Inteligência Artificial em Jogos de Tiro em Primeira Pessoa

CLAILSON SOARES DINÍZIO[1]
MARCO ANTÔNIO COSTA SIMÕES (ORIENTADOR)[2]

[1]UNIT – Universidade Tiradentes
CCFT – Centro de Ciências Formais e Tecnologia
Av. Murilo Dantas, 300 – CEP 49032-490 Aracaju (SE)
james_metallica@hotmail.com

[2]UNIT – Universidade Tiradentes
GPA – Grupo de Pesquisa em Automação
GPI – Grupo de Pesquisa em Informática
CCFT – Centro de Ciências Formais e Tecnologia
Av. Murilo Dantas, 300 – CEP 49032-490 Aracaju (SE)

[2]UFPE – Universidade Federal de Pernambuco
REDIS – Grupo de Redes e Sistemas Distribuídos
Cin – Centro de Informática
Caixa Postal 7851 – CEP 50732-970 Recife (PE)
macs3@cin.ufpe.br

Resumo: Este artigo representa um estudo de como as técnicas de Inteligência Artificial (IA), especificamente a técnica chamada de Máquina de Estado Finito (FSM – *Finite State Machine*), podem ser aplicadas no desenvolvimento de agentes para serem utilizados em Jogos de Tiro em Primeira Pessoa (FPS – *First Person Shooter*), que a cada dia vêm trazendo inovações e níveis de evoluções mais complexos como os jogos da série *Unreal*. Desenvolver agentes para jogos (chamados de *bots*) representa a mais nova tentativa de traduzir as condições e capacidades humanas de resolver diversos problemas desde gerenciar seus recursos de vida até planejar táticas defensivas e de ataque. É apresentado, também, um estudo de caso através do qual se desenvolveu um protótipo que demonstra a técnica FSM em um agente que joga um tipo de jogo do *Unreal Tournament* (UT) chamado de *Deathmatch* (DM). Utilizando para atingir tal objetivo a linguagem de programação Java para o desenvolvimento do agente, o jogo UT e o projeto *Gamebots* para a conexão com o ambiente do jogo.

Palavras Chaves: Autômatos Finitos, Jogos de Tiro em Primeira Pessoa, *Unreal Tournament*, agente, *Gamebots*.

1 Introdução

Desde o início do surgimento da informática, muito se procurou fazer para concretizar a idéia de que um computador pudesse jogar e aprender com seus erros, melhorando assim seu desempenho. É nessa área que entra a Inteligência Artificial, que tem feito com que, a cada dia que passa, programas especializados surjam, e que tais programas possam agir cada vez mais como humanos, atuando e tomando decisões.

Na área dos jogos tais técnicas se desenvolveram inicialmente aplicadas aos simples jogos de xadrez, damas, jogo-da-velha, etc. Hoje em dia, as técnicas de Inteligência Artificial (IA) aplicadas em jogos de computador estão mais desenvolvidas, permitindo a criação de jogos mais complexos, que permitem a criação de programas mais inteligentes e de ambientes mais reais para interagir com os mesmos.

Um dos gêneros de jogos de computador que se tornaram mais famosos no mundo e que vêm arrastando uma legião de seguidores, são os Jogos de Tiro em Primeira Pessoa ou FPS (*First Person Shooter*) - tipo de jogo em

que o jogador controla o personagem como se fosse ele próprio no jogo, tendo uma visão em primeira pessoa do jogo, utilizando armas para cumprir os mais diversos tipos de missões. As avançadas técnicas de IA estão trazendo Agentes (Programas que agem de forma autônoma) mais inteligentes que podem até aprender com seus erros. Deste modo, os inimigos controlados por computador nestes tipos de ambientes interativos, não mais esperam para atirar, mas caçam seus inimigos à medida que vão atravessando o ambiente.

Um dos jogos de tiro em primeira pessoa em que esforços para a construção de agentes que agem como humanos, tem tido considerável crescimento, é o *Unreal Tournament* (UT) [UNREAL, 1999], em que existe a possibilidade de utilizar seu ambiente para utilização de *bots* (agentes) criados por um desenvolvedor externo, pois permite que programas externos se conectem a ele para controlá-los, sendo que esses *bots* podem ser desenvolvidos em qualquer linguagem que tenha suporte para troca de mensagens via rede, através do servidor do UT [ADOBBATI et al., 1999].

Este trabalho, portanto, mostra como a IA é aplicada a jogos de FPS, descrevendo como um *bot* pode ter habilidades comparáveis com as habilidades humanas. Será descrito como são desenvolvidas as técnicas de IA nestes tipos de jogos através do desenvolvimento de um agente com as habilidades necessárias para jogar uma partida de *Deathmatch* (onde todos os jogadores tentam acumular pontos 'matando' outros jogadores, até conseguir uma pontuação máxima ou até o tempo acabar) e que possa interagir com o ambiente do *Unreal Tournament*. Para este desenvolvimento foi utilizado o projeto *Gamebots* [ADOBBATI et al., 1999] e a técnica Máquina de Estado Finito (FSM).

As seções seguintes revisam resumidamente alguns conceitos da IA aplicados em jogos de FPS e o projeto *Gamebots*, para posteriormente descrever a técnica FSM. Seguindo com as dificuldades encontradas em se construir um protótipo utilizando o *Gamebots*, juntamente com os resultados alcançados até o momento e a conclusão.

2. Principais Conceitos de IA Aplicados em Jogos de Tiro em Primeira Pessoa

Dentro dos conceitos da Inteligência Artificial aplicados em jogos de FPS existem quatro componentes principais que são o Comportamento, a Movimentação, a Animação, e o Combate.

2.1 Movimentação

O componente de movimento é designado para determinar como o agente deve se mover no ambiente do jogo. O movimento é o que faz o agente evitar obstáculos, seguir a navegação do sistema de nós, e achar caminhos pelos complexos ambientes até alcançar seu destino. O subsistema de movimento nunca determina para onde mover, só como fazer. Ele simplesmente recebe comandos de outros componentes que dizem para onde mover, e é responsável por fazer o agente se mover para o ponto especificado de uma maneira apropriada.

Fundamental para o componente de movimentação é o sistema de **Busca de Caminhos**. Este sistema é responsável por achar o caminho de uma coordenada de localização qualquer para uma outra. Dado um ponto de partida, uma designação de estado, e uma meta ou destino, ele achará uma série de **pontos de caminhos** que incluem um caminho ótimo para algum destino intermediário. É possível que ele informe que nenhum caminho pode ser achado, ou gerar um erro aparente em seu repertório de movimentos humanos simulado. O Buscador de Caminhos do jogo *Unreal Tournament* usa uma estrutura de dados pré-computada para guiar os movimentos. Este é um algoritmo de controle complexo que constrói uma espécie de lista ligada chamada de **Pontos de Navegação** (*Navigation Points*) que fazem parte do sistema de navegação do *bot*. O ambiente de um jogo de FPS é relativamente estático, assim faz sentido pré-gerar um banco de dados que é altamente aperfeiçoado para executar uma busca de caminhos mais rápida em uma seção particular do ambiente. Na execução, o banco de dados da busca de caminhos é inicializado e parametrizado O desempenho deste sistema direto de busca de pontos de navegação está altamente correlacionada a como o banco de dados de busca é otimizado [UNREAL WIKI, 2002].

A Figura 1 mostra alguns tipos de pontos de navegação do UT, onde podem ser vistos os *PathNodes* (Nós de Caminhos) que são utilizados pelos *bots* para poderem se locomover pelo ambiente, e é um conjunto desses Nós de Caminhos que é retornado para o *bot* quando o mesmo solicita um caminho qualquer para o sistema de Busca de Caminhos. Outro tipo de ponto de navegação é o *InventorySpot* (Nó de Inventário), onde se localizam todos os itens de inventário que o *bot* pode pegar no chão como armas, munições, armaduras, etc. Um outro tipo de ponto de navegação que pode ser visto na Figura 1 é o *PlayerStart* (Ponto de Nascimento), que são pontos definidos dentro do ambiente, onde os *bots* podem nascer, ou seja, aparecer no ambiente. Existe um atributo nesse tipo de ponto de navegação que indica se ele será usado para um jogo de time ou não. Ou seja, se esse

ponto for usado para jogo de time (onde os *bots* se dividem em times para poderem cumprir determinados objetivos), somente nascerá nesse ponto os *bots* que forem do time especificado pelo atributo.

Uma vez feito isto, podemos projetar os algoritmos apropriados para executar os comandos de movimento em comandos de movimento individual. Um comando como, "mova (X,Y,Z)", por exemplo, será responsável por usar os sistemas de busca de caminhos globais e locais para achar e executar um caminho para (X,Y,Z), fazendo com que o mecanismo (*engine*) apropriado seja chamado para assegurar que o agente se mova para o caminho especificado. Se nenhum caminho estiver disponível, ou quando o agente alcançar o fim de seu caminho, o comando de movimento informará de volta ao controlador de movimento que movimento foi executado.

Esta técnica também é útil para manusear tipos diferentes de movimentos. Tipos diferentes de comandos de movimento podem manusear tarefas como caminhar, correr, nadar, e morrer, com parâmetros apropriados para

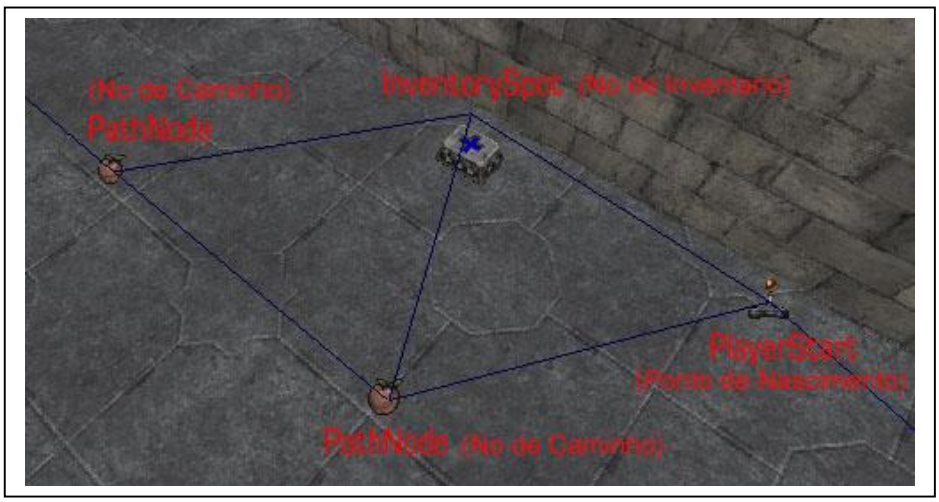

Figura 1: Diferentes tipos de Pontos de Navegação com os caminhos entre eles, mostrado pelo editor de mapas do *Unreal.*

determinar a aceleração do agente, alcance de movimento, taxa de giro, animações, e outras características de movimento [UNREAL WIKI, 2002].

2.2 Animação

O componente de animação é responsável por controlar a articulação do esqueleto do *bot* no jogo e a exibição durante o jogo. Seu papel principal é selecionar, parametrizar, e executar sucessões de animação do agente. Também é responsável por gerar animações que não podem ser executadas por sucessões de animação pré-estabelecidas. [UNREAL WIKI, 2002].

2.3 Combate

O componente de combate é responsável por avaliar a situação tática atual do agente, selecionar táticas de combate, mirar e atirar nos oponentes, decidir quando apanhar uma arma nova, e assim por diante. Considerando que o combate é o centro de atenção na maioria dos jogos de FPS, o desempenho deste subsistema será crucial à percepção do agente de IA.

Quando o mecanismo de IA inicia o combate, o controle da maioria de seus comportamentos é transferido a um componente de controle de combate. O controlador de combate é responsável por todas as tarefas relacionadas a combate, como selecionar um oponente, selecionar uma arma, manobrar, atirar, e procurar armas adicionais e munições.

A parte mais difícil do problema de combate é determinar como avaliar a situação atual inteligentemente, selecionar e executar uma tática apropriada em resposta [UNREAL WIKI, 2002].

2.4 Comportamento

O componente de comportamento é o *framework* integrado que determina a meta atual do agente, ordens, estado, destino imediato, e a comunicação com os outros subsistemas para coordenar o movimento físico para uma determinada área. É um subsistema de alto nível em jogos de FPS e fica acima de todos os outros subsistemas na hierarquia dos componentes de IA em jogos de FPS.

Existe um grande número de modos para modelar um controlador de comportamento e depende das exigências de requerimentos do jogo. A maioria dos jogos de FPS usa uma Máquina de Estado Finito (FSM – *Finite State Machine*) para esta parte de IA.

Os comportamentos são, cada, um representado por um objeto que é responsável por se comunicar com os subsistemas de movimentação, animação e combate para representar os comportamentos adequadamente. Desenvolver estes comportamentos é muito fácil, desde que os outros componentes já estejam prontos e que disponibilizem uma rica quantidade de comportamentos básicos para o componente de comportamento assimilar [UNREAL WIKI, 2002].

2.4.1. Máquina de Estado Finito

Uma Máquina de Estado Finito (*Finite State Machine* - FSM) ou Autômato Finito (AF) é um sistema que tem um número limitado de estados de operações. Um exemplo real pode ser um interruptor, que pode estar ligado ou desligado, ou um relógio de alarme que pode estar parado contando o tempo ou soando um alarme. Qualquer sistema que tem um número limitado de possibilidades onde algo pode ser definido através de um estado (até mesmo combinações) pode ser representado como uma máquina de estado finito [HOWLAND, 1999].

Segundo Brownlee [200_?], as FSM são compostas de 4 elementos principais:

- **Estados** - Que definem comportamento e podem produzir ações.
- **Transições de Estado** - Que são movimentos de um estado para outro.
- **Regras ou Condições** - Que devem ser conhecidas para permitir uma transição de estado.
- **Eventos de Entrada** - Que são gerados externamente ou internamente, e que podem ativar regras, podendo conduzir a transições.

FSM é tipicamente usado como um tipo de sistema de controle onde o conhecimento é representado nos estados, e ações são controladas por regras.

A técnica FSM é uma técnica de Inteligência Artificial que se originou nos campos da matemática e teoria da computação, inicialmente usada para representação de linguagens. É intimamente ligada a outras técnicas fundamentais de representação de conhecimento, como as **Redes Semânticas** e uma extensão destas, chamada de **Espaço de Estado** [BROWNLEE, 2000].

Como qualquer sistema baseado em regras, se todos os antecedentes de um nó são verdadeiros, então a regra é ativada. É possível que múltiplas regras sejam disparadas, ocorrendo um conjunto de conflitos. Pode haver somente uma transição do estado atual, assim uma estratégia de resolução de conflitos consistente é exigida para selecionar uma única regra de ativação para executar uma transição de estado.

Isto nos traz para os dois tipos principais de FSM. A simples FSM original que é conhecida como **determinística** e significa que, dado uma entrada e um estado atual, a transição pode ser sempre prevista. O oposto é uma FSM não-determinística. Isto é, dado uma entrada e um estado atual, a transição nem sempre pode ser prevista. Pode ser o caso de múltiplas entradas recebidas em vários momentos, significando que a transição do estado atual para outro estado não pode ser conhecida até que a entrada seja recebida (evento dirigido).

Há dois métodos principais que são usados para gerar as saídas para uma Máquina de Estado Finita. Eles são conhecidos como a **Máquina de Moore** e a **Máquina de Mearly** [BROWNLEE, 2000].

A Máquina de Moore é um tipo de FSM onde as saídas são geradas como produtos dos estados. A Figura 2 exemplifica o estado **O que Fazer**, tal como ligar a luz, onde a ação ligar a luz é uma saída do estado ligada.

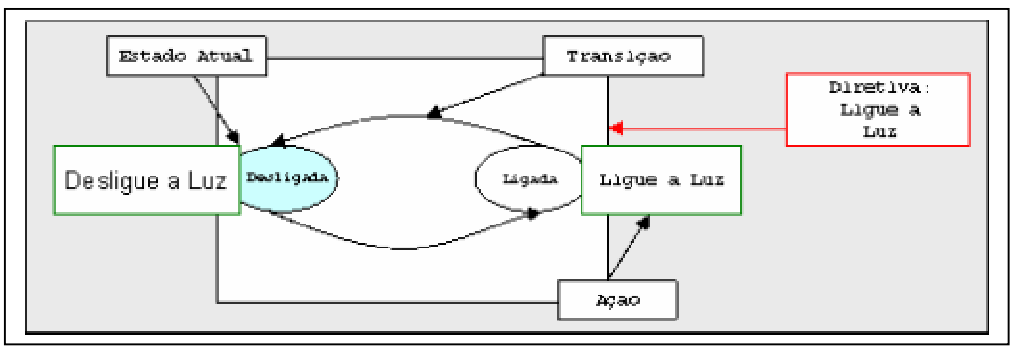

Figura 2: Máquina de Moore.

Uma Máquina de Mearly, diferente da Máquina de Moore, é um tipo de FSM onde as saídas são geradas como produtos da transição entre estados. Na Figura 3, a ação de ligar a lâmpada é afetada pelo processo de troca entre os estados.

Como toda Técnica, existem vantagens e desvantagens de utilização das FSM. Segundo Brownlee [200_?], as vantagens e desvantagens são:

Vantagens das FSM:

- A **simplicidade** facilita o trabalho de desenvolvedores sem experiência, que implementam com uma quantidade pequena de conhecimento extra.
- **Previsibilidade** (em FSM Determiníticas), dado um conjunto de entradas e um estado atual conhecido, a transição de estados pode ser prevista, permitindo um teste fácil.
- Devido à sua simplicidade, FSMs são **rápidas de se projetar**, **rápidas de se implementar** e **rápidas na execução**.

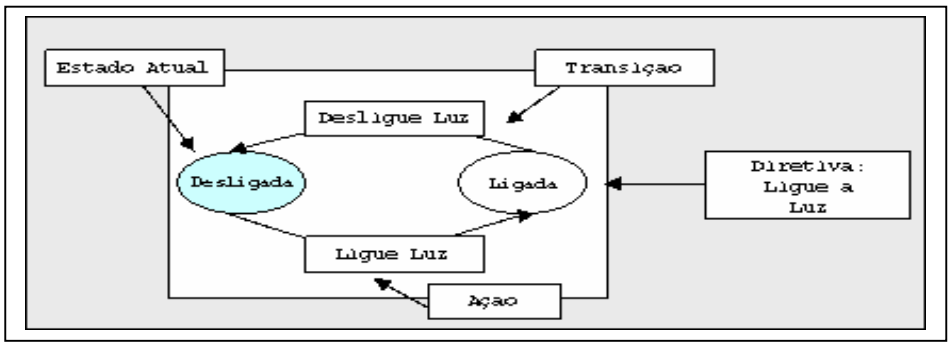

Figura 3: Máquina de Mearly.

- **Baixo nível de processamento**; bem aplicada a domínios onde o tempo de execução é compartilhado entre módulos ou subsistema. Somente o código para o estado atual necessário é executado, e talvez uma quantia pequena de lógica para determinar o estado atual.

Desvantagens da FSM:

- A **natureza previsível** das FSMs determinísticas pode não ser desejável em alguns domínios como jogos de computador (solução pode ser FSM Não-Determinísticas).
- **Não adaptável** a todos os domínios de problema, somente deve ser usado quando o comportamento do sistema pode ser decomposto em estados separados com condições bem definidas para as transições de estado. Isto significa que todos os estados, transições e condições precisam ser conhecidos e bem definidos.

Esta é a técnica mais simples de jogos de IA. Ela é freqüentemente usada em Jogos de Tiro em Primeira Pessoa como *Quake* e *Unreal*. Mostraremos agora, um exemplo de FSM de um jogo de FPS criado pela *Id Software* chamado de *Quake*. A análise se dará em cima deste, mas os mesmos conceitos podem ser usados para outros FPS, como *Unreal* e *Half Life*. Isto porque, desde a ocasião em que *Quake* foi lançado, o seu código foi autorizado para ser usado por outras companhias que produziram estes títulos altamente populares, provando o sucesso do produto original e da técnica de FSM [MATTHEWS,2000].

A Figura 4 mostra a FSM de um foguete no Quake. Um foguete em Quake é um projétil lançado de uma arma chamada Rocket Launcher que pode ser possuída e operada por um jogador humano.

A FSM da Figura 4 utiliza uma aproximação bem parecida com um Diagrama de Transição de Estado. As caixas azuis são os estados, as laranjas, os gatilhos e as setas são as transições de estado. As caixas pretas são os pontos de entrada e saída do sistema.

O diagrama mostra o ciclo de vida completo do projétil de foguete no jogo. É interessante notar que o projétil é criado como o produto de uma ação de outra FSM, chamada de *Rocket Launcher* em sua **Ação de Tiro**. Quando a instância do projétil desaparece, ela é removida do jogo, e já não mais existe.

Esta representação é uma interpretação subjetiva de código. Outra representação válida pode ser quebrar o **Estado de Colisão** nos **Estados de Toque** e **Explosão**. Mas aqui, o Estado de Explosão é visto como uma ação ou efeito executado pelo foguete em seu Estado de Colisão [BROWNLEE, 2000].

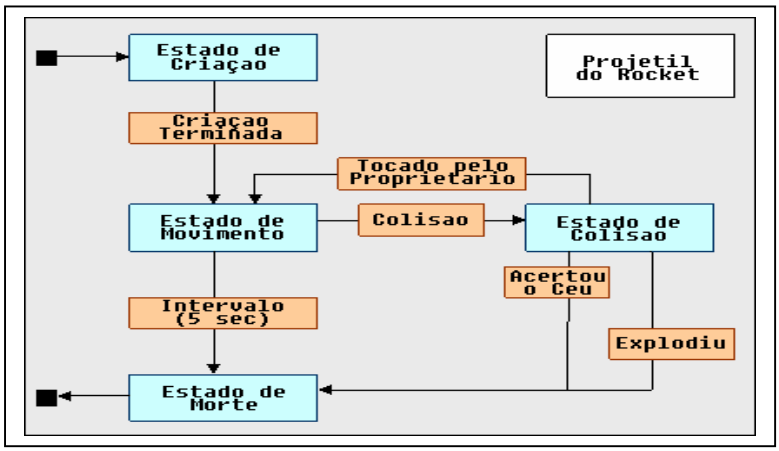

Figura 4: Representação de uma Transição de Estado de um projétil de foguete em *Quake*.

tudo que ele toca. Se tiver sucesso em ferir outro agente no ambiente do jogo, a ação de dano se torna uma entrada que pode ativar uma transição de estado no agente atingido.

Quake faz extenso uso da FSM como um mecanismo de controle para os agentes que existem no ambiente do jogo. Isto é provido por um interessante *framework* que é firmemente relacionado com o modo com que a FSM trabalha em jogos de computadores [BROWNLEE, 2000].

3. Gamebots

Gamebots é um projeto criado por um grupo do Departamento de Ciência da Computação da Universidade de *Carnegie Mellon* em *Pittsburgh* que consiste em uma modificação do *Unreal Tournament* [UNREAL, 1999] para permitir que um *bot* possa ser controlado via *sockets* de rede conectados a outros programas. Este tipo de modificação é perfeito para uso do UT como um ambiente para pesquisas de IA com agentes inteligentes. O propósito da arquitetura dos *JavaBots* (*bots* feitos em *java*) e a API de conexão é prover um nível de interface mais elevado para o protocolo de modificação do *GameBots* e tornar isto mais facilmente accessível para grupos de pesquisas poderem desenvolver agentes/*bots* para este rico domínio. *JavaBots* podem ser criados estendendo-se classes *java* e trabalhando com objetos sem ter que se preocupar com protocolos de redes, análise de mensagens, etc [SOURCEFORGE, 2002].

O centro do projeto *Gamebots* é o módulo para UT que permite que um *bot* possa ser controlado por *sockets* de rede conectados aos clientes de *bot*, como visto na Figura 5. O *Gamebots* provê informações sensoriais para os *bots* da conexão de rede. Baseado nesta informação, o cliente (*bot* ou jogador humano) pode decidir qual ação deve tomar e enviar comandos de volta para o servidor para fazer com que o *bot* se mova, atire, ou troque mensagens. Agentes exibem avançadas técnicas de IA para jogar prosperamente, como planejar caminhos, aprender um mapa do ambiente 3D, usar recursos disponíveis, coordenar seus membros de equipe, e seguir um planejamento estratégico que leva em conta seus adversários. O sistema do *Gamebots* permite que jogadores humanos joguem com os agentes, provendo, assim, oportunidade de estudar o comportamento do time humano, e construir agentes que joguem colaborativamente com humanos.

O protocolo de interação do *Gamebots* é um protocolo de textos simples de mensagens enviadas pela rede entre o servidor e os *bots*, utilizando protocolos UDP e TCP. O servidor envia informações sensoriais para os *bots* que contêm o estado atual do ambiente. Os bots interagem no ambiente enviando comandos ou mensagens de comunicação para o servidor.

Através do *Gamebots* é possível ter acesso às mensagens sensoriais enviada ao *bot* pelo ambiente do *Unreal Tournament.* As mensagens sensoriais são divididas em dois tipos, **Síncronas** e **Assíncronas**.

Mensagens Síncronas vêm em grupos, a um intervalo configurável. Estas incluem coisas como uma atualização visual do que o *bot* vê e um relatório do estado do próprio *bot*. No começo de cada grupo, o servidor transmite uma mensagem de “BEG” (que marca o início do grupo) seguida com um horário. Todos as mensagens recebidas até a mensagem “END” com o mesmo horário são parte de um grupo de mensagens síncronas. Elas são todas enviadas no mesmo momento do jogo e se referem a um único estado discreto do jogo.

Mensagens Assíncronas vêm de acordo com os acontecimentos de eventos no jogo (elas nunca aparecerão entre um "BEG" e um “END"). Eles representam coisas que podem acontecer ao acaso em qualquer ponto do jogo, em intervalos menos freqüentes como ser ferido, ou um recebimento de mensagem de outro jogador, ou na colisão com uma parede [GAMEBOTS, 2002].

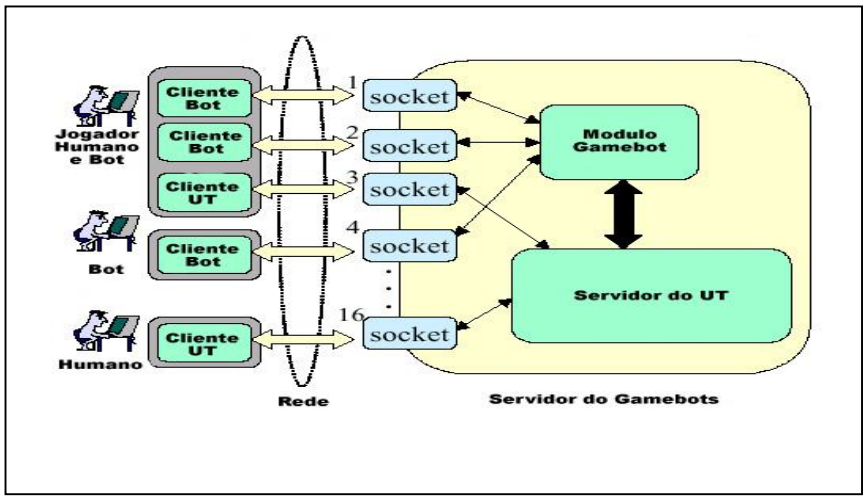

Figura 5: Arquitetura do *Gamebots* (Os jogadores humanos se conectam diretamente com o servidor do UT, enquanto que os *Bots* se conectam a ele através do Módulo do Gamebots)

4. Desenvolvimento de um agente que joga *Deathmatch* no *Unreal Tournament*

O objetivo da implementação deste protótipo é demonstrar uma técnica de IA em um Jogo de Tiro em Primeira Pessoa, mostrando assim, que é possível utilizar ambientes de jogos para pesquisas de comportamentos de *bots*.

O protótipo utiliza a técnica de IA chamada de Máquina de Estado Finito (FSM), sendo a técnica mais utilizada em jogos FPS por ser uma técnica simples e de fácil implementação. Os *bots* foram desenvolvidos na linguagem *Java*. O protótipo utilizou a API fornecida com o *Gamebots* para fazer comunicação entre o bot e o servidor do UT.

Para a implementação do protótipo foi utilizado um computador com processador AMD *Athlon Thunderbird* de 1.2 MHz, com 786MB de memória RAM e uma placa de vídeo *GeForce* 2 MX 400 com 64 MB de memória de vídeo. Para a troca de mensagens entre o bot e o servidor do UT foi utilizado *sockets* de rede.

Os softwares utilizados nesse protótipo foram: Linguagem de Programação Java, Jogo *Unreal Tournament*, Biblioteca de Classes do *Gamebots*, Programa Tcl/Tk, VizClients. O sistema Operacional foi o *Windows* 2000 para a execução da implementação e realização dos testes.

4.1 Modelagem do Agente

O agente proposto executará suas ações no tipo de jogo do UT chamado de *Deathmatch*. Este tipo de jogo é o mais simples, onde todos os jogadores tentam acumular pontos 'matando' outros jogadores, até conseguir uma pontuação máxima ou até o tempo acabar. Assim, é muito importante que o agente tenha uma personalidade agressiva de forma a conseguir eliminar a maior quantidade de adversários possíveis para obter a pontuação máxima e vencer o jogo.

A Tabela 1 mostra as mensagens e comandos utilizados pelo agente no protótipo e a Figura 6 mostra a Máquina de Estados (FSM) Básica do Agente.

O agente guarda os pontos mais importantes do ambiente como vidas (Medkits), armas, munições e armaduras, para que na próxima vez que entrar no ambiente, o agente já esteja ciente de onde os mesmos se localizam.

De acordo com a FSM descrita na Figura 6, o agente entrará no ambiente já procurando por uma arma, indicado pelo estado inicial do agente **Pegando Arma**. Após pegar a arma, o agente entrará no estado **Explorando**, onde o mesmo pegará os itens que encontrar pelo caminho (como munições, armaduras e recursos de vida) e guardará os pontos de localização de cada item. Como descrito na sua FSM, o agente apenas trocará para o estado **Pegando Item,** se precisar do mesmo, pois pode acontecer que o agente veja um recurso de vida, por exemplo, mas sua vida esteja completa. No caso de ele ver uma munição, ele verificará se a arma que utiliza aquela munição está completa. Se estiver, o agente dará um tiro para descarregar uma parte da munição e pegará a outra no chão, completando-a. Isto acontecerá para não se deixar recursos de artilharia para o inimigo, simulando assim, uma habilidade humana neste tipo de jogo. O recolhimento dos itens se fará presente em toda a permanência do agente no ambiente.

Durante a movimentação pelo ambiente, o agente prestará atenção aos sons e ruídos que o oponente poderá fazer, como quando ele pegar algum item ou subir por um elevador. Caso escute alguma coisa, o agente deverá

realizar uma procura visual a partir do ponto em que ele escutou o som, ou seja, deverá realizar uma rotação em seu eixo a fim de determinar de onde o oponente virá, no sentido de receber uma percepção visual do mesmo. Caso não receba nenhuma percepção visual, o agente continuará pelo seu caminho. Será feito dessa forma pois o ambiente não fornece a localização e direção de onde veio o som e sim a indicação de que ocorreu o evento. Fazendo isso, ele estará no estado **Caçando**. Quando o agente conseguir ver o oponente, entrará no estado de **Lutando**. Caso o *bot* não consiga localizar visualmente o inimigo, voltará para o estado **Explorando**.

No estado **Lutando**, o agente terá uma personalidade agressiva, ou seja, sempre irá tentar matar o oponente a qualquer custo, de acordo com o objetivo do tipo de jogo *Deathmatch*. Se o oponente tentar fugir, o agente irá atrás do mesmo para tentar mata-lo. A precisão da mira é ajustada pelo *Engine* do UT, sendo preciso apenas dar as coordenadas x, y, z para o mesmo, ou seja, a localização do oponente. Após o *bot* conseguir matar o oponente, verificará se precisa de vida. Se precisar, o agente entrará no estado **Pegando Vida**, onde o mesmo procurará por um ponto de recuperação de vida conhecido, mais próximo a ele. Quando o agente conseguir recuperar sua vida ou se verificar que não precisa de vida, voltará ao estado **Explorando**.

Tipo	Mensagem	Significado
Mensagem Assíncrona	NFO	Informação sobre o jogo dada logo após a conexão com o servidor. O *bot* deve esperar por essa mensagem antes de mandar o comando INIT.
	AIN	O *bot* pegou um novo item para o inventário.
	HRN	*Bot* ouviu um barulho no ambiente. Talvez um outro jogador andando ou atirando, uma bala batendo na parede ou apenas o som de um elevador subindo e descendo.
	DIE	O b*ot* morreu.
	PTH	Um resultado boleano de um comando de checagem, verificando se o *bot* pode ou não chegar diretamente ao ponto.
Mensagem Síncrona	SLF	Informações sobre o estado do *bot*.
	PLR	Outro personagem no jogo.Apenas reporta aqueles que são visíveis (no alcance de visão e não obstruídos).
	NAV	Um ponto de navegação (*Pathnodes*) no jogo. *Pathnodes* são objetos invisíveis (pelo menos para humanos) colocados pelo nível para definir caminhos que os *bots* podem seguir. Eles provêem uma rede totalmente conectada que se estende por todo o ambiente.
	INV	Um objeto no chão que pode ser pego.
Comandos	INIT	Comando para poder inicializar o *bot* no ambiente mandada depois da mensagem NFO.
	STOP	Para todos os movimentos do *bot*.
	RUNTO	Faz com que o *bot* se mova para uma determinada localização.Especificada pelo Id do alvo ou localização.
	TURNTO	Especifica um ponto, valor de rotação ou um objeto para poder mudar de direção.
	ROTATE	Se vira em uma determinada direção.
	SHOOT	Atirar
	STOPSHOOT	Para de atirar.
	GETPATH	Pede uma lista de *Pathnodes* para poder chegar a um destino.

Tabela 1: Mensagens e Comandos utilizados pelo Agente.

Outro detalhe importante a se notar é que em qualquer estado que o agente esteja, ele pode sofrer ferimentos que acabem com sua vida e morrer, pois durante a execução de suas ações, pode encontrar um inimigo e este pode matá-lo. Acontecendo isso, a partir de qualquer estado o agente pode ir para o estado **Morte**, renascendo em outra parte do ambiente.

4.2 Resultados Obtidos

O Agente foi testado exaustivamente no ambiente do UT dentro de vários mapas, simulando satisfatoriamente as habilidades de um jogador humano.

Foi comprovado que a técnica FSM é uma técnica bastante eficiente para lidar com as situações impostas pelo ambiente, onde cada ação foi determinada por um estado.

Na implementação do agente, foi observado que os 4 componentes de uma FSM (estados, transições de estados, regras e condições e eventos de entradas) têm que ter seus usos bem definidos para tornar o agente eficiente.

Foi observado também que, como os agentes são implementados em *Java*, utilizando a máquina virtual, os mesmos perdem performance de execução em máquinas de baixo processamento e memória, devido também a grande troca de mensagens entre o servidor e a API *Java* de conexão.

As limitações que o agente encontrou no ambiente e as influências que o mesmo causou nas decisões do agente foram as seguintes:

- Limitações visuais com relação aos pontos de navegação. Se o *bot* chegava a um lugar e não via um outro ponto de navegação, o mesmo não sabia para onde ir.

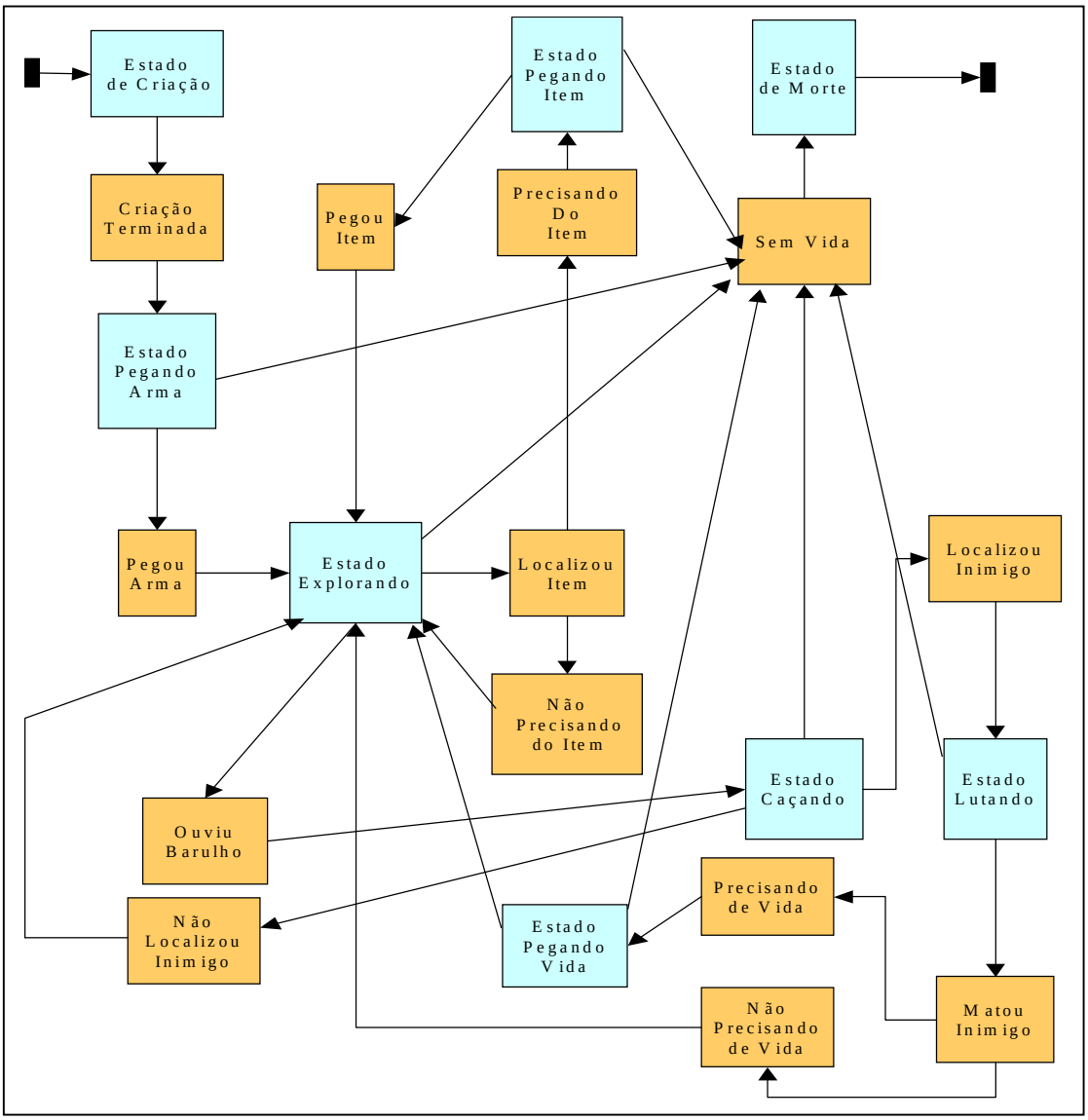

Figura 6: Máquina de Estado do Agente no *Deathmatch*.

- Limitações auditivas, pois o agente escutava o barulho, mas não sabia de onde estava vindo, tendo que fazer uma busca visual do ponto em que estava.
- Houve pequenas limitações com relação ao manuseio de elementos no ambiente (portas e elevadores), onde o agente não conseguiu manusear habilmente tais elementos.
- O agente tem uma certa limitação com relação ao eixo ‘Z’ das coordenadas 3D que representa a altura, não realizando pulos eficazmente no ambiente.

De acordo com o que foi observado durante a execução do agente no ambiente, a movimentação dentro do mesmo foi realizada de maneira bem satisfatória. Ocorreram apenas alguns problemas com relação a alguns pontos de navegação em alguns ambientes, pois existiam pontos que ficavam em lugares fechados e quando o agente entrava nestes lugares, demorava muito para sair e às vezes não conseguia nem sair. Isso acontecia porque quando o agente olhava o ponto de fora para dentro do lugar fechado, o mesmo via normalmente o ponto e continuava no seu curso, mas quando já estava dentro do espaço fechado, não conseguia ver outros pontos de dentro para fora deste espaço.

O manuseio realizado pelo agente com relação aos elevadores foi realizado de maneira satisfatória com pequenas limitações. Estas limitações se resumiam apenas a elevadores que subiam e desciam muito rápidos. Com a maioria dos elevadores o agente conseguiu usa-los bem. Já o manuseio de portas foi comprometido, pois as portas do ambiente do UT são automáticas e se abrem quando um agente está perto da mesma, ou seja, se não existisse nenhum ponto de navegação próximo à porta, o mesmo não conseguia abri-la para tentar sair de algum lugar, como foi observado em um ambiente. Neste ambiente, existe um corredor limitado por duas portas, uma em cada lado do corredor, sem mais nenhuma saída. Dentro do corredor há um ponto de nascimento e quando o *bot* nascia dentro do corredor ficava andando de um lado para o outro, dentro do corredor sem conseguir sair do mesmo, pois os pontos de navegação que existiam dentro do corredor não estavam próximos à porta. Isto acontecia também quando o agente estava do lado externo e conseguia entrar no corredor, pois do lado externo existem pontos de navegação próximos à porta.

Durante o combate o desempenho do agente foi muito bom, onde o mesmo conseguia matar os inimigos com eficiência e conseguia perseguir o inimigo de maneira satisfatória, caso o mesmo tentasse fugir.

A Tabela 2 mostra os dados estatísticos de sucesso de execução, com relação aos componentes de um jogo FPS aplicados no agente dentro do ambiente do UT. É importante observar que o agente obteve 100% de sucesso com relação ao componente de comportamento, pois o agente conseguiu ter uma postura agressiva e agir agressivamente em busca dos seus objetivos.

A Tabela 3 mostra mais alguns dados estatísticos de sucesso com relação às habilidades humanas que o agente simulou, como a estratégia, a utilização de elementos no ambiente e suas capacidades visuais e auditivas. Pode-se perceber que as limitações citadas anteriormente estão refletidas nestes dados, como no uso de portas ou na habilidade auditiva, por exemplo.

Estes dados foram gerados a partir do rastreamento da execução do agente, verificando o percentual de sucesso nas diversas ações executadas nas situações impostas pelo ambiente.

Componentes	Sucesso (%)
Movimentação	90
Animação	100
Combate	90
Comportamento	100

Tabela 2: Sucesso de Execução dos Componentes de FPS no UT Aplicados no Agente.

Habilidades	Sucesso (%)
Estratégia	90
Uso de Elevadores	95
Uso de Armas	88
Uso de Portas	35
Perseguição Durante o Combate	97
Visuais	100
Auditivas	70

Tabela 3: Sucesso de Execução com Relação às Habilidades do Agente.

4.3 Dificuldades Encontradas

Durante a implementação do agente e sua execução ocorreram problemas com relação às mensagens trocadas entre o agente e o ambiente e com o uso de algumas ferramentas. Tais problemas são relatados a seguir:

- O comando CHANGEWEAPON (parar trocar de arma) não funciona direito. Algumas vezes executa e outras não. Quando executa, funciona durante 2 ou 3 trocas de armas e depois para de

funcionar, fazendo com que outros comandos também deixem de funcionar como o comando SHOOT.

- A mensagem BMP (indica quando o agente se bateu com outro) só funciona quando há o primeiro toque entre os agentes na fase, não funcionando mais no decorrer do jogo.
- A mensagem HRP (mensagem para indicar que alguém pegou um item do chão) não funciona, ou seja, mesmo que alguém pegue um elemento no chão atrás do agente o mesmo não escutará.
- A mensagem SEE (mensagem síncrona disparada quando o agente vê outro agente) não funciona, ou seja, o agente só está vendo o oponente quando recebe a mensagem PLR.
- A mensagem PRJ (indica que um projétil de arma pode atingir o agente) não funciona, ou seja, o agente não recebe a mensagem de que há um projétil indo em sua direção, não sendo possível se desviar do mesmo.
- O comando JUMP (para pular) não está funcionando direito. Quando o comando é emitido o agente pula, mas a uma altura muito grande como se não existisse gravidade, ficando flutuando no ar.
- Não foi possível usar a ferramenta de leitura de arquivos de *log* para o uso no estudo do comportamento do agente, pois o formato do arquivo é desconhecido. Foram feitas algumas tentativas, mas sempre dava erro quando se realizava a abertura da ferramenta.
- Depois de um certo tempo de funcionamento as mensagens trocadas entre o servidor do UT e o agente começam a dar conflitos com relação ao momento de recebimento, fazendo com que o agente congele em determinados momentos.

Dentre estas dificuldades, aquelas que foram identificadas como *bugs* foram reportadas aos desenvolvedores do projeto *Gamebots*, mas nenhuma correção ou solução foi apresentada até a conclusão deste trabalho.

5. Conclusões

O objetivo deste trabalho foi estudar e demonstrar como técnicas de IA podem ser utilizadas em Jogos de Tiro em Primeira Pessoa. Com o estudo mais aprofundado da técnica Máquina de Estado Finito, foi mostrado e comprovada a eficácia desta técnica nos jogos de FPS. Isso se deve à sua simplicidade de entendimento e implementação, pois todas as ações e decisões tomadas em um jogo de FPS, se baseiam em um conjunto limitado de estados e suas transições, onde de acordo com uma percepção recebida do ambiente o agente modifica o seu estado para executar uma determinada ação. Através do estudo e a implementação de um agente usando esta técnica, foi possível provar a eficácia desta e mostrar que é possível fazer com que um agente consiga, com certas restrições, simular as habilidades e comportamentos humanos em um jogo.

Com a demonstração de uma implementação de um agente utilizando a técnica FSM, foi possível abrir novos caminhos para o estudo da simulação de habilidades humanas utilizando os jogos como ambientes e a partir do que foi feito, ficam algumas sugestões para trabalhos futuros:

- Realizar testes mais aprofundados com o ambiente do UT para deixar mais eficiente a performance do agente no que diz respeito à exploração e uso dos Pontos de Navegação;
- Desenvolver outros tipos de agentes que joguem outros tipos de jogos dentro de um FPS como o UT (*Capture The Flag*, *Assault*, etc);
- Realizar a implementação e testes com relação à personalidade dos agentes, onde a personalidade junto com as percepções do ambiente é que vão determinar quais as ações e decisões que o agente irá tomar;
- Desenvolver agentes com outros tipos de técnicas para fazer um comparativo com relação à eficiência de cada uma;
- A partir do desenvolvimento de agentes para jogos de FPS utilizando outras técnicas, promover um estudo de integração entre elas, desenvolvendo um agente multifuncional que pudesse, de acordo com a situação, escolher qual a melhor técnica que se aplica ao problema encontrado;
- Desenvolver agentes utilizando técnicas de Sistemas Multi-Agentes (SMA) permitindo usar a IA para jogos em times de agentes, tanto cooperativamente quanto entre times adversários;

A partir dos resultados positivos do protótipo, os quais mostraram satisfatoriamente que um agente pode simular comportamentos humanos em diversas condições e tendo certos objetivos a se cumprir, foi possível tirar conclusões evidentes de que é possível utilizar os jogos de computadores para o estudo da Inteligência Artificial. Este fato é de fundamental importância para nós humanos no sentido de que, através deles podemos desenvolver

ambientes virtuais e situações que são perigosas, e colocar os agentes para fazerem o trabalho perigoso, propiciando o estudo e desenvolvimento de soluções para os mais diversos problemas.

6. Referências Bibliográficas

[ADOBBATI et al., 1999] ADOBBATI, Rogelio, MARSHALL, Andrew N., SCHOLER, Andrew, TEJADA, Sheila, KAMINKA, Gal, SCHAFFER, Steven, SOLLITTO, Chris. **Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research**. [on line]. [199_?]. Disponível na Internet via WWW. URL: <www.cs.cmu.edu/~galk/Publications/01/gamebots.pdf >. Acesso em 01/02/2002.

[BROWNLEE, 2000] BROWNLEE, Jason. **Finite State Machine (FSM)**. [on line]. [200_?]. Disponível na Internet via WWW. URL: <http://ai-depot.com/FiniteStateMachines/FSM.html> Acesso em 06/09/2002.

[GAMEBOTS, 2002] GAMEBOTS. **Network API**. [on line]. 2002. Disponível na Internet via WWW. URL: <http://www.planetunreal.com/Gamebots/docapi.html> Acesso em 10/08/2002.

[HOWLAND, 1999] HOWLAND, Geoff. **Basics of Game AI**. [on line]. [199_?]. Disponível na Internet via WWW. URL: <http://www.lupinegames.com/articles/basicai.htm> Acesso em 21/08/2002.

[MATTHEWS,2000] MATTHEWS, James. **AI in Gaming**. [on line]. [200_?]. Disponível na Internet via WWW. URL: <http://www.generation5.org/app_game.shtml> Acesso em 12/07/2002.

[SOURCEFORGE, 2002] SOURCEFORGE. **JavaBot For Unreal Tournament API**. [on line]. 2002. Disponível na Internet via WWW. URL: <http://utbot.sourceforge.net/doc/>. Acesso em 12/08/2002.

[UNREAL, 1999] UNREAL Tournament. Windows 9x, 2000 e XP. Epic Games; Infogrames; Digital Extremes, [199_?]. 2 CD-ROMS: son. color+manual de informação para Pentium, AMD e compatíveis.

[UNREAL WIKI, 2002] UNREAL WIKI. **Introduction To Game AI**. [on line]. 2002. Disponível na Internet via WWW. URL: <http://wiki.beyondunreal.com/wiki/Introduction_To_Game_AI> Acesso em 20/08/2002.