o AG em ação, a linguagem Java satisfaz perfeitamente, e não será necessário a utilização de *scripts* de outras linguagens.

O uso de uma *Engine* no trabalho pode ser descartada, apesar de que uma facilita vários pontos como a renderização gráfica do ambiente e dos personagens. Como o foco está na IA, a renderização dos objetos na tela pode ser feita da maneira mais simples possível, e a linguagem Java possui bibliotecas que permitem que isso seja feito perfeitamente.

A biblioteca escolhida para essa implementação foi a AWT, que é a biblioteca de recursos gráficos padrão encontrada nas versões novas do Java.

O jogo foi compilado utilizando o compilador `javac` do Java SDK para Linux, versão 1.6.0_20. Para execução foi utilizada a máquina virtual OpenJDK Runtime Environment, também para Java versão 1.6.0_20, para o sistema operacional Linux Ubuntu 10.04. Além desse Sistema Operacional, o jogo foi testado em uma máquina com Windows XP.

5.3. Implementação

Para a implementação utilizou-se quatro classes. A classe principal é Ambiente, pois dela parte toda a sequência lógica do jogo. Também nela há a geração de novos inimigos a cada fase, portanto o Algoritmo Genético está incorporado em dois de seus métodos. A classe Inimigo faz as operações dos inimigos, principalmente relacionadas à movimentação, e por isso essa classe contém também a parte de Máquina de Estados Finitos. As classes Jogador e Tiro possuem as operações básicas do personagem controlado pelo jogador e os tiros

efetuados por esse personagem no jogo. O diagrama de classes é apresentado na figura 5.3.



Figura 5.3 - Diagrama de classes

5.3.1. A classe Ambiente

O jogo é iniciado através da classe Ambiente, quando no seu próprio construtor há o código mostrado na figura 5.4. Além de iniciar toda a parte gráfica, o método main também está nessa classe, e apenas chama new Ambiente();.

Na figura 5.4 há a chamada a alguns métodos para a criação do JFrame que mostra a janela na tela do computador. Logo após, a linha onde é instanciado um novo objeto do tipo Jogador passa para o construtor de Jogador os valores da metade da altura e largura do ambiente, fazendo dessa maneira com que o personagem do jogador inicie o jogo sempre na posição central do terreno. Posteriormente há o mais importante no que diz respeito a IA: a criação dos inimigos. Seguindo o fluxograma de um AG, a população, nesse caso os inimigos do jogador, é gerada de forma aleatória, sendo cada indivíduo (inimigo) colocado

na lista inimigos. O vetor inimigosPassado sempre guarda a população anterior, para que o AG possa através dela gerar novos filhos. Para cada inimigo criado, é passado ao construtor da classe Inimigo os valores de x e y, ou seja, as coordenadas do inimigo na tela, geradas sempre de forma randômica, baseando-se na altura e largura da tela, e o tamanho em pixels daquele inimigo. Esse tamanho é um valor aleatório entre 10 e 63. O tamanho em tela também é considerado como a aptidão do inimigo, que é mais explorada no AG.

```
public Ambiente() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(largura, altura);
    this.setResizable(false);
    this.setLocation(100, 100);
    this.setVisible(true);
    this.addKeyListener(this);
    this.createBufferStrategy(2);
    jogador = new Jogador(largura/2, altura/2);
    inimigos = new ArrayList<Inimigo>();
    inimigosPassado = new ArrayList<Inimigo>();
    for(int n = 0; n < fase * N; n++) {
        inimigos.add(new Inimigo(rand.nextInt(largura),
        rand.nextInt(altura), 10 + rand.nextInt(63)));
    }
    for (int i = 0; i < inimigos.size(); i++) {
        inimigosPassado.add(new
        Inimigo((int)inimigos.get(i).getX(),(int)inimigos.get(i).getY(),
        inimigos.get(i).getSize()));
    }
    while(true) {
        long start = System.currentTimeMillis();
        gameLoop();
        while(System.currentTimeMillis()-start < 5) {
        }
    }
}
```

Figura 5.4 - Construtor da classe Ambiente

Por fim, a figura 5.4 mostra um laço while, o qual fica infinitamente chamando o método gameLoop, que é onde a lógica do jogo acontece.

5.3.1.1. O método gameLoop

Esse método detém a sequência lógica dos acontecimentos no jogo, e é mostrado nas figuras 5.5 e 5.6, dividido nas duas partes em cada figura.

Inicialmente o método (figura 5.5) faz uma verificação para perceber se o jogador está pressionando alguma tecla, através do método contains da lista keys. Os valores 37, 38, 39 e 40 são referentes às setas direcionais do teclado, e o valor 32 à barra de espaço. Caso o jogador pressione alguma das setas, o jogador é movido e a variável dir setada com um valor entre 1 e 4.

Esse valor de dir indica a direção que foi pressionada pelo jogador, para análise nas outras classes.

```
private void gameLoop() {  
    if(keys.contains(new Integer(37))) {  
        jogador.move(1);  dir = 1;  
    } if(keys.contains(new Integer(39))) {  
        jogador.move(2);  dir = 2;  
    } if (keys.contains(new Integer(38))) {  
        jogador.move(3);  dir = 3;  
    } if (keys.contains(new Integer(40))) {  
        jogador.move(4);  dir = 4;  
    }  
    if (keys.contains(new Integer(32)))  
        if(fireLimit==0)  
            jogador.fogo(dir); fireLimit = 5;  
        else  
            fireLimit--;  
}
```

Figura 5.5 - Método gameLoop - Parte 1

A figura 5.5 mostra portanto apenas a parte do método que trata das teclas que o jogador pode pressionar.

```
        for(Tiro t:jogador.getTiros()) {
            for(int n = 0; n < inimigos.size(); n++) {
                Inimigo in = inimigos.get(n);
                Point tp = new Point(t.getX(), t.getY());
                Point inp = new Point((int)in.getX(),
(int)in.getY());
                if(tp.distance(inp) <= in.getSize()) {
                    inimigos.remove(n);
                    if(in.getSize()>10){
                        inimigos.add(new
Inimigo((int)in.getX(), (int)in.getY(), in.getSize()-5));
                    }
                }
            }
        }
        for(int n = 0; n < inimigos.size(); n++) {
            Inimigo in = inimigos.get(n);
            Point sp = new Point(jogador.getX(), jogador.getY());
            Point inp = new Point((int)in.getX(), (int)in.getY());
            if(sp.distance(inp) <= in.getSize()) {
                mostraGameOver();
            }
        }
        Ambiente.XJogador = jogador.getX();
        Ambiente.YJogador = jogador.getY();
        if(inimigos.size() <= 0) {
            fase++;
            for (int i = 0; i < 7; i++) {
                initGame();
            }
        }
        drawFrame();
    }
```

Figura 5.6 - Método gameLoop - Parte 2

O resto do método `gameLoop` é mostrado na figura 5.6. O primeiro e segundo laços *for* do método na figura verifica se os pontos entre a distância entre os pontos de todos os inimigos e a posição de um tiro é igual, e se for o inimigo é deletado e no lugar aparece um novo inimigo, com tamanho que tinha anteriormente menos 5. Caso o tamanho do inimigo já seja menor ou igual a 10 não é colocado um inimigo no lugar, ou seja, o inimigo é removido, pois foi destruído pelo jogador.

O terceiro laço *for* na figura 5.6 verifica se a posição do jogador é a mesma que o espaço ocupado por um inimigo. Nesse caso significa que o inimigo conseguiu encostar no jogador, portanto o jogador é destruído. O método `mostraGameOver` chamado apenas apresenta a mensagem de “GAME OVER!!!” na tela e finaliza a aplicação.

O resto de código da figura 5.6 apenas passa a posição do jogador as variáveis estáticas `XJogador` e `YJogador`, que são utilizadas em outras classes, aumenta a fase e chama `initGame` caso o número de inimigos seja zero (estando todos os inimigos destruídos inicia-se a fase seguinte) e chama `drawFrame`, que irá mostrar tudo que está acontecendo na tela. É importante ressaltar que a chamada a `initGame` está em um laço que roda sete vezes. Isso ocorre porque cada vez que `initGame` é chamado, é gerada uma nova lista de inimigos. Testando o jogo chamando apenas uma vez `initGame` a nova população não apresenta uma evolução clara. Gerando sete novas gerações a cada fase obteve-se uma população de inimigos mais evoluída do que a fase anterior. Isso foi uma característica necessária para o AG funcionar corretamente, apresentando realmente uma evolução.

5.3.1.2. O método `initGame`

Esse método é chamado toda vez que se inicia uma nova fase, e ele é responsável por atualizar a lista de inimigos. Também por ele é chamado o Algoritmo Genético rodando no jogo. A figura 5.7 apresenta o método.

Como demonstrado na figura 5.7, quando o `initGame` é chamado o jogador é novamente instanciado e recebe os valores para estar na posição central da tela, e uma nova lista de inimigos é gerada. No primeiro *for* observa-se que o número de inimigos é o valor da fase multiplicado pelo valor N, que nos testes foi alterado entre 5 e 10. É importante analisar dentro desse laço a chamada ao método `geraFilhos`, que é quem faz efetivamente a parte de Algoritmo Genético do programa. No último código da figura 5.7, o segundo laço *for*, a nova lista de inimigos é colocada em `inimigosPassado`, para ser utilizada no próximo chamado a `initGame`.

```
private void initGame() {
    jogador = new Jogador(largura/2, altura/2);
    inimigos = new ArrayList<Inimigo>();
    for(int n = 0; n < fase * N; n++) {
        inimigos.add(new Inimigo(rand.nextInt(largura),
rand.nextInt(altura), geraFilhos()));
    }
    inimigosPassado = new ArrayList<Inimigo>();
    for (int i = 0; i < inimigos.size(); i++) {
        inimigosPassado.add(new
Inimigo((int)inimigos.get(i).getX(),(int)inimigos.get(i).getY(),
inimigos.get(i).getSize()));
    }
}
```

Figura 5.7 - Método `initGame`

5.3.1.3. Algoritmo Genético – métodos `geraFilhos` e `roleta`

A grande inteligência por trás no AG está no método `geraFilhos`. A figura 5.8 demonstra como esse método foi implementado.

Primeiramente na figura 5.8 é feito um somatório com o tamanho de todos os inimigos. O tamanho dos inimigos é considerado como sendo a aptidão do inimigo, pois quando maior ele é, mais difícil é de ser derrotado. Por exemplo, se

um inimigo possui o tamanho máximo, 63, o jogador precisa acertar vários tiros nele para que ele seja derrotado, pois seu tamanho vai reduzindo de 5 em 5 até ser menor que 10. Já um inimigo de tamanho mínimo 10 necessita apenas 1 tiro para ser completamente destruído.

Do valor somado jogado na variável som é criado um valor randômico x1 e outro x2 entre 1 e som, que será passado ao algoritmo da roleta. O algoritmo da roleta tende a pegar os mais aptos a sobreviver para continuar no jogo. Para isso, a roleta é sempre chamada para buscar quem será o pai e a mãe do novo filho. O algoritmo da roleta é mostrado na figura 5.9.

O funcionamento do código na figura 5.9 é o seguinte: ele recebe o valor que foi calculado em x1 ou x2 na figura 5.8, e assim que um novo somatório de tamanho dos inimigos for maior que aquele valor, é retornado a aptidão do indivíduo em questão, isto é, o tamanho daquele inimigo.

Para exemplificar, imagine uma população de 4 inimigos, com tamanhos 10, 20, 30 e 60. No método geraFilhos, é calculado a soma dos tamanhos e jogado em som. Nesse caso, a soma seria 120. Os valores de x1 e x2 recebem valores aleatórios entre 0 e 120, portanto suponha-se que x1 receba 57 e x2 receba 88. Ao chamar a roleta com o valor 57, será feita a soma dos tamanhos até que essa soma seja maior que 57. Portanto, o valor 10 será somado a 20 e então a 30, tendo resultado 60, que é maior que 57, portanto será retornado o valor 30. Ao chamar a roleta com o valor 88, será feita a soma dos tamanhos até que essa soma seja maior que 88. Portanto, o valor 10 será somado a 20 e então a 30 e por fim 60, tendo resultado 120, que é maior que 88, dessa forma será retornado o tamanho 60. Essa forma de buscar os valores da roleta tende a trazer os indivíduos com maior valor, pois a chance de ao se somar um tamanho grande ser maior que o x1 ou x2 é maior do que um tamanho de valor pequeno.

Logo após o método roleta ser executado existem os valores do tamanho do pai e da mãe, armazenados nas variáveis pai e mae, respectivamente na figura 5.8. Após isso o valor do tamanho do pai é convertido para binário com tamanho em 6 casas decimais, dessa forma o valor binário máximo permitido é 111111, que é o 63 (tamanho máximo que um inimigo pode ter) em decimal. Isso explica o

tamanho máximo dos inimigos gerados serem 63.

```
private int geraFilhos() {
    int som = 1;
    for(int i=0; i<inimigosPassado.size(); i++)
        som += inimigosPassado.get(i).getSize();

    int x1 = rand.nextInt(som); int x2 = rand.nextInt(som);
    int pai = roleta(x1);    int mae = roleta(x2);
    char [] bin = {'0','0','0','0','0','0'};
    String s = Integer.toBinaryString(pai);
    int k = 5;
    for (int i = s.length()-1; i >= 0 ; i--, k--) {
        if(k<0) break;
        bin[k] = s.charAt(i);
    }
    char [] bin2 = {'0','0','0','0','0','0'};
    String s2 = Integer.toBinaryString(mae);
    k = 5;
    for (int i = s2.length()-1; i >= 0 ; i--, k--) {
        if(k<0) break;
        bin2[k] = s2.charAt(i);
    }
    char [] binF = {'0','0','0','0','0','0'};
    for (int i = 0; i < 3; i++) binF[i] = bin[i];
    for (int i = 3; i < 6; i++) binF[i] = bin2[i];
    for(int j=0; j<6; j++){
        double tx = rand.nextDouble();
        if(tx <= 0.002)
            if(binF[j] == '1') binF[j] = '0'; else binF[j] = '1';
    }
    int filho=0, z=0;
    for (int i = 5; i >= 0; i--, z++)
        filho += Integer.parseInt(""+binF[i]) * Math.pow(2,z);
    return filho;
}
```

Figura 5.8 - Método geraFilhos()

```

private int roleta(int aux) {
    int somatorio, retorno = 0;
    somatorio = 0;
    for(int i=0; i<inimigosPassado.size(); i++){
        somatorio=somatorio+inimigosPassado.get(i).getSize();
        if(somatorio >= aux){
            retorno= inimigosPassado.get(i).getSize();
            break;
        }
    }
    return retorno;
}

```

Figura 5.9 - Algoritmo da roleta

O tamanho mínimo 10 é apenas para que seja mostrado na tela, menos de 10 pixels é um valor muito pequeno. O valor do pai fica armazenado no vetor bin da figura 5.8. Também o valor do tamanho da mãe é transformado em binário seguindo o mesmo princípio do pai, sendo armazenado no vetor bin2. Por fim há a geração do filho, que pega os 3 primeiros cromossomos do pai e os 3 últimos da mãe. Para exemplificar, pode-se imaginar que o pai tenha tamanho 45 e a mãe tamanho 42. Em binário, o pai é o número 101101 e a mãe 101010. O filho é portanto o 101010, 101 do pai concatenado com 010 da mãe. Nesse exemplo, o filho também terá tamanho 42. Observa-se que dessa forma o tamanho do pai interfere diretamente, sendo que se for maior que o da mãe é garantido um maior desenvolvimento no valor do filho.

Após o filho já possuir um valor como foi demonstrado no parágrafo anterior, ele pode ou não sofrer mutação. Essa mutação pode alterar seu tamanho, e nesse estudo de caso é usada a mutação da maneira mais simples, como mostrado no penúltimo laço *for* da figura 5.8. Para cada cromossomo, no caso bit no vetor tamanho do inimigo, é buscado um valor randômico em double entre 0 e 1 e caso esse valor seja menor que 0,002 é efetuada a mutação. A mutação faz com que se altere o valor do bit naquela posição, de 0 para 1 ou 1 para 0.

Finalmente no fim da figura 5.8 o vetor do tamanho do filho é convertido

em decimal e retornado ao chamador.

5.3.1.4. O método drawFrame

Para a classe Ambiente o último método importante que deve ser apresentado é o drawFrame, que de maneira simples renderiza graficamente os objetos na tela, ou seja, desenha na tela tudo o que está acontecendo. A figura 5.10 mostra a implementação do método drawFrame.

```
private void drawFrame() {
    BufferStrategy bf = this.getBufferStrategy();
    g = null;
    try {
        g = bf.getDrawGraphics();
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, largura, altura);
        for(int n = 0; n < inimigos.size(); n++) {
            Inimigo in = inimigos.get(n);
            in.move();
            in.desenhaInimigo(g);
        }
        jogador.desenhaJogador(g);
        mostraFase(g);
    } finally {
        g.dispose();
    }
    bf.show();
    Toolkit.getDefaultToolkit().sync();
}
```

Figura 5.10 - Método drawFrame

Na figura 5.10, dentro do try na estrutura try catch, há os métodos setColor e fillRect. O primeiro faz a cor ser configurada como preta e o segundo desenha um retângulo na tela. Também dentro do try estão as chamadas a move e

desenhaInimigo, da classe Inimigo, que faz os inimigos se movimentarem e serem desenhados na tela, além de desenhaJogador da classe Jogador, que desenha o personagem do jogador na tela e mostraFase, um método que mostra a fase e o número da mesma na tela. O método chamado através de Toolkit é importante pois faz com que os objetos sejam sincronizados de forma a não ficar dando a impressão de movimento e travamento ao movimentar o personagem do jogador e os inimigos.

5.3.2. A classe Inimigo

Na classe Inimigo estão os parâmetros que um inimigo possui, que são básicos: x para a coordenada horizontal, y para a coordenada vertical e size para o tamanho do inimigo na tela.

Além de desenhar o inimigo na tela, essa classe tem como objetivo fazer a movimentação do mesmo, e é por isso que a Máquina de Estados Finitos é implementada no método move dessa classe.

O construtor da classe Inimigo é mostrado na figura 5.11, e trata de definir o x e y iniciais além do tamanho do inimigo. As variáveis dx e dy são criadas para que o x e y possam ser diferentes à todo momento. Também gera um valor randômico para prob. Esse valor será necessário para indicar se aquele inimigo deve ou não ir atrás do jogador. No método move é feita a comparação de prob.

```
public Inimigo(int x, int y, int size) {  
    this.x = x;  
    this.y = y;  
    this.size = size;  
    prob = rand.nextInt(10);  
    while(this.dx==0)  
        this.dx = rand.nextDouble()*2-1;  
    while(this.dy==0)  
        this.dy = rand.nextDouble()*2-1;  
}
```

Figura 5.11 - Construtor da classe Inimigo

5.3.2.1. O método move

A primeira parte do método move é apresentado na figura 5.12. Ele é responsável por todas as possibilidades de movimentação do inimigo na tela. Primeiramente ele incrementa x e y de maneira com que o x e o y sejam o valor atual mais o valor dx e dy, que foram definidos de forma randômica no construtor da classe. Logo após começa a parte da Máquina de Estados Finitos. No algoritmo mostrado na figura 5.12, o estado do inimigo é percorrer o jogador. A comparação para prob determina uma chance de 70% do inimigo ir atrás do jogador, ou seja, apenas alguns inimigos perseguem o personagem do jogador. Isso ocorre para o jogo não ficar muito difícil.

```
public void move() {  
    x += dx;  
    y += dy;  
  
    if(prob<8){  
        if(x>Ambiente.XJogador)  
            x--;  
        else  
            x++;  
        if(y>Ambiente.YJogador)  
            y--;  
        else  
            y++;  
    }  
}
```

Figura 5.12 - Método move - Parte 1

A figura 5.13 mostra a 2ª parte do método move. Essa parte também refere-se à FSM, e trata do estado fuga. Nesse estado, caso exista um tiro do jogador na direção do inimigo, o inimigo deve correr do tiro. Primeiro verifica-se a existência de tiros na lista de tiros do jogador, e depois compara-se a direção de cada tiro e a posição do inimigo. Por exemplo, no primeiro *if* dentro do *for* compara-se se a direção é igual a um. Nesse caso, o tiro está indo para a esquerda

da tela. Compara-se também se a posição x do tiro é maior que a posição x do inimigo. Se for, significa que há a possibilidade do tiro acertar o inimigo, pois está indo em sua direção x. Compara-se por fim a diferença entre o y do tiro e o y do inimigo. Se essa diferença for menor que 50, o inimigo recua o seu y. Assim é feito analogamente a comparação para todas as outras três possíveis direções que um tiro possui.

```
if(Ambiente.jogador.getTiros().size() > 0){
    for(int i=0; i<Ambiente.jogador.getTiros().size();i++){
        if(Ambiente.jogador.getTiros().get(i).getDir() == 1
        && Ambiente.jogador.getTiros().get(i).getX() > x &&
        Math.abs(y - Ambiente.jogador.getTiros().get(i).getY()) < 50){
            y+=-1;
        } else if(Ambiente.jogador.getTiros().get(i).getDir()
        == 2 && Ambiente.jogador.getTiros().get(i).getX() < x
        &&Math.abs(y - Ambiente.jogador.getTiros().get(i).getY())<50){
            y+=-1;
        } else if(Ambiente.jogador.getTiros().get(i).getDir()
        == 3 && Ambiente.jogador.getTiros().get(i).getY() > y &&
        Math.abs(x - Ambiente.jogador.getTiros().get(i).getX())<50){
            x+=-1;
        } else if(Ambiente.jogador.getTiros().get(i).getDir()
        == 4 && Ambiente.jogador.getTiros().get(i).getY() < y &&
        Math.abs(x - Ambiente.jogador.getTiros().get(i).getX())<50){
            x+=-1;
        }
    }
}
```

Figura 5.13 - Método move - Parte 2

A última parte do método move da classe Inimigo é apresentado na figura 5.14. Essa parte verifica se o inimigo não saiu da tela que é vista. Isso é feito comparando o valor de x a zero e ao tamanho da largura do ambiente, e o de y a zero e ao tamanho da altura do ambiente. Caso alguma dessas comparações

retorne valor verdadeiro o x e y são configurados de forma que o inimigo saia do outro lado da tela. Por exemplo, se o x for igual a zero significa que o inimigo está a esquerda da tela, saindo do terreno. Portanto, é atribuído o valor da largura (800) a ele, de forma que ele apareça do lado direito da tela, e o y nesse caso não é alterado. Assim é feito analogamente para os outros possíveis casos do inimigo sumir da tela.

```
else {  
  
    if(x <= 0) {  
        x = Ambiente.largura;  
    } else  
        if (x >= Ambiente.largura) {  
            x = 0;  
        }  
  
    if(y <= 0) {  
        y = Ambiente.altura;  
    } else  
        if(y >= Ambiente.altura) {  
            y = 0;  
        }  
    }  
}
```

Figura 5.14 - Método move - Parte 3

5.3.2.2. O método desenhaInimigo

O método desenhaInimigo é mostrado na figura 5.15. Observa-se que ele recebe o objeto de Graphics do objeto chamador do tipo Ambiente e configura a cor para vermelho, posteriormente desenhando através dos métodos drawOval e fillOval um círculo nas posições das coordenadas x e y e o tamanho guardado na variável size.

```

public void desenhaInimigo(Graphics g) {
    g.setColor(Color.RED);
    g.drawOval((int)x, (int)y, size, size);
    g.fillOval((int)x, (int)y, size, size);
}

```

Figura 5.15 - Método desenhaInimigo

5.3.3. A classe Jogador

O construtor da classe Jogador apenas define as coordenadas do jogador na tela, variáveis x e y, e instancia uma nova lista de tiros desse jogador. A figura 5.16 mostra o construtor. Essa classe possui um método move, que movimenta o personagem do jogador na tela, além dos métodos desenhaJogador para desenhar o personagem, fogo para adicionar um tiro à lista de tiros e desenhaTiros para mostrar os tiros na tela. Vale ressaltar que as colisões entre tiros e inimigos e jogador e inimigos é feita na classe Ambiente.

```

public Jogador(int x, int y) {
    this.x = x;
    this.y = y;

    tiros = new ArrayList<Tiro>();
}

```

Figura 5.16 - Construtor da classe Jogador

5.3.3.1. O método fogo

O método fogo, apresentado na figura 5.17, adiciona um novo tiro à lista de tiros do jogador. Esse método é chamado dentro da classe Ambiente. Para um novo tiro, são passados como parâmetros ao construtor o x e y do jogador, pois as coordenadas de um tiro são sempre as mesmas do jogador, ou seja, a bala de um tiro é inicialmente mostrada na mesma posição do personagem do jogador, movimentando-se em determinada direção logo a seguir.

```

public void fogo(int dir) {
    try {
        tiros.add(new Tiro(x, y, dir));
    } catch ( ConcurrentModificationException e) { }
}

```

Figura 5.17 - O método fogo

A direção para a qual o tiro irá se movimentar também é passada como parâmetro, tendo sido recebida como argumento pela variável dir.

5.3.3.2. O método desenhaTiros

O método desenhaTiros é mostrado na figura 5.18, e faz a renderização gráfica dos tiros na tela. Como observado na figura, para cada tiro na lista de tiros é chamado o método move da classe tiro e desenhaTiro, esse último passando o objeto de Graphics que foi definido como parâmetro no método. Por fim, no método desenhaTiros, se a coordenada x do tiro for zero ou maior que a largura do ambiente ou a coordenada y for zero ou maior que a altura, o tiro é removido da lista de tiros, pois nesse caso o tiro já saiu do terreno.

```

private void desenhaTiros(Graphics g) {
    try {
        for(int n = 0; n < tiros.size(); n++) {
            Tiro t = tiros.get(n);
            t.move();
            t.desenhaTiro(g);
            if(t.getX()<0 || t.getX() > Ambiente.largura ||
t.getY()<0 || t.getY() > Ambiente.altura) {
                tiros.remove(t);
            }
        }
    } catch ( ConcurrentModificationException e) {
    }
}

```

Figura 5.18 - O método desenhaTiros

Diferente da movimentação de um inimigo, um tiro que sai do terreno é descartado. Um inimigo que sai do terreno reaparece do outro lado da tela.

5.3.3.3. O método `desenhaJogador`

O método `desenhaJogador` é mostrado na figura 5.19. Observa-se que ele recebe o objeto de `Graphics` do objeto chamador do tipo `Ambiente` e configura a cor para azul, posteriormente desenhando através dos métodos `drawRect` e `fillRect` um retângulo nas posições das coordenadas `x` e `y` e o tamanho de 10 pixels.

```
public void desenhaJogador(Graphics g) {  
    g.setColor(Color.BLUE);  
    g.drawRect(x, y, 10, 10);  
    g.fillRect(x, y, 10, 10);  
    desenhaTiros(g);  
}
```

Figura 5.19 - Método `desenhaJogador`

5.3.3.4. O método `move`

O método `move` da classe `jogador` é bem simples, pois diferente da movimentação de um inimigo, onde a movimentação é controlada pelo computador, no caso do personagem do jogador foi o jogador que pressionou uma tecla.

```
public void move(int dir) {  
    switch (dir) {  
        case 1: x--;      break;  
        case 2: x++;      break;  
        case 3: y--;      break;  
        case 4: y++;      break;  
        default: break;    }    }
```

Figura 5.20 - O método `move`

O método `move`, mostrado na figura 5.20, recebe a direção configurada no objeto de `Ambiente` no momento quando o jogador pressionou uma tecla, guardada no argumento `dir`, e para cada caso altera a posição necessária: para `dir` igual a 1, o valor de `x` é decrementado, pois a tecla pressionada foi seta esquerda, fazendo com que o personagem caminhe à esquerda; para `dir` igual a 2, o valor de `x` é incrementado, pois a tecla pressionada foi seta direita, fazendo com que o personagem caminhe à direita; para `dir` igual a 3, o valor de `y` é decrementado, pois a tecla pressionada foi seta acima, fazendo com que o personagem caminhe para cima na tela; e finalmente, para `dir` igual a 4, o valor de `y` é incrementado, pois a tecla pressionada foi seta abaixo, fazendo com que o personagem caminhe para baixo na tela. Deve-se atentar para o fato que `x` e `y` são as variáveis que guardam as coordenadas atuais do personagem do jogador.

5.3.4. A classe `Tiro`

O construtor da classe `Tiro` apenas define as coordenadas do tiro na tela, variáveis `x` e `y`, além da direção que o tiro deverá tomar quando for criado. A direção é guardada na variável `dir`. A figura 5.21 mostra esse construtor. Essa classe possui um método `move`, que movimenta o tiro feito pelo jogador na tela, além do método `desenhaTiro` para desenhar cada objeto tiro na tela.

```
public Tiro(int x, int y, int dir) {  
    this.x = x;  
    this.y = y;  
    this.dir = dir;  
}
```

Figura 5.21 - Construtor da classe `Tiro`

5.3.4.1. O método `desenhaTiro`

O método `desenhaTiro` é mostrado na figura 5.22. Observa-se que ele

recebe o objeto de Graphics do objeto chamador e configura a cor para branco, posteriormente desenhando através do método drawOval um círculo na tela, nas posições das coordenadas x e y instanciadas no construtor da classe Tiro e o tamanho de 5 pixels. Não foi utilizado o método fillOval, deixando o tiro apenas desenhado na tela e não preenchido com branco, para não ficar muito forçada a vista dos jogadores.

```
public void desenhaTiro(Graphics g) {  
    g.setColor(Color.WHITE);  
    g.drawOval(x, y, 5, 5);  
}
```

Figura 5.22 - Método desenhaTiro

5.3.4.2. O método move

O método move é apresentado na figura 5.23. Ele usa o valor da variável dir para saber em qual direção irá movimentar o tiro. Se dir for igual a 1, o tiro se movimenta para a esquerda, portanto o valor da coordenada x é diminuído. Caso dir seja 2, o tiro se movimenta para a direita, portanto o valor da coordenada x é aumentado. Para dir igual a 3 o tiro se movimenta para cima, portanto o valor da coordenada y é diminuído. E por fim, para dir igual a 4 o tiro se movimenta para baixo, portanto o valor da coordenada y é aumentado.

```
public void move() {  
    switch (dir) {  
        case 1:    x-=3;        break;  
        case 2:    x+=3;        break;  
        case 3:    y-=3;        break;  
        case 4:    y+=3;        break;  
        default:   break;  
    }  
}
```

Figura 5.23 - Método move

Diferente da movimentação do jogador e dos inimigos, onde cada alteração de coordenadas há o incremento ou decremento de 1 unidade, observa-se na figura 5.23 que no caso de um tiro é incrementado ou decrementado 3 unidades. Isso se dá devido ao fato de que tiros na mesma velocidade do inimigo seria praticamente impossível que o tiro colidisse com o inimigo.

5.4. Utilização de IA x não utilização de IA

Uma boa análise a ser feita na implementação do estudo de caso é se realmente é necessário a utilização de IA para garantir a inteligência do jogo. O que acontece é que durante muitos anos a IA foi questionada por muitos cientistas no ponto de vista que não há realmente inteligência por parte da máquina, apenas regras pré-determinadas, que fazem com que acreditemos existir alguma forma de inteligência naquela aplicação.

Não levando em consideração outros ramos em que aplica-se as técnicas de IA, pode-se analisar a IA dos jogos em termos de como ela poderia transformar os personagens em personagens inteligentes ou como a estratégia do jogo por parte da máquina se comporta de maneira inteligente.

Neste trabalho praticamente todas as utilizações de IA encontradas estão na movimentação dos personagens na tela, e algumas poucas vezes na criação do ambiente no qual os personagens estão inseridos. Deve-se ressaltar que o que se relaciona ao ambiente é apenas a forma como os personagens são gerados quando as fases são incrementadas.

No estudo de caso deste trabalho vários pontos podem ser levados em consideração. Utiliza-se basicamente o AG para que o desenvolvimento da população no RPG ocorra. Utilizando-se o princípio de que os AG's podem ser utilizados como na teoria da evolução, onde uma população evolui de maneira que a próxima geração seja filha da geração atual e que carregue a carga genética da atual geração, além de que a mutação possa ocorrer de maneira de indivíduos possam ficar mais fortes ou até mesmo mais fracos, a população do RPG pode então se desenvolver de forma uniforme, isto é, os guerreiros de uma próxima

geração terão habilidades parecidas com os guerreiros da atual geração, o que torna o jogo mais realista.

Entende-se que o AG da maneira como é trabalhado neste documento cuida da evolução de um povoado de forma que essa evolução se torne bem realista. Muitos escritores criticam esse tipo de algoritmo, justificando que não há realmente alguma inteligência na maneira como os guerreiros pensam: eles foram apenas programados para agir daquela forma. Porém, o estudo de caso mostra que até certo ponto o AG colabora bastante no desenvolvimento, principalmente quando se busca uma jogabilidade melhor. Toda vez que um jogador estiver jogando ele terá diferentes povoados pela frente, com guerreiros menos ou mais evoluídos, e até os guerreiros do seu povoado terão a cada jogo um padrão de força maior ou menor. Dessa forma, a cada vez que o jogo for executado o jogador poderá atuar com uma estratégia diferente.

Além do AG, também foi utilizado Máquina de Estados Finitos, FSM, a qual mostrou uma certa eficiência por parte da movimentação dos personagens inimigos na tela. Porém, essa técnica apesar de poder ser considerada dentro da parte de movimentação da IA, a mesma que trata de algoritmos de menor caminho, não é analisada como IA pela maioria dos autores. Realmente, também não há nada que faça o personagem ser inteligente ao ponto que o jogador queria que ele fosse, ele se torna inteligente apenas no sentido de saber se irá perseguir o jogador ou correr dele.

Caso algum desenvolvedor opte por não utilização de Algoritmo Genético ou Máquina de Estados Finitos em seu RPG, e também opte por não usar outros tipos de algoritmos de Inteligência Artificial, ele basicamente utilizaria muito de alguns algoritmos que gerem aleatoriedade. Isso também garante uma jogabilidade melhor, mas acontece que os povoados não terão um padrão: existirão povoados com alguns guerreiros muito fortes e outros muito fracos, não sendo o que acontece na realidade, além do que a pouca inteligência presente nesse estudo de caso, da tomada de decisão do inimigo via Máquina de Estados Finitos, seria descartada.

5.5. Conclusão

Apesar de que a utilização de inteligência artificial gere um trabalho a mais do desenvolvimento de um jogo eletrônico, que já possui um caráter multidisciplinar, é recomendado no caso de jogos de RPG a utilização de AG's. Não foi encontrado nenhum outro tipo de algoritmo que faça o desenvolvimento da população do jogo de forma mais adequada.

A linguagem Java mostra-se apta à utilização de AG's, visto que é uma linguagem completa e para todos os recursos necessários no estudo de caso ela atendeu bem.

Uma boa prática que pode ser utilizada em trabalhos futuros é a utilização de outros tipos de algoritmos da IA não trabalhados nesse estudo de caso, como por exemplo as RNA. No jogo, os personagens controlados pelo computador apenas trabalham de forma aleatória, buscando atacar o adversário, através da FSM. Um bom uso poderia ser de utilizar uma RNA para que a máquina aprenda os movimentos do jogador e possa se organizar taticamente de maneira diferente a cada jogador.

Nenhum tipo de algoritmo de IA que não seja AG e FSM mostrou-se necessário neste estudo de caso, visto que o mesmo trata de fazer com que o povoado do adversário se desenvolva de modo diferente a cada execução, para que o jogador aprecie novos guerreiros a cada jogo, e possa então ter diversas estratégias diferentes.

Com a utilização da FSM, para os estados perseguição e fuga do inimigo, o jogo mostrou uma dificuldade considerável. A utilização de probabilidade para o inimigo ir ou não atrás do jogador foi necessária, pois sem ela o jogo ficou difícil. Quanto ao AG, a utilização de sete gerações a cada fase também foi necessária, pois sem ela a evolução dos inimigos a cada fase não foi claro.

O próximo capítulo, capítulo 6, trata de fazer a conclusão geral da monografia, mostrando o que pode-se esperar da IA no futuro, através da análise da IA na história dos jogos eletrônicos, além de apresentar a conclusão de se utilizar ou não IA nos jogos, visto as outras várias disciplinas envolvidas em um

jogo, como rede, áudio, física, etc. Também é apresentado a conclusão geral de como o AG, FSM e RPG casam bem, e quais as desvantagens dessa implementação.

6. CONCLUSÃO

Os primeiros jogos eletrônicos surgiram no final da década de 1960 e início da década de 1970, com poucos recursos gráficos até então disponíveis. A evolução desses jogos pode ser considerada rápida, visto que acompanharam a evolução dos próprios computadores.

Jogos como *Pong* marcaram gerações, e até os dias atuais são bastante jogados, sendo replicados em aplicações pela internet. Deve-se analisar que os recursos gráficos ditaram o desenvolvimento dos jogos, pois durante algumas décadas os desenvolvedores preocuparam-se em criar jogos que agradassem esteticamente os jogadores. Isso realmente se mostrou eficaz, pois muitos jogos foram vendidos por causa das cores e outros recursos gráficos que eram capazes de entregar ao usuário, no caso o jogador.

Não somente de recursos gráficos, mas também a movimentação de personagens na tela sofreu grandes avanços ao longo da história. Pode-se concluir esse avanço analisando o poder de movimentação de jogos como *Asteroids* e *Pac-man*, tanto pelo uso de padrões de movimento quanto máquina de estados.

A preocupação em utilizar movimentos mais bem trabalhados ainda na década de 1980 se dá devido à exigência dos jogadores. A jogabilidade começou a ditar o ritmo das pesquisas e criação de jogos eletrônicos, onde não somente as interfaces bem trabalhadas agradam o público, mas também jogos que possuem jogabilidade boa.

Com a preocupação de trabalhar com esse tipo de cenário, têm-se o início da Inteligência Artificial nos jogos. Tanto técnicas mais simples como Padrões de Movimento ou Máquina de Estados Finitos, que não são relacionadas com a

disciplina de IA, assim como as técnicas reconhecidamente como IA clássica (Redes Neurais, Algoritmos Genéticos e Lógica Fuzzy) foram incorporadas em alguns jogos.

Entende-se claramente que os jogos eletrônicos utilizam várias disciplinas da Ciência da Computação, e conclui-se essas disciplinas podem ser perfeitamente utilizadas ao mesmo tempo. Não é necessário a utilização de uma *Engine* para a criação de um jogo eletrônico, assim como a sua utilização pode facilitar a renderização de personagens mais bem trabalhados. Ao não se utilizar uma *Engine* os recursos de diferentes disciplinas podem ser colocadas no jogo independente da linguagem de programação em que se esteja desenvolvendo o jogo. A utilização de uma *Engine* também possui a possibilidade de utilização de recursos de várias disciplinas. O capítulo 2 mostrou a história e evolução dos jogos, e o capítulo 3 apresentou alguns componentes de algumas disciplinas que são comumente utilizados em um jogo. Percebe-se que os jogos realmente aproveitaram o máximo dos conceitos da Ciência da Computação, tais como IA ou redes de computadores. Cada vez mais busca-se utilizar Algoritmos Genéticos para desenvolvimento do ambiente e personagens ao longo de um jogo, ou Redes Neurais para aprendizado dos personagens que estão inseridos em determinado jogo. Também cada vez mais o estudo das redes de computadores mostra força, visando principalmente os jogos *multiplayer*.

O capítulo 3 mostrou também que não só de técnicas de computação está relacionado um jogo eletrônico. Outros elementos como por exemplo o roteiro são muito trabalhados em um jogo bem feito. Percebe-se que não somente profissionais da computação são utilizados em um jogo. Normalmente profissionais de outras áreas é que criam o roteiro e fazem o *Game Design*. Cabe a desenhistas muitas vezes criarem personagens para os jogos, e a computação então é incorporada à esses personagens, para que ganhem vida, se movimentem no ambiente e interajam com o jogador.

O capítulo 4 buscou mostrar como uma das várias disciplinas da computação contribuem ao jogos, no caso a escolhida Inteligência Artificial. O

capítulo 5 demonstrou um estudo de caso onde Algoritmo Genético, da IA, é incorporado a um jogo eletrônico.

Foi observado que as técnicas RNA, AG e Lógica Fuzzy são muito poderosas, e teoricamente cabem perfeitamente para serem utilizadas em jogos, RNA para aprendizado, AG para desenvolvimento e evolução de personagens e Lógica Fuzzy para tomada de decisões dos personagens. Porém, na prática, essas técnicas são incorporadas aos jogos, e conceitualmente são técnicas de IA, mas não foi concluído que a sua utilização deixa os personagens do jogo realmente inteligentes, e talvez estejamos um pouco longe disso. O jogo desenvolvido como estudo de caso no capítulo 5 mostrou o uso de AG para gerar novos guerreiros durante o jogo, e algum tipo de inteligência para esses personagens ou para que o jogador se sentisse jogando contra outro ser humano foi demonstrado utilizando a FSM. Com AG e FSM em conjunto, observou-se a criação de um jogo com dificuldade elevada para o jogador.

O foco dos próximos trabalhos na área desta monografia talvez sejam voltados para as RNA's, pois são teoricamente as que podem trazer mais rápido o conceito de inteligência para um jogo. Ainda durante alguns anos serão vistos jogos com padrões de movimento e movimentos aleatórios, e justifica-se até essa utilização, visto que os algoritmos de IA não são tão fáceis de programar, e não trazem tanta diferença notória em um jogo.

REFERÊNCIAS BIBLIOGRÁFICAS

BALDANCE, Fernando Henrique Martins, REIS, Otávio Augusto de Queiroz. Projeto e Implementação de Técnicas de Desenvolvimento de Jogos para Computador. Varginha: Centro Universitário do Sul de Minas – UNIS MG, Unidade de Gestão da Educação Presencial – GEDUP, 2006.

BATTAIOLA, André Luiz. Jogos por Computador – Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação. São Carlos: Departamento de Computação da Universidade Federal de São Carlos, 2000.

BORGES, Deise Miranda, BARREIRA, Rafael Gonçalves, SOUZA, Jackson Gomes de Souza. Comportamento de personagens em jogos de computador. Palmas: Centro Universitário Luterano de Palmas, 2009.

BORTOLI, Allyson Gabriel, VIER, Filipe Winibaldo, ULLON, Vivien de Lima Nunez. História da Informática. Foz do Iguaçu: Universidade Federal do Oeste do Paraná – UNIOESTE, 2007.

BRUN, André Luiz. Algoritmos Genéticos. EPAC – Encontro Paranaense de Computação.

CASTRO, César C. de, CASTRO, Marina Cristina F. de. Redes Neurais Artificial. In: _____. Capítulo 4. Multilayer Perceptrons. PUCRS.

COSTA E SILVA, Luiz Fernando. Modelo de Rede Neural Artificial Treinada com o Algoritmo *Backpropagation*. Juiz de Fora: Universidade Federal de Juiz de Fora, Departamento de Ciência da Computação, 2003.

CRAWFORD, Chris. Chris Crawford on Game Design. In: _____. Chapter 2. Some Milestone Games. New Riders Games. 2003.

DA SILVA, Marcos Antonio. Aplicação de Lógica Nebulosa para Previsão do Risco de Escorregamentos de Taludes em Solo Residual. Rio de Janeiro: Universidade do Estado do Rio de Janeiro, 2008.

DAZZI, Rudimar Luís Scaranto. Sistemas Especialistas Conexionistas: Implementação por Redes Diretas e Bidirecionais. Florianópolis: Universidade de Santa Catarina, 1999.

EDWARDS, Betty. Desenhando com o Lado Direito do Cérebro. In: _____. Nosso Cérebro: Os Lados Direito e Esquerdo. Editora Tecnoprint S.A, 1984.

ENE. Disponível em “http://www.ene3.com/archives/PS2/fif_110.jpg”. Acessado em 14 de maio de 2011.

FILITTO, Danilo. Algoritmos Genéticos: Uma Visão Explanatória. Saber Acadêmico: Revista Multidisciplinar da Uniesp, 2008.

GALDINO, Carlos Henrique Silva. Inteligência artificial aplicada no desenvolvimento de jogos de computador. Itajubá: Universidade Federal de Itajubá, Instituto de Ciências Exatas, 2007.

GAMES, Rockstar. Disponível em “<http://www.rockstargames.com/games#/?game=25>”. Acessado em 14 de maio de 2011.

HUIZINGA, Johan. Homo Ludens. 4.ed. São Paulo – SP: Editora Perspectiva S.A, 2000.

KISHIMOTO, André. Inteligência Artificial em Jogos Eletrônicos. Copyright 2004, André Kishimoto, 2004.

LUZ, Mairlo Hideyoshi Guibo Carneiro da. Desenvolvimento de Jogos de Computador. Itajubá – MG: Universidade Federal de Itajubá – Unifei, Departamento de Matemática e Computação – DMC, 2004.

MILLINGTON, Ian, FUNGE, John. Artificial Intelligence for Games. Second Edition. Morgan Kaufmann Publishers, 2009.

NEWSGAMER. Disponível em “<http://www.newsgamer.com.br/index.php/jogos-classicos-do-atari-serao-disponibilizados-na-psn/>”. Acessado em 14 de maio de 2011.

PEABODY, Sue. The Art of Computer Game Design by Chris Crawford. Washington State University Vancouver, 1997.

PIRES, Eduardo José Solteiro. Algoritmos Genéticos: Aplicação à Robótica. Porto: Faculdade de Engenharia da Universidade do Porto, 2008.

POZZER, Cesar Tadeu, DREUX Marcelo, FEIJÓ, Bruno. Representação Gráfica de Histórias Interativas . Rio de Janeiro: PUC-Rio, outubro de 2003.

RUSSEL, Stuart, NORVIG, Peter. Inteligência Artificial: Uma abordagem moderna. Editora Campus, 2003.

SANTANA, Roberto Tengan. I.A. Em Jogos: a busca competitiva entre o homem e a máquina. Praia Grande: Faculdade de Tecnologia de Praia Grande, 2006.

SCHNEIDER, Marvin Oliver. Sistemas Inteligentes. Tema 1: Introdução. Rede Neural x Rede Artificial. Campinas: Pontifícia Universidade Católica de Campinas, 2001.

SCHWAB, Brian. AI Game Engine Programming. In: _____. Basic Definitions and Concepts. In: _____. Adventure Games. In: _____. Finite-State Machines. In: _____. Fuzzy-State Machines. In: _____. Genetic Algorithms. In: _____. Neural Networks. In: _____. Conclusions and the Future. 1.ed. Hingham, Massachusetts: Charles River Media, Inc. 2004. p. 3-28. p. 87-94. p. 281-308. p. 413-489. p. 577-578.

SILVA, Fábio de Melo. Concepção e Realização de um Modelo Computacional de Jogos Interativos no Contexto da Aprendizagem Colaborativa. Maceió: Universidade Federal de Alagoas, 2008.

SOARES, Gustavo Luís. Algoritmos Genéticos: Estudo, novas técnicas e aplicações. Belo Horizonte: Universidade Federal de Minas Gerais, 1997.

SOARES, Marcelo Costa. Camada de Inteligência em Jogos para Celulares. Uberlândia: União Educacional de Minas Gerais – UNIMINAS, 2005.

SPORTS, EA. Disponível em “<http://www.ea.com/soccer/fifa-ultimate-team/ps3>”. Acessado em 14 de maio de 2011.

TATAI, Victor Kazuo. Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador . Campinas: Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação , 2003. p. 8-47. p. 74-96.

TONSIG, Sérgio Luiz. Redes Neurais Artificiais Multicamadas e o Algoritmo *Backpropagation*. Campinas: Pontifícia Universidade de Campinas, 2000.

TRÉ, Daniel. Redes Neurais Artificiais Aplicadas ao Jogo da Velha 3D em Pinos. Petrópolis: ISTCCP, 2009.

VELASQUEZ, Carlos Eduardo Lé. Modelo de Engenharia de Software para o Desenvolvimento de Jogos e Simulações Interactivas. Porto: Universidade Fernando Pessoa, 2009.

WOLF, Mark J.P.. The Video Game Explosion: A History from PONG to PlayStation® and Beyond . Westport, Connecticut – London: Greenwood Press, 2008.

BIBLIOGRAFIA

BORTOLI, Allyson Gabriel, VIER, Filipe Winibaldo, ULLON, Vivien de Lima Nunez. História da Informática. Foz do Iguaçu: Universidade Federal do Oeste do Paraná – UNIOESTE, 2007.

FUJITA, Eduardo. Algoritmos de IA para Jogos. Londrina: Universidade Estadual de Londrina, Departamento de Computação, 2005.

GIOSOFT. Disponível em “<http://www.giosoft.net/Development/Java-Asteroids-Tutorial.html>”. Acessado em 21 de novembro de 2011.

LOULA, Angelo Conrado. Comunicação Simbólica entre Criaturas Artificiais: um experimento de Vida Artificial . Campinas: Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, 2003. p 1-5.

MAGNI, Antônio. Implementação do Algoritmo de Backpropagation. 2003.

RAMOS, Henrique Moraes. A História dos Jogos de Computadores. Santa Maria: Universidade Federal de Santa Maria – UFSM, 2007.