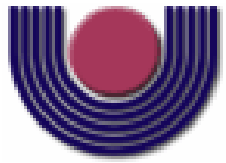

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Informática

Curso de Bacharelado em Informática

Um jogo de nave evolutivo

Yuri Rodrigues Guimarães

CASCAVEL

2009

YURI RODRIGUES GUIMARÃES

UM JOGO DE NAVE EVOLUTIVO

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Jorge Bidarra

Co-orientador: Prof. Marcio Seiji Oyamada

CASCAVEL

2009

YURI RODRIGUES GUIMARÃES

UM JOGO DE NAVE EVOLUTIVO

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Informática*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Jorge Bidarra (Orientador)
Colegiado de Informática, UNIOESTE

Prof. Marcio Seiji Oyamada (Co-orientador)
Colegiado de Informática, UNIOESTE

Prof. Claudia Brandelero Rizzi
Colegiado de Informática, UNIOESTE

Cascavel, 23 de novembro de 2009.

DEDICATÓRIA

Dedico este trabalho a toda minha família e amigos que sempre me ajudaram, e a todos que tem um sonho e lutam por ele.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

MMORPG	*Massive Multiplayer On-line Role-Playing Game*
ABRAGAMES	Associação Brasileira de Jogos Eletrônicos
IAD	Inteligência Artificial Distribuída
KSE	*Knowledge Sharing Effort*
FIPA-ACL	*Foundation for Intelligent Physical Agents – Agent Communication Language*
RBS	*Rules Based Systems*
RBC	Raciocínio Baseado em Casos
EI	Entidades de Informação
NPC	*Non-player Characters*

Sumário

Resumo

O desenvolvimento de jogos computacionais não é uma tarefa trivial e, portanto, demanda muito esforço e trabalho. Empresas de jogos que utilizam Inteligência Artificial, atualmente procuram definir agentes que se comportem como humanos, pois, segundo [Schwab, 2004], os jogadores demandam melhores inimigos. Segundo [Osório, 2007], uma abordagem para a solução do problema é utilizar o aprendizado para "aprender a "imitar" o comportamento humano em um jogo". Logo, neste trabalho, são descritas técnicas de Computação Gráfica, Inteligência Artificial, entre outras, voltadas ao desenvolvimento, especificação e implementação de jogos, com o objetivo de criar um jogo de nave evolutivo, que aprenda a partir das ações tomadas pelo jogador. O jogo apresenta dois modos: o Controlado e Assistido. O modo Controlado alimenta uma base de dados e serve para guiar a nave no modo Assistido, desenvolvendo um agente capaz de se movimentar de forma adequada ao jogo desenvolvido.

Com a utilização das técnicas de Máquina de Estados, Raciocínio Baseado em Casos (RBC) e seleção de casos armazenados na base de dados foi possível definir o comportamento do agente.

Palavras-chave: Agentes computacionais, desenvolvimento de jogos, RBC, aprendizado, jogo de nave.

Capítulo 1

Introdução

Atualmente, vivemos em uma era digital. Com a utilização do computador, as pessoas procuram comunicação, informação e entretenimento. Uma das fontes de entretenimento são os jogos digitais, utilizados por diversão ou até mesmo de forma profissional, por pessoas que participam de campeonatos que chegam a ter níveis internacionais. Portanto, o interesse de desenvolvedores desse software também vem crescendo.

1.1 Objetivo

O principal objetivo deste trabalho é especificar, modelar e implementar um jogo de nave evolutivo. Esse jogo proporciona ao usuário dois modos: Jogador e Assistido. Em modo Jogador, o sistema captura informações da partida enquanto o usuário interage com o jogo (esquivando-se de tiros). As informações relevantes são, então, inseridas em uma base de dados, e serão utilizadas para determinar o comportamento de um agente inserido em um ambiente baseado no escopo de um jogo de nave 2D. A evolução do comportamento desse agente pode ser visualizada no modo Assistido.

Com o desenvolvimento deste jogo, pretende-se adquirir *know-how* na área de Inteligência Artificial.

1.2 Motivação e Justificativa

Estudos bibliográficos realizados na área de jogos evidenciam o atual investimento na área de Inteligência Artificial (IA) [The Elder Scrolls, 2009] [Half-Life 2, 2009] [Osório, 2007]. Segundo [Champandard, 2003], a IA é uma fonte infinita de desafios e estudo sobre como definir o comportamento de um ser inteligente, por meio do uso de computadores. O

desenvolvimento de agentes inteligentes para jogos que façam uso de técnicas de sistema adaptativos e com aprendizado de máquina (*Machine Learning*) [Mitchell, 1997][Rezende, 2003] é um grande desafio na atualidade.

Uma das motivações para o desenvolvimento deste trabalho é a crescente evolução da área de jogos, tão fascinante e interessante. Visto que a IA é uma das áreas de destaque em jogos, foram aplicadas técnicas dessa área para o desenvolvimento do jogo proposto.

1.3 Metodologia

O desenvolvimento desse trabalho, segue a metodologia descrita a seguir:

- Realizou-se uma revisão bibliográfica na área de agentes computacionais, RBC e jogos computacionais. Nesses estudos técnicas utilizadas no desenvolvimento de um jogo 2D foram encontradas.
- A partir dessa revisão sobre agentes e identificação das técnicas apropriadas ao desenvolvimento do trabalho, foi obter o embasamento teórico necessário à sua implementação.
- Por fim, desenvolveu-se a modelagem, documentação e implementação do trabalho.

1.4 Estrutura do Texto

Nos capítulos seguintes o trabalho encontra-se dividido em: Empresas de Jogos e Instituições Acadêmicas (Capítulo 2), onde são apresentados os seguimentos do ramo comercial de jogos, empresas e instituições acadêmicas e jogos desenvolvidos; Uma revisão bibliográfica com técnicas de desenvolvimento (Capítulo 3) associadas ao escopo de jogos computacionais, apresentando, a definição de um jogo computacional, ciclo de desenvolvimento, técnicas de detecção de cenário e colisão, agentes computacionais; O próximo capítulo (Capítulo 4) descreve o jogo desenvolvido e apresenta sua documentação, apresentando detalhes do trabalho desenvolvido, através de diagramas, documentação e especificação textual; No ultimo capítulo (Conclusão) apresenta-se os resultados deste trabalho, avaliando as técnicas aplicadas durante a fase de implementação.

Capítulo 2

Empresas e Instituições Acadêmicas Desenvolvedoras de Jogos Computacionais

Embora a América Latina não tenha uma grande tradição no desenvolvimento de jogos, esse quadro vem se alterando gradativamente [Alves, 2008]. O mercado de jogos no Brasil está voltado a diversos segmentos: a maior parte dos trabalhos vem sendo feita sobre o segmento tradicional, que é o entretenimento. Outros segmentos vêm ganhando destaque, tais como: *advergames* (jogos com enfoque publicitário), *business games* (simulação de negócios com enfoque pedagógico) e *middlewares* (ferramentas voltadas ao desenvolvimento e manutenção de jogos).

2.1 Empresas do Ramo

O número de empresas de desenvolvimento de jogos que atuam no ramo brasileiro ainda é pequeno [Alves, 2008]. A Associação Brasileira de Jogos Eletrônicos (ABRAGAMES) conta com 29 empresas e 14 instituições corporativas afiliadas, com um total de 43 instituições voltadas ao desenvolvimento de jogos no país [Alves, 2008]. As instituições afiliadas estão distribuídas nos estados de São Paulo (10 instituições), Pernambuco (4 instituições), Sergipe (1 instituição), Rio Grande do Sul (5 instituições), Santa Catarina (3 instituições), Espírito Santo (1 instituição), Minas Gerais (1 instituição), Paraná (3 instituições) e Amazonas (1 instituição). Em uma pesquisa realizada pela ABRAGAMES em 2005, foram registradas 55 empresas no segmento de desenvolvimento de jogos, com uma média de 15 funcionários por empresa. Na época dessa pesquisa, as empresas nacionais, juntas, faturavam em torno de 20 milhões por ano [Alves, 2008].

Em Pernambuco, localizada no Porto Digital, destaca-se a empresa *Jynix Playware* [Jynix, 2009], que atua no mercado desde janeiro de 2000, focando na criação de produtos

inovadores, para a satisfação do cliente. O desenvolvimento de projetos na empresa utiliza metodologias ágeis, como *Scrum* e *eXtreme Programing* (XP).

A empresa *TechFront* está estabelecida em Curitiba-PR e Florianópolis-SC e é considerada o maior estúdio de desenvolvimento de jogos do Brasil na atualidade, sendo a única empresa brasileira licenciada para desenvolver jogos para Nintendo DS e Wii, que são videogames muito populares. A empresa tem experiência com a tecnologia Flash, desenvolvimento para Playstation 2 e XBOX, desenvolvimento de *engines* e arte 3D [TechFront, 2009].

Blizzard Entertainment é uma empresa americana que já atua no ramo de jogos desde fevereiro de 1991. Já desenvolveu vários títulos de jogos e atualmente atua no desenvolvimento de jogos do tipo *MMORPG* (*Massive Multiplayer On-line Role-Playing Game*) e jogos de estratégia. Seus principais títulos desenvolvidos atualmente são *Diablo*, *Starcraft* e *Word of Warcraft* [Blizzard, 2009].

Sega é uma empresa desenvolvedora de *software* e *hardware* para jogos, com sua sede principal na cidade de Tóquio, no Japão. Fundada em 1940, iniciou no mercado desenvolvendo consoles e jogos. Seu console de maior sucesso foi o *Sega Mega Drive*, porém a empresa já não atua mais neste ramo desde 2001. Em 2004, a empresa *Sammy* comprou seus direitos pelo valor de 1,1 bilhões de dólares, tornando a *Sega* uma das maiores empresas de desenvolvimento de jogos do mundo [Sega, 2009].

2.2 Jogos Computacionais em Instituições Acadêmicas

Dentre as instituições acadêmicas, uma que merece destaque é a Universidade do Estado da Bahia (UNEB), propondo, em 2003, o projeto de ensino *on-line*, cujo objetivo inicial era contribuir com a criação de jogos com caráter cultural para o campo acadêmico [Manovich, 2005]. Devido à falta de financiamento o ambiente não pôde ser desenvolvido, porém serviu como base para o desenvolvimento do projeto Tríade.

Este projeto contribuiu para a criação de dois jogos voltados para a comemoração dos trinta anos do Pólo Petroquímico da Bahia e produziu um novo jogo sobre a Revolta dos Alfaiates. Sendo financiado pela FAPESB, o projeto encontra-se em etapa inicial, com definição e estruturação do roteiro e das ferramentas de desenvolvimento. O objetivo da criação dos jogos mencionados é que seu processo de desenvolvimento siga a mesma lógica

dos jogos comerciais, proporcionando aos seus jogadores um espaço de aprendizagem escolar de entretenimento.

Uma instituição que também realiza contribuições na área de jogos é a PUC-PR, que promove eventos de jogos [Portal de Jogos Eletrônicos – Inicial, 2009] e disponibiliza tutoriais sobre o desenvolvimento de jogos através da Internet [Tutoria de Jogos – PUC-PR, 2009]. Esse material é desenvolvido para atividades complementares referentes a jogos computacionais e os trabalhos desenvolvidos são apresentados na feira de cursos da faculdade, tendo a possibilidade de serem submetidos à SBGAMES (Simpósio Brasileiro de Jogos e Entretenimento Digital). A PUC-PR oferece, também, um curso de Pós-Graduação na área de desenvolvimento de jogos digitais.

Uma instituição acadêmica pioneira no desenvolvimento de jogos é a UFPE (Universidade Federal de Pernambuco), que teve a primeira disciplina relacionada a esse tema no Brasil [Ramalho, 2009]. Dentre as linhas de pesquisa propostas no mestrado acadêmico, são realizados estudos na área de jogos educacionais, jogos com Inteligência Artificial e Engenharia de Software aplicada a jogos [CIN – UFPE – Pós-Graduação, 2009]. A UFPE já foi sede da SBGAMES, no ano de 2006 [SBGAMES, 2009].

A PUC-RJ apresenta disciplinas oferecidas a nível de graduação na área de jogos: GameAI – IA em Jogos 3D, *Design* de Jogos e Desenvolvimento e *Design* de Jogos 3D [PUC – RJ: Cursos, 2009]. Além disso, a PUC-RJ sediou a SBGAMES em 2009 [SBGAMES, 2009], disponibilizando vídeos das palestras na Internet [PORTAL PUC RIO, 2009].

2.2.1 Os Incentivos para o Desenvolvimento de Jogos Acadêmicos

Em fevereiro de 2006, foi criado um edital do MCT/FINEP/MEC - Jogos Educacionais, que teve o objetivo de selecionar instituições capazes de construir jogos eletrônicos educacionais em sintonia com os parâmetros curriculares nacionais. Esse edital teve grande receptividade por parte das instituições de ensino e pesquisa, que enviaram cerca de duzentos projetos, dos quais foram selecionados treze.

Dentre as instituições selecionadas estavam: Universidade Federal do Pará (UFPA), Universidade Federal do Rio Grande do Sul (UFRGS), Universidade Federal da Bahia (UFBA), Fundação Universidade Federal do Rio Grande (FURG), Universidade Federal do Paraná (UFPR), Universidade do Estado da Bahia (UNEB), Serviço Nacional de Aprendizagem Industrial - Departamento Regional da Bahia (SENAI/BA), Universidade

Federal de São Paulo (UNIFESP), Universidade Federal da Paraíba (UFPB) e Universidade Federal de Minas Gerais (UFMG).

Com o edital, é possível identificar o potencial do Brasil, tanto na discussão teórica sobre jogos, como no processo de desenvolvimento pelas instituições de ensino e pesquisa.

2.3 Jogos de Mercado

Um jogo computacional de destaque desenvolvido no Brasil é o Taikodon, que segue o gênero *MMORPG*. O universo do jogo se passa no século XXIII, simulando o cotidiano da humanidade após ter sido expulsa do planeta Terra por um campo de força que a bloqueou. Nesse jogo é possível participar de batalhas entre naves em tempo real, onde seu personagem pode assumir o papel de mercador, batedor e demolidor. Seu desenvolvimento durou cerca de 4 anos, com investimentos em torno de 15 milhões, por parte da empresa *Dever* [Taikodon, 2009].

Outro jogo que destacou-se foi o *Erinia*, por ser o primeiro *MMORPG* brasileiro. Seu projeto foi iniciado em 2004 e a historia do jogo é voltada para o folclore brasileiro, envolvendo lendas e contos nacionais.

Word of Warcraft, *MMORPG* produzido pela Blizzard, segue o gênero ação/aventura. Nesse jogo se executa um jogo cliente que está ligado a uma rede de servidores, cujo acesso deve ser pago. Na história do jogo existem várias raças, em duas facções: Aliança e Horda. O cenário do jogo se passa em um mundo baseado na era medieval, onde jogadores *on-line* travam batalhas em grupos, assim como nas guerras medievais, cumprindo missões e objetivos [Blizzard, 2009].

2.4 Trabalhos Acadêmicos Sobre Jogos Computacionais

O trabalho de [Osório, 2007] apresenta conceitos sobre desenvolvimento de jogos e entretenimento digital implementando agentes inteligentes e aborda técnicas da IA, como: arquiteturas de agentes, mecanismos de controle e cooperação entre múltiplos agentes, além de técnicas de aprendizado de máquina e técnicas evolutivas.

O artigo de [Kishimoto, 2009] realiza uma investigação sobre o uso da Inteligência Artificial em jogos computacionais, para computadores e videogames, desde o início da

indústria até os dias atuais, incluindo história, evolução e estado da arte da Inteligência Artificial aplicada a jogos, as técnicas utilizadas e os benefícios obtidos com essa prática.

Em [Barbosa, 2009], apresenta-se o desenvolvimento de um jogo computacional por meio de uma versão do jogo *Space Invaders*. O objetivo desse trabalho é descrever os conceitos básicos para construção de um jogo 2D utilizando linguagem JAVA.

[Luiz, 2009] apresenta uma introdução ao desenvolvimento de jogos, citando seus vários gêneros, e também discute o estado atual do mercado de jogos no Brasil e no mundo. Ele aponta estruturas de desenvolvimento, conceitos de *Game Design*, Inteligência Artificial e conceitos básicos para a implementação de um jogo.

Capítulo 3

O Desenvolvimento de Jogos Computacionais

Neste capítulo são apresentadas técnicas utilizadas no desenvolvimento de jogos computacionais. Dentre elas, destacam-se conceitos referentes à Inteligência Artificial, que complementam o desenvolvimento de jogos, auxiliando a produção dos mesmos.

3.1 Definindo Jogos Computacionais

Segundo [Battaiola, 2000], "um jogo de computador pode ser definido como um sistema interativo que permite ao usuário experimentar, uma situação de conflito. Quase sempre, como plano de fundo dessa situação, existe um enredo, que define do que se trata o jogo, as regras que o jogador deve seguir e os objetivos que ele deve se esforçar para atingir".

Para [Battaiola *et al*., 2001] [LaMothe *et al*., 1994], um jogo de computador é composto por três partes básicas: enredo, interface interativa e *engine* (*framework*), sendo o seu sucesso associado à combinação ideal desses três módulos.

Com a definição de [Battaiola *et al*., 2001] e [Lamothe *et al*., 1994] é evidente que o desenvolvimento de jogos é uma área multidisciplinar, exigindo conhecimento não só da Informática, mas também de outras áreas do conhecimento, como artística e musical, bem como profissionais qualificados em testar os jogos, gerentes de projeto, projetistas de jogos (*Game Designer*), entre outros.

Para o desenvolvimento de *software*, de modo geral, é necessário seguir uma série de regras bem definidas, a fim de atender as necessidades para as quais ele foi desenvolvido. O desenvolvimento de jogos foca na diversão e entretenimento de seus usuários, procurando interagir com eles da melhor forma possível, por meio da utilização de uma boa interface.

3.2 O Ciclo de Desenvolvimento de Jogos

Assim como qualquer outro *software*, um jogo deve passar por etapas para o seu desenvolvimento, as quais visam desenvolver o produto de forma ordenada, possibilitando o gerenciamento, comunicação e cumprimento de cronograma da equipe no período previsto. Segundo [Rollings e Morris, 2004] erros na etapa inicial podem encadear mais erros nas etapas posteriores e o custo para a correção de um erro em uma etapa posterior pode chegar a ser 200 vezes maior em relação à sua correção no período inicial. Por isso, deve-se ter um bom conhecimento do processo de desenvolvimento de um jogo antes de iniciar um projeto, procurando, assim, minimizar esses erros.

Segundo [Perucia *et al*., 2007], a vantagem de se ter uma boa arquitetura desenvolvida para a criação de jogos é a possibilidade de reutilização de módulos em outros jogos. Em seu livro, ele apresenta as etapas do ciclo de desenvolvimento de um jogo, descritas a seguir:

Planejamento: É realizada uma reunião com o objetivo de obter novas idéias sobre o desenvolvimento do jogo. Caso seja necessário, deve-se realizar mais de uma reunião. A etapa de planejamento é utilizada para definir como o jogo será desenvolvido, a originalidade do jogo, o público-alvo e as especificações técnicas.

***Game Design*:** Nesta fase ocorre a definição das regras do jogo. Para isso, determina-se o comportamento dos elementos do jogo, como personagens, golpes, interface, jogabilidade, armas, fases, balanceamento e imersão de jogo. Utiliza-se a imaginação e criatividade como instrumento para a definição do jogo. Ao definir essas regras, é feita a elaboração do *Document Design* (nesse documento se tem todas as definições feitas pelo *Game Designer)*.

O *Document Design* contém as definições de projeto que serão utilizadas por outros integrantes da equipe envolvidos no desenvolvimento do jogo. Em projetos pequenos, apenas esse documento é suficiente, porém, em projetos maiores e mais complexos, envolvendo muitas fases e missões, deve-se realizar um planejamento para cada fase, de forma que com que todas as pessoas envolvidas no processo de desenvolvimento tenham um bom conhecimento das tarefas realizadas.

Cronograma Geral: Com a definição do *Document Design* é possível determinar a ordem e o tempo necessário para o desenvolvimento dos processos que envolvem a criação do jogo. Isso permite aos diretores de equipes definirem as metas a serem atingidas.

***Level Design*:** Define o mapeamento geral das fases do jogo e as missões necessárias para a conclusão de cada fase. Esse documento é imprescindível para que membros da equipe possam desenvolver as fases. O desenvolvimento das fases só e iniciado após a conclusão desse documento.

***Beta*:** Primeiro realiza-se o desenvolvimento de versões *alphas* entre os membros de cada equipe de desenvolvimento. Posteriormente, a união destas versões *alphas* resulta em uma versão *beta* com todas as fases e interatividade do jogo integrada. Com o desenvolvimento da versão *beta*, iniciam-se os testes.

Teste: Esta etapa tem o objetivo de detectar erros ou *bugs* até se chegar a uma Versão Final e uma Versão Demo. Também nessa fase é realizada coleta de dados a fim de melhorar o jogo desenvolvido.

- **Versão Final**: Esta versão é considerada a versão comercial do jogo, supostamente com erros e *bugs* detectados e corrigidos.

- ***Demo***: A versão *Demo* é uma versão de demonstração que não contempla todos os benefícios de um jogo completo, por exemplo, pode-se ter um número limitado de estágios (fases). Geralmente, ela é lançada alguns meses antes do jogo completo, auxiliando na detecção de erros ou *bugs* e criando expectativa em jogadores, dando início à divulgação do produto.

A Figura 3.2 apresenta o diagrama que representa o ciclo de desenvolvimento de um jogo. Os retângulos azuis mais claros representam uma etapa. Os retângulos mais escuros representam os dados de saída de uma determinada etapa. Cada retângulo de uma etapa está associado ao seu respectivo retângulo de saída de dados, conforme ilustrado na Figura 3.1.

Figura 3.1: Exemplo da ilustração de uma etapa e seus dados de saída

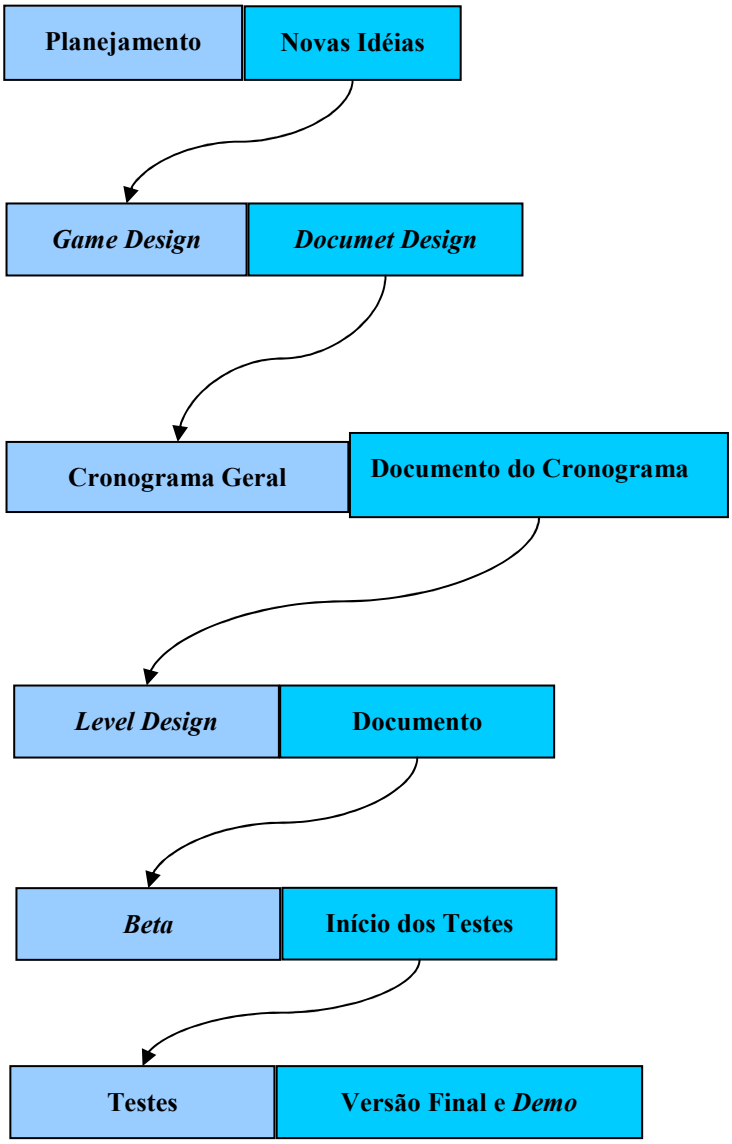

Figura 3.2: Ilustração do ciclo de desenvolvimento de um jogo, baseado em [Perucia *et al*., 2007]

Além do processo de desenvolvimento, uma empresa precisa adotar processos que contemplem a comercialização e distribuição de seus jogos, a fim de obter lucro sobre o produto desenvolvido.

Nos itens a seguir serão apresentadas técnicas imprescindíveis ao desenvolvimento da interface gráfica dos jogos computacionais, referentes ao cenário e personagem do jogo e detecção de colisão (Seções 3.3 e 3.4). Posteriormente, serão apresentadas técnicas relevantes ao desenvolvimento da lógica dos personagens (Seções 3.5 e 3.6) e técnicas de comunicação de agentes computacionais jogo (Seções 3.7, 3.8 e 3.9).

3.3 Desenvolvimento do Cenário

Para a representação do cenário, uma das formas possíveis seria a utilização de uma grande imagem. Porém, se o cenário for muito grande, a imagem que o representa ocuparia muito espaço em memória, tendo conseqüências indesejáveis, como atrasos ao plotar a imagem e dificuldade para determinar o local por onde um personagem poderia passar ou não [Brackeen, Barker e Vanhelsuwé, 2003].

Para evitar esses problemas, a técnica *Tile-Based Map* pode ser utilizada. *S*egundo [Perucia *et al*., 2007], *tiles* são imagens adicionadas em um cenário e divididas em pedaços iguais de tamanho fixo. Chama-se *tile* o quadrado resultante dessa divisão.

Essa técnica consiste na divisão da imagem que representa o cenário em partes menores, permitindo representá-lo em forma de matriz. Cada célula da matriz representa uma parte da imagem. A Figura 3.3 ilustra essa representação.

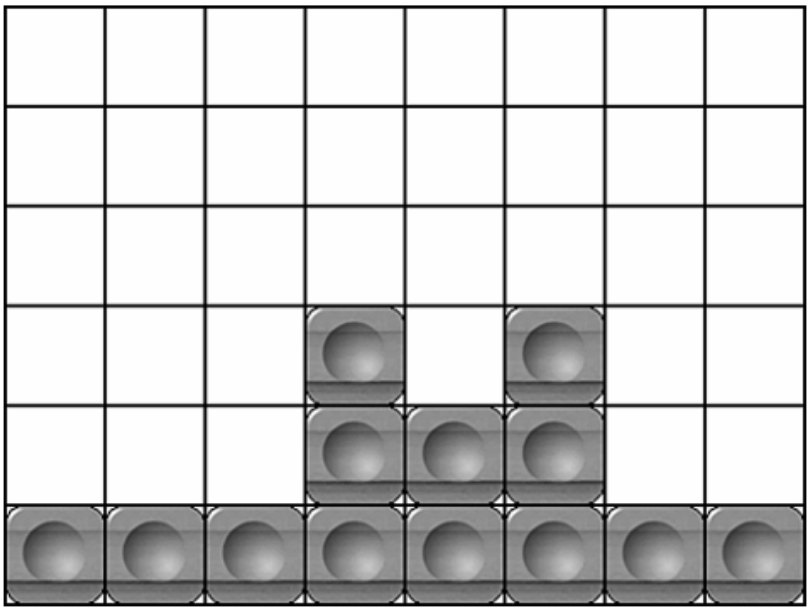

Figura 3.3: Exemplo de *Tile-Based Map* [Brackeen, Barker e Vanhelsuwé, 2003]

Com a utilização dessa técnica obtêm-se vantagens no desenvolvimento do cenário, citadas a seguir:

- Facilita na detecção de colisão [Brackeen, Barker e Vanhelsuwé, 2003];
- Melhor representação do cenário;

- Possibilidade de determinar a localização de objetos e jogadores de forma simplificada [Brackeen, Barker e Vanhelsuwé, 2003];

- Economia na alocação de memória, pois com um pequeno pedaço do cenário alocado, é possível desenhar um cenário inteiro [Brackeen, Barker e Vanhelsuwé, 2003].

Essa técnica também apresenta algumas desvantagens, tais como:

- Quando se determina o tamanho do *tile* maior que a imagem real do objeto, este objeto ocupa um espaço maior do que deveria no cenário.

- Supondo um mundo dividido em *tiles*, o bloqueio da movimentação de um personagem feita na diagonal superior direita só pode ser feito por um objeto localizado na diagonal superior direita. Não é válido pensar que, por existirem objetos acima e à direita (em relação ao personagem), a sua passagem deveria ser impedida quando ele tenta passar pela diagonal superior direita, pelo fato de ocuparem um espaço quadrado contíguo.

3.3.1 *Sprites*

Segundo [Perucia *et al*., 2007], *sprites* são estruturas com imagens próprias que permitem a criação de animações e o livre posicionamento na tela. De forma geral, objetos com animação ou movimentação em um jogo são ditos *sprites*, podendo ter uma ou mais imagens associadas a ele, possibilitando a realização de animações. Os personagens são representados por *sprites*, pois contêm características distintas dos elementos que compõem o cenário do jogo.

As animações em *sprites* podem ser de dois tipos: contínua ou finita. Animações contínuas reiniciam após chegar ao último quadro de uma seqüência de animação. Animações finitas terminam ao chegar ao último quadro de uma seqüência de animação.

No desenvolvimento da lógica dos personagens e inimigos do jogo utilizam-se técnicas de Inteligência Artificial, apresentadas nas próximas seções.

3.4 Algumas Técnicas Utilizadas no Desenvolvimento de Jogos Computacionais

Segundo [Russel e Norvig, 2002], Inteligência Artificial pode ser definida como a criação de programas que emulam 4 segmentos: pensamento humano, pensamento racional, agir como humano e agir racionalmente. A Inteligência Artificial aplicada a jogos foca no segmento que procura agir como humanos, tendo menos dependência dos outros segmentos que não são focados [Schwab, 2004]. A Tabela 3.1, retirada do trabalho de [Schwab, 2004], apresenta com as técnicas de IA utilizadas em jogos computacionais desde 1962 até 2001.

Tabela 3.1: Histórico da utilização das técnicas de IA em jogos [Schwab, 2004].

Ano	Descrição	Técnica Utilizada
1962	Primeiro jogo de computador, *SpaceWar*, para 2 jogadores.	Nenhuma
1972	Lançamento do jogo *Pong*, para 2 jogadores.	Nenhuma
1974	Jogadores tinham que atirar em alvos móveis em *Pusuit* e *Qwak*.	Padrões de Movimento
1975	*Gun Fight*, personagem com movimentos randômicos.	Padrões de Movimento
1978	*Space Invaders*, contém inimigos com comportamento definido por padrões.	Padrões de Movimento
1980	*Pac –Man*, onde o movimento dos inimigos é definido por um padrão, que pode ser diferente para cada jogador, definindo várias formas de caçar o jogador.	Padrões de Movimento
1990	*Herzog Wei,* primeiro jogo de estratégia em tempo real. Continha um algoritmo de busca impreciso.	Máquinas de Estados
1993	*Doom*, primeiro jogo de tiro em primeira pessoa, primeiro jogo desenvolvido em linguagem de alto nível (linguagem C) e possibilitando partidas em rede.	Máquinas de Estados
1996	*BatleCruiser: 3000AD*, primeiro jogo comercial desenvolvido com redes neurais.	Redes Neurais
1997	Deep Blue realizou partidas contra o campeão mundial de xadrez, Garry Kasparov.	*Alpha-beta pruning, machine learning e selective search*.
1998	*Half-Life*, jogo de tiro em primeira pessoa, visto como o título de melhor IA até sua época.	*Scripts e finite state machine*.
2001	*Black & Withe*, contém inimigos que aprendem com decisões feitas pelo jogador.	*Redes neurais, reinforcement e observational learning*.

As técnicas de IA utilizadas atualmente são inúmeras. Algumas empresas, como a *Bethesda Softworks*, desenvolvem suas próprias técnicas de IA, como a *Radiant AI* [The Elder Scrolls, 2009]. Com essa técnica, os personagens não controlados pelo jogador ou *NPC's* (*non-player characters*), possuem a capacidade de tomar suas próprias decisões, baseados no

mundo em sua volta. Assim, eles procuram atingir o comportamento adequado a seus personagens controlados pelo computador. Segundo [Dalmau, 2004], as técnicas da IA utilizadas em jogos são: Padrões de Movimento, Máquinas de Estados, Sistemas Baseado em Regras, Algoritmos de Busca e Algoritmos Genéticos. A seguir, são apresentadas as definições dessas técnicas, segundo [Dalmau, 2004] e outros autores.

3.4.1 Padrões de Movimento

Foram utilizados nos primeiros jogos da história [Schwab, 2004]. Definem padrões de comportamento para personagens, por exemplo, um personagem pode realizar ronda em uma determinada área [Lamothe, 1999]. Segundo [Bourg e Seemann, 2004] Padrões de Movimento são implementados por meio da escolha de um padrão desejado, o qual deve ser codificado por uma central de dados. A central de dados consiste em instruções específicas de movimento, tais como andar, girar, intensidade do movimento para um determinado padrão. A partir de padrões primitivos é possível criar padrões mais complexos, como movimento em círculos, quadrado e ziguezague.

3.4.2 Máquinas de Estado

Segundo [Bourg e Seemann, 2004] uma Máquina de Estados é uma máquina abstrata com um ou mais estados diferentes e predefinidos. O comportamento de um agente é definido pelos estados e a mudança de comportamento é definida peças transições entre esses estados. As máquinas de estado são utilizadas até hoje em jogos, pelo fácil entendimento e depuração de erros. O jogo *Pac-Man* utiliza máquinas de estados para definir o comportamento dos fantasmas (inimigos) [Schwab, 2004].

Máquinas de Estados contêm um estado inicial, transações e estados. Um estado inicial é um estado indicado por uma seta sem origem. Transições são setas com estado de origem e estado de destino, sendo que o destino pode ser o mesmo estado da origem ou um estado distinto. Estados representam origens e destinos de uma transição, como ilustrado a seguir:

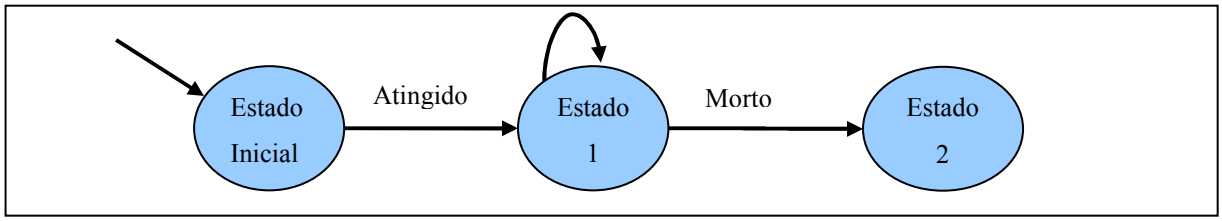

Figura 3.4: Exemplo de Máquina de Estados

As Máquinas de Estado são muito utilizadas, pois são capazes de controlar o comportamento de agentes com pouco processamento. [Karlsson, 2005], afirma "ser fácil e intuitivo definir comportamentos por meio dessa abordagem, principalmente com o auxílio de ferramentas visuais". Porém existem algumas limitações, como o fato de ter comportamentos potencialmente repetitivos, devido à própria definição de Máquina de Estados [Karlsson, 2005].

3.4.3 Sistemas Baseado em Regras

Sistemas Baseado em Regras ou *RBS* (*Rules Based Systems*) é a técnica mais difundida em aplicações do mundo real e em aplicações de jogos envolvendo IA [Bourg e Seemann, 2004].

Esses sistemas contêm regras que são utilizadas para realizar inferências ou tomada de decisões. Apresentam um comportamento simples, fazendo uso de conjunto de parâmetros que são tratados por um conjunto de regras, retornando geralmente uma tomada de decisão ou comportamento. São utilizados em diagnósticos médicos, proteções contra fraudes e também na análise de falhas de engenharia [Bourg e Seemann, 2004].

Em [Karlsson, 2005], é apresentado um exemplo da utilização de Sistemas Baseado em Regras, supondo a existência de parâmetros como: saúde do personagem, saúde do inimigo, distância do inimigo e quantidade de munição, e também ações como: atacar inimigo, fugir e procurar o inimigo.

A ação que o agente poderá tomar ao avaliar os parâmetros através de regras pode ser expressa por: SE (distância do inimigo < 50 E saúde do agente > 50) ENTÃO atacar inimigo. As condições podem ser ligadas por conectivos E e OU. A ação de atacar o inimigo somente será realizada se essa condição for satisfeita.

Uma vantagem na utilização dessa abordagem, segundo [Bourg e Seemann, 2004], é na "capacidade dos agentes imitarem o modo como as pessoas tendem a pensar e a razão de

um conjunto de fatos conhecidos e os seus conhecimentos sobre o problema específico do domínio", além da facilidade na implementação. Segundo [Karlsson, 2005] Sistemas Baseado em Regras, permitem a modelagem de comportamentos complexos de forma mais concisa em relação à técnica utilizando Máquinas de Estados, mas podem "necessitar de muito espaço em memória, além de muito poder de processamento" [Karlsson, 2005].

3.4.4 Algoritmos de Busca

Em jogos de computadores, *NPC's* movimentam-se de um ponto de origem a um ponto de destino, entre os quais podem existir obstáculos. Esses obstáculos devem ser contornados se possível, fazendo o personagem controlado pelo computador aparentar certo nível de inteligência.

Para representar tal comportamento deve ser utilizado um algoritmo de busca, que deve procurar o menor caminho entre um ponto de origem e um ponto de destino sobre um cenário, levando em consideração seus elementos [Karlsson, 2005]. O algoritmo mais utilizado é o A*, embora se utilizem também o algoritmo de *Dijkstra* e *Waypoints* [Dalmau, 2004] [Lamothe, 1999].

O algoritmo A* tem dificuldades em jogos com objetos que tenham posições dinâmicas e mudanças [Karlsson, 2005].

3.4.5 Algoritmos Genéticos

Essa técnica foi inventada por John Holland em 1960, o seu objetivo não era desenvolver um algoritmo, mas sim desenvolver formas de como o processo de adaptação poderia ser incorporado em um computador [Mitchell, 1996]. Segundo [Mitchell, 1996], o método proposto por John consiste em alterar uma população de cromossomos (*strings* de *bits* 0 ou 1) para uma nova população utilizando uma espécie de seleção natural, juntamente com operações inspiradas na genética, como: *crossover*, mutação e inversão. Cada cromossomo é constituído de genes (*bits*), os quais representam a instância de um alelo particular. O operador de seleção escolhe os cromossomos da população que irão se reproduzir, onde os mais aptos se reproduzem mais que os menos aptos. O cromossomo, então, permuta sub-partes dos genes, tentando imitar a recombinação biológica dos seres humanos. A inversão inverte a ordem de uma parte contígua do cromossomo, reorganizando a ordem dos genes.

A utilização de Algoritmos Genéticos pode se dar através da geração de uma população, criando diferentes indivíduos de acordo com um DNA virtual, sendo este representado por um vetor de valores, cada um sendo um parâmetro da espécie a ser modelada [Dalmau, 2004]. Também podem ser utilizados na mutação e evolução de personagens [Dalmau, 2004] [Lamothe, 1999].

3.5 RBC (Raciocínio Baseado em Casos)

Esta seção descreve o Raciocínio Baseado em Casos, que é a técnica principal utilizada neste trabalho. A idéia básica do RBC é resolver um novo problema relembrando uma situação anterior similar e, então, reutilizar a informação e o conhecimento daquela situação em situações similares [Riesbeck e Schank, 1989]. A resolução de problemas ocorre na recuperação e adaptação de experiências passadas, chamadas de casos, os quais são encontrados em uma base de casos. O caso recuperado pode resolver um novo problema a partir da adaptação de uma solução decorrente de problemas anteriores e similares [Wangenheim e Wangenheim, 2003].

O trabalho de [Cunha *et al*., 2009] envolve aprendizagem de soluções para um jogo educacional que visa ensinar crianças sobre a vida marinha, servindo como ferramenta de apoio à explicação de conteúdos.

3.5.1 Elementos Básicos do RBC

Os elementos básicos do RBC, segundo [Wangenheim e Wangenheim, 2003], são: Representação do Conhecimento, Média de Similaridade, Adaptação e Aprendizado.

- **Representação do Conhecimento:** é apresentada em forma de casos, que descrevem experiências concretas. Caso seja necessário, outros tipos de conhecimento sobre o domínio da aplicação podem ser armazenados no RBC, tais como: Casos Abstratos e Generalizados, Tipos de Dados e Modelos de Objetos usados como informações.
- **Média de Similaridade:** realiza a busca de um caso relevante na base de casos para a resolução de um problema atual, indicando, ainda, se o caso recuperado é ou não similar ao novo problema.

- **Adaptação:** como nem sempre casos passados são utilizados na resolução de novos problemas, então sistemas de RBC avançados contêm mecanismos e conhecimento para adaptar casos recuperados, verificando se satisfazem ou não as características de um determinado problema.
- **Aprendizado:** realiza atualização e evolução na base de casos. Armazena casos resolvidos com sucesso, proporcionando sua recuperação no futuro.

3.5.2 Funcionamento do RBC

O funcionamento do RBC pode ser expresso através de um ciclo. Dentre os vários modelos, o mais aceito na representação do RBC, segundo [Wangenheim e Wangenheim, 2003], é o proposto por [Aamodt e Plaza, 1994], que compreende quatro tarefas principais:

- **Recuperar** o(s) caso(s) mais similar(es) na base de casos;
- **Reutilizar** este(s) casos(s) para resolver o problema;
- **Revisar** a solução proposta;
- **Reter** a experiência representando o caso atual (ou partes desta experiência) para reutilização futura.

A Figura 3.5 ilustra o ciclo típico de como o RBC funciona:

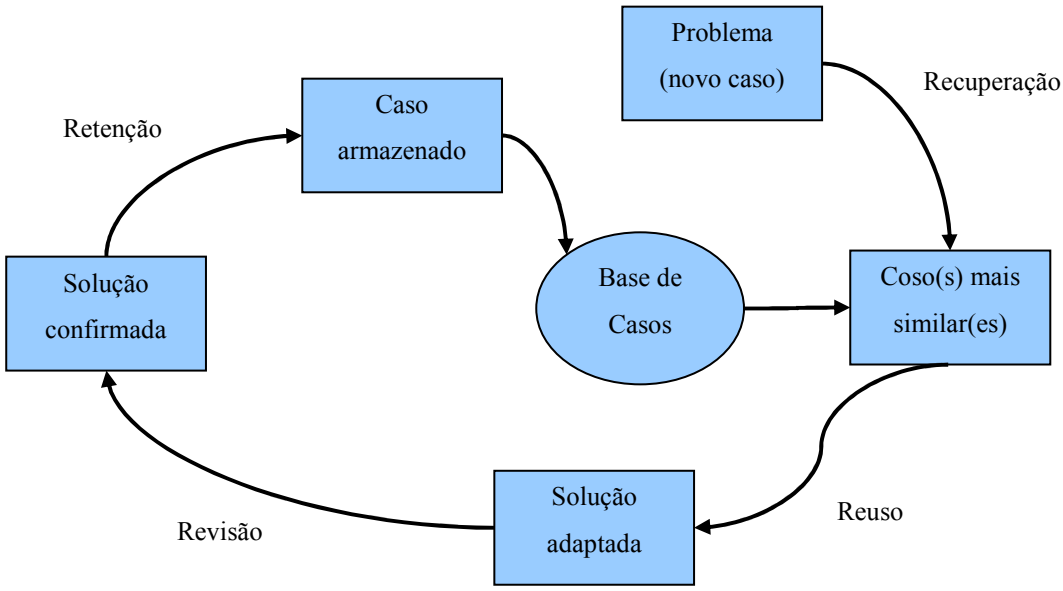

Figura 3.5: Ciclo RBC [Aamodt e Plaza, 1994]

A partir da detecção de um novo problema é realizada uma busca na base de dados, procurando os casos mais similares. Os casos recuperados podem ser mais similares ao atual ou menos similares (Recuperação). Caso seja necessário reutilizar um caso, ele é recuperado (Reuso). A solução pode ser corrigida caso seja necessário (Revisão). Quando um problema é resolvido, uma nova experiência pode ser armazenada na base de casos, ficando disponível para novas consultas de similaridade (Retenção).

3.5.3 Representação de Casos

O conhecimento do RBC é armazenado na base de casos. Segundo [Wangenheim e Wangenheim, 2003], caso é "uma peça de conhecimento contextualizado representando uma experiência ou episódio concretos. Contém a lição passada, que é o conteúdo do caso e contexto em que a lição pode ser usada.". Os casos podem conter várias formas e seu conteúdo dependerá do domínio da aplicação.

A forma de representação com melhor resultado é a atributo-valor, apresentada na Tabela 3.1, que resolve uma grande parcela dos problemas de aplicação de RBC [Wangenheim e Wangenheim, 2003].

Tabela 3.2: Representação de um caso [Wangenheim e Wangenheim, 2003]

Problema (Sintomas):	
Problema:	Impressora não funciona
Modelo:	Robotron Matrix 600
Luz de estado do papel:	apagada
Luz de estado da tinta colorida:	apagada
Luz de estado da tinta preta:	apagada
Estado do interruptor:	ligado
Solução:	
Diagnóstico:	Curto-circuito
Ação:	Troca da fonte de alimentação

3.5.4 Recuperação de Casos

Durante a recuperação de casos para a solução do problema atual, procura-se um caso na base de casos, no contexto do problema atual, para determinar sua solução. Quem define o

caso apropriado é o método de similaridade. O caso recuperado não precisa, necessariamente, ser idêntico à situação atual [Wangenheim e Wangenheim, 2003].

3.5.4.1 Similaridade pelo Método de Indexação

Este método utiliza Eis (entidades de informação) para seus dados associados aos casos envolvidos durante o processo de recuperação. As Eis são elementos utilizados para distinguir os casos recuperados, para a resolução do problema atual. São muito utilizados na recuperação de informações e de gerência de banco de dados [Wangenheim e Wangenheim, 2003].

3.5.4.2 Similaridade Vizinho mais Próximo

Esta técnica se mostra simples no processo de definição da similaridade entre dois casos. Aplica uma relação geométrica aos casos da base, logo, a distância espacial entre os casos define a similaridade. Sendo assim, a busca nesta base de casos se resume a procurar o vizinho geometricamente mais próximo. A Figura 3.6 apresenta três casos representados, definidos como: caso 1, caso2 e caso Q.

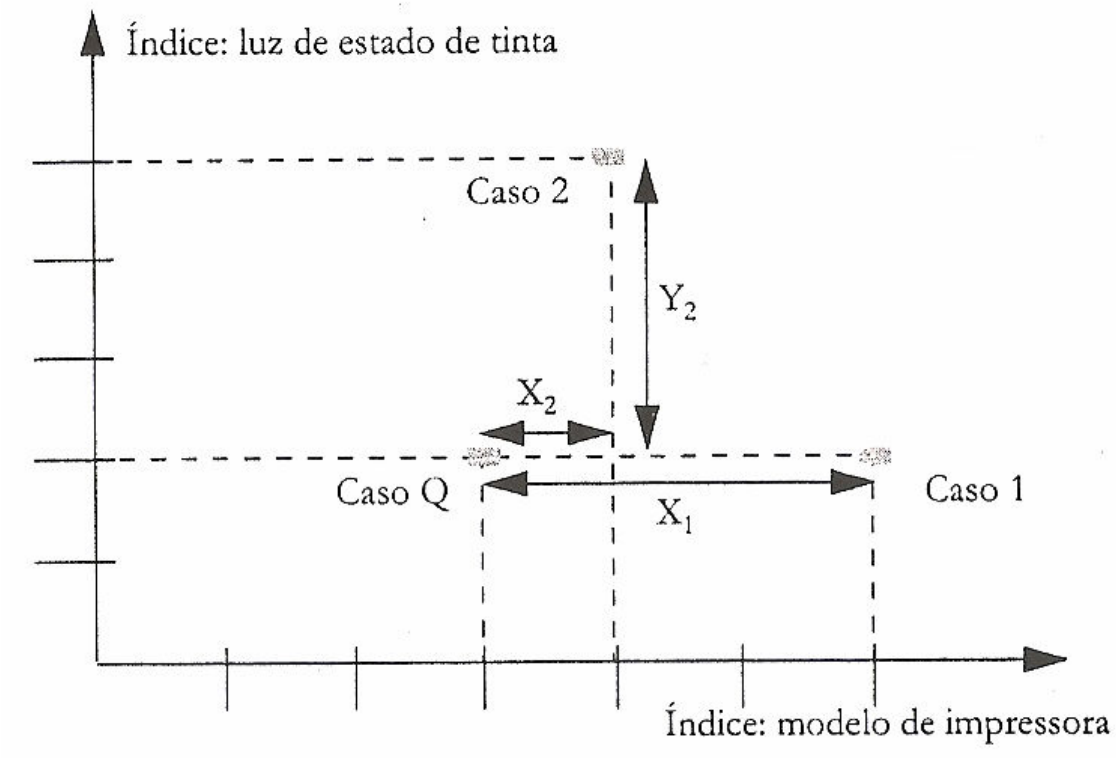

Figura 3.6: Distância dos vizinhos mais próximos [Wangenheim e Wangenheim, 2003]

A similaridade dos casos 1 e 2 é definida em relação ao caso Q, utilizando a fórmula a seguir:

- Distância entre o caso Q e o caso 1: d = X1 + Y1 = 0 + 3 = 3.
- Distância entre o caso Q e o caso 2: d = X2 + Y2 = 1 + 3 = 4.

O vizinho mais próximo de Q é definido pela menor distância, logo, o caso 1, como mostrado na fórmula acima.

3.5.5 Reutilização de Casos

O processo de aplicar uma solução ao problema atual a partir de um caso similar recuperado na base é definido como recuperação, consistindo principalmente da adaptação da solução do caso anterior ao caso atual. A adaptação define quais aspectos do caso devem ser adaptados e quais alterações devem ser feitas para essa adaptação [Wangenheim e Wangenheim, 2003].

3.5.6 Revisão de Casos

Quando se tem um caso definido para a solução de um problema e ele não é correto, ocorre uma oportunidade de aprender, então o sistema realiza uma revisão nesse caso. A revisão tem duas etapas: avalia se a solução gerada pelo reuso é correta, então a base e incrementada com o caso de sucesso. Se a solução for errada, deve-se reparar a solução, utilizando conhecimento específico do domínio ou informações fornecidas pelo usuário [Wangenheim e Wangenheim, 2003].

3.5.7 Retenção de Novos Casos

A retenção de novos casos adiciona novos conhecimentos na base de dados do RBC, incorporando novas soluções ao conhecimento que já existe no sistema. O objetivo é aumentar continuamente o conhecimento sempre que um problema novo é resolvido, com isso o poder de resolução de problemas do RBC também aumenta [Wangenheim e Wangenheim, 2003].

3.5.8 Exemplos de Aplicação do RBC

O RBC vem sendo utilizado em diversas aplicações, tais como: análise financeira, assessoramento de riscos, manutenção técnica, controle de processos, controle da qualidade, diagnóstico médico, sistemas de suporte a *software*, previsão, planejamento, projeto, classificação, interpretação de imagens, avaliação imobiliária, comércio eletrônico, suporte a usuário, gestão do conhecimento, etc. [Wangenheim e Wangenheim, 2003]

3.6 Agentes Computacionais

Segundo [Wooldridge e Jennings, 1994], a resposta para a pergunta "O que é agente?" é tão imprecisa quanto a pergunta "O que é inteligência?". O problema para definir o termo "agente" está nas diversas áreas que trabalham de forma estritamente relacionada com este termo, dificultando uma única definição que esteja de acordo com todas as áreas envolvidas. Embora não exista um consenso sobre a definição de agentes na Inteligência Artificial, será apresentada uma definição seguindo estudos na área.

Agentes são elementos que possuem a capacidade de tomar decisões próprias (autonomia). Eles podem ser reais ou virtuais e são dotados da capacidade de se comunicar. Quando inseridos em um ambiente (exemplo de ambiente: partida de futebol, floresta, páginas de *web*, etc.), eles têm a capacidade de interagir com outros agentes, através de sensores e atuadores. Sensores são mecanismos de percepção que possibilitam ao agente adquirir informações do ambiente no qual ele encontra-se inserido, por exemplo: olhos, ouvidos e outros órgãos. Atuadores são mecanismos que são responsáveis por realizar uma determinada ação do agente, por exemplo: mãos, pernas, boca e outras partes do corpo.

A Figura 3.7 apresenta um diagrama exemplificando a interação entre um agente e seu ambiente, e como pode ser realizada a seleção de ações que o agente precisa realizar para cumprir sua meta. Segundo [Bukland, 2005], o comportamento de um agente pode ser definido em 3 camadas.

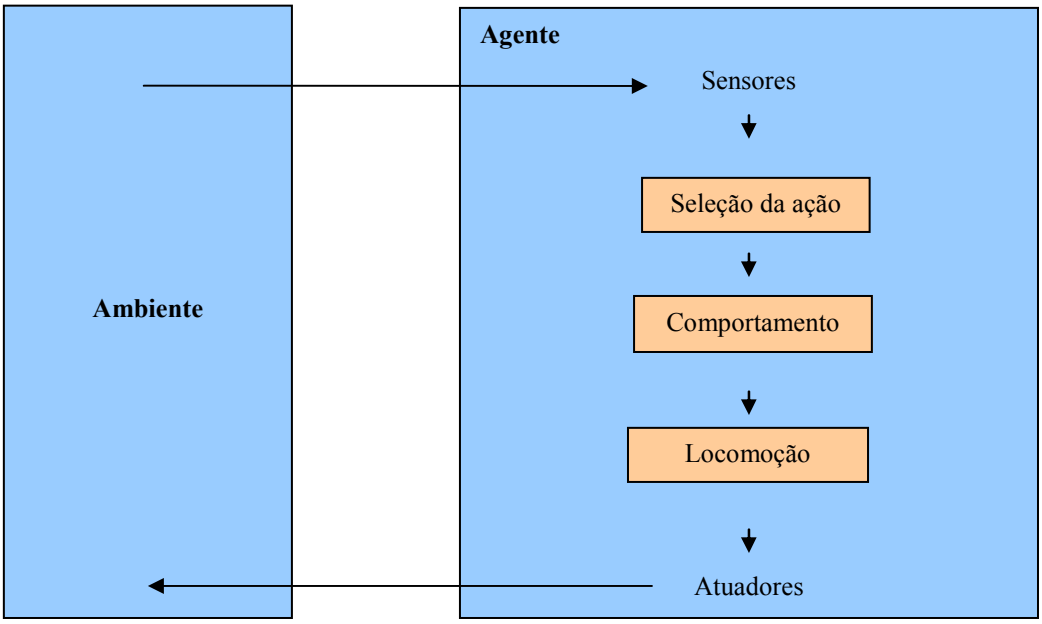

Figura 3.7: Interação de um agente e seu ambiente [Bukland, 2005] [Russel e Norvig, 2002]

- Seleção da ação: Representa a escolha de qual ação o agente irá executar, dentre as quais ele foi projetado para realizar.

- Comportamento: Compreende a parte de análise, onde se realiza todas as pré-condições necessárias a realização de uma ação. As pré-condições podem ser definidas por condições.

- Locomoção: Define como o agente realiza ação.

Basicamente, existem dois tipos de agentes: os reativos e os cognitivos. O agente reativo é a forma mais simples de se desenvolver um agente, sua complexidade pode aumentar até se chegar a um agente totalmente cognitivo.

3.6.1 Agentes reativos

Segundo [Álvares, 1998] agente reativo é baseado em modelos etológicos ou biológicos, por exemplo: sociedade de formigas, colméia de abelhas, etc. Seu comportamento é definido através de estímulos e respostas. Os estímulos do ambiente onde o agente encontra-se inserido são captados por meio de sensores e interpretados pelo agente, depois a resposta é fornecida através de atuadores em forma de reação a um estímulo.

3.6.2 Agentes Cognitivos

Os agentes cognitivos têm seu desenvolvimento baseado em organizações sociais humanas, como grupos e hierarquias. Compreendem o ambiente de forma explicita, pelo fato de conter memória, podem tomar decisões, a fim de determinar ações futuras. Segundo [Álvares, 1998] os sistemas que utilizam agentes cognitivos geralmente utilizam poucos agentes.

3.6.3 Ambiente de Tarefa

São definidos por [Russel e Norvig, 2002] como os "problemas" para os quais os agentes racionais são as "soluções". Logo, ambiente é o local onde o agente atua, executando objetivos ou realizando tarefas. Existem diversos tipos de ambientes. A seguir, apresenta-se a definição de alguns exemplos, segundo [Russel e Norvig, 2002].

Completamente observável versus **parcialmente observável:** Quando os sensores de um agente permitem perceber todo o ambiente em cada instante, o ambiente é definido como completamente observável. Um ambiente pode ser definido como parcialmente observável por apresentar ruído e sensores imprecisos ou por falta de estados nos dados do sensor.

Determinístico versus **estocástico:** Se o estado atual e ação do agente podem ser utilizados para definir o próximo estado do ambiente, então o ambiente é definido como determinístico. Caso contrário ele é estocástico.

Estático versus **dinâmico:** Se o ambiente pode ser alterado enquanto o agente atua nele, então esse ambiente é definido como dinâmico. Caso contrário ele é estático.

3.7 Sistemas Multiagente

Sistema multiagente é uma subárea da Inteligência Artificial Distribuída (IAD). Segundo [Bittencourt, 2009], o seu objetivo é estudar as técnicas de raciocínio que podem ser necessárias ou úteis para que agentes computacionais participem de sociedades de agentes. Estuda o comportamento de agentes inseridos em um ambiente, como eles interagem entre si e se comunicam na busca de uma solução, baseada em seus objetivos. A comunicação dos agentes se dá por meio de trocas de mensagens. Os objetivos podem ser determinados a um grupo de agentes ou a um agente individual. Os agentes podem ser réplicas uns dos outros

(cada agente é uma cópia do outro) ou distintos (cada um tem sua própria característica). Agentes podem ter comportamento de cooperação ou ser competitivos entre si [Bittencourt, 2009]. Os sistemas multiagente são definidos em dois tipos, apresentados a seguir.

3.7.1 Sistemas Multiagente Reativos

Estes sistemas contêm agentes reativos em grande quantidade. Esses agentes possuem um comportamento simples, apenas reagindo a estímulos do seu ambiente, ou seja, formado pelo par Estímulo-Resposta (Ação-Reação), não possuindo inteligência.

As principais características de sistemas multiagente reativos, segundo [Ferber e Gasser, 1991], são:

- **Não há representação explícita do conhecimento:** O conhecimento (regras de comportamento) do agente é implícito, manifestando através de seu comportamento.
- **Não há representação do ambiente:** O comportamento do agente se dá através das percepções do ambiente, não existindo representação interna explícita do ambiente.
- **Não há memória das ações:** Os agentes reativos não possuem um histórico de ações passadas, sendo assim, nenhuma ação passada pode influenciar em decisões futuras.
- **Organização Etológica:** A organização de sistemas multiagente reativos é similar à organização observada em animais que vivem em grandes comunidades, como formigas, abelhas, cupim, entre outras.
- **Grande número de membros:** Estes sistemas, em geral, compreendem um grande número de agentes, com populações podendo chegar a milhares de agentes.

3.7.2 Sistemas Multiagente Cognitivos

Segundo [Álvares e Sichman, 2001], os sistemas multiagente cognitivos trabalham com poucos agentes inseridos em relação aos sistemas multiagente reativos. Em seu ambiente, realizam tarefas mais complexas do que os agentes reativos, e por isto, são dotados de inteligência. Os agentes cognitivos têm a capacidade de comunicar-se entre si, negociar informações ou serviços e ainda planejar uma ação futura. O planejamento de ações é possível

pelo fato de agentes cognitivos conterem conhecimentos, competências, intenções e crenças, podendo, assim, coordenar ações visando a solução de problemas ou a conquista de metas.

As características de sistemas multiagente cognitivos, segundo [Ferber e Gasser, 1991], são:

- Os agentes cognitivos contêm uma representação explícita do ambiente e dos outros agentes existentes no ambiente;
- Podem manter histórico das interações e ações passadas;
- A comunicação entre os agentes é feita através de envio e recebimento de mensagens;
- Para alcançar seus objetivos, os agentes utilizam o raciocínio;
- Sistema geralmente é formado por poucos agentes.

3.8 Comunicação Entre Agentes

A comunicação entre agentes proporciona a troca de mensagens com o objetivo de permitir colaboração, negociação e cooperação entre os agentes inseridos em um sistema multiagente. Para garantir todas essas características, deve-se utilizar uma linguagem que todos os agentes entendam, sendo possível compartilhar as informações (conhecimento) entre os agentes e coordenação das atividades em grupo entre os agentes do ambiente.

Existem diversas maneiras de garantir a comunicação entre agentes em um ambiente. Segundo [Baker, 1997] os agentes podem se comunicar diretamente uns com os outros, por meio da comunicação direta. Também é possível a comunicação através de agentes facilitadores, chamado de sistema federado (comunicação assistida), outra forma de comunicação ocorre quando um agente se comunica com todos os outros de uma só vez, chamada de Comunicação por difusão *(broadcast*). Por fim, existe também a comunicação através de quadro-negro (*blackboard*).

3.8.1 Comunicação Direta

Estabelece uma comunicação ponto-a-ponto onde os agentes se comunicam de forma direta, através de protocolos de comunicação, para garantir a troca de informação. Nesse tipo

de comunicação se faz necessário que cada agente saiba da existência dos outros agentes. Como não exista nenhum agente coordenador, não existe risco de gargalos na comunicação dos agentes quando houver alta demanda na troca de mensagens entre eles no sistema, porém, quanto maior o número de agentes envolvidos na comunicação, maior será o custo para realizá-la. A Figura 3.8 ilustra a comunicação direta entre agentes.

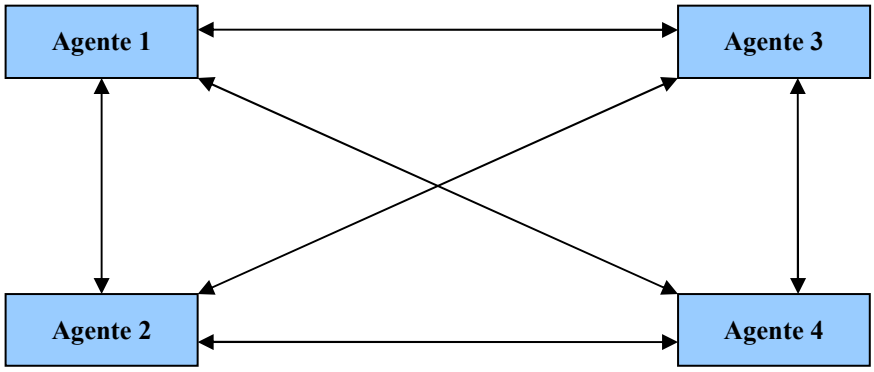

Figura 3.8: Exemplo de comunicação direta entre agentes [Silva, 2003]

3.8.2 Sistemas Federativos (Comunicação Assistida)

Nesta comunicação, segundo [Baker, 1997], o sistema utiliza um ou mais agentes especiais, para coordenar a comunicação. Os agentes responsáveis por intermediar as mensagens são chamados de facilitadores. A Figura 3.9 ilustra essa comunicação.

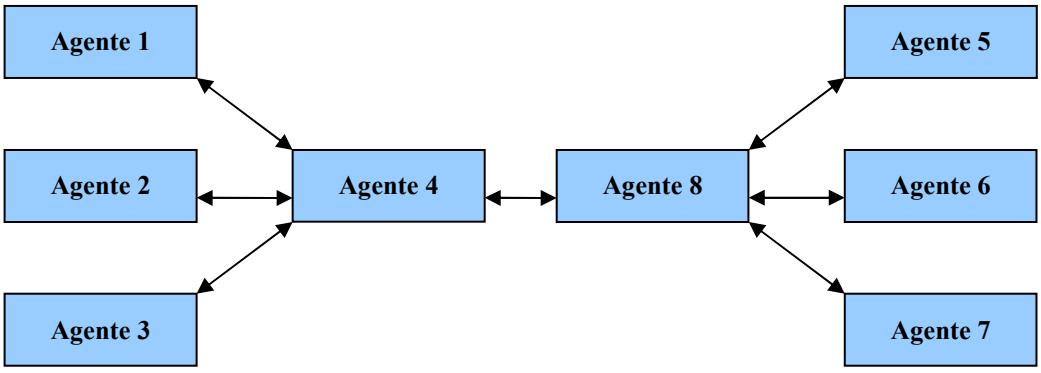

Figura 3.9: Exemplo de comunicação assistida [Silva, 2003]

3.8.3 Comunicação por Difusão (*Broadcast*)

A comunicação por difusão é utilizada quando se deseja enviar mensagens a todos os agentes do sistema ou quando não se sabe para quem enviar a mensagem.

3.8.4 Quadro-Negro (*Blackboard*)

A comunicação por quadro-negro é utilizada de forma análoga a um quadro-negro, onde um agente posta uma mensagem no quadro, realizando uma ação de escrita, e os outros agentes podem acessar a mensagem por meio do quadro-negro, realizando operações de leitura.

3.9 Linguagens de Comunicação

Na comunicação dos agentes, é possível utilizar linguagens de comunicação, através das quais é possível definir um padrão para a troca de mensagens entre os agentes, assim garantindo a comunicação entre eles. Segundo [Meneses, 2001] a comunicação humana e a teoria dos atos comunicativos foram utilizadas para definir a comunicação entre os agentes através de linguagens. Essa teoria usa conceitos e performativas para conduzir as ações, sendo definida abaixo, segundo [Gudwin, 2003]:

- Derivada da análise lingüística da comunicação humana;
- Com uma linguagem, um falante de uma língua não somente efetua uma declaração, mas realiza uma ação;
- Mensagens são ações ou atos comunicativos.

Segundo [Gudwin, 2003], “os atos comunicativos são interpretados a partir da mensagem do contexto”, e “nem sempre essas interpretação é obvia”. Sendo assim, a comunicação pode ser ambígua, levando a interpretações erradas.

A seguir, são descritas algumas dificuldades na comunicação, relatadas por [Gudwin, 2003]:

- “Saia da minha frente!”: indica um comando enviado por um agente.
- “Por favor, saia da minha frente”: indica um pedido enviado por um agente.
- “Você poderia sair da minha frente?”: pergunta feita por um agente.

- “Eu gostaria que você saísse da minha frente”: informação relatada por um agente.

Assim, faz-se necessário explicar o ato comunicativo juntamente com a mensagem enviada durante a comunicação dos agentes.

3.9.1 KQML (*Knowledge Query and Manipulation Language*)

A KQML foi criada pelo *Knowledge Sharing Effort* (KSE), com o objetivo de servir como formato de mensagem e protocolo de gerenciamento de mensagens. Ela contém um protocolo de comunicação de alto nível, proporcionado troca de mensagens independente de conteúdo. Durante a troca de mensagens, seu objetivo principal é a especificação da informação, responsável por garantir a compreensão do conteúdo, se preocupando menos com a mensagem.

De acordo com [Gudwin, 2003], o significado de performativas (atos comunicativos) reservadas e padrões da linguagem KQML é pouco claro: “normalmente estão associadas à intuição”. Alguns problemas e dificuldades da KQML são:

- Ambigüidade e termos vagos;
- Performativas com nomes inadequados, segundo [Gudwin, 2003] “algumas performativas têm nomes que não correspondem diretamente ao ato comunicativo a ela associado”.
- Falta de performativas. Segundo [Gudwin, 2003], “alguns atos comunicativos não estão representados entre as performativas disponíveis”.

3.9.2 FIPA-ACL (*Foundation for Intelligent Physical Agents – Agent Communication Language*)

A FIPA-ACL é uma linguagem baseada em ações de fala. Ela apresenta sintaxe semelhante à KQML, diferenciando-se no conjunto de performativas. Contém um conjunto de tipos de mensagens e descrições dos efeitos da mensagem sobre os agentes que enviam e recebem mensagens [Fipa, 2009].

De acordo com [Gudwin, 2003], a FIPA-ACL “deve vir a substituir o KQML, pois resolve a maioria dos problemas criticados por diferentes autores na concepção da KQML”.

Capítulo 4

Especificação e Modelagem

Neste capítulo são apresentadas as características do sistema desenvolvido, técnicas utilizadas para descrever o seu funcionamento e características particulares da implementação de um jogo computacional.

4.1 Descrição do Jogo Desenvolvido

O jogo tem uma nave que pode ser controlada ou assistida pelo jogador. Em modo controlado, o sistema extrai informações do jogador, armazenando estas informações na base de casos, que inicia vazia. No modo assistido, o comportamento da nave será totalmente autônomo, utilizando-se da base de dados preenchida pelo sistema, quando em modo jogador. A nave, em modo assistido, utiliza técnicas de agentes computacionais, baseados no modelo cognitivo.

Esse agente nave tem como objetivo defender-se e contra-atacar agentes inimigos sempre que possível. A modelagem do seu comportamento será feita utilizando Máquinas de Estados e a técnica de RBC para controlar sua movimentação. Os agentes inimigos são naves com características reativas. Tais naves inimigas têm o objetivo de abater o agente nave, que representa o agente inteligente ou o próprio jogador, dependendo do modo de jogo escolhido. Os ataques realizados pelos agentes inimigos podem ocorrer em grupos, porém estes ataques não têm características determinísticas, ou seja, a seqüência de ataques das naves inimigas não irá se repetir. Busca-se, com este tipo de comportamento, proporcionar ao jogador maior diversão e entretenimento, sendo esses os principais objetivos no desenvolvimento de jogos.

4.2 Questões de Projeto

Com a definição do trabalho, surgiram questões pertinentes ao desenvolvimento do sistema, as quais são apresentadas nas subseções a seguir.

4.2.1 Como Criar os Personagens?

Todo jogo requer a definição e criação dos personagens que farão parte dele. O processo de criação de um personagem em 2D envolve a utilização de editores de imagem, onde é possível desenhar o personagem *pixel* a *pixel*, criando, assim, uma imagem que o represente. O tipo de imagem gerada pode ser variado como jpg, png, etc.

Existem alguns aplicativos que auxiliam no processo de utilizar personagens em jogos como: *RPG Maker* (http://www.rpgmakerbrasil.com/), *Game Maker* (http://www.yoyogames.com/make) e até mesmo o *Paint* (http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/mspaint_overview.mspx?mfr=true) pode ser utilizado [Game Maker, 2009]. Nos aplicativos *RPG Maker* e *Game Maker* existem imagens de personagens previamente desenvolvidas, não havendo necessidade de criar um personagem *pixel* a *pixel*. Já no *Paint* não existem personagens predefinidos, entretanto o aplicativo permite a criação de imagens de personagens *pixel* a *pixel*.

Uma vez que desenhar uma nave de forma satisfatória utilizando-se do *Paint* (ou um aplicativo similar) levaria muito tempo, optou-se, neste trabalho, por utilizar imagens já desenvolvidas de outros jogos. Tais imagens possuíam um plano de fundo indesejado que precisava ser retirado. Por esse motivo, elas foram editadas com um plano de fundo transparente, sendo possível realizar sua plotagem sobre qualquer plano de fundo sem alterar o cenário. Na remoção do plano de fundo das imagens foi utilizado o editor de imagens Gimp [Gimp, 2009].

4.2.2 Como Detectar Colisões e Desenvolver Cenários?

Para o desenvolvimento do cenário foi utilizada a técnica de *Tile-Based Map*, muito difundida entre jogos em 2D. Segundo [Brackeen, Barker e Vanhelsuwé, 2003], essa técnica consiste em criar um cenário a partir de um conjunto de blocos (*tile*), os quais têm a forma de quadrados. Para a representação do cenário esse conjunto de blocos é representado em forma de matriz. Neste trabalho, o tamanho dos blocos na matriz ficou definido em 60 x 60 *pixels*.

A divisão do mundo (cenário) em blocos facilita o tratamento de colisão, pois cada objeto contido no cenário localiza-se em uma posição (x, y). Esses blocos conterão objetos ou personagens, que podem se mover em várias direções. A posição de destino almejada pelo personagem deve ser verificada antes da locomoção, sendo que essa verificação caracteriza a detecção de uma colisão. Após detectar a colisão é possível tratá-la de forma adequada.

4.2.3 Como Utilizar a Técnica de RBC com Uma Máquina de Estados?

O comportamento do nosso agente no modo assistido é definido por uma Máquina de Estados e sua locomoção pela técnica RBC. Seus estados representam as ações tomadas pelo agente, sendo essas ações: atirar, mover, morrer, agente retorna ou permanece no centro. Porém, onde usar o RBC na tomada de decisão do agente?

A única ação definida pelo agente que requer raciocínio é a ação de mover, as demais ações não necessitam de lógica associada. Então, quando o agente, em modo assistido, necessitar se locomover, utiliza-se a técnica RBC para relembrar situações passadas onde ele obteve sucesso na movimentação. Ao recuperar o melhor caso, o agente realiza a ação mover.

4.3 Características Gerais de Implementação

Para descrever o sistema proposto, apresenta-se como será desenvolvida sua arquitetura, proporcionando uma visão geral do sistema desenvolvido. Segundo [Perucia *et al*., 2007], a arquitetura de um jogo geralmente é dividida em abstração de interface de *hardware* e abstração de interface de jogo.

A abstração de interface de *hardware* facilita a manipulação dos elementos físicos do jogo, como *mouse*, teclado, *joystick*, som, rede, entre outros, servindo como uma camada intermediária entre o *hardware* e o aplicativo (neste caso, o jogo).

Segundo [Perucia *et al*., 2007], o isolamento do *hardware* com essa camada de abstração possibilita algumas vantagens, descritas a seguir:

- Isola especificidades de *hardware*;
- Facilita a atualização do hardware sem interferir no código do jogo;
- Facilita a portabilidade do jogo para outra plataforma;
- Permite o desenvolvimento de componentes reusáveis de fácil uso e configuração;
- Permite ao desenvolvedor preocupar-se somente com o jogo;

- Torna o jogo mais limpo e consistente.

Já a abstração de interface de jogo proporciona uma idéia geral de como é o funcionamento do jogo, com seus subsistemas e interações, sem se preocupar com o *hardware*. Essa abstração de jogo, segundo [Perucia *et al*., 2007], apresenta os seguintes subsistemas:

- Interface;
- Tratador de eventos;
- Gerenciador de dados;
- Gerenciador de física;
- *Engine* gráfica;
- *Engine* sonora;
- *Engine* lógica;
- Camadas de abstração de *hardware*;
- Sistema de configuração de jogo;
- Sistema de menus;
- Sistema de ajuda;
- Sistema de música.

Dependendo da arquitetura do jogo, alguns dos itens citados anteriormente, tanto na abstração de interface de *hardware* como na interface de abstração de jogo, podem ser adicionados ou removidos para se adequar ao jogo desenvolvido.

Neste trabalho, será utilizada a abstração de *hardware* e alguns subsistemas apresentados sobre a abstração de jogo, como sistema de menus, camadas de abstração de *hardware* e interface.

4.4 Descrição do Funcionamento do Jogo

Para a descrição do funcionamento do jogo proposto, utiliza-se Máquina de Estados, tornando possível representar, também, o comportamento dos agentes desenvolvidos. Segundo [Perucia *et al.*, 2007], Máquinas de Estado são ferramentas muito importantes para o planejamento do jogo. Então, com o objetivo de se ter uma visão geral do funcionamento do

jogo, deve-se desenvolver um autômato para representá-lo. A seguir, são apresentados os estados que representam o funcionamento do jogo desenvolvido.

- **Menu:** O estado menu apresenta as opções iniciais do jogo, onde é possível iniciar o jogo, em modo jogador ou assistido, ou sair do jogo.
- **Jogo:** Este estado representa a execução do jogo, compreendendo os seus dois modos. Ao pressionar a tecla *Esc*, pode-se sair do jogo. Existem interações neste estado apenas se escolhido o Modo Jogador.

A Figura 4.1 ilustra o diagrama representa uma máquina de estados que exemplifica o funcionamento do sistema, proporcionado uma idéia geral do seu funcionamento.

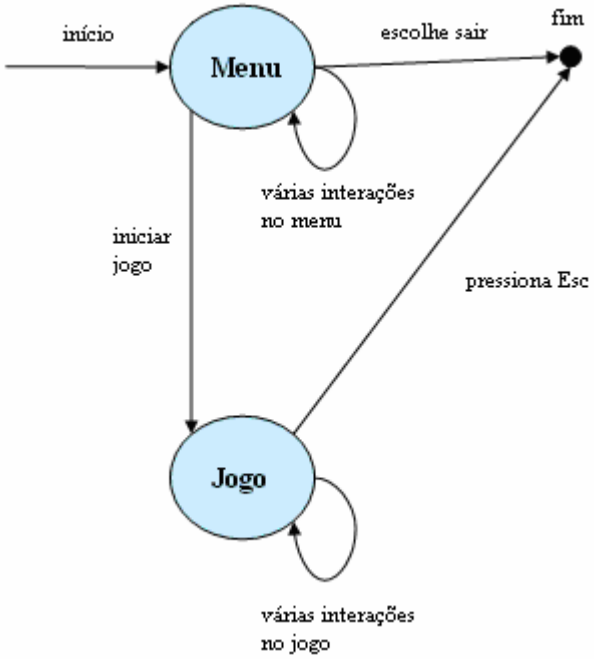

Figura 4.1: Ilustração do funcionamento do jogo em alto nível de abstração

O funcionamento do jogo segue a Máquina de Estados ilustrada na Figura 4.1. Esses estados de alto nível de abstração podem ser especificados em estados de níveis inferiores, detalhando ainda mais o sistema.

4.4.1 Funcionamento do Estado Menu

Diminuindo o nível de abstração do estado Menu, apresenta-se a Figura 4.2.

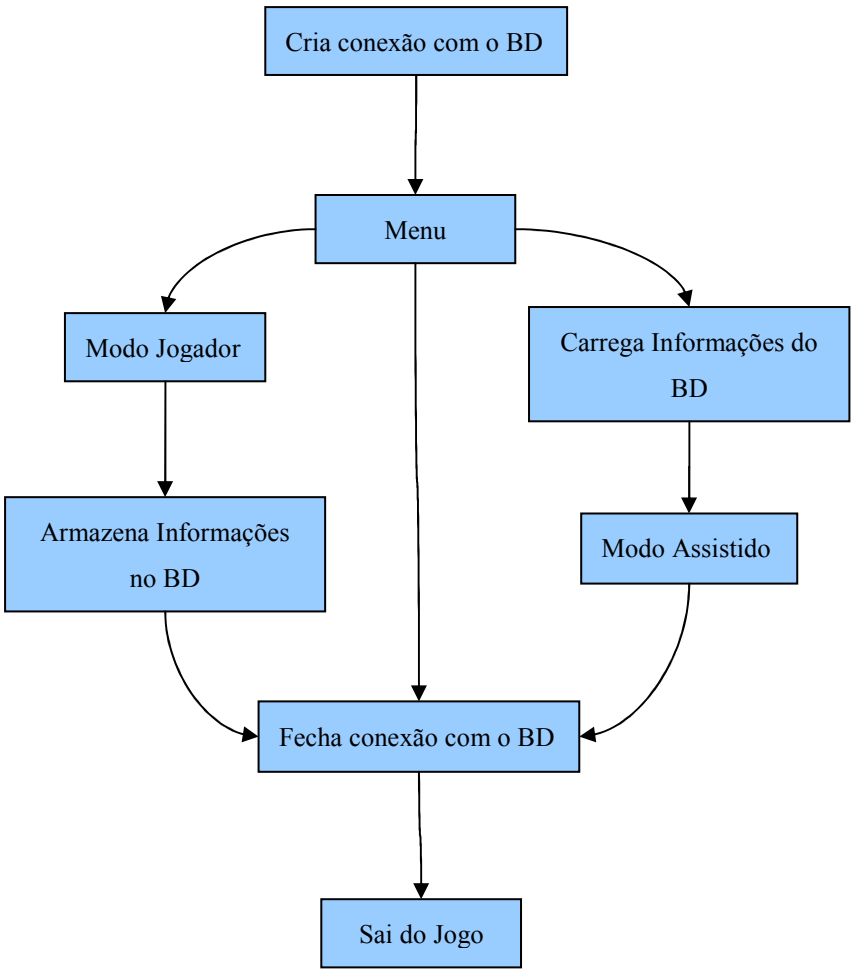

Figura 4.2: Funcionamento do estado Menu

Inicialmente, o sistema cria uma conexão com o banco de dados, assim, evita-se *delay* durante a execução do jogo. Depois de criada a conexão com o banco de dados, é possível escolher dois modos de jogo ou sair do jogo.

Se for escolhido o Modo Jogador, o jogo é iniciado, as informações coletadas pelo sistema durante o jogo são armazenadas em memória até que o usuário aperte a tecla *Esc*, para terminar o jogo. Então o sistema salva essas informações no banco de dados, de forma que o processo de escrita no banco de dados não afete o desempenho do jogo.

Em Modo Assistido, inicialmente, os dados armazenados em Modo Jogador são carregados para a memória, evitando perda de desempenho durante processos de leitura de informações no banco de dados. Se o jogador escolher o Modo Assistido e não existir casos armazenados em sua base ele não ira se mover. Neste modo o usuário apenas visualiza o a sua

nave atirando e esquivando-se dos tiros inimigos, podendo acompanhar a evolução do agente de forma visual. Quando apertar a tecla *Esc*, o jogo é encerrado.

Enquanto estiver no Menu principal do jogo, o usuário também pode sair do sistema sem, necessariamente, escolher um modo de jogo. Neste caso, nenhuma informação é armazenada ou carregada do banco de dados.

4.4.2 Funcionamento do Modo Jogador

Diminuindo a abstração do estado Modo Jogador, apresenta-se a Figura 4.3.

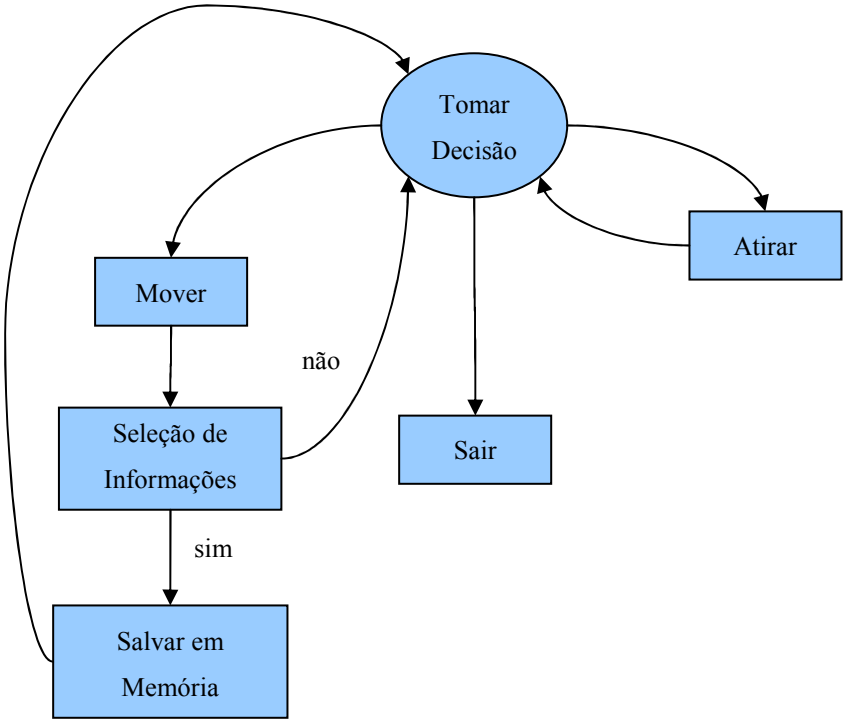

Figura 4.3: Funcionamento do estado Modo Jogador

O usuário pode interagir com o jogo através do teclado. Para atirar, o usuário deve utilizar a tecla “f”. Caso o disparo acerte algum inimigo, ele será removido do jogo. Para mover sua nave, o usuário deve utilizar as setas do teclado. Quando uma dessas teclas for pressionada, o sistema verifica se a nave ainda está viva e, então, identifica os tiros próximos a ela em uma área de 40 *pixels* em volta da sua posição atual. Depois dessa fase é definido um caso, atribuindo valores aos seus sintomas. Se o caso passar pelo processo de seleção, ele é armazenado em memória, para depois ser persistido em disco. Se o caso não passar pelo processo ele não é armazenado em memória. Para sair do Modo Jogador, basta apertar o botão

Esc. Além do usuário, existem dois agentes atuando no sistema: um atualiza os tiros do cenário e o outro atualiza os inimigos do ambiente. O comportamento dos agentes será detalhado melhor na Seção 4.6.

4.4.3 Funcionamento do Modo Assistido

Neste estado existem três agentes atuando no ambiente são eles: Agente Nave, Agente Gerenciador de Tiros e Agente Gerenciador de Inimigos. O usuário apenas visualiza a nave se movendo e atirando contra os inimigos, quando todos os inimigos estiverem mortos, ele poderá sair do jogo. O comportamento dos agentes será detalhado melhor na Seção 4.6.

4.4.4 Estado Inicial do Jogo

Supondo a escolha do Modo Jogador no menu principal do jogo. O Jogo inicia com a nave localizada no centro do cenário. Nesse cenário existem três naves inimigas, que realizam movimentação no sentido horizontal, acima do jogador. Os inimigos atacam o jogador efetuando disparos contra ele de forma aleatória, sendo que os tiros se deslocam no sentido vertical. O jogador pode, então, por meio do teclado desviar dos tiros inimigos e contra atacá-los.

No Modo Assistido, a nave que representa o jogador também encontra-se no centro do cenário, assim como no Modo Jogador. Ela realiza movimentação e ataques de forma autônoma. Os inimigos se movimentam e atacam da mesma forma do Modo Jogador. A Figura 4.4 representa a tela inicial do sistema.

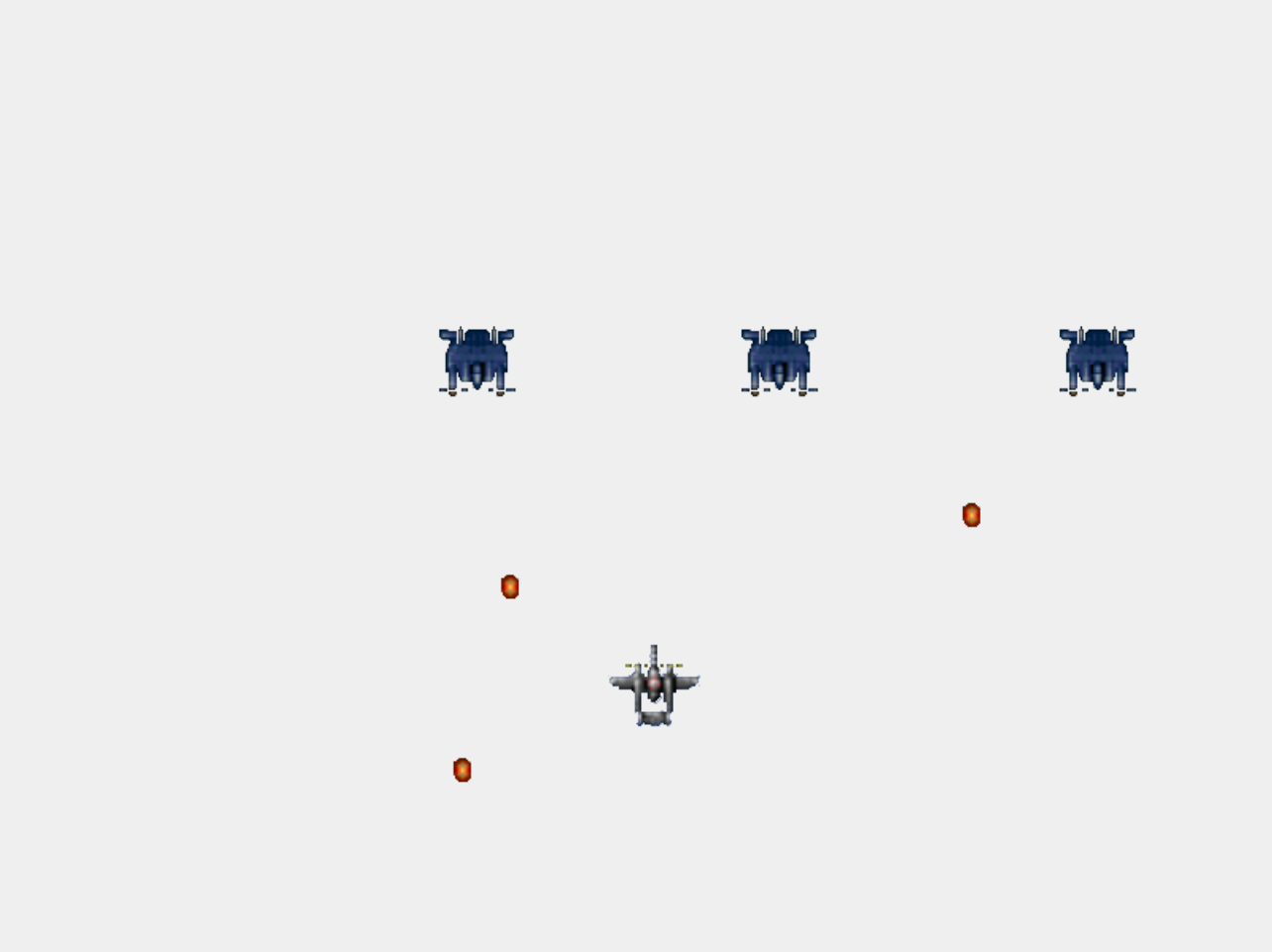

Figura 4.4: Ilustração da tela inicial do sistema

As naves alinhadas na parte na parte superior da imagem são os inimigos. As bolas vermelhas são os tiros dos inimigos e os tiros azuis são os do jogador. A nave mais abaixo representa o jogador, quando em modo Jogador, ou o agente, quando em modo Assistido. As imagens utilizadas no sistema foram retiradas do jogo *Sonic Wings Especial* [Konami, 2009].

4.4.5 Funcionamento da Seleção de Informação

Este estado define o padrão de comportamento mais adequado ao agente, pois, no problema apresentado, nem sempre um caso resolvido é interessante para o comportamento do agente. Por exemplo, quando vem um tiro sobre o agente e outro tiro à sua direita, é interessante que ele se mova para a esquerda. No entanto, o RBC não restringe a inserção de casos onde o usuário desloca-se para a direita, o que pode gerar casos possíveis, porém indesejados. Para resolver esse problema, foi desenvolvido um processo de seleção dos casos

armazenados na base. Esse processo faz com que sejam armazenados na base os casos mais adequados, não permitindo que sejam armazenados os seguintes casos, no Modo Jogador:

- Quando o tiro vem sobre a nave e a ação do usuário é ficar parado;
- Quando o tiro vem pela direita e a ação do usuário é mover-se para a direita;
- Quando o tiro vem pela esquerda e a ação do usuário é mover-se para a esquerda.

4.5 Casos do RBC

Na solução apresentada, o armazenamento de casos na base é feito com o *framework Hibernate* [Hibernate, 2009], *Hibernate JPA* e banco de dados *PostgreSQL* [PostgresSQL, 2009]. Os casos definidos correspondem a situações em que o agente inteligente se encontra em perigo em relação aos tiros que estão à sua volta. Inicialmente, a base de casos encontra-se vazia, esperando por informações retiradas e armazenadas durante partidas realizadas em Modo Jogador. Após a base ser preenchida por um conjunto de casos resolvidos pelo jogador, é possível utilizar o modo assistido de forma adequada.

Os casos definidos para o RBC abordam três tipos básicos de casos: tiro(s) tem uma trajetória que passa(m) exatamente sobre a posição (na mira) do agente, tiro(s) tem uma trajetória que passa(m) ao lado do agente. Esses tiros laterais podem estar à sua direita ou à sua esquerda. Com a combinação dos três casos, podem ser gerados todos os outros casos. As figuras (Figura 4.5, Figura 4.6 e Figura 4.7) ilustram os casos básicos:

- Caso: tiro(s) tem uma trajetória que passa(m) exatamente sobre a posição do agente;

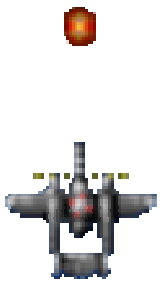

Figura 4.5: Caso RBC, tiro(s) sobre o agente

- Caso: tiro(s) tem uma trajetória que passa(m) à esquerda do agente;

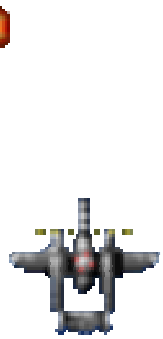

Figura 4.6: Caso RBC, tiro(s) passam a esquerda do agente

- Caso: tiro(s) tem uma trajetória que passa(m) à direita do agente.

Figura 4.7: Caso RBC, tiro(s) passam a direita do agente

A partir da combinação desses três casos, é possível realizar a tomada de decisão sobre o deslocamento do agente, resultando no comportamento inteligente desejado do agente.

4.5.1 Representação do RBC

A modelagem dos casos pode ser demonstrada através de tabelas, onde são apresentados os sintomas e uma ação a ser tomada pelo agente. Os sintomas definem o estado do ambiente. O sintoma 1 representa o caso onde o tiro(s) passa sobre (na mira) o agente. O sintoma 2 indica que existe um tiro lateral, o qual pode estar à esquerda, representando o sintoma 3, ou à direita, representando o sintoma 4. A coluna Ação determina qual ação o agente deve tomar, com base nos sintomas descritos. Ela pode assumir os valores E

(esquerda), D (direita) e P (parado). A Tabela 4.1 mostra exemplos de casos reais armazenados na base.

Tabela 4.1: Casos básicos do RBC.

Sintoma 1	Sintoma 2	Sintoma 3	Sintoma 4	Ação
FALSE	FALSE	FALSE	FALSE	P
FALSE	TRUE	TRUE	FALSE	P
TRUE	FALSE	FALSE	FALSE	E
TRUE	TRUE	FALSE	TRUE	E
FALSE	TRUE	FALSE	TRUE	P
FALSE	TRUE	FALSE	TRUE	E
FALSE	TRUE	TRUE	TRUE	P
TRUE	TRUE	TRUE	FALSE	D
FALSE	TRUE	TRUE	FALSE	D

Os sintomas têm valores associados, apenas para identificá-los dentro de um *switch* no sistema. Eles possuem pesos diferenciados: sintomas 1 e 2 têm peso 10, o sintomas 3 tem peso 5 e o 4 tem peso 1. Os sintomas 3 e 4 têm valores diferentes para que não existam casos diferentes com o mesmo peso.

A tabela de casos pode ser gerada no *Pgadmin III*, com a SQL apresentada a seguir. Para a criação da tabela, o campo OWNER TO deve ser alterado para o nome de usuário do banco de dados utilizado:

```
CREATE TABLE casos
(
  id serial NOT NULL,
  sintomaprincipal1 boolean,
  sintomaprincipal2 boolean,
  sintomasecundario1 boolean,
  sintomasecundario2 boolean,
  acao text,
  CONSTRAINT casos_pkey PRIMARY KEY (id)
)
WITH (OIDS=FALSE);
ALTER TABLE casos OWNER TO yuri;
```

4.5.2 Similaridade

A similaridade utilizada é baseada no método de indexação, definido na Seção 3.5.4.1. Neste caso, as entidades de informação são os sintomas: sintomaprincipal1, sintomaprincipal2, sintomasecundario1 e sintomasecundario2, que constituem os elementos que definem o caso a ser recuperado.

4.6 Comportamento do Agente Inteligente

O comportamento do agente inteligente utiliza uma técnica de IA chamada de RBC (Seção 3.5), que será responsável pelo aprendizado do agente. Juntamente com essa técnica, foram utilizadas Máquinas de Estados na definição do seu comportamento. Segundo [Schwab, 2004], Máquinas de Estado já vêm sendo utilizadas em conjunto com técnicas de aprendizado. Um exemplo disso é a técnica conhecida como *Fuzzy-State Machines* (*FuSMs*), que combina Máquina de Estados com Lógica Fuzzy.

4.6.1 Representação do comportamento do Agente Inteligente

Segundo [Perucia *et al.*, 2007], o comportamento do agente pode ser expresso por uma Máquina de Estados Finitos. Portanto, neste trabalho, demonstra-se o comportamento do agente através de uma Máquina de Estados.

Essa Máquina de estados contém 4 estados: Atirar, Agente retorna ou está no centro, Mover-se e Morrer. O estado Atirar indica que existe um inimigo na mira, então o agente deve atirar. Levando em consideração que a velocidade do tiro do agente e a velocidade dos inimigos são iguais (valor igual a 10 *pixels* de deslocamento) e os inimigos realizam uma trajetória perpendicular em relação ao agente, utilizando a regra do triângulo retângulo é possível definir o local ideal para se disparar um tiro. O estado Agente retorna ou está no centro faz com que o agente retorne à posição de origem (centro do cenário). Com este comportamento, evita-se que o agente fique nas laterais do cenário, onde se tornaria um alvo mais fácil para os inimigos, pois nas laterais a área de movimentação é menor. O estado Mover-se indica que o agente encontra-se em situação de risco em relação aos tiros inimigos,

logo, deve movimentar-se, desviando dos tiros que o ameaçam. O estado Morrer indica que o agente foi abatido, então sua imagem é removida da tela.

As transições Inimigo no Alvo, Inimigo fora do alvo, Perigo, Fora de perigo e Abatido realizam as mudanças de estado. A sua compreensão é fundamental para o entendimento da tomada de decisão realizada pelo agente, na Máquina de Estados. Inimigo no Alvo indica que o inimigo está em uma posição onde, se o agente efetuar um disparo, o inimigo detectado será atingido. Inimigo fora do alvo indica que, no momento atual, não existe inimigo no alvo. Perigo indica que existe pelo menos um tiro detectado pelo radar da nave, logo, esse tiro representa um perigo ao agente por estar próximo a ele. O radar é uma área quadrada em torno do agente, sua área de cobertura é de 40 *pixels* a partir das bordas da imagem do personagem. Fora de perigo indica que não existe nenhum tiro detectado pelo radar. Abatido indica que algum tiro atingiu a nave.

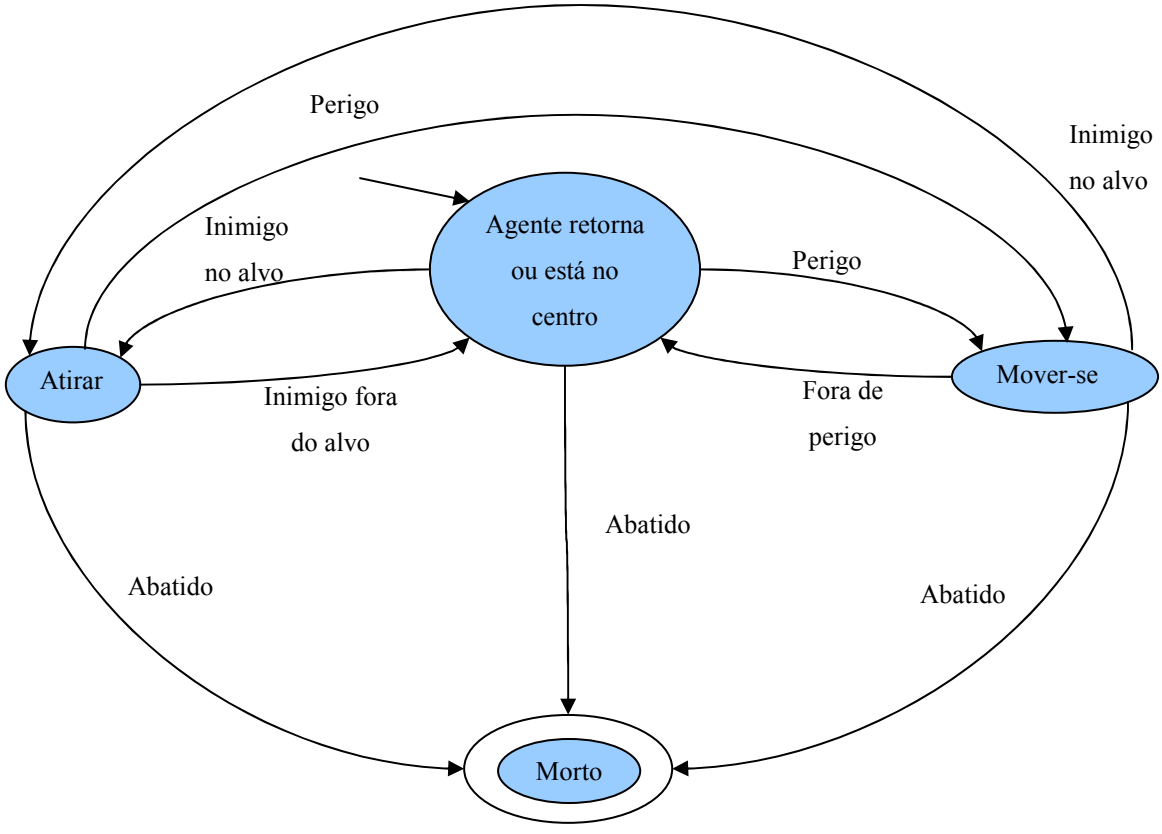

Figura 4.8: Máquina de Estados do agente inteligente

O RBC é chamado quando o agente se encontra no estado Mover-se. Nesse estado o agente define o problema atual e realiza a busca de similaridade utilizando o método de

indexação, retornando o caso mais similar. Caso este caso exista, ele é, então, adotado como critério de movimentação.

4.6.2 Comportamento do Agente Gerenciador de Inimigos

O Agente Gerenciador de Inimigos tem características reativas, é responsável por atualizar todos os inimigos, deslocá-los, solicitar disparos e caso algum dos inimigos seja abatido, removê-lo da partida. A Máquina de Estados da Figura 4.9 define o método de atualização efetuado pelo agente nos inimigos do cenário.

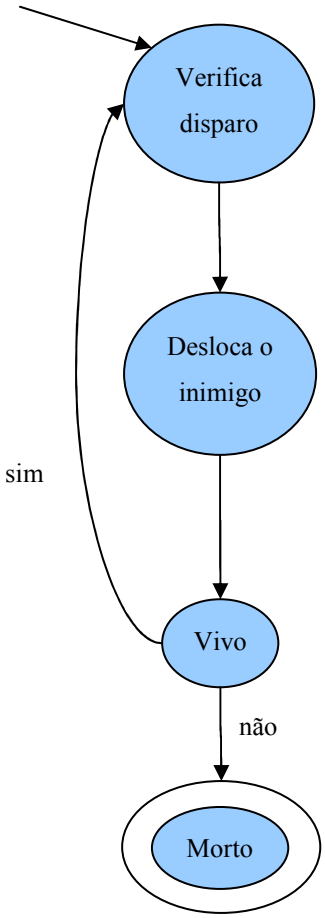

Figura 4.9: Máquina de Estados do Gerenciador de Inimigos

No estado Verifica Disparo, o agente verifica o tempo que define o intervalo de disparos, que tem valor inicial 30. Caso o valor seja menor ou igual a 0, um disparo é efetuado

e o intervalo de disparos recebe valor 30, mais uma vez. Caso contrário, o valor do intervalo de disparos é decrementado. A fórmula utilizada para decrementar o tempo de disparo é IT = IT – randômico (esse valor randômico varia entre 0 e 2), sendo que a variável IT é o intervalo de tempo. No estado Desloca Inimigo o agente desloca o inimigo em 10 *pixels* para a direita da tela. No estado Vivo o agente verifica se o inimigo atual está vivo ou não. Caso esteja vivo, quando voltar a atualizar este inimigo, iniciará o processo novamente no estado Verificar Disparo. Caso contrário, o inimigo foi morto e deve ser removido do jogo, alterando seu estado para Morto.

4.6.3 Comportamento do Agente Gerenciador de Tiros

Este agente tem características reativas, sendo responsável por atualizar todos os tiros do cenário, inserir e removê-los. A seguir a Figura 4.10 representa o funcionamento do agente gerenciador de tiros.

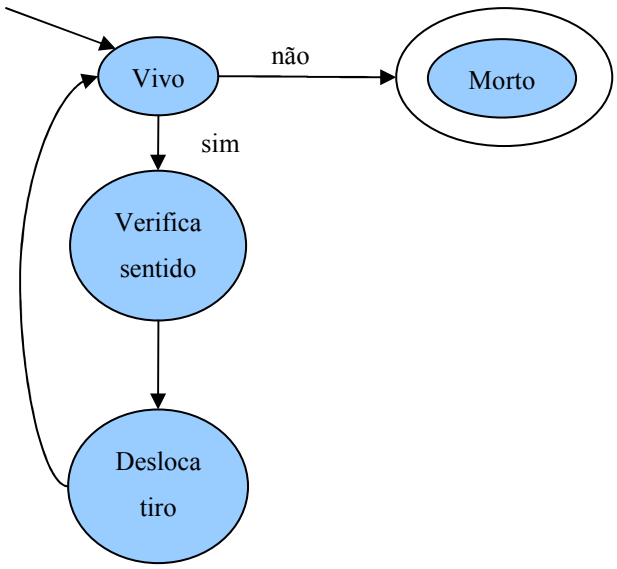

Figura 4.10: Máquina de Estados do Gerenciador de tiros

Durante o estado Vivo é verificado o tempo de vida do tiro (valor inicial 70). No estado Verifica sentido é verificado o sentido do tiro, para realizar seu deslocamento para cima ou para baixo. No estado Desloca tiro, desloca-se o tiro no sentido correto e decrementa-

se o tempo de vida do tiro em uma unidade. O tempo de vida do tiro é utilizado para determinar se ele saiu da tela, quando isso ocorrer ele deve ser removido do jogo.

4.6.4 Comportamento dos Agentes no Ambiente do Jogo

A seguir apresenta-se um diagrama que define as interações entre os agentes no jogo, em Modo Assistido, na Figura 4.11:

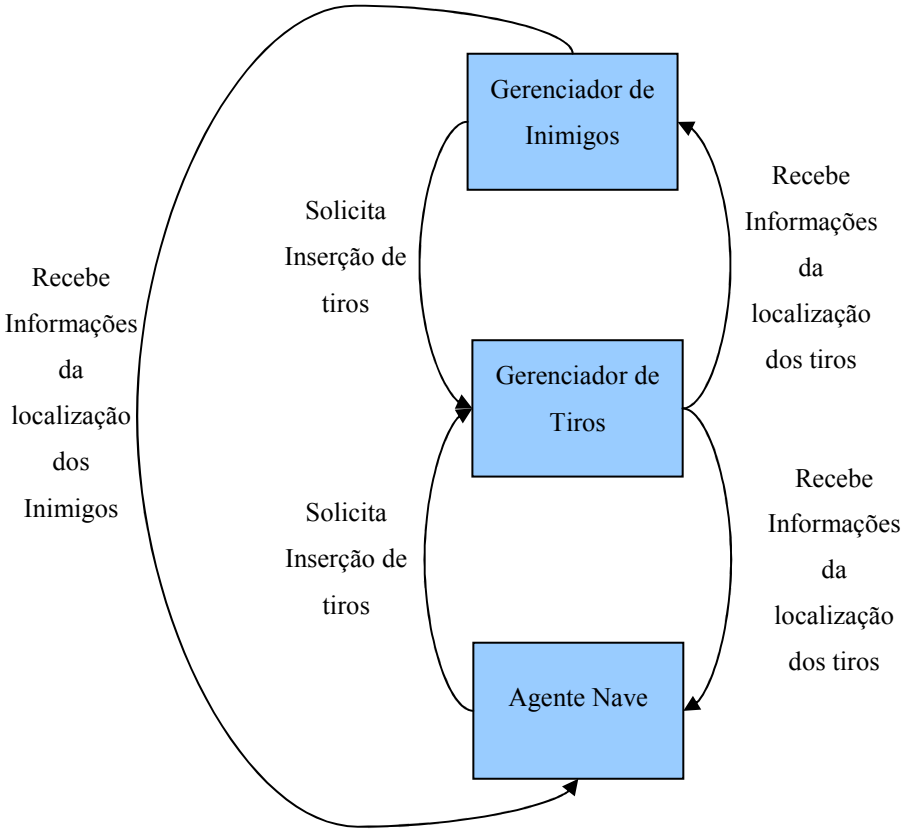

Figura 4.11: Interação dos Agentes em Modo Assistido

O agente gerenciador de inimigos recebe informações dos tiros para definir quais foram os inimigos abatidos ou não, removendo os atingidos e atualizando os vivos, também solicita a inserção de tiros no jogo ao agente que regência os tiros. O agente que gerencia os tiros desloca os tiros e remove os tiros que estão fora da tela, para isso verifica o tempo de vida de cada tiro atualizado. O agente nave também envia dados aos demais agentes do sistema sobre a localização dos tiros. O Agente Nave recebe informações dos tiros, para determinar se continua vivo ou já foi abatido. Também recebe informações das posições dos

inimigos para determinar a hora certa de efetuar o disparo contra um inimigo, quando um inimigo está no alvo então o disparo é solicitado ao agente gerenciador de tiros.

4.7 Documentação

Nesta seção apresenta-se toda a documentação associada à etapa de implementação deste trabalho. A documentação de um jogo em particular envolve a elaboração de um *Document Designer,* o qual foi desenvolvido e encontra-se abaixo. São apresentados, também, o dia o diagrama de classes e o de casos de uso do sistema.

4.7.1 *Document Designer*

A elaboração de jogos eletrônicos envolve documentação. Segundo [Perucia *et al*., 2007], em projetos pequenos, apenas o desenvolvimento do *Document Designer* é suficiente para documentar todo o projeto de desenvolvimento de um jogo. Baseando-se no *Document Designer* apresentado por [Flynt e Salem, 2005], foi realizada uma adaptação ressaltando os itens mais relevantes descritos por ele. A seguir, apresenta-se o *Document Designer* do jogo desenvolvido nesta monografia.

Raptor: *Game Design Document*

Introdução

Com o passar dos anos, a população na terra cresce constantemente. Com isso, a escassez de reservas naturais é inevitável. Se aproveitando desta situação, a organização *Nogir* coloca em prática o seu plano de dominar o mundo. Devido à escassez de recursos hídricos no mundo, essa organização instalou bases militares nas maiores bacias hidrográficas do mundo: Bacia Amazônica – Brasil 7.050.000 km², Bacia do Congo – Zaire 3.690.000 km², Bacia do Mississippi – EUA 3.328.000 km², Bacia do Obi – Rússia 2.975.000 km², Bacia do Nilo – Egito 2.867.000 km², Bacia do Níger – Nigéria 2.092.000 km² e Bacia do Rio Amarelo – China 1.807.199 km². Através do controle dos recursos hídricos mundial, a *Nogir* deixa o planeta refém de suas vontades.

Em uma reunião feita com máxima urgência pela ONU (Organização das Nações Unidas), foi definido que cada país deve enviar um avião caça com o seu melhor piloto, para

percorrer o mundo e destrur todas as bases criadas pela organização *Nogir*, com isso, restaurando a paz no mundo.

Gênero

Ação /tiro.

Título

Raptor.

Controle

Como o jogo possui duas modalidades, a nave só poderá ser controlada via teclado no modo jogador, enquanto no modo assistido o jogador apenas visualiza o comportamento do personagem interagindo com os inimigos, sem poder controlá-lo.

Unidades e Personagens

Em Raptor, o jogador é uma nave que se utiliza de manobras e tiros em sentido vertical para abater os seus inimigos. Os inimigos também são naves e sua representação é diferente da utilizada para o jogador. Os inimigos realizam ataques com tiros no sentido vertical. Jogadores e inimigos não possuem atributos relevantes.

Classes de personagens

As classes de personagens são: inimigo e jogador. Essas classes possuem as mesmas características. A diferença entre as classes está no seu comportamento.

Armas

A arma que o personagem jogador dispõe é uma metralhadora de tiros simples. Qualquer dano é fatal a uma nave atingida.

Dinâmica de fases

Acontece o confronto entre inimigos e o jogador em uma única fase. Os inimigos atacam em trio, efetuando rajadas de tiros em sentido vertical, o jogador pode desviar dos tiros, se necessário, e contra-atacar, atirando nos inimigos. O jogo acaba quando o jogador for

abatido ou abater todos os inimigos. Se o jogador sobreviver aos ataques inimigos e abatê-los, ele vence o jogo. Se ele for abatido pelos inimigos, perde o jogo.

Movimentação

A movimentação do jogador é feita utilizando as setas do teclado cima, baixo, esquerda e direita, sendo estas as únicas direções que o personagem pode utilizar para se locomover.

Ataques

Os ataques feitos pelo jogador são rajadas de tiros no sentido vertical. Os disparos são acionados pelo botão "f". Os ataques realizados pelos inimigos são rajadas de tiros feitas no sentido vertical.

Vida

O jogador só tem uma vida, portanto, quando abatido uma vez, ele perde o jogo.

Descrição das fases

O jogo só tem uma fase, que não possui nenhum preenchimento. A fase é composta por uma cena fixa, que não se desloca da cena inicial. Não existe nenhum tipo de colisão com elementos do cenário, pois não existe cenário.

Áudio

O jogo não possui nenhum efeito de áudio.

Através do *Document Designer* é possível se ter uma noção global da idéia do jogo. O desenvolvimento deste documento auxilia no desenvolvimento, fazendo uma equipe trabalhar em conjunto sobre a mesma idéia.

4.7.2 Diagrama de Casos de Uso

No diagrama de casos de uso (Figura 4.12) são apresentadas as funcionalidades do sistema.

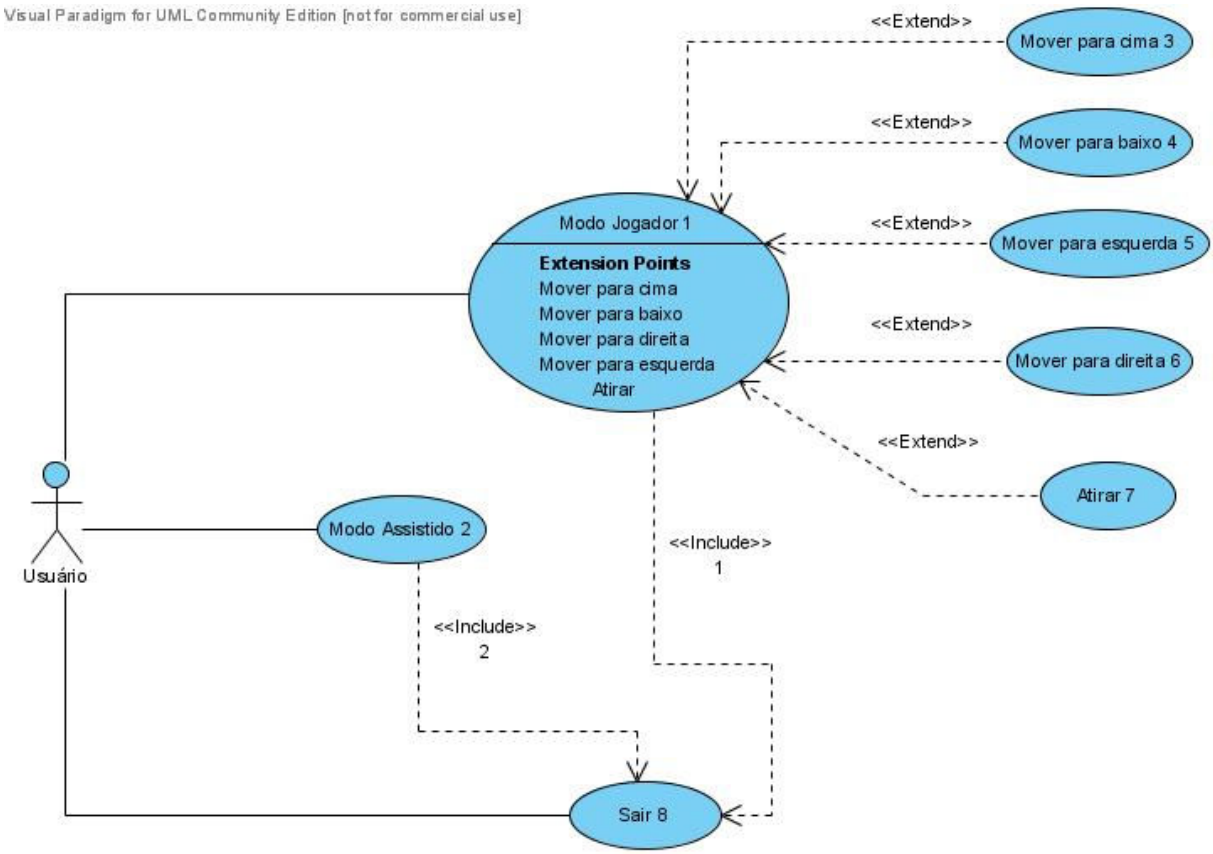

Figura 4.12: Diagrama de Casos de Uso

Modo Jogador

*Caso de uso 1: O usuário deve interagir no ambiente, atacando e desviando sua nave dos tiros inimigos, enriquecendo a base de informações do RBC, que será utilizada no modo assistido.

Informação Característica

*Objetivo no Contexto: Interagir com o sistema, enriquecendo a base de dados que será utilizada no modo assistido.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Usuário.

*Pré-Condições: O modo jogador deve ser escolhido, no menu principal.

*Condição Final de Sucesso: O usuário deve interagir com o sistema o suficiente para enriquecer a base de dados com informações relevantes ao modo assistido.

*Condição Final de Falha: O usuário não adicionar nenhuma informação à base de dados.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Adicionar informações relevantes à base de dados, movendo-se e atacando os inimigos.

*Passo 2: Aperta o botão *Esc* para sair do sistema <<Include 1>>.

Cenário Secundário de Sucesso

*Passo 1, apertar seta para cima: <<Extend 3>>

*Passo 1, apertar seta para baixo <<Extend 4>>

*Passo 1, apertar seta para esquerda <<Extend 5>>

*Passo 1, apertar seta para direita <<Extend 6>>

*Passo 1, apertar o botão "f" <<Extend 7>>

Mover para Cima

*Caso de uso: O usuário deve apertar a seta para cima do teclado, movendo o personagem controlado por ele para cima.

Informação Característica

*Objetivo no Contexto: Mover a nave controlada pelo personagem para cima.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Contexto.

*Pré-Condições: O modo jogador deve ser escolhido.

*Condição Final de Sucesso: A nave movimentar-se para cima.

*Condição Final de Falha: A nave não se movimentar.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Com o teclado, apertar a tecla para cima.

Mover para Baixo

*Caso de uso: O usuário deve apertar a seta para baixo do teclado, movendo o personagem controlado por ele para baixo.

Informação Característica

*Objetivo no Contexto: Mover a nave controlada pelo personagem para baixo.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Contexto.

*Pré-Condições: O modo jogador deve ser escolhido.

*Condição Final de Sucesso: A nave movimentar-se para baixo.

*Condição Final de Falha: A nave não se movimentar.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Com o teclado, apertar a tecla para baixo.

Mover para Esquerda

*Caso de uso: O usuário deve apertar a seta esquerda do teclado, movendo o personagem controlado por ele para a esquerda.

Informação Característica

*Objetivo no Contexto: Mover a nave controlada pelo personagem para a esquerda.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Contexto.

*Pré-Condições: O modo jogador deve ser escolhido.

*Condição Final de Sucesso: A nave movimentar-se para esquerda.

*Condição Final de Falha: A nave não se movimentar.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Com o teclado, apertar a tecla para esquerda.

Mover para Direita

*Caso de uso: O usuário deve apertar a seta direita do teclado, movendo o personagem controlado por ele para a direita.

Informação Característica

*Objetivo no Contexto: Mover a nave controlada pelo personagem para a direita.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Contexto.

*Pré-Condições: O modo jogador deve ser escolhido.

*Condição Final de Sucesso: A nave movimentar-se para direita.

*Condição Final de Falha: A nave não se movimentar.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Com o teclado, apertar a tecla para cima.

Atirar

*Caso de uso: O usuário deve apertar tecla “f”, efetuando disparos contra os inimigos.

Informação Característica

*Objetivo no Contexto: Atirar contra os inimigos.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Contexto.

*Pré-Condições: O modo jogador deve ser escolhido.

*Condição Final de Sucesso: A nave deve atirar contra os inimigos.

*Condição Final de Falha: A nave não atirar.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Apertar a tecla “f”.

Modo Assistido

*Caso de uso: O usuário pode verificar o progresso do aprendizado da sua nave, visualizando o seu comportamento.

Informação Característica

*Objetivo no Contexto: Visualizar o nível de aprendizagem adquirido pela nave.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Sistema.

*Pré-Condições: O modo assistido de ser escolhido.

*Condição Final de Sucesso: A nave movimentar-se de forma adequada.

*Condição Final de Falha: A nave não se movimentar de forma adequada, indicando, ainda, que faltam informação relevantes à sua base de dados, as quais devem ser aprendidas.

*Ator Primário: Sistema.

Cenário Principal de Sucesso

*Passo 1: Visualizar o comportamento do sistema.

*Passo 2: Finalizar o sistema <<Include 2>>.

Sair

*Caso de uso: O usuário deve apertar a tecla *Esc* para sair do jogo ou sair do menu principal. Quando sai do modo jogador, salva informações na base de dados do RBC. Quando sai do modo assistido, apenas finaliza o sistema.

Informação Característica

*Objetivo no Contexto: Finalizar o sistema. Dependendo do modo de jogo escolhido, salvar informações na base de dados do RBC.

*Escopo: Sistema Computacional.

*Nível: Objetivo de Sistema.

*Pré-Condições: O modo jogador ou assistido deve ser iniciado ou o usuário deve estar no menu principal.

*Condição Final de Sucesso: O sistema ser finalizado e, se estiver em modo jogador, salvar informações da partida na base do RBC.

*Condição Final de Falha: O jogo não encerrar ou não salvar as informações da partida, quando em modo jogador.

*Ator Primário: Usuário.

Cenário Principal de Sucesso

*Passo 1: Quando em modo jogador, o sistema salva as informações da partida. Em modo assistido, apenas sai do sistema. Se estiver no menu principal, sai do sistema.

4.7.3 Diagrama de Classes

No diagrama de classes (Figura 4.13) é possível verificar abstrações de *hardware* e *software* realizadas no sistema. Como o foco do sistema é desenvolver um agente que possa aprender com o usuário, melhores práticas de Engenharia de Software podem alterar o diagrama de classes.

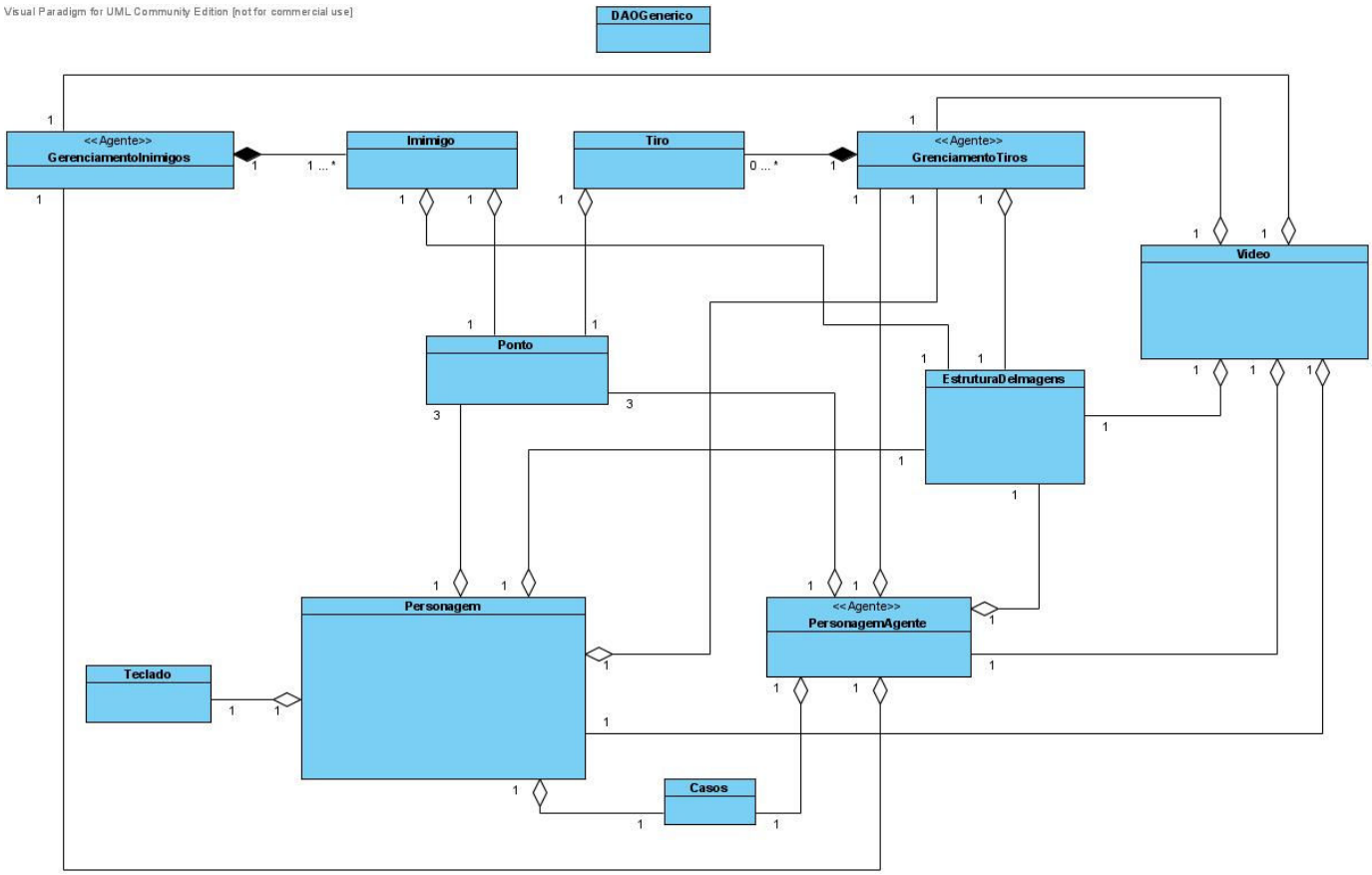

Figura 4.13: Diagrama de Classes

As classes PersonagemAgente, GerenciamenotoInimigos e GerenciamentoTiros são os agentes do sistema. As duas classes de Gerenciamento contêm uma lista encadeada que gerenciam os inimigos e tiros do Jogo. O comportamento destes agentes já foi descrito na Seção 4.6.

As classes Inimigo e Tiro são elementos que fazem parte das classes GerenciamentoInimigos e GerenciamentoTiros respectivamente. A classe Inimigo contém informações sobre a localização, imagem, valor dado ao intervalo dos tiros disparados e atributo que define se o inimigo esta vivo. A classe Tiro contém informações sobre localização, imagem, sentido e tempo de vida do tiro.

As classes Ponto e EstruturaDeImagens são classes auxiliares, que tem o papel de dar suporte as classes que contêm suas instancias. A classe Ponto realiza operações sobre dois inteiros que definem uma localização (x,y) de um elemento do jogo. A classe imagens é uma lista encadeada onde cada elemento da lista é uma imagem.

A classe Personagem define a nave em Modo Jogador, contêm a imagem que representa o jogador, define a localização da nave, o valor para deslocamento, armazena os

casos resolvidos pelo jogador durante a partida, tem acesso a eventos do teclado para interagir com o jogador e também define o tamanho do radar utilizado para definir os casos.

A classe Teclado captura eventos do teclado, definindo as ações de deslocar o personagem, atirar e sair do jogo.

A classe Casos define os casos armazenados na base do RBC, sua estrutura representa a tabela manipulada na base de dados. A classe DAOGenerido é uma classe publica que define as ações realizadas no banco de dados, como: armazenamento de casos, carregar os casos e realizar buscas na base.

A classe Vídeo representa o monitor, é responsável por manipular todas as imagens do jogo fornecidas pelas naves, tiros e inimigos do jogo.

Conclusão

Após estudar diversas técnicas computacionais relacionadas a jogos e aplicá-las durante o desenvolvimento do sistema, fica evidente que a implementação de um jogo não é uma tarefa trivial. Além das técnicas aplicadas ao jogo desenvolvido (Máquinas de Estados e RBC), também foi utilizado um processo de seleção dos dados. Através destas técnicas foi possível verificar o comportamento satisfatório do agente, principal objetivo do trabalho. Durante a realização de testes, verificou-se que existe um padrão de comportamento do agente, o qual surte um efeito melhor em sua movimentação, como, por exemplo, priorizar os casos em que o agente permanece parado. Observa-se a movimentação satisfatória do agente inteligente (modo assistido) com cerca de 10 casos armazenados na base de dados.

Todas as *threads* no sistema utilizam um tempo de *sleep*. Esse tempo é definido subtraindo-se o tempo de execução da *thread* com o tempo de *sleep* (20 milisegundos). Caso o valor seja menor ou igual a zero, deve-se utilizar um valor mínimo (5 milisegundos), proporcionando, assim, a atualização adequada das imagens do jogo.

O desenvolvimento do sistema, do ponto de vista de um jogo computacional completo, ainda se encontra em um nível simples. Seria interessante desenvolver mais o jogo, aumentando a sua complexidade, e testar como o sistema de aprendizado implementado com RBC se comportaria em um sistema mais complexo.

Apêndice A

Processo de Instalação

Como a utilização de banco de dados em nosso sistema complicou o processo de instalação, a seguir apresento os passos necessários para instalação do jogo desenvolvido e programas associados ao trabalho desenvolvido.

Programas necessários: Postgres 8.3, Maquina virtual JAVA 1.6 ou superior (recomendado) e Netbeans 6.5.

Processo de Instalação:

1. Ter acesso ao projeto desenvolvido para o jogo, com o Netbeans 6.5.
2. Instalar o banco de dados Postgres e deixar o servidor do banco de dados rodando.
3. Acessar o arquivo XML que auxilia na configuração do BD. Alterar o campo <property name="hibernate.connection.url">. Neste campo deve ser inserido a url para conexão com o BD.
4. Importar bibliotecas Hibernate, Hibernate JPA e PostgresSQL JDBC Driver, caso elas não estejam adicionadas ao projeto.
5. Definir no banco de dados especificado através da url do passo "3" uma tabela. Esta tabela pode ser criada com a SQL descrita a seguir. Obs: o campo OWER TO deve ser alterado, com um usuário do BD definido.

```
CREATE TABLE casos
(
id serial NOT NULL,
sintomaprincipal1 boolean,
sintomaprincipal2 boolean,
sintomasecundario1 boolean,
```

```
sintomasecundario2 boolean,
acao text,
CONSTRAINT casos_pkey PRIMARY KEY (id)
)
WITH (OIDS=FALSE);
   ALTER TABLE casos OWNER TO yuri;
```

6. Deve-se então carregar os dados do banco de dados para a aplicação. Utilizando o Netbens 6.5, isso pode ser feito, clicando com o botão direito do mouse no projeto, escolhendo a opção novo, depois a opção classes de entidade do banco de dados. Então ira surgir uma janela, siga os passos especificados pelo *fremework* Netbens, para carregar os dados da tabela casos definida no banco de dados Postgres.

Seguindo estes passos o processo de instalação estará completo.

Referências Bibliográficas

[1] AAMODT, A; PLAZA, E. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. **AICOM**, Amsterdam, Vol. 7, 1, 39-55, Março,1994.

[2] ÁLVARES, L. O. **Sistemas Multiagentes**. III Simpósio Nacional de Informática. Setembro, 1998.

[3] ÁLVARES, L. O; SICHIMAN, J.S. **Introdução Aos Sistemas Multiagentes**. UFRGS, Instituto de Informática e USP Escola Politécnica, Porto Alegre – RS e São Paulo – SP, 2001.

[4] ALVES, L. R. G. Estado da Arte dos games no Brasil: Trilhando caminhos. In: ACTTAS DA CONFERÊNCIA ZON | DIGITAL GAMES 2008, 2008. **Proceedings…** Braga – Portugal, 2008. p.12.

[5] BAKER, A. **JAFMAS – A Java-Based Agent Framework for Multiagent Systems**. Cincinnati: Departamento de Eletronica & Engenharia da Computação, 1997. Tese.

[6] BARBOSA, S. T; CARVALHO, C. V. A. **Tutorial para o Desenvolvimento de Jogos 2D usando a Linguagem Java**. http://www.uss.br/web/hotsites/revista_teccen3/artigo07.pdf, consultado na internet em: 17/11/2009.

[7] BATTAIOLA, A. L. Jogos por Computador – Histórico Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação. In: ANAIS DA XIX JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI), 2000. **Proceedings...**, Santa Rita do Sapucaí-MG,2000, p.83-122.

[8] BATTAIOLA, A. L; DOMINGUES, R. G; FEIJO, B; SCWAREMAN, D; CLUA, E. W. G; KOSOVITZ, L. E; DREUX, M; PESSOA, C. A; RAMALHO, G. Desenvolvimento de Jogos em Computadores e Celulares. **Revista de Informática e Teórica e Aplicada**, Outubro, 2001.

[9] BITTENCOURT, G. **Inteligência Artificial Distribuida**. www.das.ufsc.br/gia/iaft-apoio/tra-iad.ps, consultado na Internet em: 19/11/2009.

[10] BLIZZARD. **Blizzard Entretainment**. http://us.blizzard.com/en-us/, consultado na Internet em: 17/11/2009.

[11] BOURG D.M; SEEMANN, G. AI for Game Developers. In: MEDIA INC., 2004. **Proceedings…** Sebastopol – CA: [s.n].

[12] BUKLAND, M. **Programming Game AI by Example**. Wordware Publishing. 2005.

[13] BRACKEEN, D; BARKER, B; VANHELSUWÉ, L. **Developing Games in Java™**. New Riders Publishing. 2003.

[14] CHAMPANDARD, A, J. **AI for Game Developers – Synthetic Creatures with Learning and Reactive Behaviors**. Indianapolis: New Riders, 2003.

[15] CIN – UFPE – PÓS-GRADUAÇÃO. http://www.cin.ufpe.br/~posgraduacao/selecao2010/temas-mestrado-2010.htm, acessado na Internet em: 10/11/2009.

[16] CUNHA, M. M; COSTA, F. P. D; PERES, A. L; SANTOS, C. L. **Jogos Eletrônicos como Ferramenta de Auxílio no Processo de Explicação de Conteúdos no Meio Educacional**. http://www.edapeci-ufs.net/ANAIS/02/006MARCELO.pdf, consultado na Internet em: 1/11/2009.

[17] DALMAU, D. S. **Core Techniques and Algorithms in Game Programming**. Indianapolis: New Rides, 2004.

[19] FERBER, J; GASSER, L. Intelligence Artificielle Distribuée. In: TUTORIAL NOTES OF 11th CONFERENCE ON EXPERT SYSTEMS AND THEIR APPLICATIONS. **Proceedings ...** Avignon – France, 1991.

[20] FIPA. **FIPA ACL Message Structure Specification.** http://www.fipa.org/specs/fipa00061/SC00061G.pdf, consultado na Internet em: 31/8/2009.

[21] FLYNT, J. P; SALEM, O. **Software Engineering for Game Developers.** Local: Thomson Course Technology PTR, 2005.

[22] GAME MAKER. **Yoyo Games.** http://www.yoyogames.com/make, consultado na internet em: 31/08/2009.

[23] HIBERNATE.ORG – HIBERNATE. **Hibernate**. https://www.hibernate.org/, consultado na internet em: 20/11/2009.

[24] JYNIX. **Jynix Playware**. http://www.jynx.com.br/, consultado na Internet em: 17/11/2009.

[25] KARLSSON, B. F. F. **Um Middleware de Inteligênica Artificial para jogos Digitais**. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, 2005. Dissertação.

[26] KISHIMOTO, A. **Inteligência Artificial em Jogos Eletrônicos**. http://www.programadoresdejogos.com/trab_academicos/andre_kishimoto.pdf, consultado em 17/11/2009.

[27] KONAMI. **Sonic Wings Especial.** http://www.hamster.co.jp/products/soft.cgi?ps&87137, consultado na internet em: 13/11/2009.

[28] LAMOTHE, A; RATCLIFF, J; SEMINATORE, M; TYLER, D. **Tricks of the Game Programming Gurus**. 1° ed. Sams Publishing, 1994.

[29] LAMOTHE, A. **Tricks of the Windows Game Programming Gurus – Fundamentals of 2D and 3D Game Programing**. 1ª Edição. Indianapolis: Sams, 1999.

[30] LUIZ, M. **Desenvolvimento de Jogos de Computador**. http://www.programadoresdejogos.com/trab_academicos/marlo_luiz.pdf, consultado na Internet em: 17/11/2009.

[31] MANOVICH, L. **Novas Mídias Como Tecnologia e Ídéia: Dez Definições**. São Paulo, 2005.

[32] MENESES, E. X; SILVA, F. S. C. Integração de Agentes de Informação**.** In: Anais do Congresso da Sociedade Brasileira de Computação, 2001. **Proceedings...**, Fortaleza-CE: 2001,p.209-253.

[33] MITCHELL, M. **An Introduction to Genetic Algorithms**. Massachusetts: MIT Press, 1996.

[34] MITCHELL, T. M. **Machine Learning**. In: Series in Computer Science. **Proceedings…**, New York: McGraw-Hill, 1997, p.414.

[35] OSÓRIO, F; PESSIN, G; FERREIRA, S; NONNENMACHER, V. Inteligência Artificial para Jogos: Agentes especiais com permissão para matar... e raciocinar!. In: VI Brazilian Symposium on Computer Games and Digital Entertainment – SBGAMES, 2007. **Proceedings…** São Leopoldo – RS: [s.n.], 2007.

[36] PAINT. **Windows XP.**
http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/mspaint_overview.mspx?mfr=true, consultado na internet em: 31/08/2009.

[37] PERUCIA, A.S; BERTHÊM, A.C; BERTSCHINGER, G.L; MENEZES, R.R.C. **Desenvolvimento de Jogos Eletrônicos – Teoria e Prática**. 2ª Edição. São Paulo – SP, 2007.

[38] PORTAL DE JOGOS ELETRÔNICOS - INICIAL. **3ª Mostra de Jogos Eletrônicos**. http://espec.ppgia.pucpr.br/jogos/, consultado na Internet em: 10/11/2009.

[39] PORTAL PUC RIO DIGITAL. **Vídeos SBGAMES 2009**. http://puc-riodigital.com.puc-rio.br/cgi/cgilua.exe/sys/start.htm?infoid=5480&sid=135&tpl=view_integra.htm, consultado na Internet em: 17/11/2009.

[40] POSTGRESQL: The world's most advanced open source database. **PostgresSQL**. http://www.postgresql.org/, consultado na internet em: 20/11/2009.

[41] PUC-RJ: CURSOS. **Cursos de Graduação**. http://www.icad.puc-rio.br/conteudo/cursos.html, consultado na Internet em: 14/11/2009.

[42] GUDWIN, R.R. **Semiônica: Uma Proposta de Contribuição à Semiótica Computacional**. Campinas-SP: UNICAMP - Centro de Ciências Exatas e da Terra/ Ciência da Computação, 2003.Tese.

[43] GIMP. **Gimp – The GNU Image Manipulation Program.** http://www.gimp.org/, consultado na internet em: 31/08/2009.

[44] RAMALHO, G. **Projeto e Implementação de Jogos**. http://www.cin.ufpe.br/~game/, acessado na Internet em: 14/11/2009.

[45] REZENDE, S. **Sistemas Inteligentes: Fundamentos e Aplicações**. Barueri-SP: Editora Manole, 2003.

[46] RIESBECK, C. K; SCHANK, R. C. **Inside Case-Based Reasoning**. 1ª Edição. Hillsdale – NJ: Lawrence Erlbaum Associates, 1989.

[47] ROLLINGS, A; MORRIS, B. **Game Architecture and Design: A New Edition**. United States of America – New York: New Rides, 2004.

[48] RPG MMAKER. **RPG Maker Brasil.** http://www.rpgmakerbrasil.com/, consultado na Internet em: 31/8/2009.

[49] RUSSEL, S; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2° ed. Prince Hall, 2002.

[50] SBGAMES 2006. **V Brazilian Symposium on Computer Games and Digital Entertainment**. http://www.cin.ufpe.br/~sbgames/, acessado na Internet em: 13/10/2009.

[51] SBGAMES 2009. **VIII Brazilian Symposium on Computer Games and Digital Entertainment**. http://wwwusers.rdc.puc-rio.br/sbgames/09/, consultado na Internet em: 13/11/2009.

[52] SCHWAB, B. **AI Game Engine Programming**. 1ª Edição. Local: Cengage Learning, 2004.

[53] SEGA. **Sega**. http://www.sega.com/language/?lt=EnglishUK,EnglishUSA,Dutch,Belgium,German,French,Spanish,Italian,Japan,Australian&pf=/, consultado na Internet em: 17/11 2009.

[54] SILVA, L. A. M. **Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development Framework.** Fortaleza – CE: UNIFOR – Centro de Ciências Tecnológicas, 2003. Tese.

[55] TAIKODON. **Taikodon**. http://www.taikodom.com.br/downloads, acessado na Internet em: 10/05/2009.

[56] TECHFRONT. **TechFront – Play it Forward**. http://www.techfront.com.br/about/, consultado na Internet em: 17/11/2009.

[57] THE ELDER SCROLLS. **The Elder Scrolls IV: Oblivion**. http://www.elderscrolls.com/games/oblivion_overview.htm, consultado na Internet em: 1/11/2009.

[58] THE ORANGE XBOX - HALF-LIFE 2 EPISODE ONE. **Half-Life 2: Episode One**. http://orange.half-life2.com/hl2ep1.html, consultado na Internet em: 10/11/2009.

[59] TUTORIA DE JOGOS – PUCPR. **Tutoria de Jogos**. http://www.ppgia.pucpr.br/~radtke/jogos/, acessado na Internet em: 11/11/2009.

[60] WANGENHEIM, C. G. V; WANGENHEIM, A. V. **Raciocínio Baseado em Casos.** 1ª Edição. Barueri –SP: Malone Ltda, 2003.

[61] WOOLDRIDGE, M; JENNINGS, N. R. **Intelligent Agents: Theory and Practice**. Knowledge Enginnering Review,Outubro,1994.Tese.