ATORES VIRTUAIS AUTÔNOMOS PARA SISTEMAS NARRATIVOS INTERATIVOS

SAMIR ARAÚJO DE SOUZA

ATORES VIRTUAIS AUTÔNOMOS PARA SISTEMAS NARRATIVOS INTERATIVOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: LUIZ CHAIMOWICZ

Belo Horizonte

Julho de 2009

Souza, Samir Araújo de
S726a Atores virtuais autônomos para sistemas narrativos interativos / Samir Araújo de Souza. — Belo Horizonte, 2009
xxii, 102 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Luiz Chaimowicz

1. Inteligência artificial - Teses. 2. Narrativa interativa - Teses. 3. Agentes inteligentes (software). I. Orientador. II. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Atores Virtuais Autônomos para Sistemas Narrativos Interativos

SAMIR ARAUJO DE SOUZA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ESTEBAN WALTER GONZALEZ CLUA
Centro Tecnológico, Escola de Engenharia - UFF

PROF. RICARDO POLEY MARTINS FERREIRA
Departamento de Engenharia Mecânica - UFMG

PROFA. GISELE LOBO PAPPA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 27 de julho de 2009.

Dedico esta dissertação a todos os que trabalham para que um dia possamos viver a Singularidade.

Agradecimentos

A Deus, a minha família, ao meu orientador, por me mostrar um caminho menos tortuoso até aqui, e a minha noiva, que acreditou em mim e não me abandonou quando disse que casaríamos depois que eu terminasse o Mestrado (obviamente cumprirei minha promessa).

"A vida é uma comédia para aqueles que pensam e uma tragédia para aqueles que sentem."
(Horace Walpole)

Resumo

Narrativa Interativa (Interactive Storytelling) é uma área de pesquisa multidisciplinar que tem atraído um grande número de pesquisadores nos últimos anos. Dentro dessa área, podem-se destacar duas linhas de pesquisa principais: histórias emergentes e não emergentes. Até o momento, existem poucos trabalhos que exploram a perspectiva de criação de sistemas de Narrativa Interativa que conduzem histórias emergentes, usando agentes inteligentes autônomos como atores virtuais. É proposto neste trabalho um modelo de uma mente sintética, que utiliza diversos conceitos de Inteligência Artificial (IA), para controlar agentes que interpretam papéis específicos de personagens em uma dada narrativa. Serão apresentados neste trabalho o modelo arquitetural da mente sintética, os detalhes da implementação de um protótipo para validar o modelo e os resultados da realização de experimentos utilizando o protótipo.

Abstract

Interactive Storytelling (IS) is a multidisciplinary research area that has attracted a large number of researchers in recent years. There are two main research lines within the major area: emergent and non emergent stories. However, there are few works exploring the perspective of creating IS systems that conduct emergent stories using autonomous intelligent agents as virtual actors. In this paper we propose a model of a synthetic mind for virtual actors that will make them interpret a role of a given character contextualized by a complete story. We discuss the model architecture, implement a prototype, and present some proof-of-concept experiments to show its effectiveness.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A facilidade de acesso que as pessoas possuem atualmente às diversas tecnologias de entretenimento, comunicação, educação à distância e busca de informações faz com que a geração de conteúdo para dispositivos baseados nestas tecnologias seja bastante diversificada e inovadora. Transformar informação e conhecimento em produtos que sejam consumidos pelo maior número de pessoas, em uma região alvo específica ou em todo o mundo, é o objetivo de toda organização que trabalha com geração de conteúdo eletrônico digital. Pode-se entender como conteúdo eletrônico digital todo o tipo de mídia acessível através de um computador pessoal, dispositivo móvel, videogame, TV digital, entre outros dispositivos digitais, seja esta mídia publicada na Internet, distribuída pelo ar (ondas eletromagnéticas) ou até mesmo disponibilizada através de um meio físico de armazenamento de dados qualquer.

Uma tendência que vem ganhando força e influenciando produtores durante a concepção de conteúdo digital é a de que a interatividade potencializa o valor final do produto. Contudo, não há indícios claros de que a interatividade por si só, quando adicionada ao conteúdo, o torna mais interessante, atraente ou eficiente. Richards [2006] realizou uma pesquisa sobre interatividade e concluiu que em conteúdos educativos e de treinamento há realmente impactos positivos, ou seja, os usuários se interessam mais pelo produto e tendem a absorver a mensagem transmitida com mais eficiência. Porém, em conteúdos voltados para o entretenimento e comunicação de informações o mecanismo de interatividade pode gerar um desconforto aos usuários, se não dimensionado na medida correta. Nestes casos, tornar o conteúdo não interativo e transformar o usuário em um espectador passivo seria o mais indicado.

Para um melhor entendimento do contexto apresentado, faz-se necessário definir o que seria efetivamente interatividade em conteúdos digitais. Para Crawford [2005] interatividade é *"um processo cíclico no qual dois atores alternadamente escutam, pen-*

sam e falam". No caso dos conteúdos digitais interativos os atores seriam o usuário e o sistema que transmite o conteúdo. As ações de escutar pensar e falar podem ser interpretadas como perceber a ação do outro usuário, processar a entrada (sinal emitido pelo ator) e exibir uma saída, respectivamente. Já Richards sugere algo mais genérico sobre interatividade e afirma que três temas emergem ao se tratar esse assunto: *"é uma forma de comunicação; é uma área de estudo multidisciplinar; o controle dado ao usuário para realizar a interação é a chave"*. Pensando novamente em conteúdo digital interativo, mas sob a ótica da teoria apresentada em Richards [2006], é possível dizer que a interatividade é o resultado da comunicação entre um usuário e um sistema onde ambos realizam o controle do andamento da transmissão do conteúdo. O usuário usa o seu controle para extrair do sistema o que ele procura. E o sistema controla o que será transmitido ao usuário de acordo com os estímulos recebidos. Isso pode soar um pouco vago, por isso Richards afirma que é uma área de estudo multidisciplinar. É preciso realizar estudos em diversas áreas para se possa desenvolver um modelo de interatividade aplicável a sistemas de transmissão de conteúdo digital de forma eficiente e justificável.

Jogos digitais podem ser vistos como mídias digitais de entretenimento. Neste tipo de mídia, o conteúdo pode ser concebido e transmitido ao espectador no formato de uma narrativa. O autor elabora a história e define o roteiro, que é apresentado ao jogador através das ações dos personagens, cenários, objetivos e também pela interatividade. Projetar interfaces interativas inteligentes, atrativas e principalmente justificáveis para conteúdos direcionados ao entretenimento é algo bastante complicado, pois envolve diversos valores culturais, além do fator "jogabilidade", que interfere diretamente no formato de interatividade do conteúdo [Miller, 2004], mas esta questão, apesar de ter uma correlação com a concepção de sistemas de *"Interactive Storytelling"* ou Narrativa Interativa (NI), está além do escopo deste trabalho.

Na grande maioria dos jogos, a história não passa de um simples elemento para contextualizar a tecnologia utilizada no processo de desenvolvimento [Glassner, 2004]. Exceções são os jogos digitais de *Role-Playing Game* (RPG) [Tychsen, 2006] onde efetivamente uma história é narrada pelos personagens ao jogador durante sua jornada. É importante ressaltar que os roteiros construídos para os jogos digitais tradicionais, como por exemplo os RPGs, não podem ser modificados, ou seja, o jogador não pode fazer o que realmente desejar, dadas às restrições impostas a ele pelos desenvolvedores. Dessa forma, na tentativa de manter a história como fundamento para o uso da tecnologia e não o contrário, além de permitir uma maior interatividade entre o jogador e a aplicação, diversos pesquisadores vêm investigando e trabalhando com o conceito de Narrativa Interativa.

Em sistemas de NI, a narrativa é o foco da interatividade. Ou seja, o fato do usuário interagir com o ambiente gerenciado pelo sistema pode resultar na modificação da história narrada, ao contrário dos jogos digitais tradicionais, que apenas dão prosseguimento a um roteiro que não pode ser alterado.

O dicionário Michaelis define narrativa como:

> 1 Ato ou efeito de narrar. 2 Conto, descrição, discurso, narrativa. 3 Exposição verbal ou escrita de um ou mais fatos. 4 A parte do discurso em que o orador divide e desenvolve o assunto.

Aplicando essa definição ao contexto de conteúdos digitais para sistemas computacionais, juntamente com os conceitos de interatividade abordados anteriormente, pode-se dizer que a Narrativa Interativa é uma forma de se contar uma história, de maneira não necessariamente linear, onde o usuário pode assumir o papel de um personagem e, interagir com outros personagens e com o ambiente onde estão inseridos, através de suas ações.

O roteiro da história apresentado em um sistema de NI pode ser algo mais rígido, oferecendo poucas opções de interação ao usuário, ou mais flexível, permitindo que ele faça praticamente o que quiser (obviamente, dentro das limitações do universo da história). Quanto mais alto o grau de interatividade e intrusividade do usuário na história, menos rígido e detalhado o roteiro se torna. E se o roteiro é pobre em detalhes ele acaba se transformando em apenas uma linha base para a história, permitindo que o seu curso original seja alterado durante a execução da narrativa. Mas isto tem seus efeitos colaterais. Roteiro e interatividade são em geral elementos incompatíveis (paradoxo roteiro versus interatividade) [Crawford, 2005]. Quanto mais o usuário interferir no andamento da história - isso também depende de quão rígidas são as regras de interatividade do sistema - mais o roteiro se perderá e talvez a história resultante deixe de fazer sentido.

Existem duas abordagens mais comumente empregadas no processo de criação de Narrativas Interativas. A primeira utiliza um sistema que mantém a narrativa da história o mais próximo possível do roteiro original, concebido pelo autor/roteirista. E a segunda faz uso de sistemas que utilizam roteiros mais simplificados (sem muitas definições), possibilitando que histórias emergentes surjam a partir das interações entre os personagens. Os vários pesquisadores na área de NI se dividem em relação a aplicabilidade dessas abordagens. Crawford [2005], por exemplo, não acredita na segunda abordagem. Ele diz que "Computadores não sabem contar histórias" e por isso defende que os desenvolvedores de NI devem disponibilizar um número limitado de opções aos usuários para que eles não possam mudar o rumo da história. Por outro lado diver-

sos trabalhos como Aylett et al. [2005] e Silva et al. [2001], que exploram justamente o desenvolvimento de histórias emergentes criadas por agentes inteligentes, vêm demonstrando que esta vertente é bastante promissora e que não pode ser de forma alguma ignorada.

Os avanços nas pesquisas em Inteligência Artificial (IA) e o acesso facilitado a processadores poderosos vêm possibilitando a criação de sistemas inteligentes cada vez mais eficientes. O surgimento de um sistema inteligente auto-consciente é algo que acontecerá em algumas décadas, segundo o futurista Kurzweil [2006]. Entretanto, restringindo os objetivos a um contexto mais limitado, como a construção de sistemas de NI, as tecnologias e o conhecimento vigentes bastam.

O trabalho apresentado nessa dissertação segue a segunda linha de pesquisa dentro da grande área NI (geração de histórias emergentes). Utilizando-se agentes inteligentes para conduzir narrativas, este trabalho descreve um modelo de uma mente sintética criada para controlar atores virtuais de sistemas de NI, baseados em histórias emergentes [Araújo & Chaimowicz, 2009]. Foram utilizados conceitos de Inteligência Artificial Genérica e mecanismos de interpretação de conhecimento baseado em senso comum (*commonsense knowledge*, Minsky [1985]) para a construção do modelo. O modelo proposto dá aos atores virtuais autonomia e capacidade de lidar com situações não previstas pelos desenvolvedores do sistema e pelo autor da história, além de mantê-los focados nos objetivos planejados pelos desenvolvedores para cada um deles, mantendo a consistência da narrativa.

Nota-se que, para isto, foi necessário realizar uma ampla pequisa sobre agentes inteligêntes, além de investigar diversas técnicas de IA ao se compor o modelo apresentado neste trabalho. O modelo em si é uma importante contribuição não apenas para pesquisa e desenvolvimento em jogos e Narrativa Interativa, mas também poderá servir de base para estudos comportamentais de agentes inteligentes e simulação de vida artificial.

1.1 Motivação

Os primeiros experimentos que tentaram criar sistemas computacionais para a geração de histórias foram realizados na década de 1970 [Schank & Abelson, 1977; Meehan, 1977]. Porém, dois dos primeiros trabalhos que representaram o início das pesquisas em Narrativa Interativa, onde o usuário tinha o poder de assumir o papel do protagonista e podia utilizar os mesmos recursos disponíveis para os demais personagens, além de conseguir modificar de forma significativa o rumo da história, são Laurel [1993]

e Thomas et al. [1992]. A partir de então o número de pesquisadores e trabalhos desenvolvidos na área vêm aumentando consideravelmente.

Crawford, um dos mais importantes *game designers* de jogos digitais, criou diversos jogos e acompanhou a evolução e a estagnação da indústria de jogos nas décadas de 80 e 90, respectivamente. Em 1991 ele abandonou a indústria de jogos com a justificativa que os desenvolvedores, inclusive ele próprio, não conseguiam mais inovar em termos de conteúdo. A evolução tecnológica permite que melhorias em gráficos, som, etc., sejam realizadas, mas a falta de inovação e criatividade tornara os jogos muito parecidos e sem brilho. A partir de então ele passou a buscar por algo inovador, ao iniciar suas pesquisas sobre NI [Crawford, 2005]. Crawford acreditava, quando tomou essa decisão, que através da utilização efetiva de narrativas nos jogos, juntamente com um sistema interativo, poderia-se então criar algo novo e muito mais interessante. Outros pesquisadores se juntaram à causa e o tema "Narrativa Interativa" se consolidou como uma área de pesquisa multidisciplinar que vêm se fortalecendo cada vez mais em grupos de pesquisa por todo o mundo.

Diversos congressos que abordam tópicos relacionados a NI, como a *"Joint Conference on Interactive Digital Storytelling (ICIDS)"*, *"Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)"*, *"International Conference on Virtual Storytelling (ICVS)"*, entre outros, ocorrem todos os anos com um crescente número de participantes interessados no assunto. A Inteligência Artificial tem um grande destaque nestes eventos, dado o significativo número de artigos publicado anualmente sobre técnicas de IA aplicadas ao conceito de Narrativa Interativa.

Quando McCarthy propôs o termo Inteligência Artificial para definir "a ciência e a engenharia de se criar máquinas inteligentes", todos acreditavam que em pouco tempo seria possível criar programas de computador capazes de emular a mente humana (IA forte). Com o passar dos anos, essa tese caiu em descrença pela comunidade científica, pois se caracterizava como uma tarefa extremamente complexa e fôra, de certa forma, subestimada pelos pesquisadores da época. Em seguida a IA forte foi abandonada pela maioria dos descrentes pesquisadores. A IA, desde então, continuou a existir e evoluiu bastante, mas sempre focada em técnicas específicas para serem aplicadas a problemas específicos (IA fraca). Contudo, nas últimas décadas, esforços da comunidade acadêmica foram realizados com o objetivo de retomar com mais intensidade as pesquisas sobre a IA forte, que passou a ser chamada de "Artifical General Intelligence" (AGI) ou Inteligência Artificial Genérica[1] pelos cientistas responsáveis pelo movimento. Criou-se então o Artificial General Intelligence Research Institute (AGIRI)[2] em 2001 com o in-

[1]Tradução livre de Artificial General Intelligence para o português
[2]http://www.agiri.org

tuito de promover a criação de IAs Genéricas. Este movimento já resultou em diversos modelos matemáticos e protótipos de sistemas de AGI [Wang, 2006; Iklé et al., 2006]. Sendo assim, é importante mencionar que a aplicação da Inteligência Artificial Genérica ao contexto da Narrativa Interativa é um dos principais fatores que motivaram a realização desta pesquisa.

O objetivo principal de um sistema computacional de NI é "narrar" uma história. Uma história é contada por personagens, que estão inseridos em um cenário específico e podem interagir entre si e com o usuário. Sendo assim, é pertinente questionar sobre quem ou o que controlaria os personagens que não são comandados pelos usuários. Logo, entram em cena os "Agentes Inteligentes" ou *Non-Player Character* (NPC), que são entidades computacionais que possuem um corpo (virtual), são dotados de sensores para "perceber o mundo", atuadores para interagir com o ambiente e uma mente artificial para comandar todas as suas funcionalidades. A mente de um NPC é basicamente um sistema de Inteligência Artificial utilizado para receber os dados dos sensores, raciocinar (inferindo conclusões), além de decidir qual será a próxima ação a ser executada pelo NPC. A IA que controla os agentes nos sistemas de NI é normalmente configurada de duas formas:

- Centralizada: uma espécie de diretor inteligente controla as ações de todos os personagens, que por sua vez são configurados como simples marionetes na "mão" do diretor.

- Descentralizada: cada agente possui uma mente individual e é capaz de tomar suas próprias decisões sem a necessidade de consultar outro agente.

Um cenário interessante seria aquele em que um sistema híbrido, (que utilizasse IA descentralizada) onde os atores agiriam norteados por objetivos próprios e seriam dotados com o máximo de informações sobre o seu personagem. Consequentemente eles deveriam cumprir o seu papel (previsto pelo autor) na história mas sem uma obrigação formal e irredutível deste cumprimento. Além disso, este sistema permitiria que o usuário realizasse ações quaisquer, praticamente sem restrições, possibilitando que o rumo da história, de uma forma não prevista pelos desenvolvedores, fosse alterado. É fato que um sistema capaz de realizar tais feitos seria algo surpreendente, mas o número de possibilidades de ações do usuário poderia causar inconsistências na história, o que colocaria os agentes em uma estado de "letargia" por não saberem o que fazer. Por outro lado, pode-se dizer que isso depende do tipo e da capacidade da Inteligência Artificial que está por trás dos agentes. A inteligência Artificial "ideal", neste caso específico, seria a que conseguisse simular a mente humana com o maior número de detalhes possível.

Mas como Inteligências Artificiais que fazem tal coisa existem apenas em histórias de ficção, é preciso utilizar o que se tem de mais eficiente para se alcançar uma solução aproximada, o que serviria muito bem para uma aplicação de entretenimento interativo. Os conceitos e técnicas de AGI existentes se configuram como uma possível solução para este problema.

Diferentemente das técnicas de IA fraca utilizadas na criação de agentes inteligentes, técnicas de AGI podem suprimir uma série de limitações das anteriores, além de permitir a criação de agentes mais robustos e preparados para lidar com situações diversas. Por exemplo, situações de incerteza e com pouca ou nenhuma informação sobre o contexto do problema são difíceis e computacionalmente inviáveis de serem tratadas com técnicas de IA fraca. Porém, existem técnicas de AGI capazes de tratar problemas nestas condições, o que torna a AGI um mecanismo bastante adequado para ser utilizado em sistemas de NI.

Segundo Woodcock [1998], no final da década de 90 os desenvolvedores de jogos comerciais podiam utilizar no máximo 5-10% da CPU para jogos de tempo real e de 5-50% para jogos de turno na realização de tarefas de Inteligência Artificial. Porém, com a evolução dos *hardwares* dos consoles de *videogame*, as unidades de processamento ganharam mais poder e, na geração mais recente, os fabricantes optaram por utilizar CPUs com múltiplos núcleos. O objetivo desta decisão era atender de uma vez por todas as necessidades de processamento demandas por um jogo moderno. Por outro lado, os jogos computadorizados, de um modo geral, têm características que requerem a execução serializada de suas tarefas, ou seja, é muito difícil processar paralelamente entrada de dados, atualização dos elementos do jogo, física, gráficos, etc., pois se uma tarefa for executada fora da ordem, o jogo pode ficar dessincronizado, aumentando significativamente a possibilidade de ocorrerem problemas[Brakeen et al., 2003]. Mas este é o grande desafio destas arquiteturas de processadores para os desenvolvedores, conseguir otimizar os serviços, executando-os com o máximo de paralelismo e evitando que um núcleo fique ocioso enquanto outro fique sobrecarregado. Ao vencer este desafio, a utilização de processadores com múltiplos núcleos se torna uma ótima oportunidade para que pesquisadores e desenvolvedores consigam confeccionar sistemas de Inteligência Artificial mais eficientes e realistas para os jogos.

Observando todos estes fatores e possibilidades geradas pela evolução da tecnologia dos videogames, pode-se dizer que é factível a criação de sistemas mais complexos, dotados de recursos de IA mais sofisticados e mais presentes durante todo o ciclo de execução da aplicação. Além disso, a necessidade de inovação no conteúdo dos jogos digitais e a utilização de Narrativas Interativas para atacar este problema, empregando-se Inteligência Artificial Genérica na criação de NPCs que interagem com jogadores hu-

manos de forma mais "inteligente" e mais divertida, são os principais elementos que motivaram a realização deste trabalho.

1.2 Objetivos

O objetivo deste trabalho é a criação de um modelo sintético de uma mente artificial para atores virtuais de Narrativas Interativas. Empregando conceitos de Inteligência Artificial Genérica e controlando os atores de forma descentralizada, pretende-se criar agentes capazes de executar papéis de personagens dentro de uma história. A mente artificial dos agentes deverá acessar uma base de memórias que contém todas as experiências do personagem. As experiências "implantadas" previamente em sua memória, além das que o agente irá adquirir durante sua atuação, influenciarão diretamente o processo de tomada de decisões. Adotando uma abordagem híbrida, onde há objetivos iniciais pré-definidos pelos desenvolvedores e roteiristas da história, os agentes deverão buscar a todo momento alcançar seus objetivos. Possíveis interações que um agente venha a ter durante seu ciclo de vida podem alterar sua lista de objetivos, incluindo novos, excluindo existentes e mudando prioridades. Com isto, poderão ser observados comportamentos emergentes e não previstos pelos desenvolvedores. Os agentes deverão estar preparados para lidar com situações não previstas pelo autor e desenvolvedores do sistema de NI, dando prosseguimento ao fluxo da história, mesmo que esta tenha sido bastante modificada.

Para validar este modelo, um protótipo foi modelado e desenvolvido. Este protótipo foi utilizado para avaliar se o modelo da mente artificial atende aos seus requisitos básicos. O objetivo do protótipo é apenas validar os componentes do modelo e o resultado da integração de suas partes, permitindo que o agente realize tarefas de planejamento e resolução de problemas. Está fora do escopo deste trabalho a construção de uma aplicação completa de Narrativa Interativa, que possua, por exemplo, um sistema de animações, suporte a Processamento de Linguagem Natural, cenários variados, etc.

É importante mencionar que apesar do termo mente artificial ter sido usado para descrever um mecanismo de controle de agentes inteligentes, não se pretende criar IAs auto-conscientes e que hajam como seres-humanos. Muito pelo contrário, o objetivo principal é a criação de agentes sintéticos e específicos para sistemas de Narração Interativa que, no máximo, dêem a ilusão de inteligência a espectadores humanos.

1.3 Organização do trabalho

Essa dissertação está organizada da seguinte forma: no capítulo 2 será apresentado o referencial teórico estudado para a realização deste trabalho. O capítulo 3 aborda o modelo de mente sintética proposto. O capítulo 4 descreve os detalhes da implementação do protótipo, o modelo conceitual dos seus componentes, e o que foi desenvolvido para a validação do modelo de mente artificial proposto neste trabalho. Os experimentos realizados, utilizando o protótipo, são detalhados no capítulo 5. E por fim, são apresentadas as conclusões.

Capítulo 2

Referencial teórico

Ronald B. Tobias compilou em [Tobias, 2003] vinte roteiros mestres, preparados para serem usados como moldes na criação de histórias de gêneros diversificados. Um roteiro mestre é uma espécie de agrupamento de estilos literários, que são classificados por suas características mais genéricas. Por exemplo, estilos de roteiros bastante conhecidos como aventura, investigação, vingança, etc. são descritos em seu trabalho como roteiros mestres. Os elementos dos roteiros mestres servem para os autores se nortearem ao escreverem textos sobre aquele dado assunto.

Um elemento que deve estar presente em toda boa narrativa é a tensão. Em cada roteiro mestre, a tensão se caracteriza de uma forma diferente, mas de maneira geral ela pode ser vista como o resultado do choque entre forças opostas, ou seja, expectativas contra o que realmente aconteceu. A tensão pode despertar diversos tipos de sentimentos no espectador, relacionados aos personagens, como ódio, empatia, amor, compreensão, entre outros. E são justamente estes sentimentos que criam um laço entre o usuário do sistema narrativo e os personagens da narrativa [Tobias, 2003]. Logo, quando se fala na criação de um sistema narrativo de qualquer natureza, é muito importante ter em mente que a consistência narrativa - se os fatos e acontecimentos da narrativa fazem sentido em seu contexto - é um fator indispensável. Em sistemas que permitem a geração de histórias emergentes, esse ingrediente possui um valor maior, pois deverá ser levado em consideração durante toda a concepção da arquitetura do sistema. De nada adianta criar sistemas que geram histórias emergentes que não fazem o menor sentido para o usuário ou histórias entediantes, onde não há tensão entre os personagens.

2.1 Narrativa interativa

O roteiro de uma narrativa define uma sequência de eventos e a ocorrência destes eventos, observada por um espectador, se configura na história contada pela narrativa. Quem executa estes eventos dentro de uma narrativa são os personagens da história. É importante ressaltar que o ambiente também pode interferir no sequenciamento dos eventos do roteiro, ao executar ações quaisquer (como efeitos naturais; chuva, vento, terremotos, etc.) que façam com que os personagens mudem os seus objetivos vigentes. No contexto da Narrativa Interativa, esta sequência de eventos deixa de ser algo fixo e linear, ou seja, apesar de histórias serem estruturas naturalmente lineares, elas podem ser compreendidas como uma rede de idéias inter-relacionadas [Crawford, 2005]. Tendo esta concepção de história em mente, pode-se afirmar que alterações na ordem que estas idéias são transmitidas ou até mesmo no conteúdo delas, desde que resultem na mensagem contida na história original, continuam originando narrativas de uma mesma história. Diversos fatores poderiam alterar a narrativa desta maneira, a interatividade é uma delas. Supondo que o usuário assuma o papel de um dos personagens da história e passe a interagir e a interferir no andamento dos eventos, é bem provável que ocorram mudanças na sequência das idéias transmitidas por cada personagem em situações diversas durante a narrativa.

A mudança do roteiro de uma narrativa de forma não prevista pelo autor pode gerar efeitos colaterais negativos. Um *dead end*, por exemplo, é um deles. Como explicado em [Barros & Musse, 2008], de forma geral um *dead end* é ocasionado pela uma combinação de estados, após a realização de determinadas ações pelo usuário, que impede que a história continue. Este problema pode ser visto também como um afunilamento no roteiro, onde apenas uma ação satisfaz a condição que permitirá o prosseguimento da narrativa. Por exemplo, se uma determinada porta precisa ser aberta, mas o usuário jogou a única chave que a abriria em um local que não há como recuperá-la, e os desenvolvedores não prepararam um caminho alternativo no roteiro para a continuação da história e, portanto, não há mais ações previstas para os personagens controlados pela IA, então o usuário ficaria vagando pelo ambiente virtual sem saber o que fazer e a narrativa nunca terminaria. Um *dead end* é um problema típico de sistemas de NI que utilizam apenas planejadores para definir a próxima ação dos personagens e não possuem mecanismos mais elaborados que possam evitá-lo.

Analisando diversos trabalhos na literatura sobre a criação de ferramentas para a execução de Narrativas Interativas, é possível notar duas vertentes seguidas pelos pesquisadores da área. A primeira compreende os modelos de ferramentas que realizam um controle bastante apurado do roteiro, definindo de forma bem clara e objetiva,

quais são as possíveis sequências de ações que cada ator irá executar durante toda a narrativa. Normalmente são utilizados árvores de possibilidades [Levi & Sirovich, 1976] ou planejadores de ações [Russel & Norvig, 2003] neste tipo de abordagem. Já a segunda vertente compreende os modelos que não restringem as possíveis modificações no roteiro e dão mais "liberdade" aos atores para que situações emergentes, incluindo as não previstas pelos desenvolvedores, conduzam a narrativa.

2.1.1 Narrativa não linear representada por hierarquias de possibilidades

Para que o controle de cada evento do roteiro possa ser realizado com mais precisão, é necessário que um mapeamento deste roteiro seja realizado em estruturas de dados que o tornem o mais previsível possível e organizem os eventos de forma hierarquizada. Por exemplo, em trabalhos como [Cavazza et al., 2001, 2002] foram utilizados planejadores do tipo *Hierarchical Task Network* (HTN) [Russel & Norvig, 2003, apud Kutluhan Erol [1994]]. Este tipo de planejador faz uso de grafos E/OU [Levi & Sirovich, 1976] para armazenar os possíveis estados e/ou ações utilizados no controle de cada personagem. Cada nó do grafo pode ser um estado que o agente se encontra, seja este um estado interno/externo, ação já executada, etc., e os seus nós filhos são as próximas ações/estados possíveis de serem executados/alcançados a partir do estado vigente. Um exemplo de um grafo E/OU é mostrado na Figura 2.1.

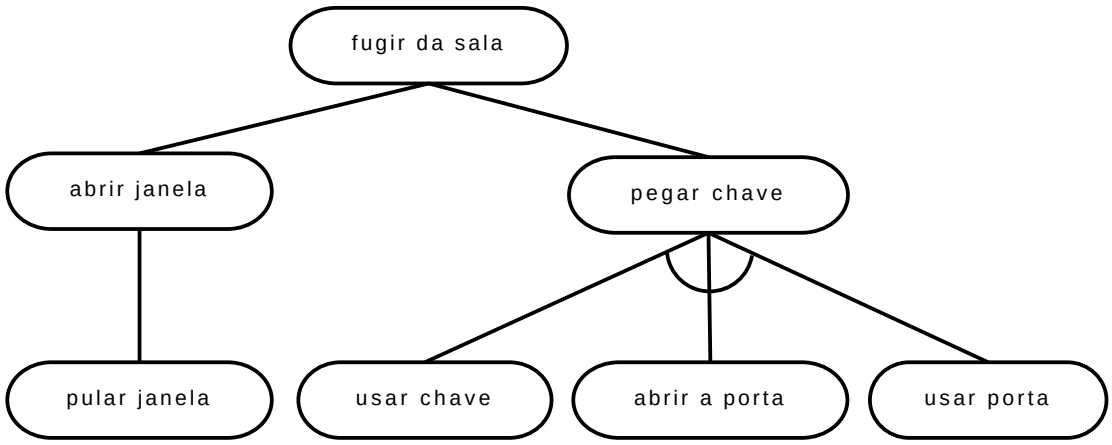

Figura 2.1. Exemplo de um grafo E/OU.

Neste exemplo, se o NPC optar por "pegar a chave" ao invés de "abrir a janela", seja por que ele já possuía uma chave em seu bolso ou não havia notado a existência de uma janela na sala, o ramo da primeira opção será descartado e o ramo da segunda passa a ser utilizado deste momento em diante, orientando-o até o fim da narrativa.

Um outro tipo de planejador, bastante utilizado para realizar a tarefa de sequenciamento de eventos, baseia-se no modelo S.T.R.I.P.S. [Russel & Norvig, 2003, apud Fikes & Nilsson [1990]]. Este planejador utiliza basicamente duas estruturas de trabalho: estados e ações. Todo o ambiente, incluindo os atributos que descrevem a situação "física e mental" vigente de cada agente, é descrito na forma de estados. Os estados podem estar ativos ou não (valores gradativos do estado de ativação também são aceitáveis como indicadores. Ex. 0.1-0.99 ao invés de apenas 0 ou 1). Uma ação possui dois atributos: pré-condições e efeitos ou pós-condições. As pré-condições são uma relação de estados que devem ser satisfeitos para que a ação possa ser executada. Já os efeitos são estados que serão modificados caso a ação seja executada. Isto posto, percebe-se que, dependendo da configuração das ações, os efeitos da execução de uma ação habilitam uma segunda ação a ser executada e assim sucessivamente (ou seja, uma ação depende da outra). Isto coloca em ordem sequencial um conjunto hierárquico de ações.

Charles et al. [2003] aprimorou um de seus trabalhos, que inicialmente fazia uso HTNs, substituindo o planejador antigo por um baseado em S.T.R.I.P.S.. Neste trabalho o autor relata que o processo de criação dos roteiros se torna bem mais fácil, dado que os eventos são tratados de forma individual e fragmentada. Neste tipo de abordagem, os eventos se tornam ações com pré-requisitos e efeitos. Os pré-requisitos são estados (internos do agente e do mundo) que devem estar ativos para que a ação possa ser executada (como se fossem *flags* de habilitação). E os efeitos são os estados que serão alterados se esta ação for executada. A estrutura continua hierárquica, assim como quando o sistema utilizava HTN, porém há um reaproveitamento maior das ações especificadas, o que facilitou o processo de criação e manutenção do roteiro.

Um problema, já mencionado anteriormente, que pode ocorrer em sistemas de NI que utilizam roteiros fixos em estruturas hierárquicas e que dão mais liberdade do que o permitido ao usuário, é o *dead end*. Para evitar ou pelo menos contornar este problema, Riedl & Young propuseram em seus trabalhos [Riedl & Young, 2003; Riedl et al., 2003; Young, 2001] duas abordagens:

- Ajuste do plano de ações sob demanda: supõe-se que um NPC possui uma lista de três ações a serem realizadas em sequência ordenada. Então o usuário altera o ambiente da cena realizando, por exemplo, a segunda ação da lista do ator. Logo, a execução deste evento não é mais necessária e portanto o planejador refaz a lista de ações do ator, que passa a conter apenas a primeira e a última ação, não alterando o fluxo da narrativa.

- Impedir a execução de ações "indevidas": supõe-se que em um determinado ponto do roteiro da narrativa é indispensável a presença de um dado recurso, seja ele um personagem ou objeto, para que haja prosseguimento na história. Se o usuário tentasse eliminar este recurso com uma determinada ação, então o sistema geraria um evento imediatamente que impediria a conclusão desta ação. Por exemplo, em uma cena onde há um baú que contém a próxima pista do mistério a ser resolvido pelo protagonista, se o usuário tentar eliminar a única chave que abre esse baú, o sistema faria algo como "a chave ficaria presa na mão do usuário" ou simplesmente ignoraria tal ação, fazendo com que a chave e o baú continuem na cena e o narrativa possa ter continuidade.

Apesar destas abordagens manterem o roteiro conforme planejado pelo autor, a interferência no livre arbítrio do usuário pode frustrá-lo, fazendo com que ele perca o interesse pelo sistema ao descobrir que não pode seguir seu próprio caminho dentro da narrativa, ou seja, não há real liberdade para se interagir da forma desejada.

O LOGTELL [Ciarlini et al., 2005] é uma ferramenta baseada em modelagem e simulação de histórias. O usuário cria a narrativa, através de um modelo que representa as estruturas básicas da história, usando lógica temporal. Este modelo descreve os eventos básicos realizados pelos personagens, além de possuir regras de inferência para a resolução de objetivos. Depois de completo, este modelo é executado por um módulo do LOGTELL que faz então sucessivos ciclos de processamento para gerar o roteiro da narrativa, que é caracterizado pela sequência de ações criadas para cada um dos personagens. O processamento realizado nesta etapa leva em consideração as regras de inferência presentes no modelo, mecanismos de planejamento e reconhecimento de planos, além de permitir que o usuário interfira manualmente na definição dos eventos da narrativa. Após a criação do roteiro, o usuário tem a opção de visualizar a encenação da narrativa em um simulador gráfico tridimensional, mas sem a possibilidade de interagir com o que está sendo exibido.

A primeira implementação de um sistema de NI com um maior grau de interatividade durante a narrativa foi o Façade [Mateas & Stern, 2003, 2005]. Através de uma implementação simplificada de um processador de linguagem natural [Mateas & Stern, 2004b], o usuário, no papel de um personagem da história, se comunica com outros personagens, redigindo frases quaisquer no console da aplicação. A história narrada em Façade dura aproximadamente vinte minutos e é dividida em momentos. Em cada momento, uma ou mais cenas curtas, chamadas *beats*, são selecionadas e executadas (levando em consideração as ações do usuário) pelo módulo *Drama Manager*. Por exemplo, se o usuário ofender os personagens com frases rudes, *beats* específicos para tentar

contornar o "clima ruim" ou até mesmo para retirar o usuário da cena são selecionados.

A narrativa em Façade é modificada de acordo com a interação do usuário à cada vez que a aplicação é executada. A configuração dos *beats* em uma partida depende das ações do usuário e o número de sequências possíveis é limitado ao número de *beats* disponíveis em cada momento da narrativa. Logo, os personagens não fazem nada que não esteja previamente preparado. Todos os comportamentos dos NPCs são definidos em *scripts*, programados em uma linguagem específica para criação de sequencias de ações, chamada *A Behavioral Language* (ABL) [Mateas & Stern, 2004a], o que facilita a edição dos *beats* e a geração de novas situações dentro da mesma história.

Magerko utilizou em alguns de seus trabalhos [Magerko & Laird, 2002; Magerko et al., 2004; Magerko, 2006] um módulo que cria modelos probabilísticos do comportamento do usuário, coletando e processando suas ações. Um agente que atua como um Diretor, controlando os NPCs e outros eventos do ambiente, utiliza estes modelos para tentar prever futuras ações do usuário. Como na abordagem de Magerko et al. o roteiro da narrativa é bastante rígido, eventos não previstos podem atrapalhar seu correto andamento. Portanto, a "previsão" das futuras ações do usuário permite ao Diretor replanejar as ações dos NPCs de forma a sempre induzir o usuário a executar ações que mantenham o roteiro em curso.

Uma técnica bastante utilizada, que garante um nível variável de interatividade ao usuário e realiza bem o controle das ações que devem ser executadas para que o roteiro projetado pelo autor seja mantido, é a disponibilização de opções de ações finitas ao usuário [Szilas, 1999, 2008]. Esta técnica também foi utilizada por [Crawford, 2005] em sua ferramenta de autoria de Narrativas Interativas *Erasmatron* [Crawford, 1999, 2005]. Ambos os pesquisadores adotaram em suas ferramentas o conceito de não dar liberdade total ao usuário, permitindo que estes apenas escolham uma das opções que lhes for oferecidas em momentos específicos da narrativa. Crawford, por exemplo, utilizou em sua ferramenta um conceito chamado *Verb Thinking*, que basicamente é pensar nas coisas em termos do que elas fazem e não do que elas são. Este conceito é naturalmente utilizado pelo autor da narrativa ao utilizar o Erasmatron. Durante a edição da narrativa, os verbos são transformados nas possíveis ações dos personagens. No caso do personagem controlado pelo usuário, tais ações formam o seu menu de opções disponível a cada momento de interatividade com a história.

A narrativa no Erasmatron é dividida em sequências. Uma sequência define quais acontecimentos ocorrerão em uma cena. A interação do usuário ocorre a cada passo de uma sequência, onde a lista de opções é apresentada ao usuário - o autor sugere no máximo 7 opções - para que uma seja escolhida e executada. Cada ação é vinculada a um papel. Os papéis servem para classificar os personagens quanto

às suas personalidades e outras características. Após o usuário executar uma das opções, o papel vinculado àquela ação é ativado e todos os personagens presentes na cena que estão conectados a este papel deverão executar alguma ação em resposta. Mas, antes de executar uma ação, os personagens controlados por NPCs utilizam um método de avaliação da ação que simula o que ocorreria se a ação fosse executada em um plano imaginário. Este plano é uma cópia simplificada dos objetos e personagens envolvidos na ação vigente. Quando a ação é executada neste plano, a resposta dos outros personagens é avaliada com o objetivo de evitar ou provocar complicações e conflitos, dependendo dos atributos de personalidade do NPC e do nível de tensão desejado e projetado pelo autor da narrativa.

2.1.2 Histórias emergentes

> Histórias são conjuntos de fatos que são transmitidas de uma maneira especial, marcante e mais fácil de serem entendidas do que outras formas de comunicação. (Crawford)

Diferentemente das narrativas organizadas em estruturas hierarquizadas e com pouca ou nenhuma liberdade de alteração no roteiro, histórias emergentes são criadas justamente pela geração de eventos sob demanda. Estes eventos podem ser realizados por NPCs, pelo ambiente ou pelo próprio usuário ao interagir com o sistema. Geração de eventos sob demanda significa que os eventos ocorridos durante a narrativa não foram previamente definidos pelo autor e sim planejados à medida que os personagens da história vão interagindo entre si. O sequenciamento destes eventos faz emergir uma das possíveis versões da história a ser contada ao espectador. O fato curioso é que, dependendo da estrutura do sistema que desenvolve a narrativa, variações bastante distintas da mesma história podem surgir de uma mesma base narrativa devido à forma com que o usuário interfere, através de suas interações, nas ações e objetivos do personagens.

Apesar deste tipo de abordagem dar bastante "liberdade" aos personagens da história, o paradoxo roteiro versus interatividade deve ser levado em consideração e, portanto, uma base narrativa consistente precisa ser definida para que as possíveis versões emergentes da história façam sentido e garantam o entretenimento do usuário (apesar destas duas características serem bastante subjetivas e dependerem diretamente do autor). Uma base narrativa consistente pode ser alcançada utilizando-se, além de regras detalhadas aplicadas ao mundo da narrativa (Seção 2.1.3), um agente inteligente (Seção 2.3) para realizar o papel de um "diretor virtual". Este agente teria acesso a todas as informações sobre os eventos realizados no ambiente virtual e poderia interferir,

através de ações indiretas, no andamento da narrativa. Ações indiretas podem ser entendidas como modificações dos estados dos elementos que constituem o ambiente e não controlando diretamente outros agentes. Por exemplo, criar um incêndio na floresta ou fazer aparecer objetos próximos aos personagens. Todas as ações do agente diretor devem servir para tentar manter o curso da narrativa conforme planejado pelo seu autor. Portanto, a função básica do diretor é dar forma à narrativa mas sem definir especificamente quais eventos devem ser realizados pelos personagens.

Pode parecer a princípio que a utilização de um agente diretor suprimiria o fator emergente da narrativa, tornando-a previsível e semelhante às narrativas cujos eventos são organizados hierarquicamente. Contudo, como o diretor não controla diretamente as ações dos personagens e apenas dá uma sugestão de direcionamento da narrativa ao interferir no ambiente, o usuário e os próprios NPCs podem tomar decisões que levem a história a rumos diversificados. Obviamente, se a ação do diretor for planejada com base em uma previsão das futuras ações dos personagens, as chances da narrativa emergir conforme o seu planejamento aumentam. Esta previsão pode ser feita de diversas formas, como por exemplo criar modelos de comportamento dos personagens e tentar inferir a partir destes as futuras ações ou até mesmo consultar diretamente o planejador de ações dos agentes personagens.

Ruth et al. [2007] sugere o conceito de um agente chamado *Story Manager* para desempenhar o papel do diretor da narrativa. Inspirado no *Game Master* ou simplesmente mestre de (RPG) [Tychsen, 2006], o autor descreve em Ruth et al. [2007] as características e as atribuições do *Story Manager* (SM). Para facilitar o trabalho do diretor, a história é dividida em partes identificadas por *way-points* [Weyhrauch, 1997]. Os *way-points* servem para separar os momentos da narrativa e são caracterizados pelo conteúdo emocional e pelo contexto do trecho da narrativa (organizados de forma a envolver emocionalmente o usuário com os personagens e com a história de um modo geral). O principal papel do *Story Manager* é conduzir os personagens de um *way-point* a outro. Basicamente, ele inicializa os objetivos de cada personagem e configura os NPCs, modelando-os de acordo com o contexto do momento da narrativa. O *Story Manager* pode realizar alterações nos objetivos dos agentes durante o andamento de um momento da narrativa conforme julgar necessário. As tarefas de um SM são:

1. Monitorar o status das ações: detecta se uma dada ação foi bem sucedida ou não e utiliza esta informação adicional na reorganização dos objetivos dos agentes.

2. Reorganizar os objetivos: criar novos ou abandonar os que não podem mais ser alcançados pelos personagens.

3. Planejar o próximo ciclo: cada etapa de um momento pode ser vista como um ciclo de objetivos a serem alcançados. Como estes objetivos são criados sob demanda, planos parciais (contendo objetivos curtos) são estendidos até que o plano vigente ativo, utilizado para conduzir os personagens ao *way-point* desejado, seja concluído.

Uma outra abordagem de agente diretor também apresentada em Ruth et al. [2007] é o *"Double Appraisal Story Facilitator"*. O *Story Facilitator* (SF) é um agente que atua em um sistema multiagente chamado *FearNot!* [Aylett et al., 2005] e tem como responsabilidade o gerenciamento da história através do sequenciamento de episódios. Um episódio é uma pequena cena interpretada pelos atores virtuais e observada diretamente pelo SF. Cada ação executada pelos personagens é classificada com um fator de impacto emocional. O impacto emocional é medido pela intensidade das alterações emocionais nos agentes, geradas pela execução de uma ação. Durante um episódio, os agentes planejam suas ações com base não apenas em seus objetivos mas também levando em consideração o impacto emocional da ação em si mesmo e nos outros personagens. Ou seja, a escolha de uma ação depende basicamente da soma de objetivos, motivação e emoções. Antes de uma ação ser executada, o agente avalia, através do cruzamento de informações da ação e dos modelos mentais (personalidade e estados mentais vigentes) dos agentes, qual seria o impacto emocional em cada um dos agentes presentes na cena, para então decidir qual é a melhor ação a ser executada.

Tabela 2.1. Atributos de um episódio de uma narrativa em FearNot!.

Atributo	*Descrição*
Nome	Nome do episódio
Local	Localização no ambiente virtual onde ocorrerá o episódio
Personagens	Personagens que participarão do episódio
Pré-condições	Conjunto de condições necessárias para que o episódio possa ser executado
Objetivos	Objetivos dos personagens que serão comunicados aos agentes
Eventos	Condição que quando satisfeita executa uma série de ações
Condições de término	Conjunto de condições, semelhantes ao do item Pré-condições, que quando satisfeito indica que o episódio chegou ao fim
Introdução	Conjunto de ações que faz a introdução do episódio

Em FearNot! o usuário não pode assumir um personagem nem interferir diretamente no rumo da narrativa, apenas pode dar sugestões a um agente. Dessa forma,

dependendo das sugestões que o usuário dá e das decisões tomadas pelo agente que recebe tais sugestões, o episódio pode ter desfechos distintos. Durante o andamento de um episódio, o SF faz o monitoramento de todas as mensagens trocadas entre os agentes, além de possuir a habilidade de saber tudo o que acontece no ambiente virtual. Além disso, cada episódio possui uma relação de atributos, apresentada na Tabela 2.1, utilizada para classificá-los e permitir que o SF faça um planejamento mais adequado ao andamento da narrativa. Dependendo do resultado da análise das ações executadas pelos agentes e dos atributos de cada um dos episódios disponíveis, o SF escolhe uma sequência de episódios que evita quebras no fluxo da narrativa e garante continuidade nos temas tratados de um episódio para outro.

2.1.3 O ambiente "físico" da narrativa

Interatividade requer um mundo e regras (Crawford)

Todo processo interativo está sujeito ao ambiente e suas regras. O mundo em que a história se passa deve ser algo totalmente controlado pelo desenvolvedor da narrativa. Uma técnica que funciona bem para a criação de tais universos é a concepção de uma *Sandbox* [Wright et al., 2006]. A *Sandbox* é um ambiente flexível e expressivo que suporta ações ad-hoc por parte dos agentes, além de permitir que eventos controlados ocorram em seu interior, graças às regras intrínsecas a este modelo. Com uma *Sandbox*, os desenvolvedores conseguem recriar digitalmente todo o cenário da narrativa, aplicando-lhe restrições específicas para evitar que problemas em tempo de execução ocorram, ou restrições tais que irão conduzir os atores a ambientes diversos dentro do universo. Por exemplo, se o ambiente ficcional possui uma física mais realista utilizada para regular a interação entre os corpos do ambiente, agentes, dotados de informações sobre ações e reações físicas, poderão inferir que ao chegarem à beira de um precipício eles deverão tentar achar outro caminho ao invés de continuar sua trajetória rumo à morte certa.

Os projetistas de Narrativas Interativas também podem utilizar as regras do ambiente para conduzir a história, mantendo o seu curso o mais próximo do roteiro original. Como visto na Seção 2.1, existem abordagens intrusivas que impedem que determinadas ações sejam executadas, afim de manter a qualquer custo o roteiro da história. Para tentar evitar medidas drásticas como esta, é sugerido em Crawford [2005] que não devem ser dadas ao usuário opções geradas em tempo real, tudo deve ser pensado previamente, evitando riscos de mudanças inesperadas no roteiro da história. Entretanto, ao se projetar adequadamente o ambiente da narrativa, com regras genéricas e cenários favoráveis ao contexto e ao clima da história idealizada pelo autor, seria mais

interessante se o usuário pudesse realizar as ações que ele desejar, logicamente dentro dos limites do ambiente. Isto por dois motivos:

1. Em narrativas extensas, a utilização de regras genéricas ao invés de várias regras específicas para tentar impedir cada possível "ação inválida" do usuário, facilitaria o processo de desenvolvimento e permitiria que universos bastante extensos e complexos fossem criados.

2. Com a definição de regras genéricas, porém relevantes, o usuário seria naturalmente punido pelos seus atos, caso estes sejam passíveis de pena no contexto da história.

A criação de um ambiente com regras genéricas e consistentes é um fator bastante significativo para a imersão do usuário na história, uma vez que ele, tendo consciência que todos os personagens da narrativa estão sujeitos às mesmas regras, poderia as utilizar para conduzir o roteiro por caminhos que lhe sejam mais divertidos e gratificantes.

2.2 Senso comum - conceitos, representação e raciocínio

> Senso comum é a coleção de preconceitos que adquirimos aos 18 anos (Albert Einstein)

Pode-se entender como senso comum todo o conhecimento de uso geral que as pessoas adquirem durante suas vidas. Conhecimento de uso geral é aquele compartilhado pela maioria das pessoas de uma dada comunidade, utilizado como base para raciocínios sobre fatos e acontecimentos do dia-a-dia. A formação deste conhecimento normalmente depende do ambiente onde ocorre a formação do indivíduo, destacando-se os costumes e a cultura exercitada no meio onde este indivíduo está inserido [Minsky, 2006]. O conhecimento de senso comum depende do contexto para ser utilizado e, portanto, mais de um tipo de conhecimento pode ser aprendido sobre um mesmo problema. Por exemplo, uma "campainha tocando" pode significar diversos acontecimentos como:

- Alguém ligando: se emitido do aparelho telefônico.
- Visita: se emitido da campainha da residência.
- Crime: se emitido de um alarme de um carro estacionado.

Este tipo de conhecimento pode ser adquirido através de experiências pessoais, aprendizado com os pais, pessoas conhecidas ou não. Normalmente os conhecimentos "experimentados" são os que possuem maior relevância por terem um reforço prático armazenado na memória do indivíduo.

O conhecimento de senso comum pode ser utilizado para se construir agentes inteligentes, com o objetivo de torná-los mais capazes de lidar com situações não previstas. Ou seja, teoricamente, é possível criar uma aplicação de AGI que utilize senso comum para resolver problemas de praticamente qualquer natureza e em contextos variados. Cientistas como Minsky e Lenat, acreditando nisso, cunharam diversas teorias e, juntamente com outros cientistas, criaram diversos protótipos de aplicações que utilizam esta idéia [Lieberman et al., 2004].

Em [Quillian, 1991, 1968] é apresentado um modelo para estruturar fatos e elementos que constituem a memória de um dado indivíduo. Informações sobre diversos fatos e o significado de "coisas" são organizados em uma rede complexa que interconecta e mantém próximos, elementos que possuem algum tipo de ligação. Quillian chamou esta estrutura de "Memória Semântica". Posteriormente esta estrutura foi rebatizada como "Rede Semântica". Através de um elemento, parte constituínte de uma Rede Semântica, outros elementos e funcionalidades, que possuem alguma proximidade semântica com este primeiro, podem ser acessados pelas conexões da rede. Quillian realizou experimentos, utilizarando este modelo como um dicionário, consultado por usuários através de uma interface de *software* que possibilitava a realização de perguntas em linguagem natural e cujas respostas baseavam-se no conhecimento armazenado na Rede Semântica.

Lenat iniciou em 1984 um projeto chamado CYC[1] com o objetivo de criar uma enorme base de conhecimento de senso comum a ser utilizada em sistemas computacionais [Lenat, 1995]. Segundo ele, "Uma vez se tenha realmente uma enorme quantidade de informação integrada como conhecimento, então um sistema homem-software será super-humano, da mesma forma que a humanidade com a escrita é super-humana se comparada a humanidade antes da escrita". Além da base de conhecimento o CYC oferece um sistema de inferência, utilizando lógica de senso comum. O CYC utiliza uma Rede Semântica como estrutura de dados para armazenar a base de conhecimento. As Redes Semânticas se tornaram um padrão para a construção de bases de conhecimento de senso comum, devido à facilidade de representar o problema com tal estrutura de dados. A WordNed [Miller, 1998] é um outro bom exemplo desta combinação, criada em 1985 por Miller com o intuito de mapear a língua inglesa em um dicionário de

[1] http://www.cyc.com

palavras rotuladas semanticamente. Através destes rótulos, palavras de categorias semânticas próximas podem relacionar-se entre si. Estes relacionamentos podem então serem utilizados para a construção de vários sistemas de informação [Fellbaum, 1998], o que tornou a WordNet um importante repositório de conhecimento de senso comum.

Ainda não existem aplicações interativas funcionais capazes de utilizar de forma eficiente todo o conhecimento de senso comum disponibilizado a ela. Afinal de contas, não basta apenas catalogar e organizar o máximo de conhecimento se não houver um sistema de inferência capaz de extrair conclusões úteis desta massa de dados. Segundo Lieberman et al. [2004], não há tecnologia para a criação de uma aplicação de perguntas e respostas eficiente, onde o usuário faz questionamentos ao sistema e recebe respostas razoáveis. Este tipo de aplicação ainda é custosa computacionalmente, além de na maioria dos casos frustrar o usuário com respostas incorretas e que demoram muito tempo para serem processadas [Lieberman et al., 2004].

Diferentemente da lógica clássica, a lógica de raciocínio baseada em senso comum, utilizando-se Redes Semânticas e Frames (descrito mais adiante) por exemplo, é não monotônica. Na lógica monotônica, um subconjunto qualquer dos axiomas de uma dada base de conhecimento é sempre um subconjunto desta mesma base, mesmo após a adição novos axiomas. Contudo, se um novo axioma contradizer um já existente na base de conhecimento, esta torna-se inconsistente. A partir de uma base de conhecimento inconsistente, pode-se inferir qualquer coisa, o que a torna não é confiável. Já na lógica não monotônica, a falta de informações ou a presença parcial das informações sobre um dado contexto não impede que inferências sejam realizadas. Além disso, adicionar novos fatos, que contradizem outros já existentes na base de conhecimento, não invalida o conjunto de todos os fatos.

Utilizar lógicas monotônicas para se criar IAs genéricas é inviável, pois estas exigem enormes quantidades de axiomas para descrever um universo relativamente simples. IAs genéricas devem ser preparadas para lidar, o tempo todo, com pouca informação e muita incerteza. Por isso lógicas não monotônicas são as principais ferramentas para se criar AGIs [Iklé et al., 2006].

Sun [1991] descreve em seu trabalho um modelo para a representação de regras de inferência para lógica de senso comum, utilizando um sistema baseado em regras [Bourg & Seeman, 2004] codificado em Redes Conexionistas [Minsky, 1985], chamado pelo autor de Rede Neural para Raciocínio Baseado em Regras (RNRBR). Este modelo, é constituído de duas Redes conexionistas CL e CD. Na rede CL os nós representam o nível conceitual do conhecimento, ou seja, fatos, crenças, objetos, etc.. Já a rede CD compreende os elementos de nível sub-conceitual, ou seja, propriedades e funcionalidades dos conceitos, primitivas de percepção, objetivos internos ou estados de elementos

descritos pelos conceitos. De maneira geral, os nós da rede CD qualificam conceitos da rede CL. Por exemplo, a sentença "o gato é branco" poderia ser representada com um nó conceito "gato", presente na rede CL, e um nó propriedade "branco", presente na rede CD, além de uma aresta conectando os dois nós.

Os relacionamentos entre os nós de ambas as redes são formalizados através de conexões. Para cada uma destas conexões são atribuídos pesos. Estes pesos indicam quão forte é o relacionamento entre os nós. As regras de inferência e/ou axiomas são representados na rede CL através de conexões entre os nós. Por se tratar de uma rede conexionista, toda regra é convertida em valores numéricos que por fim são atribuídos (na forma de pesos) às arestas que conectam os nós envolvidos. Os sub-conceitos também podem-se relacionar entre si na rede CD. Este relacionamento também é qualificado por pesos e serve basicamente para descrever a proximidade semântica dos nós e para regular a ativação dos elementos da rede.

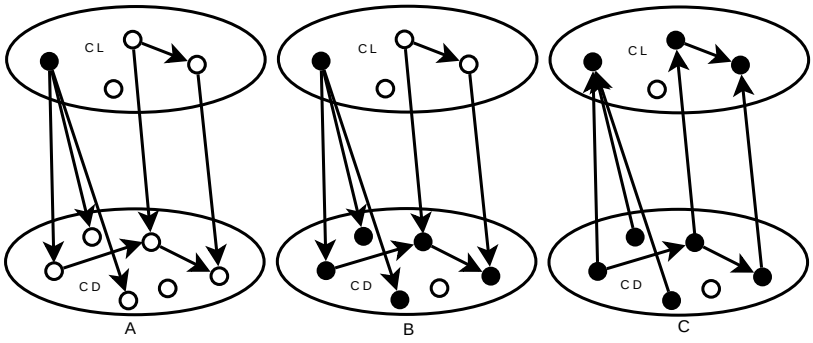

Figura 2.2. Processo de raciocínio de uma RNRBR

O processo de raciocínio em uma RNRBR depende basicamente da ativação direta ou indireta dos nós. A ativação direta é aquela que ocorre devido a um estímulo externo, seja ele manual (usuário) ou realizado por um módulo externo da aplicação que utiliza a rede. Já a ativação indireta ocorre quando um nó, utilizando suas conexões, ativa um segundo nó. A formalização do processo é descrita abaixo e, para facilitar o entendimento, suas etapas podem ser visualizadas na Figura 2.2. Onde A) Entrada de dados, B) Ativação dos sub-conceitos, C) Conclusão do raciocínio. As conexões entre os nós das redes CL e CD são bidirecionais e, portanto, as setas destas conexões indicam apenas o fluxo do processo de ativação dos nós.

1. Um nó é ativado na rede CL com um dada intensidade (peso).

    a) Todas as conexões deste nó são avaliadas e se seus pesos forem superiores a um limiar computado pela equação 2.1, os nós correspondentes às outras

extremidades das conexões são então ativados. Este processo se repetirá para cada uma destas novas conexões. Este limiar será então comparado ao peso dos nós em tempo de execução para saber se a outra extremidade da conexão será ou não ativada.

$$ATV_b = a * (A \xrightarrow{r} B) \tag{2.1}$$

onde ATV_b é valor de ativação do nó B quando o nó A estiver ativo; a é uma constante de ajuste do peso; $(A \xrightarrow{r} B)$ é o valor da regra condicional existente entre dois nós, lida como se A então B vale r.

2. As conexões que relacionam o nó que contém o maior peso de ativação na rede CL com os nós correspondentes na rede CD são então ativadas.

   a) Todas as conexões dos nós ativos na rede CD tem seus pesos avaliados e, se ultrapassarem um dado limiar de similaridade apresentado na equação 2.2, ativarão os nós adjacentes às conexões.

$$(A \approx B) = \frac{|F_A \cap F_B|}{|F_A|} \in [-1, 1] \tag{2.2}$$

onde A e B são os nós conectados avaliados; F_A e F_B são os pesos de ativação dos nós A e B respectivamente.

3. Por fim, os nós ativados na segunda etapa, ativam os nós correspondentes na rede CL através de suas conexões, finalizando o processo de inferência.

Minsky desenvolveu todo um arcabouço teórico sobre um possível modelo de funcionamento da mente humana. Em Minsky [1985], o autor apresenta o conceito de uma sociedade formada por simples agentes[2](o conceito de agentes será apresentado na seção 2.3), ou recursos, que ao trabalharem em conjunto são capazes de resolver problemas complexos. Diferentemente de uma RNRBR, a sociedade da mente se baseia em Redes Semânticas e, apesar do trabalho de Sun possuir várias semelhanças com o de Minsky na forma como ocorre o raciocínio, a sociedade da mente vai muito além de apenas raciocínio. Basicamente, a sociedade é formada por recursos que podem desempenhar desde papéis muito simples como pegar e soltar objetos até outros mais complexos

[2]O termo "agente", no contexto da sociedade de agentes, se refere a um recurso e não a um agente (corpo+mente) propriamente dito. Devido a esta ambiguidade na interpretação deste termo, Minsky optou por chamar tais estruturas de recursos, ao invés de agentes em seus trabalhos posteriores

como coordenar outros recursos, reconhecer padrões, memorizar fatos, etc. Ou seja, há uma hierarquia de recursos que são organizados para otimizar o processo de comunicação entre si. Recursos mais específicos não trocam informações com outros recursos, apenas recebem ordens de superiores imediatos, enquanto que os recursos localizados em níveis superiores na hierarquia devem se preocupar com o contexto da sua tarefa e com o sequenciamento das atividades a serem realizadas por seus subordinados.

Tabela 2.2. Camadas do modelo mental.

Camada	*Descrição*
6	Emoções auto-conscientes
5	Raciocínio auto-reflexivo
4	Raciocínio reflexivo
3	Raciocínio deliberativo
2	Reações aprendidas
1	Reações instintivas

O modelo mental de Minsky é dividido em camadas [Minsky, 2006]. Em cada camada, hierarquias de recursos desempenham papéis bastantes distintos. As camadas superiores servem para regular o processo cognitivo ocorrido nas camadas inferiores. Por exemplo, se uma camada **A** determina que o corpo[3] deverá realizar uma determinada tarefa, sendo que esta poderá causar danos à parte física do indivíduo e já foi experimentada anteriormente, a camada imediatamente superior **B** envia um sinal à camada **A** para impedi-la de continuar com esta ação. Pode ser visto na tabela 2.2 a disposição das camadas no modelo mental.

- Reações instintivas: a mente já "nasce" com funcionalidades instintivas básicas.
- Reações aprendidas: durante a existência do agente ele interage com o ambiente e aprende novas reações.
- Raciocínio deliberativo: problemas complexos e desconhecidos pelo agente que não podem ser resolvidos por reações simples, exigem estratégias diferentes. Logo, ele passa a raciocinar e planejar quais são as ações necessárias para se alcançar seus objetivos.
- Raciocínio reflexivo: depois de uma ação executada pelo agente, ele pode refletir sobre o porque de ter escolhido tais ações e não outras. Isso permite ao agente que ele consiga otimizar suas soluções.

[3]Supondo que o modelo utilizado no exemplo possui, além da mente, um corpo para interagir com o ambiente

- Raciocínio auto-reflexivo: o agente se projeta no contexto de uma ação para tentar identificar o seu valor. Por exemplo, reconhecer que o agente realizou uma boa ação é fruto da auto-reflexão.

- Emoções auto-conscientes: o agente possui valores, ideais e crenças. Com base nisto, ele pode refletir se suas ações ou de outros agentes condizem com seus ideais de conduta.

Nota-se que quanto mais alta a camada está localizada menos ela interfere nas ações diretas do corpo e mais generalizadas são as atividades nela realizadas.

2.3 Agentes inteligentes

> Agente é qualquer coisa, força ou substância que produz ou é capaz de produzir determinado efeito ou resultado; causa ativa; Algo que age, que exerce alguma ação; que produz algum efeito. (Dicionário Michaelis)
>
> Um agente é uma entidade que consegue perceber o ambiente que está inserido através de sensores e agir através de atuadores ([Russel & Norvig, 2003]).

A ação produzida pelos atuadores do agente deve levar em consideração os dados coletados pelos sensores em um dado momento. No caso dos agentes computacionais, uma função, implementada em um programa, faz esse mapeamento entre sensoriamento e atuadores. A Figura 2.3 apresenta uma simples idéia de como seria um agente interagindo com um ambiente. Dados coletados pelos sensores **S** são passados à uma função **F** que aciona os atuadores correspondentes **A** necessários para a execução da ação determinada por **F**.

O corpo do agente, incluindo sensores e atuadores, determina a maneira com que ele poderá interagir com o ambiente e/ou modificá-lo. A utilização dos atuadores é determinada pelo tipo de ação que o agente executa. Sequências de ações executadas pelo agente podem ser vistas como comportamentos. Quando o agente se depara com um problema, ele precisa se comportar de forma inteligente para conseguir solucioná-lo. Existem diversas definições para o termo inteligência, alguns foram apresentados no Apêndice A, juntamente com uma análise sobre o assunto, baseado na perspectiva da concepção de sistemas computacionais inteligentes. Mas, de maneira geral, para que o agente demonstre comportamentos inteligentes, é necessário que ele processe as informações que possui sobre o contexto do problema a ser tratado e decida quais

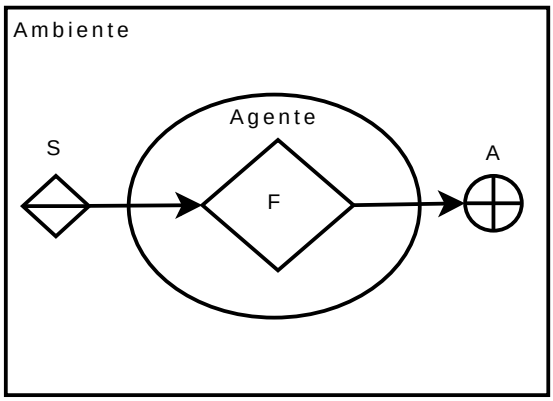

Figura 2.3. Agente interagindo com o ambiente através de sensores e atuadores.

ações devem ser executadas para que ele obtenha êxito, preferencialmente de maneira eficiente.

Segundo Russel & Norvig [2003], agentes inteligentes ou racionais utilizam uma função de desempenho para medir a sua eficiência na escolha de suas ações e, conseqüentemente, na resolução de problemas. As ações que o agente seleciona para serem executadas devem maximizar a função de desempenho. Quanto melhor o desempenho do agente, mais próximo do sucesso ele estará na resolução de problemas e, portanto, mais eficiente será o agente. Para um observador humano, o ato de maximizar uma função de desempenho pode ser interpretada como inteligência se a função mapear objetivos sensatos, evitando que o agente execute ações "estúpidas" sob o seu ponto de vista.

Para Goertzel & Pennachin, inteligência é "a habilidade de resolver problemas complexos em ambientes complexos". Ou seja, subentende-se que a resolução de problemas complexos exige um raciocínio que otimize o processo de execução das tarefas necessárias à realização desta atividade. Obviamente, se o problema é desconhecido para o agente (existe incerteza e/ou informações limitadas sobre o problema), uma etapa de aprendizagem será necessária. Nesta etapa erros são tolerados, mas desde que reconhecidos como tal, utilizados como lições e empregados na convergência do problema em uma solução. Este conceito de inteligência é um dos fundamentos da Inteligência Artificial Genérica.

Ainda não existem técnicas capazes de possibilitar a construção de agentes computacionais, que se comportem de forma eficiente e inteligente em qualquer contexto. Agentes com estas características podem ser vistos como dotados de Inteligência Artificial Genérica efetiva, o que seria algo próximo à inteligência humana. Mas em contextos fechados e reduzidos, como sistemas de Narrativa Interativa, é possível fazer

uso de técnicas mais simples de AGI para se criar agentes que executem suas tarefas de forma satisfatória. Pois mesmo comportamentos não inteligentes (pela perspectiva de um humano) para estes casos podem ter a conotação de serem "engraçados" ou "surreais" o que, de forma controlada, pode agregar valor ao sistema resultante.

2.4 Evolução do *Hardware* dos videogames: processamento disponível para a IA

Em maio de 1972, o primeiro console de videogame, batizado de Odyssey, foi lançado comercialmente no mercado americano. Criado por Ralph H. Baer, um engenheiro de televisão formado pelo *American Television Institute of Technology* (ATIT), o Odyssey introduziu um modelo de *hardware* que é utilizado até hoje, inclusive nos videogames de última geração. Este modelo consiste em um sistema de processamento de dados e geração de imagens, um dispositivo para realizar a transmissão das imagens à uma televisão, controles para mais de um jogador, além de utilizar dispositivos externos, chamados de cartuchos, para fazer a troca entre um jogo e outro. Os cartuchos do Odyssey eram simplesmente *"jumpers"*, utilizados para conectar os circuitos internos do videogame e alternar a ativação dos quase trinta jogos previamente gravados em seus circuitos [Eaters, 1998].

Quatro anos depois surge o Fairchild Channel F., o primeiro videogame com uma unidade de processamento central (CPU) e que utilizava cartuchos dotados de memória ROM para armazenar os jogos. A utilização de uma memória externa como meio de armazenamento permitia que os jogos fossem desenvolvidos e distribuídos separadamente. Todos os videogames criados após o Fairchild possuem internamente uma ou mais unidades de processamento genéricas e utilizam dispositivos de armazenamento de dados (removíveis ou não) para armazenar os jogos [Eaters, 1998]. Um ano depois do lançamento do Fairchild, a Atari lança no mercado o "Atari VCS" ou simplesmente Atari 2600. O Atari 2600 utiliza como CPU um microprocessador MOS 6507 rodando a 1.19 MHz e com capacidade de endereçar 8KB de memória (Apenas 4KB eram usados para o armazenamento dos jogos). Esta configuração é capaz de processar aproximadamente 0.0011 MIPS (Milhões de Instruções Por Segundo) [Taha & Löfstedt, 2004].

Observando, agora, os videogames dá sétima geração, lançados vinte anos após o Atari 2600, tem-se uma noção mais precisa da evolução do *hardware* dos videogames. Considerar-se-á o XBOX 360. Sua CPU é um Power PC de três núcleos, rodando à 3.2GHZ. Sua capacidade de processamento é de 19.200 MIPS ou 115 GFLOPS (Bilhões de Operações de Ponto Flutuante por segundo). Ou seja, aproximadamente 17.5×10^6

vezes mais instruções que a CPU do Atari 2600. Além disso, videogames como o XBOX 360 possuem uma unidade de processamento programável[4] exclusiva para a computação dos gráficos, chamada de GPU (do inglês Graphics Processing Unit). Os videogames da segunda geração já possuíam unidades de processamento exclusivas para a renderização dos gráficos, contudo estes processadores não eram programáveis.

Para a execução de jogos bidimensionais, a arquitetura dotada de apenas uma CPU e um processador simples de gráficos atendia muito bem. Contudo, com a produção de jogos tridimensionais, foi necessário a criação e a utilização de um *hardware* especializado para tratar as primitivas espaciais que compõem um cenário 3D, com mais eficiência que a CPU tradicional. E, portanto, a GPU foi incorporada aos videogames e passou a desempenhar esse papel. No caso do XBOX 360, sua GPU tem uma capacidade de processar 240 GFLOPS [Microsoft, 2006]. Por ter uma função específica, as GPUs normalmente têm um poder de processamento maior que as CPUs. Ou seja, no final das contas o poder computacional disponível nos consoles vigentes permite aos desenvolvedores criarem jogos muito mais complexos, dado que o *hardware* suporta cargas elevadas de processamento.

O ciclo de processamento dos jogos computadorizados é algo bastante repetitivo, utiliza extensivamente a CPU/GPU, a memória, e normalmente se resume a um laço de execução com vária etapas. As principais etapas deste laço são a leitura dos comandos do jogador (via *joystick* ou outro dispositivo de entrada), a atualização dos elementos do jogo (física, áudio, rede, etc.) e a renderização dos gráficos. Com a evolução dos processadores e memórias, mais e mais etapas foram sendo adicionadas ao laço de processamento, enriquecendo as imagens finais renderizadas, a trilha sonora, melhorando a física, aumentando o tamanho do universo físico do jogo, além de outras melhorias. A utilização da GPU reduziu bastante a demanda de processamento na CPU, mas novos papéis foram atribuídos às CPUs dos videogames modernos. Graças ao constante aumento na complexidade e no tamanho dos jogos há uma permanente necessidade de evolução no poder de processamento das CPUs de uma geração à outra.

Jogos com cenários, animações e efeitos visuais cada vez mais realistas geram no jogador a expectativa de poder interagir com o mundo virtual de forma similar à realidade, obtendo respostas às suas ações, assim como ocorre no mundo real. Além disso, o usuário espera também poder interagir e se comunicar com os personagens humanos (ou personificados) dos jogos. Para atender tais expectativas, tanto a indústria de jogos quanto o meio acadêmico tem investido bastante em pesquisa e desenvolvimento de ferramentas que tornem mais realistas as animações, expressões corporais/faciais,

[4] O programador escreve um código que será processado diretamente na unidade de processamento de gráficos ao invés de usar a CPU para isto

as forças físicas que atuam no mundo virtual, a Inteligência Artificial para controlar os personagens, além de outros elementos [Nareyek, 2007]. Todos estes recursos tecnológicos tornam o jogo uma aplicação bastante onerosa em termos de processamento. Mas, são justamente estes recursos incorporados aos jogos que definem o acelerado ciclo de evolução das CPUs/GPUs para as próximas gerações de videogames.

Atualmente existem diversos *Engines*[5] e bibliotecas que realizam simulação de física [Wikipedia, 2009b]. Muito já se investiu nesse tipo de ferramenta e, por isso, os jogos que utilizam tais recursos realmente impressionam com o nível de detalhes e realismo na simulação. Logo, pode-se notar que a simulação de física já está em um estágio bastante avançado. Por outro lado, a Inteligência Artificial, que é indispensável a praticamente qualquer jogo, ainda "engatinha" e requer muito investimento e pesquisa. Isto porque a IA aplicada aos jogos não é a mesma pesquisada no meio acadêmico, é algo bem mais simplificado e específico, se enquadrando na classe de IAs fracas. Bourg & Seeman [2004] define a IA para jogos como:

> Tudo que dá a ilusão de inteligência em um nível apropriado, tornando o jogo mais imersivo, desafiador, e, mais importante, divertido, pode ser considerado IA para jogos

O principal argumento da indústria de jogos para não usar técnicas mais avançadas de IA, como aprendizado de máquina e sistemas de inferência (mesmo versões mais leves e simplificadas dos algoritmos), são que estas podem gerar comportamentos inesperados pelos desenvolvedores, o que pode ser um problema. Mais cedo ou mais tarde jogos contendo personagens com comportamentos dinâmicos e emergentes, como por exemplo NPCs que interagem com jogadores humanos de forma única, deverão ser criados. Obviamente esta tarefa não é a das mais simples. Tornar um NPC capaz de interagir com uma pessoa adulta e corresponder às suas expectativas como se fosse um outro indivíduo humano é algo praticamente impossível atualmente, pois não existe uma tecnologia capaz de simular a mente humana em todos os seus aspectos.

Existem diversas técnicas de IA que podem ser usadas para compor um indivíduo artificial bastante convincente, como Processamento de Linguagem Natural (PLN), Aprendizado de Máquina, Sistemas Dedutivos, além de outras. Entretanto, as técnicas mencionadas são muito custosas computacionalmente e, portanto, não é recomendável usá-las em aplicações de tempo real, muito menos todas ao mesmo tempo, dadas as limitações de processamento dos videogames e computadores pessoais atuais [Mitchel, 1997]. Mas, é possível utilizar versões simplificadas de cada uma destas técnicas em

[5]Um Engine é um conjunto de bibliotecas, editores e aplicativos utilitários de diversas naturezas para uso em um propósito específico

um jogo de tempo real. Mesmo sendo simplificações dos algoritmos originais, ainda sim é possível gerar resultados surpreendentes na perspectiva do jogador.

Capítulo 3

O modelo proposto

Nesta seção será apresentado o modelo proposto para a construção de uma mente sintética artificial para controlar atores virtuais. Mas antes, é indispensável que uma relação de requisitos seja preparada e utilizada para definir justamente qual problema o modelo se propõe a resolver. Os requisitos permitem ao projetista direcionar seus esforços na busca por uma solução que atenda tais premissas de forma integrada, além de servir como guia e indicadores para o início e o término da etapa de planejamento e construção do modelo.

A implementação deste modelo deve permitir que o projetista do sistema de NI consiga modelar a personalidade e o perfil de cada ator, mudando os impactos de cada emoção no ciclo de processamento da mente, além de garantir que os agentes sejam capazes de:

- lidar com situações não previstas pelos projetistas do sistema de Narrativa Interativa.
- completar objetivos complexos (um objetivo complexo é aquele que demanda a execução de diversas tarefas para ser realizado).
- gerenciar objetivos de curto e longo prazo para alcançar suas metas.
- lidar com emoções que possam mudar os objetivos de curto e longo prazo dos agentes.
- usar fragmentos de memória para raciocinar e decidir o que fazer. Isto poderá levar o ator a realizar ações esperadas, quando um evento previsto ocorrer.

Geralmente, técnicas tradicionais de IA para jogos como *scripts* e simples sistemas baseados em regras [Brachman & Levesque, 2004] não são suficientes ou são bastante

ineficientes para se construir um sistema que atenda tais requisitos. Logo, o principal objetivo do modelo proposto é criar um mecanismo para controlar atores virtuais autônomos de sistemas de Narrativa Interativa que sejam eficientes neste contexto. Este mecanismo leva em consideração diversos elementos resultantes da interação entre os personagens da história entre si e com o ambiente virtual. Estes elementos podem modificar o comportamento dos atores virtuais, ao mudar seus objetivos e criar situações não previstas pelos agentes.

É óbvio que os requisitos supracitados não são triviais de serem satisfeitos em um único modelo, pois a integração deles requer a concepção de um sistema bastante complexo. Entretanto, é importante ter em mente que o modelo de Inteligência Artificial que será descrito nesta seção é específico para um determinado tipo de aplicação e não se pretende em hipótese alguma tentar imitar a mente humana. Ou seja, o que se pretende é controlar agentes de forma que estes executem sequências de ações que dêem a impressão de inteligência, sob a perspectiva de jogadores humanos, e sejam capazes de conduzir uma narrativa através de suas ações. Contudo, esta sequência de ações deve não somente convencer os jogadores humanos, mas também seguir uma lógica dentro do contexto da narrativa. Isso significa que, apesar dos agentes terem autonomia para realizar tarefas quaisquer, eles devem ser modelados e construídos para interpretar de forma mais fiel possível o papel de um personagem específico. Para isto, o modelo proposto deve fornecer mecanismos ao projetista da aplicação de NI para que ele consiga realizar tal tarefa.

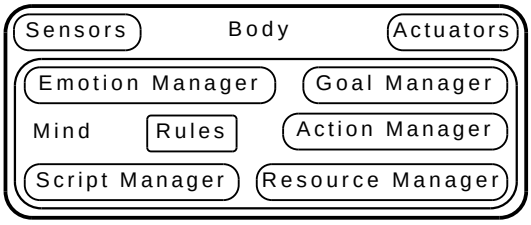

Figura 3.1. Componentes do agente.

O diagrama apresentado na Figura 3.1 representa os elementos básicos que compõem um único agente inteligente e utiliza o modelo proposto neste trabalho para controlá-lo. Atuadores, sensores e a mente são partes integrantes do corpo do agente. A mente é composta por cinco módulos, além de possuir um conjunto de regras definidos para cada modo de raciocínio do agente. O agente precisa ter sempre ativo um modo de raciocínio, por isso o elemento do diagrama chamado "Rules" faz parte da mente.

Pode-se notar que na Figura 3.1 todos os termos estão em inglês. Isto foi uma decisão proposital para tornar o modelo acessível a um maior número de pessoas. Sendo assim, todos os termos que se referem ao modelo serão referenciados em inglês, padronizando a forma de descrevê-lo.

O modelo é composto de cinco diferentes módulos mentais, como mostrado na Figura 3.1. Estes módulos trabalham em conjunto para controlar e processar o fluxo de informações que será manipulado pela mente do agente durante todo o seu ciclo de "vida". O resultado deste processamento é a realização, por parte do agente, de sequências de ações, que podem ser interpretadas como comportamentos.

Este fluxo de informações pode ser observado nas Figuras 3.2 e 3.3. Note que a coleta dos dados sensoriais é a primeira etapa do processamento do agente, passando pelos módulos que o compõe. Todo o ciclo de processamento é sempre gerenciado pelo elemento principal "Mind".

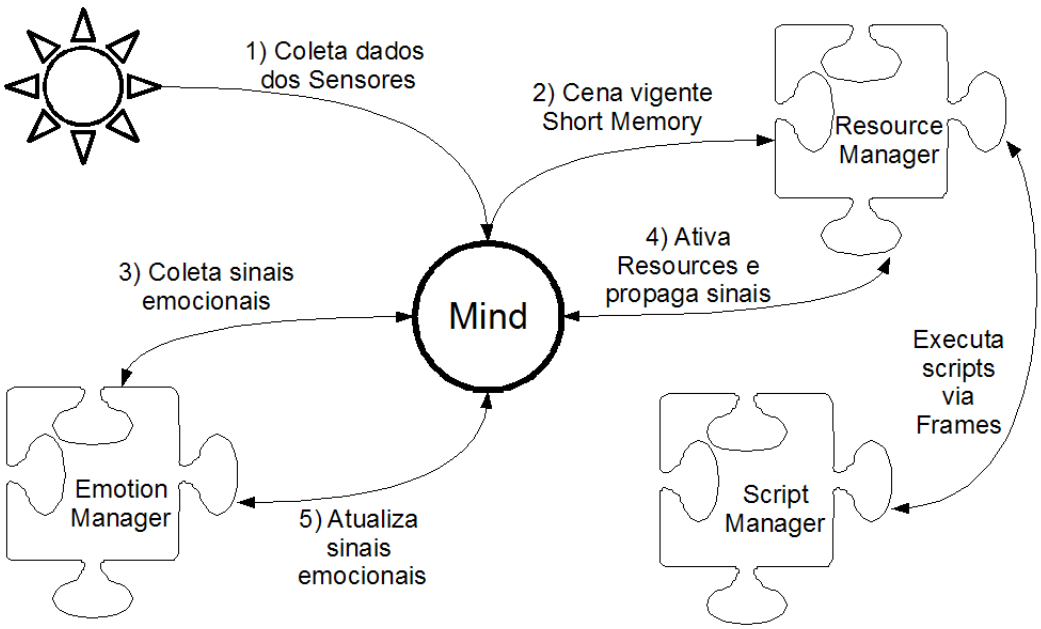

Figura 3.2. Fluxo de informações entre os módulos, primeira parte.

As próximas seções descrevem em detalhes os diversos componentes do modelo. Cada um dos 5 módulos gerenciadores (*Manager*) da mente possui uma seção exclusiva. O papel dos sensores e dos atuadores são detalhados em uma outra seção e a última seção apresenta o mecanismo de funcionamento do modo de raciocínio do agente.

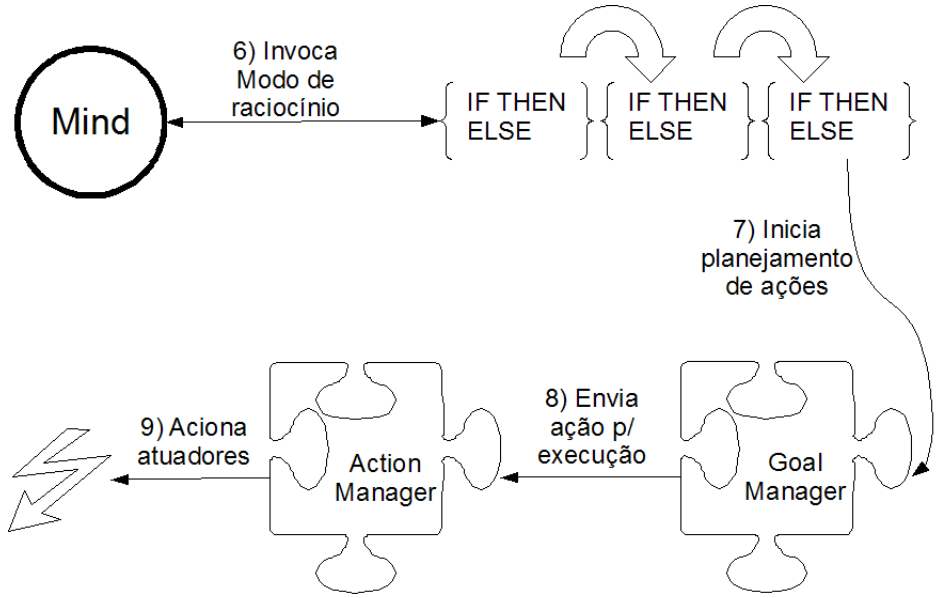

Figura 3.3. Fluxo de informações entre os módulos, segunda parte.

3.1 Resource manager

A forma com que as memórias e o conhecimento são representadas dentro da mente é uma das mais importantes características ao se projetar um modelo como o apresentado neste trabalho. Praticamente todas as operações dentro da mente artificial dependem das memórias. Logo, a melhor forma de representá-las será aquela que torne eficiente o processo de escrita e consulta de fragmentos de memória na base de conhecimento.

Uma das melhores formas de se representar conhecimento baseado em senso comum em sistemas de Inteligência Artificial é através da utilização de Redes Semânticas [Quillian, 1968]. Uma Rede Semântica pode ser usada para descrever objetos complexos e situações de uma forma simples e direta. Neste trabalho foi utilizado um tipo de Rede Semântica composta por *Frames* [Minsky, 1985]. A rede semântica formada pelos Frames é muito parecida com aquela formada por classes e objetos em uma linguagem orientada a objetos (OO). Da mesma forma que um sistema OO, os Frames podem ser compreendidos em tipos (*sorts*) ou instâncias. Um Frame tipo define a estrutura geral do que será representado por ele e sua instância representa algo (real, imaginário, abstrato ou concreto) que possa ser descrito por tal Frame. Utilizando Frames é possível representar desde objetos simples e configurações de cenas estáticas (um *snapshot* de algo em movimento) até eventos complexos e sequências de ações. No modelo proposto, o módulo *Resource Manager* é responsável por gerenciar os fragmentos de memórias descritos no formato de Frames.

O Resource Manager é utilizado por todos os demais módulos, para a realização de operações de armazenamento e consulta de memórias. Contudo, este módulo não gerencia apenas Frames e sim Resources. Frames são extensões de Resources. Um Resource é uma estrutura básica que pode ser estendida para diversos tipos de elementos com finalidades distintas, como por exemplo sinais motivacionais (Ex.: necessidades fisiológicas: sede, fome, sono, etc. ou até mesmo pequenos utilitários para realizar processamento de linguagem natural como coletores de frases ditas por outros agentes, conversores de texto para frames e vice-versa), embora este tipo de sinal não seja parte integrante do modelo e não será discutido neste trabalho. Resource é um termo bastante utilizado por Minsky [2006] para denotar simples unidades funcionais da mente. Neste trabalho, os Resources podem se relacionar, usando um tipo de conexão chamado *Similarity Link* ou conexão por similaridade. Cada Similarity Link possui um atributo de peso, que representa o grau de similaridade entre os Resources conectados. Este peso serve para o processo de propagação de Resources, que será abordado mais a frente.

Um fragmento de memória pode ser definido por um simples Frame ou de forma hierárquica, combinando diversos Frames em uma estrutura de árvore. Um Frame pode descrever diversos tipos de objetos. Os atributos de um objeto podem ser especificados através dos *Slots* dos Frames. Um Slot é uma propriedade membro de um Frame, como um membro de uma classe num sistema OO. Cada Frame pode ter ilimitados slots. Dado que um Slot pode ser preenchido por um outro Frame, eles podem ser vistos como mecanismos para se construir relações de referência entre dois Frames [Brachman & Levesque, 2004]. Um Frame pode também se relacionar com outro Frame através de herança. Uma relação de herança entre dois Frames, que é similar a uma herança OO, representa uma relação "is a" ou "é um", ou seja, o Frame filho passa a ser ("is a") um Frame do tipo pai ao herdar todas as propriedades do Frame superior. Um grupo de Resources, que se relacionam entre si via Slots, Similarity Links e/ou herança, pode ser visto como uma Rede Semântica, como mostrado na Figura 3.4. Linhas contínuas indicam herança entre Frames. Linhas tracejadas com flechas representam um relacionamento de referência onde o Slot está no Frame alvo. Elementos com os cantos arredondados são instâncias de Frames. Os números são os tempos de ativação dos recursos durante um processo de propagação de sinais.

Quando um agente está interagindo com o ambiente, diversos tipos de sinais chegam à sua mente. Estes sinais podem ser provenientes de fora de seu corpo, coletados através de sensores exteroceptivos, ou de dentro do seu corpo, usando sensores proprioceptivos. Percepções visuais e auditivas são exemplos de sinais coletados no ambiente externo. Já emoções e necessidades fisiológicas são exemplos de sinais internos. Sen-

sores proprioceptivos estão fora do escopo deste trabalho, mesmo embora eles possam ser implementados como Resources. Os sinais coletados pelos sensores irão influenciar diretamente o processo de tomada de decisão dos agentes.

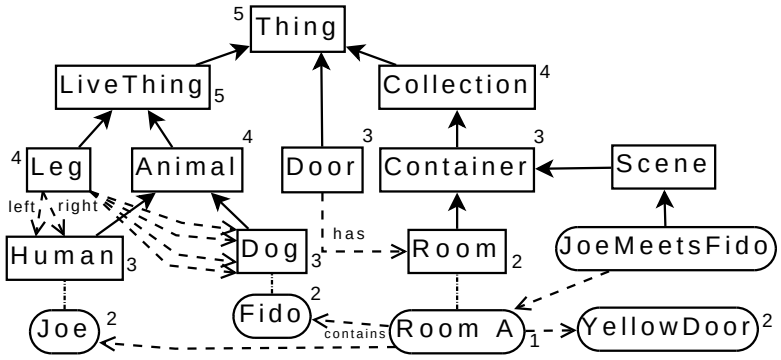

Figura 3.4. Exemplo de uma Rede Semântica.

Quando um sinal é reconhecido pela mente, um ou mais Resources, relacionados com este sinal, emergem e se tornam ativos durante um curto prazo ou enquanto a chegada deste sinal à mente persistir. Então, um sinal é propagado de cada Resource ativo para outros Resources, seguindo as relações existentes entre eles na Rede Semântica. Esta propagação, depois de inicializada, continua até que um número máximo de Resources ativos é atingida. A ativação de um Resource é classificada por um peso. Quanto mais forte for este peso, maior é a importância deste Resource para o contexto vigente. O objetivo destas restrições e das regras de propagação de Resources é evitar uma explosão de ativação de Resources desnecessários, o que atrapalharia o desempenho do sistema que implemente este modelo.

As duas próximas subseções tratam de duas configurações especiais de Frames, que são referenciados extensivamente por diversas operações da mente artificial. Porém, é importante deixar claro que estes Frames são especiais não por serem construídos de forma diferenciada dos demais, mas sim por desempenharem papéis chave no modelo.

3.1.1 Scene frame

Um Frame do tipo Scene é um contêiner que armazena todos os elementos localizados em um lugar específico (real ou imaginário), em um dado momento. Um Scene Frame pode ser entendido como uma foto tirada de local específico (uma sala, uma rua ou mesmo um local experimentado em um sonho). Além disso, um Frame do tipo Scene também armazena os relacionamentos espaciais entre os elementos (perto, longe, atrás, etc.) tomando como referência a localização e a orientação do agente ou um ponto de

observação da cena específico. Ex.: Joe está perto de Fido; Fido está em frente à porta amarela.

Cada cena possui uma classificação emocional. Ou seja, cada cena possui uma lista de emoções e suas respectivas intensidades. Dessa forma, quando um Frame do tipo Scene é ativado, as emoções a ela vinculadas emergem e alteram o estado emocional do agente, desencadeando uma cascata de ativações de outros Frames, podendo até mudar os objetivos de curto e longo prazo do agente.

Por fim, uma outra informação, associada ao Scene Frame, classifica o tipo da situação que a cena descreve. A situação de uma cena pode ser classificada como boa, neutra, ruim ou crítica. A cada ciclo de processamento da mente artificial, o agente classifica a situação do Scene Frame vigente (que representa a cena vivenciada pelo agente no momento mais recente do tempo), utilizando analogia com experiências passadas, armazenadas em forma de fragmentos de memória. Se a situação da cena vigente for ruim ou crítica, então o agente tenta encontrar uma forma de sair daquela situação. Para isso, o agente procura no Story Frame que possui o Scene Frame utilizado no reconhecimento da situação atual. Este processo será descrito com mais detalhes mais a frente.

3.1.2 Story frame

Quando uma ação é executada por algum agente dentro do mundo virtual, alterações no ambiente podem ocorrer e, consequentemente, podem alterar a cena vigente percebida pelos agentes. Logo, para que o agente tenha conhecimento do curso das ações que estão sendo realizadas no ambiente por qualquer agente, um Frame do tipo Story é utilizado. Baseado em conceitos de Trans/Story Frames, descritos em Minsky [1985] e Minsky [2006], o Story Frame é uma estrutura que armazena ações e seus impactos no ambiente num formato de histórico, usando uma sequência de outros Frames chamados Trans. Um Trans Frame é composto de três partes, uma cena anterior (representada por um Scene Frame, que descreve uma dada cena), uma ação (que é executada na cena anterior) e uma cena posterior (que é a cena anterior modificada pela ação). Quando uma lista de Trans Frames são sequenciados por um Story Frame, pode-se notar que a próxima cena do primeiro Trans Frame é a mesma referenciada pela cena anterior do próximo Trans Frame. Portanto, esta relação temporal de cena, ação, cena, ação, etc. indica o que está acontecendo dentro do ambiente.

O conjunto de Trans Frames que um Story Frame armazena pode ser entendido como uma história, uma crônica registrada na memória dos agentes. Esta história, assim como todas as outras histórias representadas por Frames do tipo Story, são

utilizadas pelo agente no processo de raciocínio, a ser detalhado mais adiante.

3.2 Goal manager

Um *goal* é um objetivo perseguido pelo agente por um motivo qualquer. Diversos fatores podem servir de motivação para a busca de um objetivo: emocionais (medo, paixão, raiva, etc.), segurança, satisfazer necessidades fisiológicas, etc.. Existem basicamente dois tipos de objetivos, de curto e longo prazo. Um objetivo de longo prazo, na maioria das situações, é mais difícil de ser alcançado. Além do mais, um objetivo de longo prazo pode somente ser alcançado através da realização de um ou mais objetivos de curto prazo. Já um objetivo de curto prazo é realizado com a execução bem sucedida de uma simples ação.

Uma ação possui pré-condições e efeitos. Pré-condições são estados que precisam ser satisfeitos para que a ação possa ser executada. Por exemplo, uma das pré-condições da ação "destrancar porta" é que o agente esteja "perto da porta". Já os efeitos da ação são justamente os elementos do ambiente virtual que foram modificados pelo agente depois da ação. É importante ressaltar que não existe uma lista de efeitos no Frame que representa a ação, assim como a que existe em sistemas baseados em regras inspirados no S.T.R.I.P.S. [Russel & Norvig, 2003, apud Fikes & Nilsson [1990]], ao contrário, os efeitos são os resultados da ação, percebidos pelos agentes através de seus sensores. Considerar-se-á, por exemplo, que a ação "abrir porta" somente pode ser executada se os estados "porta fechada" e "porta destrancada" estiverem ativos. Uma vez satisfeitas as pré-condições e executada a ação, um novo estado "porta aberta" deverá ser percebido pelo sensor de visão do agente.

Cada pré-condição pode ser configurada com uma lista de ações que, ao serem executadas, pode tornar tal pré-condição satisfeita. Dessa forma, se o objetivo do agente é "abrir porta", mas este está distante da porta e a pré-condição "perto da porta" não está satisfeita, então o agente poderá adicionar um novo objetivo de curto prazo, contida na lista de ações da pré-condição, "ir até a porta", que satisfará a pré-condição "perto da porta" e, por fim, o agente poderá completar o seu objetivo inicial.

Não necessariamente uma ação será bem sucedida ao ser executada. Ou seja, não há garantia que um *goal* de curto prazo será satisfeito apenas porque a ação correspondente a ele foi executada. Quando uma ação falha, um Frame de fracasso, contendo o motivo pelo qual falhou e o contexto da ação, é ativado. Este Frame é então usado por outros processos da mente, que poderão fazer com que o agente tente outras abordagens para resolver o problema enfrentado. Este tipo de operação é tratado pelo

Reasoning Mode, parte da mente que executa uma série de regras de raciocínio a cada ciclo de execução da mente, que será abordado na seção 3.7. Contudo, depois que as regras de raciocínio (executadas pelo Reasoning Mode) registram um novo *goal* no Goal Manager, este módulo tenta quebrar o *goal* em objetivos de curto prazo menores e mais fáceis de serem trabalhados.

Em resumo, todas as funcionalidades descritas acima são responsabilidades do Goal Manager, um módulo de gerenciamento de objetivos do modelo geral. Mas, apesar do Goal Manager organizar os objetivos em ações a serem executadas, não é papel deste módulo a execução destas ações. Para isto, um módulo chamado Action Manager, que faz parte do modelo proposto, tem a responsabilidade de controlar o fluxo de execução das ações. A sequência de objetivos de curto prazo que precisam ser alcançados pelo agente pode ser chamada de plano. Os comportamentos apresentados pelos agentes são justamente o resultado da execução destes planos.

3.3 Action manager

Executar uma ação significa, na maioria dos casos, usar os atuadores para interagir com o mundo virtual. Supondo que o agente possua mais de um atuador, em geral, cada atuador poderia executar ações independentes a qualquer momento. Mas isto apenas aumentaria desnecessariamente a complexidade do sistema. Logo, o Action Manager é responsável por receber ações a serem executadas e escaloná-las, obedecendo uma ordem de prioridade atribuída aos *goals*.

Uma ação escalonada pode se encontrar em 6 estados: não iniciada, executando, paralisada, cancelada, finalizada com sucesso ou com falha. Uma ação em execução pode ser paralisada para que outra, que irá satisfazer um *goal* com maior prioridade, entre em execução. Uma ação paralisada pode voltar à execução posteriormente. Ações finalizadas são removidas da lista de execução no ciclo de execução posterior e o resultado é repassado ao Goal Manager, para que este trate o resultado e faça o planejamento de novos goals. Uma ação pode ser cancelada caso seja parte de um *goal* que já foi satisfeito.

Como mencionado anteriormente, quando um *goal* de curto prazo tem todas as suas pré-condições satisfeitas, sua ação correspondente é enviada ao Action Manager. Então, depois desta ação ser executada e o Goal Manager ter registrado o resultado desta execução, a ação é então anotada em um Story Frame para compor o histórico de todas as ações percebidas pelo agente.

O Action Manager é o único módulo que precisa trabalhar em paralelo aos demais

processos da mente, permitindo que, enquanto uma ação é executada, os outros módulos continuem trabalhando e decidindo se novos objetivos precisam ser alcançados, dada a situação em que o agente se encontra.

3.4 Emotion manager

Picard [1997] ressalta em seu trabalho que a utilização de emoções em sistemas inteligentes contribui para o enriquecimento das respostas dadas pela aplicação de três maneiras: reconhecer as emoções do usuário e, consequentemente poder raciocinar usando estas informações; expressar emoções, utilizando agentes animados; e simular emoções (papel de um sistema emocional), utilizando sinais e estímulos produzidos por eventos no ambiente para gerenciar as intensidades de emoções baseadas em um modelo de emoções humanas. O Emotion Manager, módulo apresentado nesta seção, implementa a terceira maneira de lidar com as emoções sugerida por Picard e sintetiza a idéia de um sistema emocional capaz de simular emoções para influenciar o processo de tomada de decisões do agente.

Um dos trabalhos sobre computação utilizando emoções mais importantes é o modelo Ortony, Clore & Collins (OCC)[Ortony et al., 1988]. O OCC serviu de referência para a construção de diversos sistemas emocionais, apesar de não ter sido criado com o propósito de ser um sistema emocional. O OCC foi concebido com o objetivo de ser um modelo para avaliação das emoções contidas em informações produzidas por humanos (Por exemplo, processamento de linguagem natural e resolução de problemas de forma cooperativa), facilitando o processo de raciocínio do sistema de IA.

Reilly & Bates [1996] detalharam em seu trabalho um sistema emocional baseado no modelo OCC chamado *Em*. O sistema emocional *Em* define dois tipos de emoções, positivas e negativas. Emoções positivas são aquelas que levam o agente a um bom humor, enquanto que as negativas o conduzem a um mal humor. *Em* é um sistema simples, direto e fácil de se utilizar/manter, que foi concebido pelos autores para ser um sistema computacional de gestão de emoções. O módulo Emotion Manager se baseia no modelo matemático do *Em*, além de utilizar sua relação de emoções.

Cada emoção é caracterizada por um sinal. A intensidade deste sinal é computada através de uma função sigmoide proposta por Picard [1997] [1]. Esta função recebe um pulso como entrada e informa a intensidade da emoção como saída, dado àquele pulso. Os parâmetros desta função podem ser ajustados para que aspectos da personalidade do personagem sejam evidenciados ou suprimidos. Como o Emotion Manager lida com

[1] Procurar por "Emotions and Moods for Animated Characters" em Picard [1997] para mais detalhes

22 emoções positivas e negativas (Tabela 3.1), a personalidade do agente pode ser moldada para torná-lo mais sensível a um determinado tipo de emoção. Se é desejado a criação de um personagem covarde, por exemplo, pode-se configurar a emoção "Fear" para que seu sinal suba de forma intensa a qualquer sinal de entrada. O Emotion Manager possui um indicador do humor do agente, baseado na intensidade das emoções vigentes. Este indicador é computado usando uma função logarítmica proposta em [Reilly & Bates, 1996], que utiliza somatórios ponderados das emoções positivas e negativas como entrada.

Tabela 3.1. Hierarquia padrão de emoções de Em [Reilly & Bates, 1996].

Total	Positive	Joy
		Hope
		Happy-for
		Gloating
		Love
		Satisfaction
		Relief
		Pride
		Admiration
		Gratitude
		Gratification
	Negative	Distress
		Fear - Startle
		Pity
		Resentment
		Hate
		Disappointed
		Fears-Confirmed
		Shame
		Reproach
		Anger - Frustration
		Remorse

Dependendo do estado emocional do agente (intensidade do sinal de cada emoção do agente), fragmentos de memórias, configurados por Frames, serão ativados e/ou desativados, o que, consequentemente, terá um impacto sobre os objetivos do agente. Isso significa que se o agente estiver, por exemplo, com a intensidade elevada da emoção "Fear", um Scene Frame caracterizado por esta emoção poderá emergir, fazendo com que o agente reconheça a situação vigente como ameaçadora e acabe perseguindo um *goal* com maior prioridade como por exemplo "Chorar" ou "Fugir".

3.5 Script manager

Devido ao comportamento dinâmico dos Resources, o uso de uma linguagem de *scripts* torna-se necessário para a realização de tarefas como a criação, destruição e ativação de Resources. Além disso, a computação de rotinas implementadas por Slots de Frames é facilitada com a utilização de tal recurso, pois um Slot, como um membro de um Frame, pode também ser configurado como uma rotina a ser processada em tempo de execução.

A linguagem de *scripts* facilita o acesso aos Resources por partes externas da mente, incluindo sensores e atuadores. Para tanto, o módulo Script Manager foi projetado e incorporado ao modelo proposto, para registrar e tornar disponível, em um ambiente comum, todos os Resources gerenciados pela mente. Este ambiente comum é provido pela linguagem de *scripts*, que disponibiliza todos os Resources e diversas outras partes do modelo (a mente do agente e seus módulos, sensores, etc.), na forma de variáveis que podem ter suas interfaces chamadas por pequenos *scripts*. Este módulo torna facilitada a manutenção de todas as funções que são Slots membros de Frames, além de tornar o sistema como um todo uma ferramenta melhor para a realização de experimentos e criação de novas funcionalidades, sem a necessidade de modificar a estrutura da aplicação.

3.6 Sensores e atuadores

Sensores são responsáveis por coletar informações do ambiente e, com ajuda de outros módulos da mente, criar fragmentos de memória de curto prazo. Fragmentos de memória de curto prazo são Resources que podem ser esquecidos pelo agente após um curto período de tempo. Este tipo de memória é muito útil para planejar *goals* de curto prazo. Entretanto, se uma memória de curto prazo se tornar ativa por um longo período de tempo, ela pode ser transformada em uma memória de longo prazo, que é um tipo de memória que nunca será excluída do Resource Manager.

Cada elemento coletado pelos sensores é classificado usando a intensidade de sua percepção. Um sensor de visibilidade, mais conhecido como *Field Of View (FOV)* ou campo de visão, por exemplo, daria maior importância aos objetos posicionados em seu centro, por exemplo. Desta forma, um objeto com maior intensidade de percepção terá maior chance de se tornar uma memória de longo prazo. Cada elemento ativo percebido irá propagar sua intensidade de ativação, através da rede semântica gerenciada pelo Resource Manager, fazendo com que fragmentos de memória a ele relacionados sejam também ativados. Dessa forma, esta propagação influencia diretamente o plano de

goals do agente, dado que Resources ativos com grande intensidade serão utilizados com maior frequência no processamento das regras de raciocínio.

Já os atuadores são partes físicas do agente, utilizados pelas ações para interagir com o mundo virtual. Se o agente possuir uma garra, por exemplo, ações como "Segurar" e "Soltar" podem ser implementadas para utilizar tal atuador, permitindo que este agente possa transportar objetos dentro do mundo virtual. Conforme mencionado anteriormente, ações executadas com sucesso alteram o estado do mundo virtual. Estas alterações são justamente feitas pelos corpos físicos dos agentes, o que inclui seus atuadores. Então, depois de um agente qualquer executar uma ação como "Segurar(chave)", o agente e o alvo da ação (neste caso a chave) são percebidos pelos sensores de todos os agentes presentes no ambiente. Em seguida, alguns estados passam a ser representados na memória de cada um dos agentes, utilizando Frames para descrever algo como "Segurado(agenteX, chave)" e "Perto(agenteX,chave)", informando que o "agenteX" está segurando a "chave" e também está perto dela.

3.7 Processos de raciocínio

> A forma mais eficiente de se resolver um problema é saber como resolvê-lo.
> [Minsky, 1985]

A cada ciclo de atualização do sistema, a mente do agente executa diversas tarefas de forma sequencial, como descrito no Algoritmo 1. Uma destas operações é a execução do conjunto de regras de raciocínio do Reasoning Mode vigente.

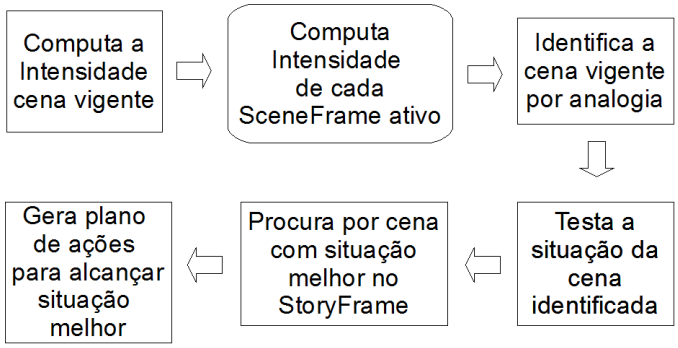

Figura 3.5. Etapas do Modo de Raciocínio Padrão.

Um Reasoning Mode é um conjunto de regras fixas, encapsulado por um objeto que utiliza algo semelhante ao padrão Strategy (Padrão de desenho do Gamma et al. [1995]), executado de forma sequencial. O Reasoning Mode é o núcleo da mente do agente, o elemento que trata todas as informações provenientes dos módulos e gera

os objetivos que serão trabalhados pelo Goal Manager. Um agente pode ter mais de um Reasoning Mode configurado em sua mente. Se o Reasoning Mode vigente não conseguir resolver um dado problema enfrentado pelo agente, um outro Reasoning Mode pode ser então acionado para tentar resolvê-lo. É como se fosse uma mudança na forma de ver o problema.

Algoritmo 1: Mind Processes

```
foreach sensors do
    foreach perceived element by sensor do
        create/update short term memory for element;
foreach emotion signals do
    pass the current emotion signal to Resource Manager;
update resource manager;
compute the activation of the emergent resources;
collect the emotion strengths of all active scenes and update emotion
manager;
create a temporary Scene Frame using all short term memory elements;
ReasoningRules();
```

O algoritmo *ReasoningRules*() é a execução do Reasoning Mode vigente. A arquitetura deste modelo é bastante customizável e qualquer conjunto de regras pode ser especificado para servir como Reasoning Mode. Porém, é sugerido no Algoritmo 2, uma lista com as regras básicas que compreendem o Reasoning Mode padrão, utilizado na construção do protótipo (Descrito no Capítulo 4). Nota-se que o Reasoning Mode apresentado no Algoritmo 2 é bastante simples e tenta sempre manter o agente em uma situação boa ou, na pior das hipóteses, neutra. As etapas do Reasoning Mode padrão em alto nível podem ser observadas na Figura 3.5.

Um Resource pode se tornar ativo, se ele receber um sinal de outro Resource, através de propagação de sinal de ativação, ou via sinais emocionais. Logo, para se normalizar todas essas fontes de sinal e conseguir computar o peso final de ativação do Resource, pode-se utilizar a equação 3.1.

$$s = \begin{cases} A(((i * d) + (\sum_e i * e * f)/n)/2), & \text{if } n \geq 1 \\ A(i * d), & \text{caso contrário} \end{cases} \tag{3.1}$$

onde s é a intensidade resultante de ativação; A é uma função de amortização usada para manter a intensidade dentro de um intervalo fixo, por exemplo 0.0-1.0 (no protótipo desenvolvido foi utilizado uma função sigmoide para realizar esta tarefa); d é um fator de decaimento de intensidade

de acordo com a distância do Resource a ser ativado, a partir do que está propagando o sinal; i é a intensidade do sinal de ativação de um Resource vinculado a um fragmento de memória de curto prazo; e é a intensidade do sinal das emoções relacionadas com o Resource (na maioria dos casos somente Scenes Frames terão emoções associadas); f é um fator emocional que afeta o Scene Frame (influência da emoção no Scene Frame em questão) e n é o número de emoções relacionadas.

Algoritmo 2: Reasoning Rules.

```
// f = scene identification factor
f ⟵ 0.5;
a ⟵ Equation 4.3(short term memory Scene) ;
foreach scene in active Scene Frames do
    b ⟵ Equation 4.3(scene) ;
    if b ≥ a * f then
        c ⟵ Story Frame which contains scene;
        if ∃ c then
            if scene ≡ bad situation then
                d ⟵ find good/neutral scene in c;
                if ∃ d then
                    create a goal using d;
            else if scene ≡ neutral situation then
                d ⟵ find a good scene in c;
execute another rules;
update goal manager;
update Emotion Manager;
```

Um Resource não permanece ativo para sempre. Logo após ele se tornar ativo, sua intensidade começa a diminuir por um fator constante (isso é uma simplificação rudimentar do processo de esquecimento do cérebro humano). Se o Resource não mais receber um novo sinal de ativação, ele se tornará inativo. Um fragmento de memória de curto prazo, quando se torna inativo, tem todos os Resources que o constituem removidos da memória (caso não sejam referenciados por outro fragmento de memória), caracterizando um processo de esquecimento. Por outro lado, se um Resource receber um sinal de ativação de forma contínua, durante um longo período de tempo (maior que um limiar de tempo customizado), este se tornará um fragmento de memória de longo prazo.

De maneira geral, a interação entre todos os módulos apresentados neste capítulo torna o agente capaz de realizar diversas tarefas durante todo o seu ciclo de vida dentro do ambiente virtual. O Resource Manager centraliza toda a informação utilizada pelos demais módulos, arnazenando fragmentos de memória e representando o conhecimento do agente. O Emotion Manager influencia o processo de ativação de Resources dado o estado emocional vigente do agente. O Goal Manager e o Action Manager são responsáveis pela gestão dos objetivos do agente, gerando e executando planos de ações. O Script Manager funciona como uma ferramenta para auxiliar a criação de estruturas de memórias dinâmicas de forma rápida e flexível. E o Reasoning Mode funciona como um centralizador das atividades mentais do agente, controlando o fluxo de informações que passam de um módulo a outro do modelo.

Capítulo 4

Protótipo

Para avaliar/experimentar o modelo proposto, foi desenvolvido um protótipo funcional que inclui um simulador gráfico simples de um ambiente virtual, onde o comportamento de agentes inteligentes podem ser observados.

Não é tarefa simples construir um protótipo que contenha todos os elementos de uma aplicação real de Narrativa Interativa. Aplicações desta natureza normalmente requerem um mundo virtual rico em número e diversidade de elementos, bem como modelos de personagens e entidades com uma enorme gama de animações, para que a narrativa possa ser contextualizada de uma forma mais precisa. É importante que os personagens tenham também um número variado de ações para interagir com o ambiente. Alguns sistemas de NI utilizam Processamento de Linguagem Natural para que o usuário possa se comunicar com os personagens controlados pela IA, mesmo que de forma bastante limitada como no trabalho de Mateas & Stern [2003].

A construção de um ambiente virtual completo para a experimentação de aplicações de NI está fora do escopo deste trabalho e, portanto, houve uma etapa de avaliação de ferramentas que poderiam ser usadas na preparação de ambientes virtuais de forma mais simples. O intuito da etapa de avaliação destas ferramentas era permitir a concentração de esforços na construção do código referente ao modelo e não ao simulador do ambiente virtual.

A primeira ferramenta avaliada, chamada Multiverse [1], é destinada a construção de mundos virtuais customizados. O Multiverse oferece diversas ferramentas para: realizar o controle de animações; gerenciar a interatividade entre os personagens e o ambiente virtual; controlar a física do ambiente; e permitir a construção de cenários abertos e/ou fechados (dentro de construções, cavernas, etc.). Porém, o seu código é fechado, e a única forma de se desenvolver para este sistema é estendendo suas

[1] http://www.multiverse.net

funcionalidades.Implementar novas funcionalidades que não foram previstas pelos desenvolvedores do Multiverse é algo bastante custoso, pois sua arquitetura não favorece a construção/integração de novos módulos ao sistema. Por exemplo, a construção de um simples sistema sensorial (visão e audição) para os agentes requer a codificação de diversos componentes que utilizam um sistema bastante ineficiente de troca de mensagens entre os elementos do sistema, o que faz com que a aplicação final bastante consuma muito recurso computacional. Logo, descartou-se a possibilidade de se utilizar o Multiverse.

Uma outra ferramenta voltada à construção de mundos virtuais chamada realXtend [2], foi avaliada. O realXtend, ao contrário do Multiverse possui seu código completamente aberto. O realXtend utiliza a plataforma OpenSimulator [3], que é um gerenciador de mundos virtuais baseado no código fonte do Second Life [4]. Apesar de possuir diversos recursos já implementados, necessários para a realização dos experimentos com o modelo proposto, a versão disponível durante o desenvolvimento deste trabalho não era estável, e alguns recursos básicos desejados não estavam funcionando corretamente (Ex.: a simples ação de um avatar segurar um objeto).

Além destes mundos virtuais dois *Motores Gráficos* de jogos foram avaliados: Unreal Engine [5] e Quake 3 Engine [6]. Contudo, o resultado final desta prospecção foi o mesmo: apesar destas tecnologias possuírem diversos recursos para se construir uma aplicação gráfica tridimensional de qualidade, o volume de trabalho demandado para se criar e integrar toda a parte sensorial (principalmente o FOV) dos agentes, além de outras funcionalidades como o painel de status do agente, seria tão grande que tomaria mais tempo do que a própria implementação do modelo.

O objetivo da busca por uma ferramenta já existente era justamente poupar tempo, porém concluiu-se que seria mais vantajoso construir um simulador simplificado, mas já adequado aos requisitos necessários à etapa de experimentação deste trabalho, do que tentar adaptar uma das tecnologias existentes. Logo, um simulador de um ambiente virtual bidimensional minimalista foi implementado. Este simulador será detalhado na seção 4.2.

[2] http://www.realxtend.org/
[3] http://opensimulator.org/
[4] http://secondlife.com/
[5] http://www.unrealtechnology.com/
[6] http://www.iddevnet.com/

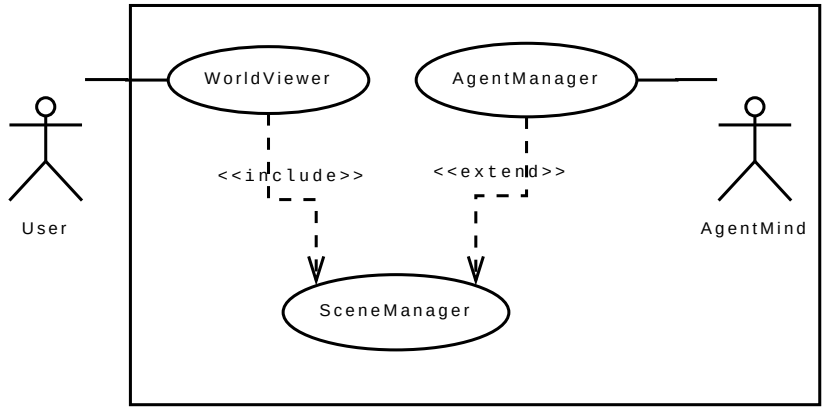

Figura 4.1. Diagrama de contexto do protótipo.

4.1 Arquitetura

Em um nível bastante abstrato, a Figura 4.1 apresenta os principais elementos que compõem o protótipo. O usuário (User) faz uso da aplicação através do visualizador do mundo, uma aplicação gráfica (detalhada na seção 4.2), que recebe os comandos do usuário e invoca o gerenciador de cenas para que a aplicação seja atualizada. O gerenciador de cena (Scene Manager) é o módulo responsável pelo carregamento e gerenciamento (criação, atualização e destruição) de todos os elementos que constituem a cena. Estes elementos são representados pela classe Entity (Figura 4.4).

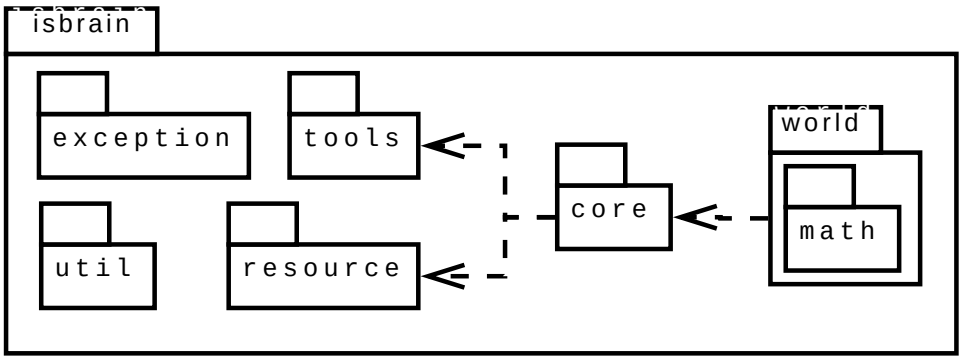

Figura 4.2. Diagrama de pacotes.

A Figura 4.2 dá uma visão geral dos elementos que compõem o protótipo implementado. O pacote **isbrain** agrupa os demais pacotes do sistema. O pacote **exception** possui todas as classes de exceções utilizadas pelo sistema. O pacote **util** (Figura 4.5 possui um utilitário para a realização de registro de mensagens. O pacote **tools** (Figura 4.3) possui ferramentas essenciais como o ScriptManager e um tratador de mecanismo de persistência da aplicação. O pacote **resource** contém o gerenciador de Resources

e Frames da aplicação. O pacote **core** contém estruturas básicas da mente, além de módulos para o gerenciamento de ações o *goals*. O pacote **world** possui as ferramentas necessárias para a realização da simulação e o pacote **math** (Figura 4.6), que é está dentro de **world** agrupa classes responsáveis por cálculos matemáticos utilizados pelo Simulador 2D.

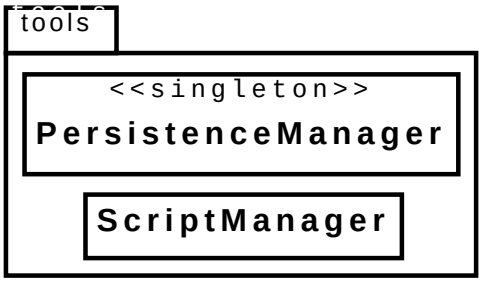

Figura 4.3. Diagrama de classes do pacote **tools**.

A classe ScriptManager exibida no diagrama da Figura 4.3 possui, como membro da classe, uma referência para o elemento que provê o ambiente de desenvolvimento Lua. Já a classe PersistenceManager, como é necessária a diversos pontos da aplicação, foi desenhada como um Singleton.

Todas as classes utilizadas pelo Simulador 2D para representar objetos no mundo virtual estão no pacote exibido na diagrama de classes do pacote **world** (Figura 4.4). **world** é o maior pacote do protótipo e inclui o pacote de classes para operações matemáticas **math**.

O módulo AgentManager (Figura 4.1) encapsula as rotinas de processamento dos agentes inteligentes. Este módulo é utilizado pelo gerenciador de cena e faz acesso aos serviços oferecidos pela mente artificial, através de sua interface. Já a mente artificial pode ser vista como um usuário externo à essa pequena aplicação de teste. Representado no diagrama de contexto pelo ator AgentMind, todo o código responsável pela gerência dos recursos da mente artificial estão disponíveis através de uma biblioteca, por isso a representação via um elemento externo à aplicação.

Todo o protótipo foi desenvolvido utilizando C++ como linguagem de programação base, e compilada com o GCC [7]. Duas bibliotecas foram extensivamente utilizadas neste protótipo, Boost [8] e STL [9]. Diversos elementos como estruturas de dados, gerenciadores de memória, controladores de tempo, threads, etc. foram aproveitados destas bibliotecas, agilizando o processo de desenvolvimento.

[7] http://gcc.gnu.org/
[8] http://www.boost.org
[9] http://www.sgi.com/tech/stl/

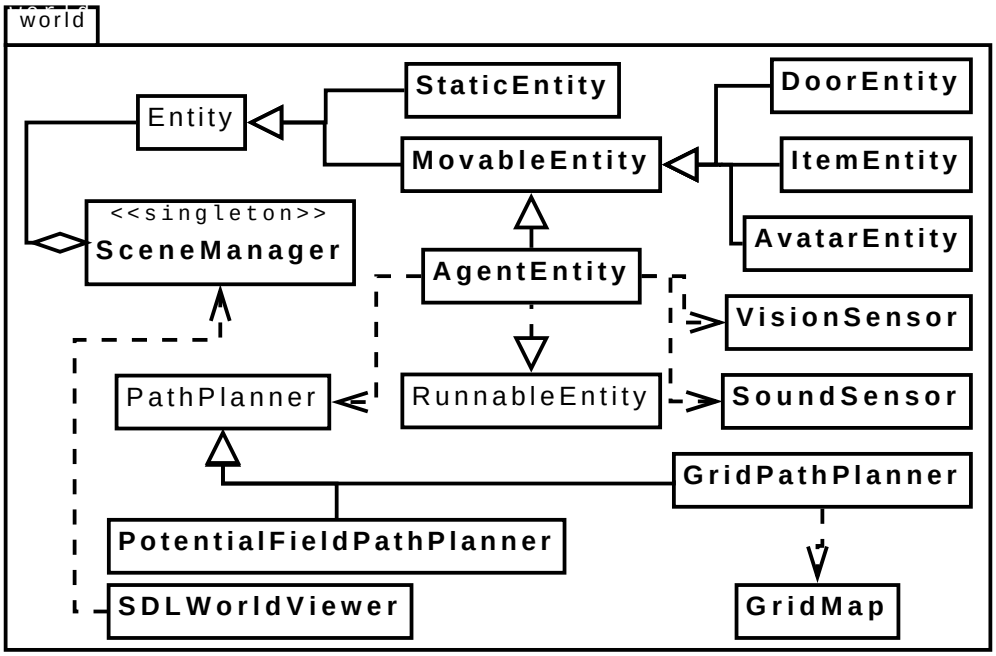

Figura 4.4. Diagrama de classes do pacote **world**.

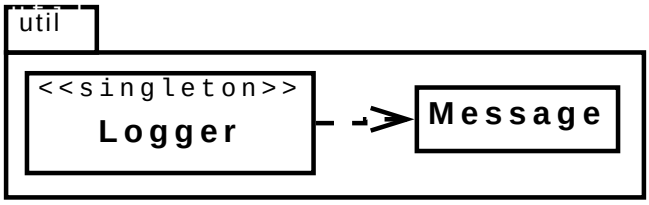

Figura 4.5. Diagrama de classes do pacote **util**.

O funcionamento do protótipo ocorre da seguinte forma: o primeiro elemento a ser inicializado é o Simulador 2D, que configura os dispositivos de *hardware* e invoca o gerenciador de cena para realizar o carregamento de todos os dados do sistema. Uma cena é composta por todos os elementos presentes graficamente no ambiente virtual simulado. Assim que a cena é então carregada, o Simulador 2D inicia o ciclo de atualizações do sistema, que se repetirá cerca de 60 vezes por segundo durante todo o período que a aplicação estiver executando. Este ciclo inclui eventos diversos como captura dos comandos enviados pelo usuário via teclado e mouse, atualização do gerenciador de física, atualização dos sensores dos agentes, renderização dos gráficos, etc. Os agentes presentes na cena possuem um ciclo de atualização separado do ciclo do Simulador 2D. Eles realizam operações ligadas ao raciocínio em frequências diferentes e de forma independente do Simulador 2D para evitar problemas de atraso na realização de suas tarefas. Em seu ciclo independente de processamento, os agentes atualizam as intensidades de ativação de seus fragmentos de memória, atualizam o Goal Manager

para se certificar do correto andamento de seus objetivos, invocam o Reasoning Mode para a execução das regras de raciocínio e, finalmente, atualizam o Action Manager para executar uma eventual ação demandada pelo seu objetivo vigente. Isso resume toda a atividade realizada pelo protótipo implementado. As próximas seções descrevem com mais detalhes o processo de desenvolvimento deste protótipo.

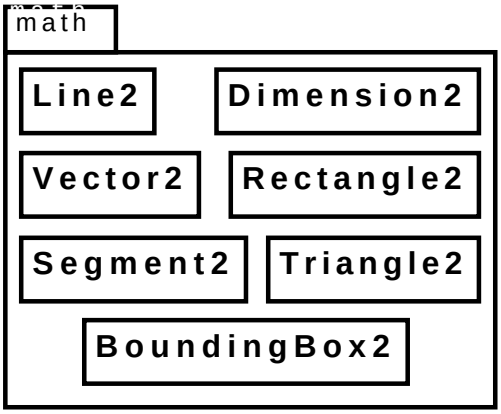

Figura 4.6. Diagrama de classes do pacote **world::math**.

4.2 Simulador 2D

O simulador 2D é a aplicação que faz o interfaceamento com o usuário. Ele é responsável por capturar os comandos do usuário, invocar todos os demais módulos do protótipo para realizar a atualização da aplicação, além de renderizar em tempo real o ambiente virtual bidimensional que está sendo simulado. Obviamente, conforme mencionado anteriormente, o simulador 2D gerencia um ambiente extremamente simplificado, se comparado aos ambientes virtuais encontrados em jogos e simuladores de mundos. Contudo, apesar de ser um simulador minimalista de um único ambiente, seus recursos são suficientes para realizar os testes necessários à validação do modelo proposto neste trabalho.

A tecnologia utilizada na renderização dos gráficos é o OpenGL (*Open Graphics Library*)[10]. Apesar desta ser uma tecnologia bastante utilizada para a renderização de gráficos 3D, fez-se uso recursos de renderização 2D de sua API (*Application Programming Interface* ou Interface de Programação de Aplicativos). Para realizar o controle do hardware (video, teclado, mouse, etc.) utilizou-se uma outra biblioteca chamada SDL (*Simple DirectMedia Layer*) [11]. A biblioteca SDL oferece recursos variados para o

[10]http://www.opengl.org/
[11]http://www.libsdl.org/

controle de dispositivos de *hardware*, que facilitam a inicialização e o uso destes recursos pelo programador.

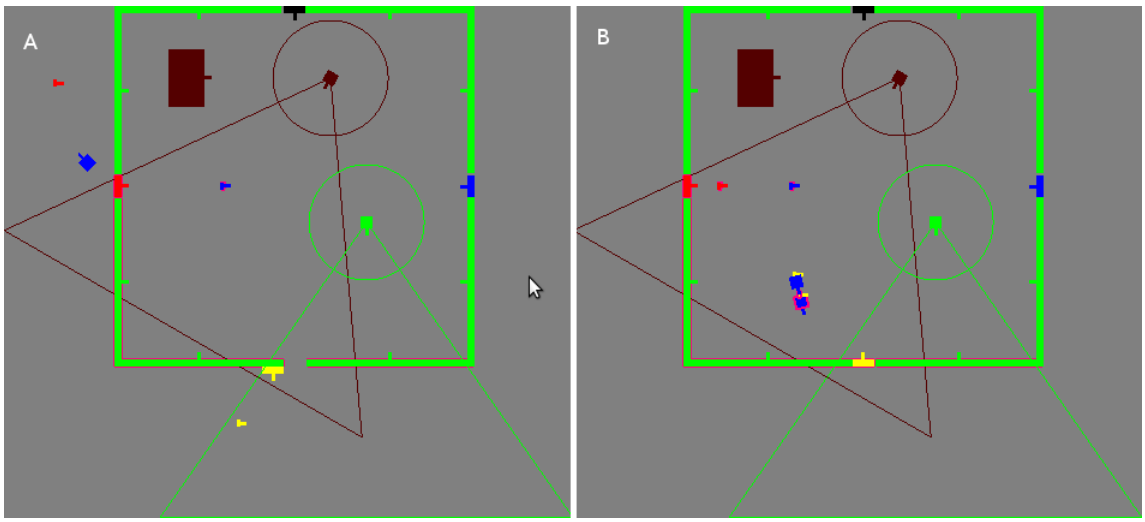

Figura 4.7. Organização do cenário em duas simulações.

Todos os cálculos de física e detecção de colisões são resolvidos pelo Engine de Física Box2D [12]. Esta ferramenta realiza, no simulador, todo o trabalho de manter em seus lugares elementos estáticos como paredes e obstáculos, além de reposicionar objetos móveis, de acordo com as forças a eles aplicadas, evitando inclusive que corpos de objetos que deveriam se colidir passem por cima uns dos outros. O Box2D também é responsável por gerenciar o trancamento das portas, através do uso de conexões ou *Joints*.

Cada porta possui uma dobradiça formada por uma *Joint* que a conecta à parede através de pontos nas extremidades de ambos os corpos (porta e parede) (na Figura 4.7 [A] percebe-se que a porta inferior foi destrancada e está totalmente aberta). Para representar uma porta trancada, foi utilizado uma conexão entre a porta e a parede oposta à conexão, utilizada como dobradiça. As conexões de dobradiça e tranca são diferentes. A de dobradiça limita os movimentos da porta em um ponto específico de rotação, enquanto que a conexão referente à tranca impede que a porta gire sobre a dobradiça. Para que uma porta seja aberta, é necessário destrancá-la com uma chave e então empurrá-la. Já em [B] é demonstrado a memória temporária do agente. O rastro deixado pela passagem de um avatar pelo FOV do agente indica que a memória do agente leva um tempo para ser atualizada com a última posição do elemento móvel. Isso serve para evitar uma sobrecarga no sistema. O tempo de atualização pode ser configurado através de uma parâmetro de configuração.

[12] http://www.box2d.org/

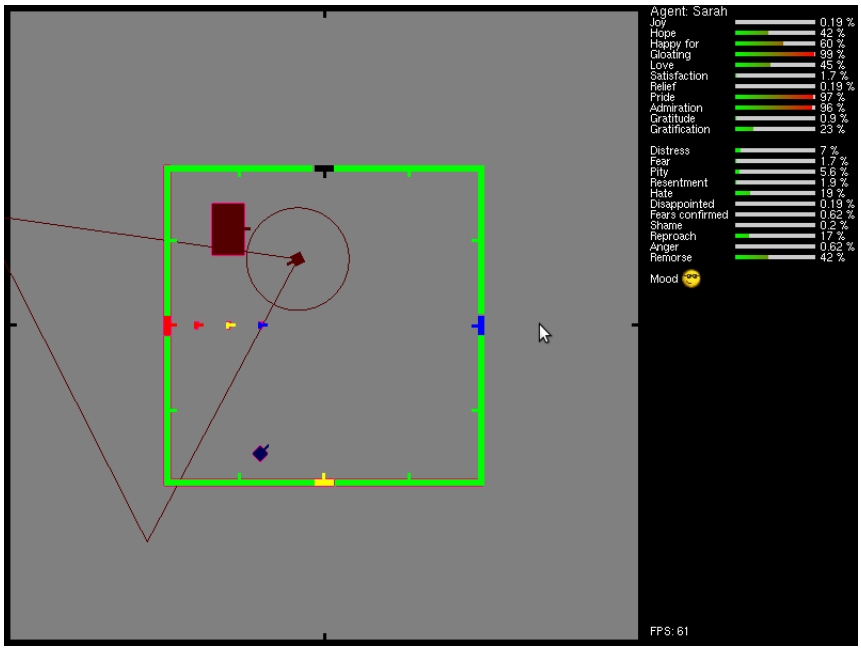

Figura 4.8. Interface do simulador 2D.

A Figura 4.8 apresenta a Interface Gráfica do simulador, que é composta pelo painel de exibição do estado do agente à direita e pelo cenário à esquerda. O agente possui um círculo ao seu redor, além de um cone projetado a partir da sua parte frontal. O círculo representa um sensor de audição e o cone um sensor de visão (FOV). Os três pequenos itens à frente do agente são chaves. A caixa à direita do agente é uma mesa. A entidade próxima à parede inferior, do lado de fora do FOV do agente, é o avatar controlado pelo usuário. A sala é fechada por paredes e portas que inicialmente estão todas trancadas. Cada chave abre somente uma porta. Cada elemento do cenário possui um pequeno retângulo que parte do centro do elemento para fora do seu corpo. Isto indica a sua orientação.

O simulador oferece uma operação básica de utilização de itens. Porém, a única operação deste tipo implementada no protótipo é a de destrancamento de portas usando uma chave. Para isto, o agente precisa mover-se até uma chave, executar uma operação de "Pegar item", depois deslocar-se até a porta correta e acionar a operação de "Usar item". A porta precisa estar fechada e trancada para esta operação ser bem sucedida.

Como mostrado na Figura 4.9, existe um painel de exibição do estado emocional do agente. O painel de estado do agente pode exibir as suas emoções ou os Resources ativos em sua memória. Esta imagem mostra as duas opções que podem ser exibidas no painel separadas por uma linha branca, porém somente uma delas pode ser exibida por vez. No caso da exibição dos Resources também são apresentados a intensidade de

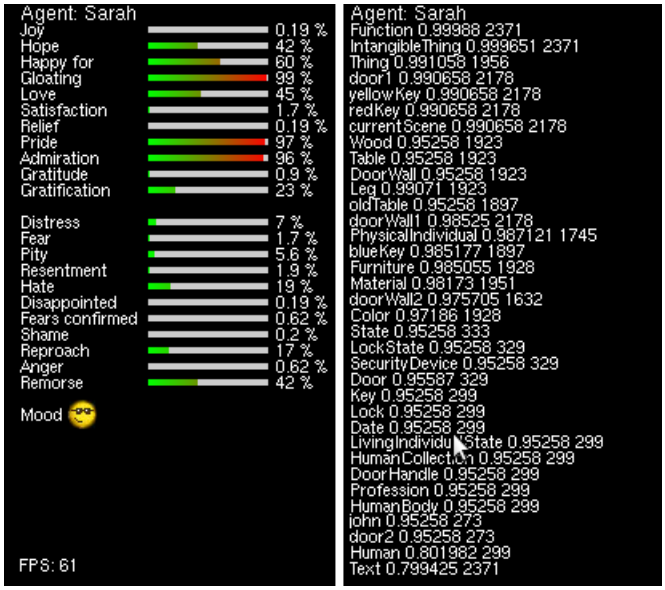

Figura 4.9. Painel para exibição de informações sobre o agente.

Figura 4.10. Emoticons que demonstram o humor do agente.

ativação do Resource e o *timestamp* da sua última atualização. No exemplo apresentado pela Figura estão relacionados os níveis (0-100%) vigentes das intensidades das emoções do agente. As emoções positivas vêm primeiro na listagem, seguidas pelas emoções negativas. Além destas barras de emoções, existe um *emoticon* que exibe o humor do agente durante a simulação. São sete níveis possíveis de humor, conforme exibidos na Figura 4.10. O humor pode variar de -1.0 (emoções negativas abaixo de 0) a 1.0 (emoções positivas acima de zero). O *emoticon* referente ao humor vigente (computado através da Equação 4.2, que será descrita na Seção 4.3) é apresentado na interface, seguindo os seguintes critérios:

1. Excelente humor (h > 0.8) e alegre: quando as emoções positivas Joy, Hope, Happy for, Gloating e Love são mais intensas que as demais positivas.

2. Excelente humor (h > 0.8) e confiante: quando as emoções positivas Satisfaction, Relief, Pride, Admiration, Gratitude e Gratificiation são mais intensas que as demais positivas.

3. Bom humor (h <= 0.8)

4. Humor neutro (= 0.0): emoções positivas igual às emoções negativas

5. Mal humor (-0.8 <= h < 0)

6. Péssimo humor (h < -0.8) e raivoso: quando as emoções negativas Disappointed, Fears confirmed, Shame, Reproach, Anger e Remorse são mais intensas que as demais negativas.

7. Péssimo humor (h < -0.8) e deprimido: quando as emoções negativas Distress, Fear, Pity, Resentment e Hate são mais intensas que as demais negativas.

Os itens de 1 a 7 correspondem aos *emoticons* da esquerda para a direita apresentados na Figura 4.10.

4.2.1 Gerenciador de cenas

Mesmo sendo apenas um protótipo, a aplicação desenvolvida é bastante complexa e adota uma arquitetura modular, exigindo um processo de integração eficiente. Dessa forma, conforme exibido na Figura 4.4, foi desenvolvido um módulo, encapsulado pela classe SceneManager, responsável por invocar todos os processos de carregamento da cena, realizar a preparação do ambiente, inicializar os agentes, além de realizar outras tarefas de gerenciamento dos dados tratados pela aplicação. Utilizando a classe PersistenceManager, o SceneManager carrega o arquivo, formatado em XML (usando para isto a biblioteca TinyXML [13], que é um parser XML bastante simples), descritor da cena, que possui uma referência para todos os elementos que devem ser nela carregados. Trechos dos arquivos manipulados pelo processo de carga do SceneManager podem ser visualizados no Apêndice B.

O PersistenceManager converte, durante a inicialização do sistema, todos os arquivos XML em uma estrutura de dados específica da aplicação (mantendo o formato

[13] http://www.grinninglizard.com/tinyxml/

de árvore) e a encapsula em uma interface bastante eficiente para a manipulação dos dados. Esta estrutura de dados fica então acessível à toda aplicação através do *Singleton* [Gamma et al., 1995] PersistenceManager.

O grafo de cena, que descreve os elementos do cenário em uma organização hierárquica, é utilizado para a construção dos Scene Frames, detalhados nas próximas seções.

4.3 Implementação do modelo proposto

Esta seção descreve de forma detalhada a etapa de implementação da parte do protótipo que diz respeito à mente do agente. Esta descrição foi realizada de forma técnica e detalhada, para facilitar o entendimento dos eventos ocorridos na aplicação durante a execução dos experimentos (apresentados no Capítulo 5).

A parte mais importante e mais complexa desenvolvida neste protótipo foi o sistema de Frames do modelo. Um sistema de Frames, como mencionado anteriormente, é muito semelhante a uma linguagem Orientada a Objetos. Logo, um interpretador para uma linguagem de Frames foi modelado e construído de forma a permitir que a massa de fragmentos de memória, que caracteriza toda a experiência do personagem interpretado pelo agente, pudesse ser modelada e representada de forma fácil e direta. O principal papel de um sistema de Frames é gerir uma hierarquia de tipos distintos de Frames, para que então instâncias destes tipos sejam utilizadas no processamento do sistema.

Conforme descrito na Seção 3, os Frames possuem Slots para guardar referências para dados diversos, incluindo referências para outros Frames. Além disso, Slots podem guardar trechos de códigos formatados como funções para serem executados em tempo real. Para que este segundo requisito fosse atendido, a mente possui uma instância de um ScriptManager, que é responsável pela execução deste tipo de procedimento. O trecho apresentado na listagem B.3 do Apêndice B é um exemplo de como os Frames são configurados pelo projetista da cena a ser simulada, utilizando um arquivo XML.

Neste protótipo, um Frame é representado por uma classe C++ que estende Resource (conforme apresentado na Figura 4.11). Esta classe pode assumir dois estados, tipo e instância. Um Frame "tipo" define a interface de um Frame, enquanto que a "instância" de um Frame é um objeto derivado de um Frame "tipo". Cada instância de um Frame possui parte de sua interface inicializada e gerenciada pelo ScriptManager, que é então disponibilizada ao ambiente da linguagem de programação Lua [14] (linguagem

[14] http://www.lua.org/

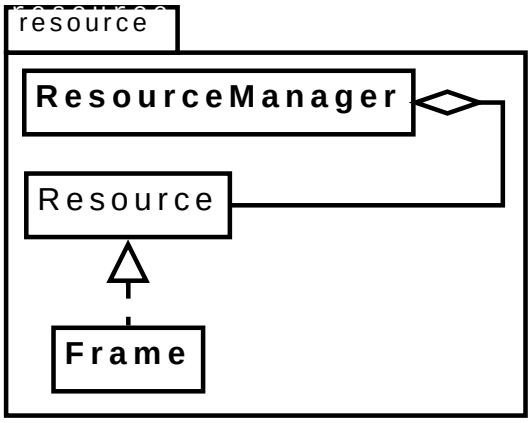

Figura 4.11. Diagrama de classes do pacote Resource.

de scripts utilizada pelo ScriptManager). O que o ScriptManager faz é disponibilizar o objeto que representa o Frame à Lua, além de criar atalhos para cada um de seus Slots, facilitando a programação dos Slots de funções.

Como um Frame é uma classe escrita em C++, é necessário a utilização de algum mecanismo para fazer a tradução entre o código escrito em C++ e os *scripts* em Lua. Para isso, foi utilizada a biblioteca Luabind [15], que realiza o mapeamento de estruturas de uma linguagem para a outra de forma bastante simplificada e direta, usando programação via *templates* [Stroustrup, 2000] para isto.

Durante a carga inicial do XML de memória do agente, os Frames são criados, inicializados e registrados no ResourceManager, ficando então disponíveis aos outros módulos da mente. A classe Mind 4.13 possui uma referência para cada um dos módulos que a compõe (Figura 3.1). Cada módulo, por sua vez, possui uma referência para o objeto instanciado da classe Mind. Assim, qualquer módulo pode acessar os demais, através da interface do objeto Mind. A classe AgentEntity (Figura 4.4), que representa o agente no mundo virtual, instancia o objeto Mind, chamando o seu método de atualização a cada ciclo de processamento do Simulador 2D.

Depois que a cena (ambiente, agentes, avatares, itens, visualizador 2D, etc.) é completamente carregada pelo SceneManager, um comando de inicialização da cena é dado pelo Simulador 2D ao SceneManager. Nesse momento, a aplicação entra em um laço sem fim pré-determinado que realiza, a cada iteração, um ciclo de processamento de toda a aplicação. Este laço só para quando um comando explícito de saída é dado pelo usuário ou quando um erro acontece.

Cada agente inteligente implementa a interface RunnableEntity (4.4), pois eles terão todo o seu ciclo de processamento gerenciados por uma *thread* separada e não

[15] http://luabind.sf.net/

pelo processo principal da aplicação durante a execução da cena. Conforme descrito no Capítulo 3, assim que o agente é carregado e passa a ser atualizado no ambiente virtual, o módulo ActionManager inicializa uma segunda *thread* que passa então a ter o seu processamento executando separadamente do laço principal do agente. Isso é necessário para que o sistema de controle do agente consiga continuar executando sem ser atrapalhado pela execução de uma ação que leva tempo para ser finalizada e vice-versa. Um outro motivo para se ter utilizado este formato de arquitetura é que, tornando o processamento do ActionManager independente do ciclo do agente, facilita o escalonamento de ações de acordo com os processos internos da mente. Ou seja, mesmo com a execução de uma ação em andamento o agente pode detectar uma situação de emergência e criar um objetivo de maior prioridade (pausando o objetivo vigente e executando o novo) para lidar com tal situação sem ter que esperar pelo fim desta ação.

Para o tratamento de erros, exceções são lançadas e tratadas em diversos pontos da aplicação usando para isto as classes do diagrama apresentado na Figura 4.12. Um sistema de registro de mensagens de erros com vários níveis de exibição de mensagens (erro, aviso, informação, etc.) apresenta os detalhes do que está acontecendo dentro da aplicação ao desenvolvedor/usuário (classes Logger e Message, mostrado na Figura 4.5) a medida que as exceções são lançadas ou quando são diretamente invocadas no código da aplicação.

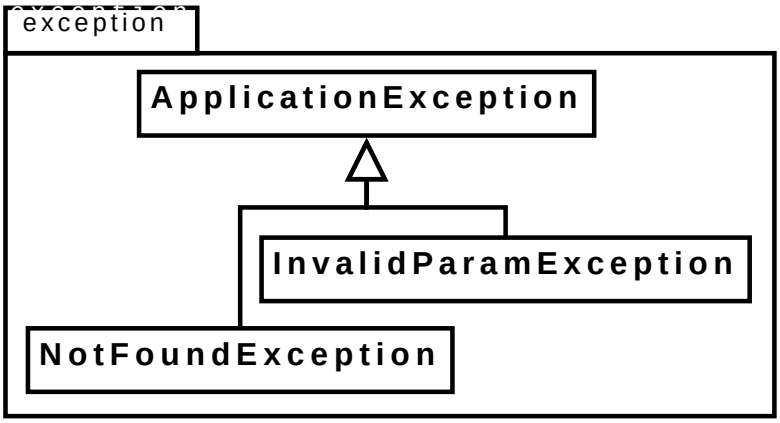

Figura 4.12. Diagrama de classes do pacote Exception.

O ciclo de execução da mente de um agente realiza a atualização dos módulos na seguinte ordem: ResourceManager, EmotionManager, ReasoningMode, GoalManager e ActionManager. Isto garante que o ReasoningMode, ao ser executado, terá disponível uma versão mais recente do estado emocional e dos Resources ativos, o que reflete de forma mais coerente o estado vigente do agente no ambiente virtual.

O ResourceManager, além de servir como um sistema de cadastro de Resources (somente do tipo Frame nesta implementação), faz o controle de ativação destes Resources. A cada chamada de sua rotina de atualização, o ResourceManager deduz um fator (customizável) de decaimento da intensidade vigente de cada Resource Ativo. Se a intensidade de um Resource fica negativa, este Resource é então excluído do ResourceManager, caso seja uma memória de curto prazo, ou apenas fica inativo, caso seja uma memória de longo prazo. Mas, um Resource somente é removido do ResourceManager caso não haja outros Resources que façam referência ao primeiro. Depois desta etapa, os Resources que permanecerem ativos têm suas intensidades propagadas pela Rede Semântica, conforme descrito no Capítulo 3.

Já o EmotionManager pode ser visto como um cadastro de sinais emocionais. Um sinal emocional, que é implementado através da classe EmotionSignal (Figura 4.13), é um operador que recebe como entrada um pulso emocional e utiliza uma função sigmoide parametrizada dada pela Equação 4.1 (retirada de [Ortony et al., 1988]) para representar a intensidade vigente de uma dada emoção do agente.

$$y = \frac{g}{1 + e^{-(x-x_0)/s}} + y_0 \tag{4.1}$$

onde y é a intensidade do sinal emocional; x é o estímulo externo desta emoção; x_0 move a curva na horizontal; s controla a inclinação da curva; y_0 move a curva na vertical; g é o ganho da curva.

Através do ajuste dos parâmetros do sinal emocional, um agente pode se tornar mais ou menos sensível a uma dada emoção, influenciando diretamente sua personalidade. O humor (Mood) é um tipo de sinal emocional diferenciado, ele é calculado utilizando-se uma função logarítmica, conforme a equação 4.2 (retirada de [Picard, 1997]).

$$I_e = log_2(\sum_e 2^{I_e}), e \in \{p \ \oplus \ n\} \tag{4.2}$$

onde I_e é o humor final, bom ou mal, dependendo do somatório de emoções assumido por e; e pode assumir dois valores, o grupo de emoções positivas (p) ou negativas(n).

Primeiro calcula-se as intensidades de bom e mal humor I_p e I_n. Então o humor final do agente é determinado da seguinte forma: se I_p for menor que I_n, o agente está de mal humor, caso contrário está de bom humor. A intensidade final do sinal emocional Mood terá como estímulo o sinal emocional vencedor, com a diferença que se o mal humor vencer, o sinal da entrada passa a ser negativo.

Uma das tarefas do ReasoningMode padrão, conforme apresentado no Algoritmo 2, é atualizar o EmotionManager com base nos Scene Frames ativos no momento. Cada Scene Frame possui sinais emocionais a ele vinculados, ou seja, uma cena guardada na memória do agente pode ser classificada emocionalmente. Os sinais emocionais de Scene Frames ativos são então coletados e utilizados como estímulos de entrada para as emoções do agente, utilizando-se o EmotionManager para isto. Mas antes desta etapa, as emoções vigentes mais intensas do agente fazem com que Scene Frames caracterizados por tais emoções sejam ativados, influenciando o seu processo cognitivo e, consequentemente, a escolha de novos objetivos.

O Algoritmo 2 (Capítulo 3) dá uma idéia geral de como funciona a implementação padrão do ReasoningMode. Esta implementação não faz parte do modelo, é apenas uma forma de demonstrar o funcionamento deste elemento da mente. Praticamente todas as operações realizadas por esta classe baseiam-se nas intensidades dos Resources ativos. Toda vez que seu método de atualização é acionado, o algoritmo de reconhecimento da situação vigente é executado. Se uma situação ruim ou crítica for detectada, o GoalManager é consultado para saber se tal situação já não está sendo tratada. Caso esta situação já estiver sendo tratada pelo GoalManager, ela é ignorada e a etapa de reconhecimento da situação é finalizada. Caso contrário, um novo *goal*, com prioridade maior que o *goal* que estiver em execução é adicionado, fazendo com que o agente pare o que estiver fazendo naquele momento e tome uma providência em relação à situação ruim detectada (gerar um novo *goal*). O novo *goal*, neste caso, é gerado a partir da ação que sucede o Scene Frame, que descreve a situação ruim detectada, no Story Frame que contém tal Scene Frame (Processo descrito no Capítulo 3).

$$s = \sum_{i} i/n \tag{4.3}$$

onde s é a intensidade final do Scene Frame; i é a intensidade de todos os Resources ativos que estão presentes na cena e n é o número de elementos da cena.

Uma situação é detectada utilizando-se um processo de analogia, baseada na comparação da intensidade do Scene Frame (Equações 4.3 e 3.1) vigente com a intensidade dos Scene Frames ativos (via emoções ou relacionamento com outros Frames). O Scene Frame vigente não possui classificação emocional nem do tipo da situação, pois ele é um Frame atualizado constantemente e reflete o que o agente está percebendo no momento mais recente da simulação sem uma análise crítica. Por isso a necessidade

desta etapa, que utiliza de um Scene Frame armazenado previamente na memória do agente para realizar o reconhecido da situação vigente.

As ações executadas pelos agentes são representadas por Action Frames. Este Frame possui um Slot construtor que recebe como parâmetros a(s) entidade(s) envolvida(s) na ação. O Slot construtor (que deve possuir o mesmo nome do Frame) é invocado em todos os objetos, instanciados a partir de um Frame "tipo" que define tal Slot, logo após a sua criação. Para facilitar o entendimento de como o Action Frame é definido, a listagem B.3, apresentada no Apêndice B, exibe a construção da ação Grab, utilizada pelo agente para pegar itens próximos a ele. Um Action Frame deve conter dois Slots básicos, "preconditions" e "update". O primeiro deve conter a lista de situações que devem ser satisfeitas para que tal ação possa ser executada pelo agente. Já o segundo Slot deve implementar um procedimento que será invocado pelo ActionManager a cada ciclo de atualização até que a ação seja completada (falha ou sucesso). Uma pré-condição (precondition) de uma ação é uma asserção comum (Assertion Frame), que deve ser satisfeita para permitir a execução da ação. Dois tipos de asserção bastante utilizado como pré-condição nos experimentos são relações unárias ou binárias (descritas pelos UnaryRelation Frames e BinaryRelation Frames que herdam de Relation que por sua vez herda de Assertion).

O GoalManager recebe do ReasoningMode vigente um Action Frame (já inicializado com os parâmetros que configuram a ação) como um novo objetivo a ser alcançado. A partir desse momento o GoalManager avalia se tal objetivo pode ser alcançado, verificando se as pré-condições da ação estão satisfeitas. Durante a varredura e teste das pré-condições, se uma asserção comum não satisfeita for encontrada, a ação é dada como já realizada e o objetivo é então marcado como "alcançado com sucesso". Isto porque uma pré-condição do tipo Assertion é justamente o foco da ação, por exemplo, no caso da ação "Abrir porta" uma pré-condição asserção seria $Porta.estado = fechado$. Logo, se o estado da porta é $Porta.estado = aberto$, o agente conclui então que a ação "Abrir porta" já foi executada, mesmo que por outro agente, determinando que não há mais a necessidade de executá-la. No caso de uma pré-condição do tipo Relation, o tratamento é um pouco diferente. O GoalManager realiza, quando encontra uma Relation não satisfeita um processo de cascata, desmembrando o *goal* vigente em *sub-goals*. Cada Frame do tipo Relation possui um Slot que relaciona as ações que podem satisfazê-la. Com esta dica, o GoalManager gera *sub-goals* utilizando para isto a(s) ações(ão) encontrada(s) neste Slot. Um *sub-goal* é um *goal* com vínculo explícito de subordinação a um outro *goal* e para que o *goal* pai seja alcançado, todos os seus *sub-goals* precisam ser alcançados. Dessa maneira o GoalManager realiza a construção de planos de ações para o agente durante todo o seu ciclo de vida.

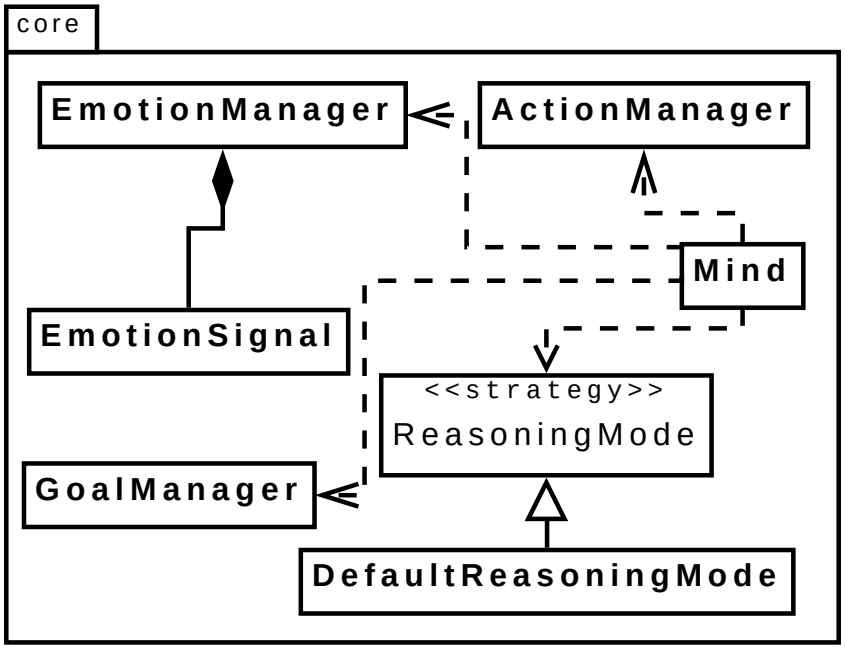

Figura 4.13. Diagrama de classes do pacote Core.

Quando um *goal* é considerado apto a ser perseguido, com todas as pré-condições da ação que o realiza satisfeitas, essa ação é então enviada à instância da classe ActionManager (Figura 4.13) para execução. O ActionManager possui uma fila de ações a serem executadas, ordenadas pela prioridade dos *goals* correspondentes a elas. O objetivo do ActionManager é eliminar todas as ações desta lista, atualizando-as até o seu sucesso ou fracasso. Esta classe invoca o Slot *update* da ação situada na primeira posição desta fila a cada ciclo de atualização (realizado pelo processo leve gerenciado pela *thread* separada deste módulo). O procedimento escrito em Lua neste Slot deve realizar operações simples, sem prender o processamento em laços longos. Se uma nova ação, advinda de um *goal* com maior prioridade da que está sendo executada pelo ActionManager é adicionada à fila e, no próximo ciclo, a ação atual é paralisada e a nova passa a ser executada. Ao final da execução desta última, a ação paralisada anteriormente volta a ser executada.

Quando uma ação em execução no ActionManager falha, o GoalManager é notificado. Em seguida ele cria uma instância do Failure Frame, que representa a falha da execução de uma ação em um dado contexto, além de conter uma sugestão para o seu motivo. Então este Frame é ativado com uma intensidade alta. No próximo ciclo de execução do ReasoningMode, uma regra para o tratamento de falhas percorre os Resources ativos e detecta que uma falha ocorreu, podendo realizar o seu tratamento utilizando regras normais ou realizando o chaveamento para outro ReasoningMode (op-

ção mais adequada para lidar com situações complexas que exigem uma forma diferente de raciocínio).

Se o agente não estiver perseguindo um *goal* e uma situação neutra for reconhecida como a vigente, então o ReasoningMode procura, no Story Frame daquele Scene Frame detectado, por uma situação boa. Se esta for encontrada, então um novo ciclo de resolução de *goals* é realizado pelo GoalManager para tentar deixar o agente em uma situação melhor. Esta abordagem otimista, de sempre tentar deixar o agente em uma situação boa, foi utilizada na construção do protótipo, mas é apenas uma forma de abordar os problemas enfrentados pelo agente.

Observa-se na Figura 4.13 que, além das dependências entre a classe *Mind* e as demais, dentro do pacote **core**, Mind possui algumas dependências com as classes dos pacotes **tools** e **resource**. Por isso, no diagrama da Figura 4.2 existem relacionamentos entre **core**/**tools** e **core**/**resource**. A classe abstrata ReasoningMode foi desenhada utilizando-se o padrão de desenho Strategy, conforme estereótipo marcado no diagrama.

Serão apresentados no próximo capítulo exemplos da utilização do sistema para facilitar o entendimento quanto ao funcionamento do protótipo.

Capítulo 5

Experimentos

O objetivo deste capítulo é descrever os experimentos realizados no protótipo implementado e apresentar os resultados juntamente com uma análise de cada um dos testes. A metodologia utilizada para a experimentação do protótipo compreende três itens: a) a utilização do modelo, para realizar a simulação e comparar os processos de raciocínio do agente que fazem uso de ativações de Resources; b) a análise dos comportamentos planejados pelos agente para verificar se o modelo é válido e produz uma saída coerente (saída esta que pareça sensata e não seja "estúpida" ao ser julgada por um usuário humano) e c) a validação das pré-condições das ações que compõem o comportamento do agente para verificar se o plano de ações gerado pelo agente está correto.

Em todos os experimentos as propriedades de taxa de decaimento da intensidade dos Resources (para o processo de esquecimento) foi mantida em 0.015 pontos por segundo. Este valor significa que um Resource ativo com uma intensidade de 1.0, se tornará inativo após aproximadamente 66 segundos, caso ele não receba estímulo algum de novas ativações durante este período. A quantidade máxima de Resources ativos foi configurada para 50. O número máximo de atualizações dos sensores por segundo é 4, ou seja, a atualização dos sensores ocorre somente em intervalos de 250ms.

Uma base de 720 Frames foi carregada pelo ResourceManager do agente para a realização dos experimentos. Essa base de Frames foi construída adaptando-se parte do banco de dados de conhecimento de senso comum do projeto OpenCYC [1] e utilizando diversas definições de Frames da FrameNet [2]. Obviamente, como poderá ser visto nos experimentos realizados, apenas uma pequena parte destes Frames foi acessada. Mas esta quantidade de Frames previamente carregada serviu para certificar que os processos realizados pela implementação do modelo independem do número de Frames,

[1] http://http://www.opencyc.org/
[2] http://framenet.icsi.berkeley.edu/

mantendo o mesmo tempo de resposta quando deixados no ResourceManager somente os Frames relacionados com experimentos e as mesmas propriedades de configuração da aplicação.

Quatro experimentos foram realizados e seus resultados apresentados neste capítulo. O primeiro avalia o processo de ativação de Resources, dado o contexto em que o agente se encontra (normalmente detectado por seus sensores e sinais emocionais). O segundo verifica se, após a contextualização feita pela ativação dos Resources, o agente é capaz de reconhecer a situação vigente utilizando suas experiências passadas armazenadas em sua memória para isto. O terceiro avalia a interferência das emoções no processo de tomada de decisões e o último experimento, realizando um teste de integração, avalia a capacidade de resolução de problemas do agente através da criação de planos de ações.

5.1 Ativação de resources por contexto

Este experimento visa validar o processo de ativação dos Resources, baseado nas percepções do agente e em seu objetivo vigente. À medida que o FOV do agente captura as entidades presentes no mapa (cena representada no ambiente virtual), os Frames correspondentes a estas entidades são ativados e então as suas intensidades de ativação são propagadas para Frames a eles relacionados. Se uma entidade está dentro da área de cobertura do FOV ela é constantemente re-ativada com uma intensidade que varia de acordo com a posição no FOV que tal entidade se encontra. Assim sendo, o objetivo deste experimento é verificar se a variação da posição e orientação do agente, que também refletirá na seleção de entidades percebidas, está ativando os Resources de maneira correta. Para isto, duas situações foram utilizadas como referências para os testes.

Na primeira situação o agente não possui objetivos em andamento, observa a porta da esquerda e depois a porta da direita, conforme apresentado nas situações A e B da Figura 5.1, respectivamente.

As listagens de Resources apresentados nas Tabelas 5.1 e 5.2 foram coletadas do protótipo durante a execução das situações demonstradas na Figura 5.1. Os Resources que são instâncias de Frames possuem o nome de seus tipos entre parênteses na frente de seus próprios nomes. Instâncias de Frames, em ambas as listagens de Resources representam objetos presentes no ambiente percebidos pelo agente. Os Resources são ordenados por três critérios: sua intensidade de ativação, *timestamp* (tempo de execução da aplicação em que a intensidade foi atribuída ao Resource) e por último seu

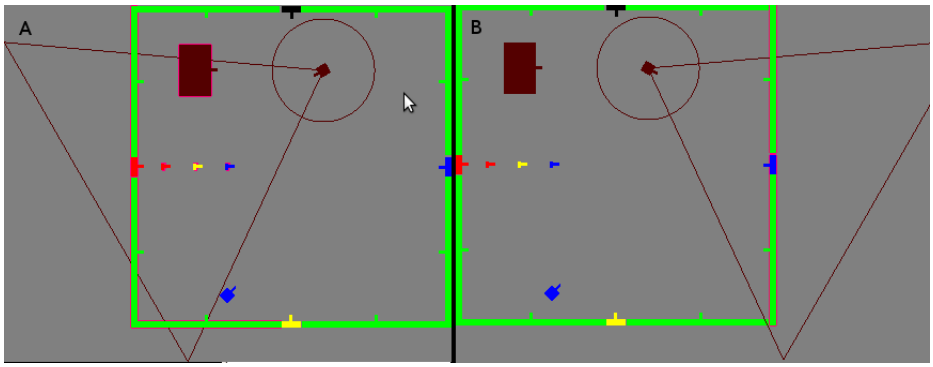

Figura 5.1. O agente observa as portas da esquerda e direita nas situações A e B, respectivamente.

nome (utilizando uma simples comparação lexicográfica).

Tabela 5.1. Resources ativos na situação A da Figura 5.1.

Nome	*Intensidade*	*Timestamp*
Thing	0.999645	11345
IntangibleThing	0.999644	11345
upperLeftDoorWall(DoorWall)	0.997732	11321
leftDoor(Door)	0.994169	11321
yellowKey(Key)	0.994169	11321
redKey(Key)	0.994169	11321
blueKey(Key)	0.994169	11321
currentScene(Scene)	0.994169	11321
oldTable(Table)	0.994169	11321
State	0.773100	542
SecurityDevice	0.733000	308
Lock	0.772580	298
Door	0.474468	308
...	...	...
DoorWall	0.454440	308

Pode-se perceber que em ambos os casos os Frames ativos são justamente as instâncias de Frames referentes aos objetos percebidos pelo FOV do agente, além dos Frames que são relacionados de alguma forma com estes objetos. Ou seja, no segundo caso, Frames relacionados com os ativados pelo FOV podem conter referências para um Frame específico, fazendo com que este tenha sua intensidade de ativação mais forte que os demais. Por exemplo, na tabela 5.1 leftDoor é um objeto do tipo Door, que referencia Key através de seus Slots, e como o objeto redKey é do tipo Key então ele recebe estímulos de ativação tanto do FOV (por ser visível) quanto da instância leftDoor.

Tabela 5.2. Resources ativos na situação B da Figura 5.1.

Nome	*Intensidade*	*Timestamp*
Wall	0.999880	5285
Thing	0.999875	5285
Individual	0.999680	5285
IntangibleThing	0.999680	5285
DoorWall	0.994761	5140
upperRightWall(DoorWall)	0.990695	5110
rightDoor(Door)	0.990694	5110
currentScene(Scene)	0.990694	5110
Door	0.896292	639
LockState	0.893800	303
Container	0.893800	303
SecurityDevice	0.893800	303
Lock	0.893800	289
...	...	...
DoorHandle	0.893800	289

Na segunda parte deste experimento, foi atribuído ao agente o objetivo de abrir a porta da esquerda (leftDoor). Ele tem a informação de que a porta está trancada e precisará pegar a chave perto da porta da esquerda (redKey) para abrí-la. Enquanto ele olha a porta da esquerda, a lista dos Resources ativos é parecida à do caso *A* da primeira parte do experimento, mas, quando ele olha para a porta da direita, percebe-se que os Resources ativos relativos ao seu objetivo vigente permaneceram no topo da lista. Este comportamento era esperado, pois o GoalManager reforça a intensidade dos Resources relacionados com o *goal* vigente, justamente para fazer com que o agente mantenha o foco, o que valida o primeiro experimento realizado com o protótipo.

As tabelas 5.3 e 5.4 relacionam os Resources ativos na segunda parte do experimento, onde o agente possui um objetivo a alcançar e, portanto, os Resources relacionados a esse *goal* permanecem no topo da lista de ativação, independente para onde o agente direciona sua visão.

5.2 Reconhecimento de situação

Este experimento foi realizado para validar o processo de reconhecimento de situações pelo agente, ou seja, dada a configuração do ambiente percebido pelo agente em um dado momento, o agente deve ser capaz de utilizar Frames do tipo Scene, armazenados previamente em sua memória, para detectar o tipo da situação em que ele se encontra.

Tabela 5.3. Resources ativos na situação A da Figura 5.1, em que o agente possui um objetivo a cumprir.

Nome	*Intensidade*	*Timestamp*
Unlock	0.999632	4122
redKey(Key)	0.994169	4122
leftDoor(Door)	0.994155	4122
Open	0.992910	4433
Thing	0.988645	4433
IntangibleThing	0.988644	4433
upperLeftDoorWall(DoorWall)	0.997732	4122
Lock	0.987654	4122
yellowKey(Key)	0.984169	4122
DoorState	0.984108	4122
Door	0.984180	4122
blueKey(Key)	0.984069	4122
...	...	...
Set	0.703380	220

Este teste foi realizado em duas etapas. Na primeira o agente constrói a cena vigente (usando o Scene Frame), percebida pelos seus sensores, e realiza a ativação/propagação dos Resources referentes à cena. Na segunda etapa o agente coleta os Scene Frames ativos e realiza a cálculo de seus pesos, usando a equação 3.1 implementada no ReasoningMode padrão (DefaultReasoningMode, Figura 4.13).

A Tabela 5.5 contém os Resources utilizados no processo de reconhecimento de situações deste experimento. Os Frames que não são instâncias (Frames de "tipo") foram omitidos para facilitar a visualização da tabela. O Frame currentScene(Scene) possui uma referência para todos os elementos presentes na cena, que foram percebidos pelo agente até o momento. Este Frame é utilizado na computação da intensidade da cena vigente, que é então utilizada na comparação com outras cenas. Os demais Scene Frames foram ativados através da propagação de intensidade pela rede semântica.

Foram adicionados à memória do agente alguns Frames para tornar possível a realização deste experimento. Os Frames criados e utilizados no experimento são apresentados nas Figuras 5.2, 5.3 e 5.4. As caixas de cantos arredondados representam os Scene Frames antes e depois da ação. Cada Scene Frame possui um nome, uma descrição de situação e uma lista de Frames estáticos no momento que ele foi registrado. Instância de Frames começam com letra minúscula e Frames de tipo com letras maiúsculas, além de utilizarem parênteses na sua inicialização. A ação responsável pela transição entre as duas cenas é representada pelo losango. As setas indicam o fluxo do evento.

Tabela 5.4. Resources ativos na situação B da Figura 5.1, em que o agente possui um objetivo a cumprir.

Nome	*Intensidade*	*Timestamp*
Unlock	0.999712	3320
redKey(Key)	0.996182	3320
leftDoor(Door)	0.996182	3320
Open	0.992910	3320
Thing	0.988645	2199
LockState	0.885100	2199
Container	0.885100	2199
SecurityDevice	0.885100	2199
Lock	0.883654	4111
upperRightWall(DoorWall)	0.880695	4111
rightDoor(Door)	0.870694	109
upperLeftDoorWall(DoorWall)	0.797732	109
yellowKey(Key)	0.777169	2199
...	...	...
Scene	0.771800	109

Tabela 5.5. Resources ativos na situação B da Figura 5.1, durante o reconhecimento da cena.

Nome	*Intensidade*	*Timestamp*
redKey(Key)	0.996182	201
leftDoor(Door)	0.996182	201
upperRightWall(DoorWall)	0.880695	120
rightDoor(Door)	0.870694	120
upperLeftDoorWall(DoorWall)	0.797732	120
currentScene(Scene)	0.777694	16
yellowKey(Key)	0.777169	221
paula(Human)	0.766187	220
...	...	...
paulaAwakened(Scene)	0.07910	120
sleepingPaula(Scene)	0.07910	120
trappedPaula(Scene)	0.06930	120
fidoBarkedToPaula(Scene)	0.06930	120

Dada a contextualização da situação vigente em que o agente se encontra, seguem abaixo os valores das intensidades de similaridade entre os demais Scene Frames e o Frame *currentScene* (computado através dos valores dos Frames apresentados na Tabela 5.5), coletados na saída do algoritmo de reconhecimento de situação:

Analisando os dados da tabela 5.6 pode-se perceber que o Scene Frame reconhe-

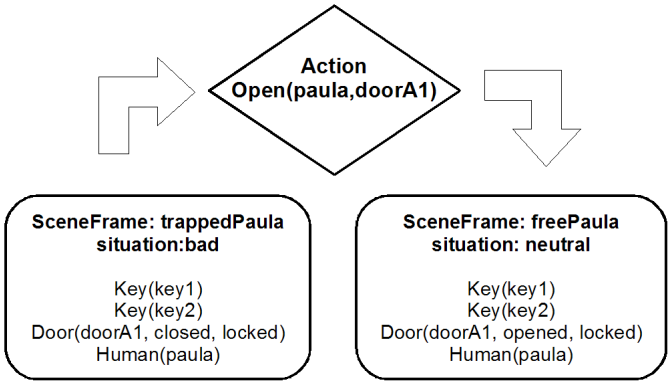

Figura 5.2. Paula abre a porta da sala.

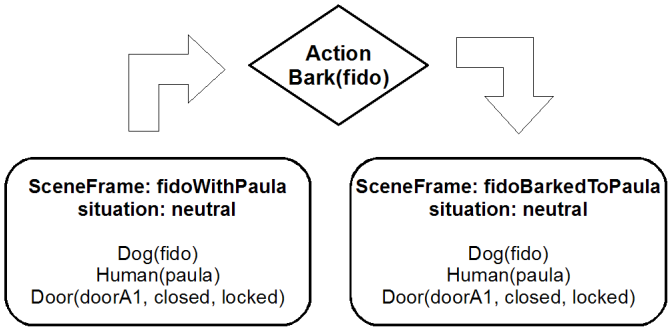

Figura 5.3. Fido late para Paula.

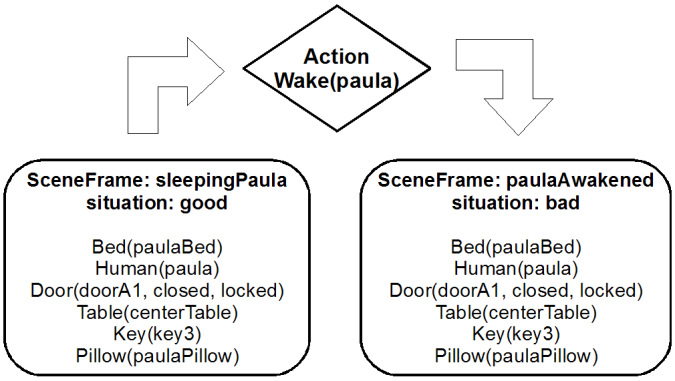

Figura 5.4. Paula acorda.

cido como a situação vigente é o *trappedPaula*, pois sua intensidade de similaridade é maior que o valor computado para o Frame *currentScene*, além de superar as demais intensidades. Isso significa que o agente identificou a situação vigente como sendo ruim, pois *trappedPaula* possui esta classificação. Se por acaso a intensidade das cenas forem menores que a da currentScene, então o agente não reconheceu a cena vigente, pois nenhuma de suas lembranças se aproxima do que ele está vendo no momento. O resultado

Tabela 5.6. Intensidade da similaridade com a cena vigente.

Scene Frame	*Intensidade*
paulaAwakened	0.334
sleepingPaula	0.334
trappedPaula	**0.920**
fidoBarkedToPaula	0.587
currentScene	0.869

final deste experimento se mostrou correto, uma vez que o Scene Frame identificado é completamente formado por elementos que estão presentes na cena vigente. Este reconhecimento permite então que o agente tome suas decisões baseados em situações familiares a ele e, portanto, de forma mais coesa. Estes resultados validam o segundo experimento realizado no protótipo.

5.3 Impacto emocional na tomada de decisão

Conforme apresentado no Capítulo 4, o módulo EmotionManager gerencia sinais emocionais que influenciam o processo de ativação de Resources. Essa influência ocorre devido ao incremento da intensidade de ativação de Scene Frames relacionados com as emoções mais fortes no momento da computação das intensidades dos Resources. Além disso, Scene Frames ativos podem alterar o estado emocional do agente, dependendo de suas intensidades. Esse é um processo de retro alimentação onde o EmotionManager reforça a intensidade de determinados Frames e os Scenes Frames ativos alteram as emoções vigentes, utilizando as emoções vinculadas ao Scene Frame. A alteração nas emoções utilizando-se Scenes Frames somente ocorre se a ativação de um dado Frame for significativa (maior que um limiar específico customizável). No caso destes testes o valor de ativação do Scene Frame tem que ser maior que 0.7. É importante ressaltar que a taxa de decaimento de intensidade de Frames e de emoções deve ser regulada adequadamente para evitar que um Scene Frame permaneça ativo por muito tempo, uma vez que as emoções a ele vinculadas o mantém ativo via retro alimentação.

O cenário deste experimento se configura da seguinte forma:

- O agente está perseguindo o objetivo "Open(leftDoor)".
- Um avatar, presente na cena, aciona um comando programado especificamente para simular uma fala com o agente, informando-o que ele ganhou na loteria. Este comando, insere na memória de curto prazo do agente um Trans Frame, protagonizado pelo agente, que representa a ação de ganhar dinheiro na loteria.

- Nesse momento, o Scene Frame de possuir dinheiro ganho na loteria (Frame posterior à ação do Trans Frame) se torna ativo com alta intensidade. Este Frame é vinculado às emoções Happy-for, Pride e Gratification.

- O agente passa a ter vontade de comemorar e adiciona um *goal* para a ação Jump (no caso, pular de alegria).

- Enquanto o agente está executando a ação de comemoração, o avatar aciona um segundo comando que insere na memória do avatar Frames que representam uma nova fala, dizendo que o que foi dito anteriormente é mentira.

  - O Trans inserido anteriormente é então marcado como falso.

- As emoções positivas do agente são zeradas e as seguintes emoções são intensificadas: Resentment, Hate e Anger.

- O agente passa a ter vontade de agredir o avatar, cancela o *goal* anterior e adiciona um novo *goal* "Hit(avatar)".

Para a realização deste experimento algumas regras foram criadas no Reasoning-Mode padrão, além da definição de alguns Frames específicos. O primeiro Frame adicional é de desejo (Desire Frame). Inspirado na arquitetura BDI (Belief-Desire-Intention) [Bratman, 1987], os Frames armazenados na memória do agente, de um modo geral, representam o que o agente acredita (Belief), o Desire Frame, uma vez ativo, dependendo do estado emocional do agente pode se tornar um *goal* (Intention). Um dos Slots do Desire Frame contém a ação a ser transformada em *goal*, caso executado. Um outro Slot deste Frame relaciona as intensidades das emoções a ele vinculadas. Uma regra específica para o tratamento de Desire Frames foi implementada. Uma vez que um Desire Frame estiver ativo, esta regra verifica se as intensidades das emoções vinculadas ao Frame são menores que as intensidades destas mesmas emoções no EmotionManager e gera o *goal* caso verdadeiro.

Os outros Frames definidos para a realização deste experimento são relacionados às estruturas que correspondem às falas. De forma bastante rudimentar, dado que a maneira desejável seria a incorporação de um sub-sistema para realizar o processamento de linguagem natural, o Frame HasSaid referencia o avatar ou agente que disse algo com o Trans Frame que representa o que foi dito. E o Frame Lie indica que um Trans Frame não é verdade. Foi criada uma regra para realizar o tratamento de uma mentira, que assim que um Frame Lie é ativado, a regra identifica o mentiroso e cria um Desire Frame de vingança, com as emoções Hate e Anger a ele vinculadas.

Segue abaixo a lista de eventos resultantes da execução deste experimento, que foi executado no mesmo cenário apresentado na Figura 4.8. Utilizou-se a mesma nomenclatura adotada no segundo experimento para a representação dos Frames. Esta sequência de eventos também está representada em diagramas (Figuras 5.5, 5.6, 5.7, 5.8, 5.9 e 5.10), para facilitar o entendimento do que acontece dentro dos módulos de controle.

1. Goal agendado: OpenDoor(sarah, leftDoor) [prioridade 99]
   i Pré-condições do goal: Near(sarah, leftDoor) == false, Equals(leftDoor.state, Unlocked) == true
   ii Subgoal em execução: Move(sarah, leftDoor) [prioridade 98] para resolver Near(sarah, leftDoor)
2. Avatar conta a mentira: Trans(Scene(currentScene), Win(Prize(Lottery(), Number(1000000))), Scene(Has(sarah, Money(1000000))))
3. HasSaid(john, Trans(...)) é criado. O Trans Frame é o mesmo descrito no item anterior
   i O terceiro parâmetro do Trans Frame acima possui a ele vinculado as seguintes emoções: Happy-for(0.8), Pride(0.6), Gratification(0.6)
4. Os Frames descritos acima se tornam ativos
5. As emoções do terceiro parâmetro do Trans Frame são atualizadas no EmotionManager
6. Regra detecta que há um pico de emoções positivas e identifica que Happy-for é a mais alta
   i O Desire Frame(Jump(), EmotionCollection(Happy-for(0.5))) é criado e ativado
7. Regra detecta que há um Desire Frame ativo e que a emoção vigente é maior que a do Desire
   i Goal agendado: Jump(sarah) [prioridade 97]
   ii Subgoal paralisado: Move(sarah, leftDoor)
8. Goal em execução: Jump(sarah)

9. Avatar desmente o que disse: Lie(Trans(...))
10. Regra detecta que uma mentira está ativa
    i Faz um decaimento de 60% (parâmetro definido empiricamente na regra) de todas as emoções positivas
    ii Registra as emoções no EmotionManager: Resentment(0.6), Hate(0.8), Anger(0.9)
    iii Recupera a origem da informação: john
    iv Novo desejo criado: Desire(Hit(john), EmotionCollection(Hate(0.5), Anger(0.6)))
11. Regra detecta que há um Desire Frame ativo e que a emoção vigente é maior que a do Desire
    i Subgoal interrompido: Jump(sarah)
    ii Goal agendado: Hit(sarah, john) [prioridade 97]
    iii Pré-condições: Near(sarah, john) == false
    iv Subgoal em execução: Move(sarah, john) [prioridade 96]
    v Subgoal finalizado: Move(sarah, john)
12. Goal em execução: Hit(sarah, john)
13. Goal em finalizado: Hit(sarah, john)
14. Goal re-iniciado: Move(sarah, leftDoor)
15. Goal finalizado: Move(sarah, leftDoor)
16. Goal em execução: OpenDoor(sarah, leftDoor)
17. Goal finalizado: OpenDoor(sarah, leftDoor)

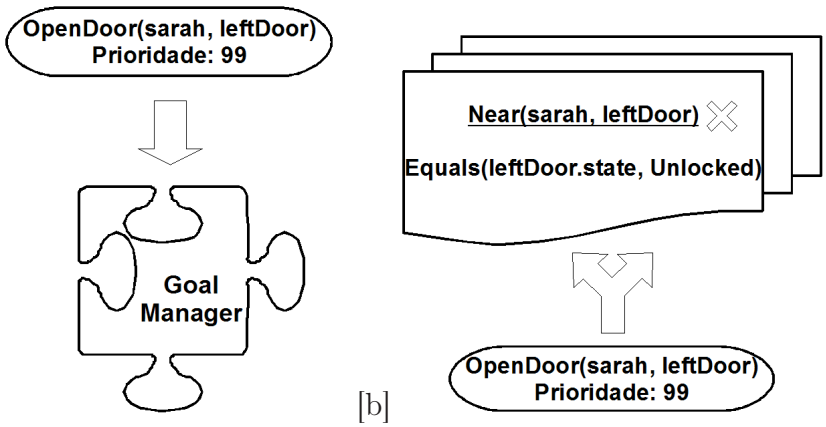

Figura 5.5. a) Goal Manager agenda novo objetivo. b) Pré-requisitos do objetivo são avaliados.

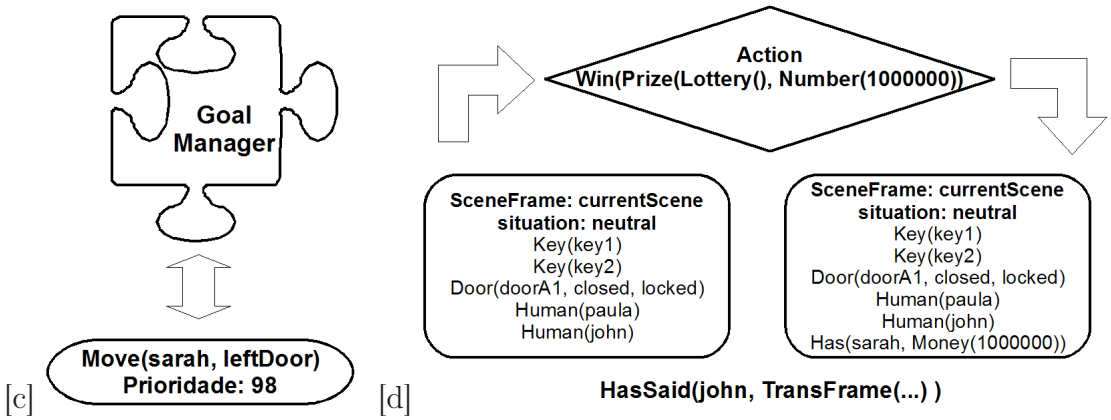

Figura 5.6. c) Goal Manager agenda sub-objetivo. d) Trans Frame é adicionado à memória do agente.

Onde os principais Frames são definidos por: **sarah**="Frame do agente"; **leftDoor**="Frame que representa a porta a ser aberta" ; **john**="Frame que representa o avatar" ; **OpenDoor(<"Frame do agente">,<"Frame da porta">)**="ação em que um agente abre uma porta" ; **Move(<"Frame do agente">,<"Frame de destino">)**="ação em que o agente se move para perto de um alvo" ; **Equals(<"Frame 1">,<"Frame 2">)**="asserção que indica que o Frame 1 é idêntico ao Frame 2" ; **Trans(<"Scene Frame anterior">,<"Frame da ação">,<"Scene Frame posterior">)**="percepção do impacto de uma ação no ambiente".

Através do histórico de eventos ocorridos dentro do módulo mental do agente, o experimento demonstrou a influência do módulo gerenciador de emoções no processo de formulação de *goals* de forma satisfatória.

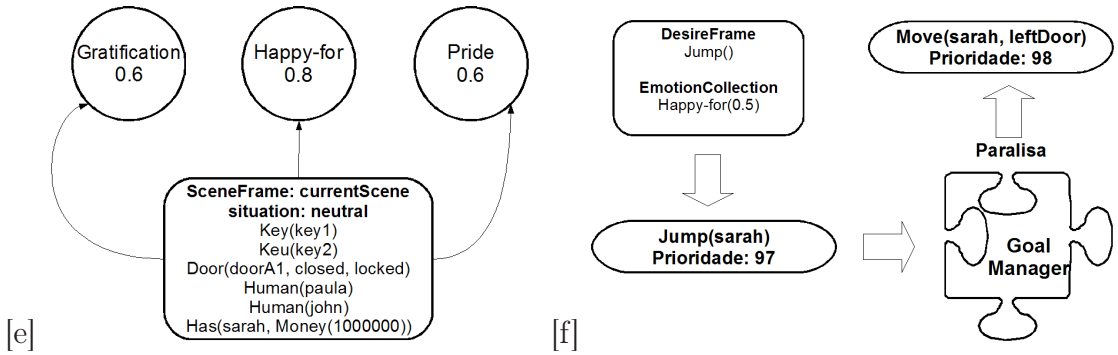

Figura 5.7. e) Emoções do agente são revisadas. f) Um desejo ligado às emoções emerge na mente do agente.

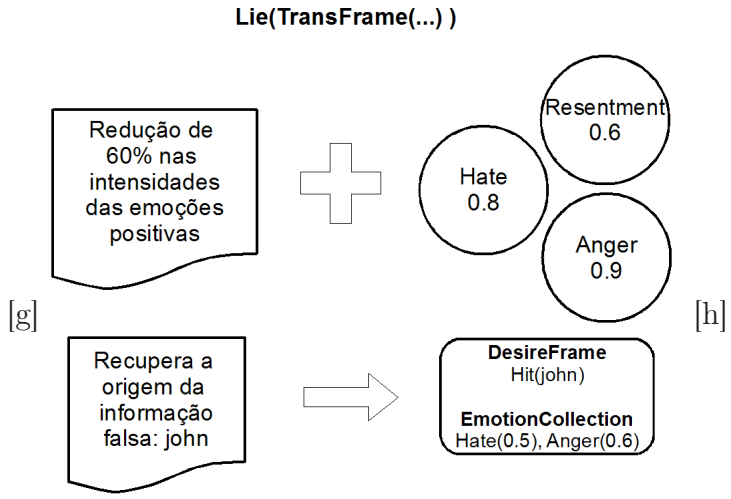

Figura 5.8. g) Um novo Trans Frame é adicionado e as emoções são revisadas novamente. h) Um novo desejo emerge.

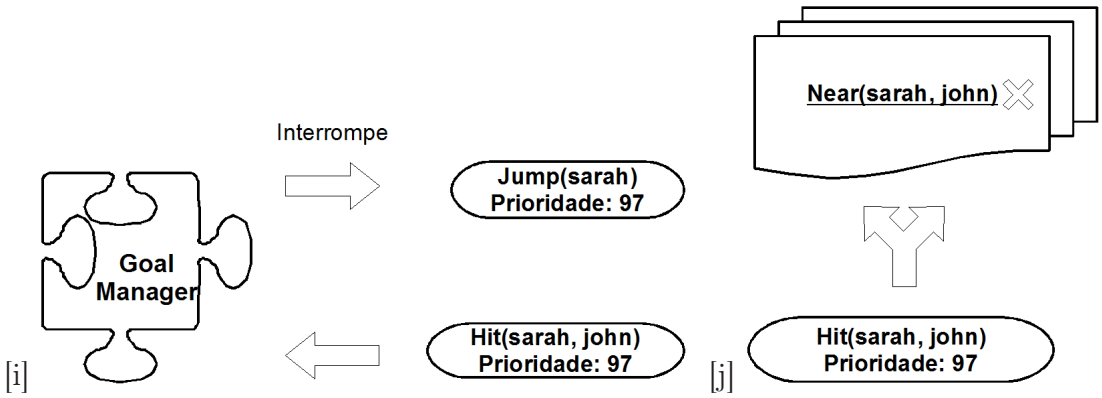

Figura 5.9. i) O Goal Manager prioriza outra ação. j) As Pré-condições da ação vigente são analisadas.

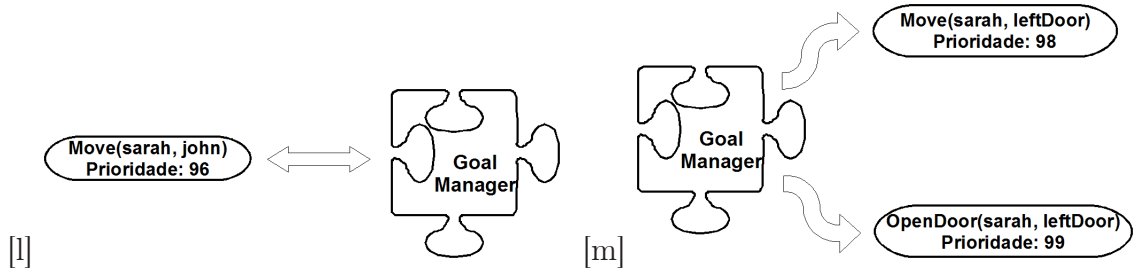

Figura 5.10. l) Uma nova ação entra em execução. m) O objetivo original do agente volta a ser perseguido.

5.4 Resolução de problemas

O objetivo do experimento detalhado nesta seção é demonstrar como o sistema se comporta durante a tentativa de resolução de um problema. O Resultado esperado é um plano contínuo de ações que leve o agente a alcançar o seu *goal* de longo prazo. A busca por um *goal* de curto ou longo prazo pode resultar em falha. Algumas falhas podem ocorrer durante a execução de *goals* de curto prazo. Porém, com o devido tratamento de falhas, o *goal* de longo prazo pode ser alcançado com sucesso.

O cenário e a posição inicial do agente são apresentados na Figura 5.1. O *goal* de longo prazo do agente é sair da sala. Para isto ele deverá simplesmente abrir e atravessar uma das portas que existem na sala. A sala é um Frame do tipo contêiner, que inclui o agente e todos os demais elementos nela presentes. Relações que representam o que está dentro e fora do contêiner, definidas por Frames do tipo Relation, serão utilizadas pelo agente para saber se ele já está fora da sala ou não. Neste experimento o agente não tem informações privilegiadas (por exemplo, saber que a porta está ou não destrancada, etc.), e deverá utilizar somente suas percepções visuais e suas experiências para guiá-lo em seu objetivo principal. Utilizando a mesma terminologia para a definição dos Frames do experimento anterior, o plano de ações gerado pelo agente será:

1. Goal agendado: GetOut(sarah, bedRoom) [prioridade 99]
   - i Pré-condições do goal: Near(sarah, Passage(bedRoom)) == true, Unblocked(Passage(bedRoom)) == true
   - ii Passage(bedRoom) é resolvido como leftDoor, pois leftDoor possui maior intensidade de ativação, é uma instância de Door que estende Passage
   - iii Subgoal em execução: Move(sarah, leftDoor) [prioridade 98] para resolver Near(sarah, leftDoor)
   - iv Subgoal finalizado: Move(sarah, leftDoor)

v Subgoal em execução: Open(sarah, leftDoor) [prioridade 98] para resolver Unblocked(leftDoor)

i Pré-condições do goal: Equals(leftDoor.state, Closed()) == true, Near(sarah, leftDoor) == true

ii Utilizando o sensor de visão, o agente já havia detectado que o estado da porta era fechada (Closed) e como ele já estava perto da porta, ambas as pré-condições foram satisfeitas.

iii Subgoal em execução: Open(sarah, leftDoor) [prioridade 98] para resolver Unblocked(leftDoor)

iv Subgoal falhou: Open(sarah, leftDoor)

v Um Failure Frame(leftDoor, Locked(), Open(sarah, leftDoor)) foi criado e ativado, indicando que não foi possível abrir a porta, dado que está trancada.

2. Uma regra de detecção de falhas captura o Failure Frame(leftDoor, Locked(),Open(sarah, leftDoor)) e ativa os Locked Frames e o Frame leftDoor com força máxima (1.0)

3. Por similaridade (conforme descrito na Listagem B.3) ao estado Locked, a ação Unlock é então ativada com força máxima, uma vez que existe um SimilarityLink que conecta ambos os Frames com intensidade máxima (strength = 1.0).

4. Uma regra detecta que a ação Unlock e uma Failure relacionada a esta ação estão ativas, gerando um novo *goal* Unlock(sarah, leftDoor). Os parâmetros da ação são definido pelos tipos de Frames de seu construtor.

5. Goal agendado: Unlock(sarah, leftDoor) [prioridade 98]

i Pré-condições do goal: Equals(leftDoor.state, Locked()) == true, Near(sarah, leftDoor) == true, Holding(sarah, Key()) == true

ii As duas primeiras pré-condições estão satisfeitas, mas a última não. Como o parâmetro da terceira pré-condição é um tipo e não uma instância de Frame, o agente pega a instância ativa mais intensa deste Frame.

iii Subgoal em execução: Grab(sarah, redKey) [prioridade 97] para resolver Holding(sarah, redKey)

i Pré-condições do goal: Near(sarah, redKey) == true, NotHoldingSomething(sarah) == true

ii O agente não está segurando nada neste momento.

   iii Subgoal em execução: Move(sarah, redKey) [prioridade 96] para resolver Near(sarah, redKey)
   iv Subgoal finalizado: Move(sarah, redKey)
  iv Subgoal finalizado: Grab(sarah, redKey)
  v O *goal* Unlock(sarah, leftDoor) é novamente avaliado e percebe-se que Near(sarah, leftDoor) == false.
  vi Subgoal em execução: Move(sarah, leftDoor) [prioridade 97] para resolver Near(sarah, leftDoor)
  vii Subgoal finalizado: Move(sarah, leftDoor)

6. Goal finalizado: Unlock(sarah, leftDoor)
7. Um SolvedFailure Frame(leftDoor, Unlocked()) é criado e ativado com força máxima.
8. O *goal* agendado GetOut(sarah, bedRoom) é novamente avaliado e percebe-se que Unblocked(Passage(bedRoom)
9. Subgoal em execução: Open(sarah, leftDoor) [prioridade 98] para resolver Unblocked(leftDoor)
10. Subgoal finalizado: Open(sarah, leftDoor)
11. Goal em execução: GetOut(sarah, bedRoom)
   i Subgoal criado: Move(sarah, <posição conhecida e não ocupada fora do quarto>)
   ii Subgoal em execução: Move(sarah, SpatialPosition(x,y))
   iii Subgoal finalizado: Move(sarah, SpatialPosition(x,y))
12. Goal finalizado: GetOut(sarah, bedRoom)

Através desse plano de ações, o agente consegue atingir seu objetivo principal, tratando eventuais falhas e manipulando corretamente as informações coletadas pelos seus sensores e pelas regras especificadas em seu ReasoningMode padrão, concluindo com sucesso o último experimento detalhado neste capítulo.

Capítulo 6

Conclusões

Foi apresentado neste trabalho um modelo de uma mente sintética, para controlar atores virtuais, baseado em conceitos de Inteligência Artificial Genérica. Este modelo é fruto de uma ampla pesquisa e investigação de teorias, métodos e conceitos de manipulação de conhecimento para a resolução de problemas e geração de comportamentos espontâneos que dão a ilusão de inteligência aos agentes. Foram muitas as tentativas e experimentações (utilizando pequenos protótipos e prospecções tecnológicas) até chegar ao resultado apresentado nesta dissertação. O trabalho não termina por aqui, mas esta etapa foi concluída com sucesso, pois todos os objetivos foram alcançados.

Os experimentos realizados permitiram avaliar os módulos da mente do agente de forma independente ao exigirem respostas de partes específicas do modelo. Os dois últimos experimentos se configuram como testes de integração do sistema, requisitando do agente a resolução de problemas que necessitam que todas as partes do modelo funcionem de forma correta e cooperativa. Além disso, apesar de o simulador utilizado nos testes não oferecer de forma completa (animações, ambientes detalhados, recursos de comunicação entre os agentes, etc.) foi possível demonstrar o potencial do modelo, lembrando que está fora do escopo deste trabalho a construção de uma sistema completo de NI.

A variedade de recursos e a flexibilidade do modelo proposto são características que o tornam uma ferramenta "poderosa" para se desenvolver agentes inteligentes para sistemas de Narrativa Interativa. É possível que a evolução de modelos como o proposto neste trabalho, promova, no futuro, a existência de bases compartilhadas de consciências de personagens. Ou seja, conjuntos de experiências de um indivíduo virtual armazenadas em fragmentos de memórias de uma base de dados comum e disponíveis para qualquer um que deseje construir sua própria narrativa emergente. Com algo deste tipo, uma nova forma de entretenimento surgiria a partir de sistemas de NI.

Com a utilização de um diretor real ou mesmo artificial (um outro agente inteligente atuando como diretor) para conduzir o foco da narrativa, seria possível construir filmes e seriados interativos que se correlacionam de certa forma, pois a consciência dos personagens, com todas as experiências vivenciadas em uma narrativa anterior, seria utilizada em outros trabalhos, dando continuidade à existência do personagem.

Pesquisadores que conhecem bem sistemas de controle de agentes inteligentes e também dominam os conceitos de Narrativa Interativa podem afirmar que o modelo proposto é algo genérico e que pode ser visto como um modelo de propósito geral e não exclusivo para sistemas de NI. É correto afirmar que este modelo pode ser empregado em outros contextos que não o de sistemas de NI. Porém, os requisitos atendidos pelo modelo são necessários para a preparação de sistemas de NI, conforme descrito no Capítulo 3. A forma com que os problemas são resolvidos pela mente do agente, a manutenção dos fragmentos de memória, a influência do sistema emocional na tomada de decisões, além outros aspectos, foram levados em consideração durante a concepção do modelo. Isto para torná-lo capaz de transformar os agentes em atores aptos a desempenharem papéis de personagens dentro de um sistema de Narrativa Interativa.

Situações como a apresentada no filme "O show de Truman" [1], onde uma narrativa macro é planejada pelo diretor e eventos emergentes conduzem a história, poderiam ser reproduzidas em sistemas computacionais de Narrativa Interativa. Traçando um paralelo entre o filme e um sistema de NI, apenas o protagonista não tinha consciência da verdade dos eventos por trás da narrativa e, num sistema de NI, este personagem poderia ser substituído por um agente inteligente. Os demais personagens seriam usuários humanos interagindo com o sistema e seguindo as regras estipuladas pelo diretor. Mesmo não sendo literalmente um jogo, um sistema de NI como este poderia ser visto como um MMOG (Massive Multiplayer Online Game) [Wikipedia, 2009a], onde os jogadores assumiriam papéis de personagens específicos dentro do contexto da narrativa conduzida por um diretor, o que seria algo parecido com RPGs tradicionais regidos por mestres de jogo [Tychsen, 2006]. Uma outra forma de enxergar este comparativo entre o filme e um sistema de NI seria a utilização de agentes inteligentes para executarem todos os papéis dos demais personagens que não o protagonista (Truman), sendo que este poderia ser controlado pelo usuário humano para interagir com o universo da história e conduzir sua própria versão da narrativa.

É importante ressaltar que o papel de um diretor (focado na narrativa) em sistemas que conduzem a história de forma emergente pode fazer a diferença, pois sem a sua interferência o resultado final pode ser algo como em um simples "Reallity Show"

[1] http://www.imdb.com/title/tt0120382/

[Wikipedia, 2009c]. Neste tipo de programa de televisão, apesar dos participantes poderem criar personagens e os utilizarem para jogar, não existe uma história e o diretor apenas organiza eventos diversos para tornar o programa mais atraente para os teleespectadores. Uma característica importante dos participantes de um "Reallity Show" é que mesmo aqueles que criam personagens para jogar, possuem a consciência de que aquilo é apenas um jogo e não existe uma história a ser desenvolvida. Por outro lado, agentes inteligentes que utilizem modelos como o proposto neste trabalho poderiam ser preparados para terem objetivos de longa duração, de preferência que durem em todo o ciclo da narrativa, sendo capazes de atuar sem a necessidade de um diretor.

É fato que não é pouco o volume de trabalho demandado para se gerar conteúdo para uma narrativa de média ou longa duração, utilizando-se o modelo proposto. Modelar todos os fragmentos de memória dos agentes não é uma tarefa complicada, mas exige que uma grande quantidade de esforço para coletar e cadastrar informações, modelando a consciência, para que um agente interprete de forma coerente o seu papel na narrativa. Contudo, conforme especulado anteriormente, é importante ressaltar que todo esse trabalho poderia ser re-aproveitado em outras sequências da narrativa principal, o que geraria um ganho de esforço na construção de narrativas posteriores.

O modelo apresentado neste trabalho, obviamente, levou em consideração o requisito de responsividade de eventos em tempo real ao usuário do sistema de NI. Por isso a maioria dos cálculos são muito simples, mas eficientes, e baseiam-se na utilização de somatórios matemáticas que envolvem as intensidades de ativação dos *Resources*. Mesmo assim, não é trivial implementar um sistemas de gerenciamento de Frames eficiente e que funcione adequadamente aos requisitos deste trabalho. O sistema de Frames implementado no protótipo utilizado para a realização dos experimentos é fruto da terceira tentativa, e mesmo assim não possui a eficiência necessária para ser empregado em uma aplicação real. As duas primeiras tentativas gastavam respectivamente 120% e 70% de tempo a mais que a terceira na realização das tarefas de criação, configuração e destruição de Frames. Portanto, não faz sentido executar um teste de desempenho para se comparar o protótipo desenvolvido com outras abordagens de controle de agentes inteligentes.

Como trabalhos futuros, pretende-se criar um modelo eficiente de um sistema de Frames integrado a uma linguagem de *scripts* conforme especificado no capítulo 3, além de realizar a implementação de forma definitiva e otimizada do modelo de mente, para ser utilizado em uma ferramenta de visualização e simulação de ambientes virtuais ricos em funcionalidades (ações, expressões corporais[faciais, gestuais, etc.] para os atores, extensível, escalável, etc.). O objetivo de se implementar o modelo para um ambiente virtual com mais recursos é o de se poder criar uma narrativa interativa

completa, onde o usuário humano poderia assumir o papel de um personagem e conduzir a história juntamente com os atores virtuais controlados pela mente artificial descrita nesta dissertação.

Apêndice A

Teorias sobre o cérebro humano e a inteligência

> Inteligência normalmente significa "a habilidade de resolver problemas difíceis" [Minsky, 1985].
>
> Inteligência é a habilidade de um sistema de processamento de informações se adaptar a um ambiente com recursos e conhecimentos limitados [Wang, 1995].

Como funciona o cérebro humano? A resposta desta pergunta solucionaria um dos maiores mistérios sobre a natureza do homem. Mas como ainda não se sabe a resposta desta pergunta, busca-se, através de diversas teorias, explicar como as pessoas são capazes de resolver problemas, memorizar enormes quantidades de dados, reconhecer padrões sensoriais (imagens, sons, cheiros, etc.) com grande rapidez, além de realizar diversas outras tarefas que não se sabe ao certo como são tratadas no cérebro humano.

Uma das teorias mais importantes sobre o cérebro, mais especificamente sobre o córtex cerebral, cunhada pelo neurocientista Vernon Mountcastle, se refere às colunas corticais e aos padrões cerebrais. O córtex cerebral é a camada mais externa, constituída de tecido nervoso cinzento, que encobre a parte interna do cérebro (tecido nervoso branco) de todos os mamíferos. Nos humanos, o córtex possui seis camadas, cada uma com a espessura de uma carta de baralho. Acredita-se que quanto mais camadas, mais "inteligente" é a espécie (os golfinhos possuem apenas três camadas). Não será detalhado aqui a parte da teoria que descreve as colunas corticais, mas de uma forma geral elas mapeiam o córtex em uma arquitetura de colunas conectadas de forma hierárquica e dispostas nas seis camadas que constituem o córtex cerebral. Isso não ajuda muito a entender como as colunas funcionam (para tal, consultar o mate-

rial original em Mountcastle [1978]), mas será importante para compreender como os padrões cerebrais são transformados em memórias e vice-versa.

Mas o que seriam estes tais padrões cerebrais? Recorrendo à parte da Biologia que estuda o cérebro humano, pode-se destacar as sinapses e os pulsos elétricos, utilizados pelas células cerebrais (neurônios) para se "comunicarem" entre si. Fazendo uma analogia com os condutores elétricos, o cérebro pode ser visto como uma malha de bilhões de fios metálicos que trafegam freneticamente energia elétrica de um ponto a outro do cérebro. Logo, os padrões podem ser compreendidos como sequências de pulsos elétricos, que passam a ter um significado se levado em consideração o intervalo de tempo que estes pulsos chegam ao cérebro advindos de algum sensor (órgão sensorial). Órgãos sensoriais (olhos, ouvidos, pele, etc.) são os responsáveis pela geração dos padrões cerebrais. Eles capturam, no ambiente externo, dados brutos de diversos tipos, fazem o devido tratamento e entregam os resultados ao cérebro no formato destes pulsos elétricos temporalmente ordenados. Dessa forma, o cérebro "apenas" tem que se preocupar em receber os padrões, reconhecê-los, catalogá-los, armazená-los e prover alguma forma para permitir a sua recuperação no futuro. É importante ressaltar que qualquer parte do córtex é capaz de tratar padrões provenientes de quaisquer fontes sensoriais.

Mas como ficam as teorias que afirmam que o cérebro é dividido em áreas responsáveis por executar tarefas específicas como reconhecer padrões visuais, lidar com a língua falada, realizar raciocínio matemático, etc.? Mountcastle [1978] acredita que o córtex não é constituído de forma modular. Segundo ele, qualquer área do córtex é capaz de lidar com informações de qualquer natureza. É provável que o cérebro funcione desta forma e, levando isto em consideração, pode-se dizer que o córtex trabalha com um "algoritmo genérico de manipulação de padrões". Ou seja, qualquer padrão que chegue ao cérebro ou trafegue por ele é tratado da mesma forma.

O modelo forjado por esta teoria permitiu o avanço de diversas pesquisas sobre a organização e o processamento dos padrões cerebrais. O criador do computador de mão Palm Pilot e entusiasta dos estudos sobre o córtex cerebral, Jeff Hawkins, publicou um livro em 2005 chamado OnIntelligence [Hawkins & Blakeslee, 2004]. Neste livro ele reforça com diversas outras suposições, relatos e experimentos as teorias de Mountcastle e ainda faz especulações sobre vários outros assuntos ainda não explicados sobre o funcionamento do cérebro. É importante ressaltar que o autor de OnIntelligence é da área de computação e um indivíduo bastante pragmático. Ele deixa claro que o seu objetivo, no final das contas, é converter as teorias criadas por neurocientistas, biólogos, psicólogos, filósofos e outros estudiosos (incluindo ele próprio) num modelo computacional aplicável à construção de máquinas inteligentes. Mas o que chama

atenção em seu livro é a forma com que ele aborda os principais fundamentos das teorias sobre o córtex cerebral, além de explicá-las de forma prática. De uma forma sintética, as principais responsabilidades do córtex, apresentadas por Hawkins, são:

- Armazenamento sequencial de padrões: Quando o cérebro precisa armazenar alguma informação, os padrões que chegam ao córtex são armazenados de forma sequencial. Isso significa que o cérebro cria elos de comunicação entre os padrões de entrada, constituindo a memória a partir de um encadeamento de padrões. Como o tempo é levado em consideração durante este sequenciamento, as cadeias geradas também possuem informações temporais, que indicam o momento de chegada e/ou ativação de cada padrão. A ativação se refere ao acionamento de um padrão já armazenado no córtex, que também pode compor uma nova memória, combinando-se com outros novos padrões.

- Capacidade de invocar padrões auto-associativos: Ao tentar se lembrar de algum fato, o cérebro humano começa a busca pelo padrão que inicializa a cadeia referente à memória que explica tal fato. Depois que este padrão foi recuperado, ele invoca o padrão consequente através do elo criado entre os dois. Este processo se repete até que toda a cadeia tenha sido recuperada e o fato relembrado. Por isso é possível se lembrar de cada letra do alfabeto, passando de A a Z com facilidade, mas poucas pessoas conseguem realizar esta tarefa ao contrário, de Z a A, pois normalmente as pessoas não são "treinadas" para isto e não possuem os padrões sequenciais necessários para se lembrar do alfabeto de trás p/ frente. É claro que se for realizado um treinamento - pensando no córtex, este treinamento seria o armazenamento de padrões temporais que representam as letras de Z a A - semelhante ao efetuado para se lembrar de A a Z, seria fácil se lembrar do alfabeto ao contrário com facilidade.

- Armazenamento de padrões num formato invariante: Imagine um indivíduo num parque brincando com uma criança. Esse indivíduo tem uma bola em suas mãos. A criança pede a bola e ele a joga. Então a criança repete o seu gesto e joga a bola para ele. Ele a pega e continua a brincadeira. Todo o mecanismo necessário para realizar as ações que o indivíduo e a criança executaram é tratado de forma "inconsciente". Eles não têm que se preocupar com a velocidade da bola, as forças que atuam sobre ela, o seu peso, etc. Simplesmente a jogam e depois a pegam. Isto porque existe uma invariante armazenada no cérebro que representa "pegar algo" e "jogar algo". Uma invariante nada mais é do que um conjunto de padrões genéricos, que serve como modelo para o processamento de padrões mais

específicos. Ainda não existe, de forma detalhada, uma teoria que explique como são formadas as invariantes mas, acredita-se que as invariantes são as responsáveis pelas pessoas conseguirem lidar com uma variedade enorme de estímulos durante o dia, sem ficar a todo momento se perguntando "o que é aquilo?" ou "como isto funciona?".

- Armazenamento de padrões em hierarquia: Como já foi dito anteriormente, as colunas corticais são organizadas hierarquicamente. Os padrões mais genéricos e invariantes ficam no alto desta hierarquia. Padrões mais específicos ficam mais abaixo nesta hierarquia. Quando o estímulo vem dos órgãos sensoriais, os padrões são ativados num fluxo de baixo para cima nesta hierarquia de colunas. Mas quando um indivíduo imagina alguma coisa ou tenta se lembrar de um algum fato, os padrões são ativados de cima para baixo, das invariantes para os padrões específicos. Esta organização permite o reaproveitamento de padrões por diversos fragmentos de memória, evitando a replicação desnecessária e formando uma malha que facilita a recuperação posterior de informações.

- Recuperação da memória auto-associativa: Existem diversos padrões sequenciais armazenados no córtex. Supõe-se que um indivíduo tenha visto uma caneta e, portanto, uma sequência de padrões que a descreve é enviada ao cérebro. O cérebro por sua vez cria uma sequência de padrões que representa a caneta, resultando no entendimento do que é uma caneta por parte do indivíduo. A princípio isto pode parecer algo inútil e redundante, mas o mecanismo de recuperação de informação no córtex é algo extremamente poderoso. Por exemplo, se ao invés de passar ao cérebro os padrões da caneta inteira, passa-se os padrões que representam apenas a ponta da caneta, ou a caneta de cabeça para baixo, ou apenas metade da caneta ou uma caneta deformada pelo fogo de um isqueiro, etc. a resposta será a sequência completa de padrões que representa a caneta e o indivíduo mesmo assim entenderá que é uma caneta. O cérebro humano completa as informações com diversos padrões, realiza buscas nas invariantes, faz conexões associativas com outras cadeias de padrões e devolve a resposta de maneira completa, desde que ela já esteja previamente armazenada. Estar previamente armazenada significa que o cérebro já tenha sido exposto anteriormente àqueles padrões.

- Antecipação dos fatos: O cérebro prevê o futuro o tempo todo, mas não é algo como "enxergar um futuro distante". As consequências de todas as ações de um indivíduo a curto prazo são antecipadas pelo seu cérebro. Por exemplo, quando

> ele chega em casa depois do trabalho e gira a maçaneta da porta, o seu cérebro antecipa o quanto de força será necessário aplicar sobre a maçaneta para girá-la e empurrar a porta. Se as forças aplicadas abrirem a porta como o de costume, a antecipação é confirmada e o cérebro continua seu papel de "vidente a curto prazo". Caso alguém tenha mexido na porta e a força aplicada não seja o suficiente para abri-la ou enquanto ele a empurra para trás ela faz um rangido que antes não fazia, haverá uma violação na previsão e o cérebro tentará buscar padrões que expliquem o porquê disso. Logo, pode-se considerar todo e qualquer tipo de aprendizado como um treinamento, que construirá padrões a serem utilizados para antecipar fatos.

Se uma comparação for realizada entre um processador de um computador e o cérebro humano sob a luz das teorias de Mountcastle e Hawkins, percebe-se que eles funcionam de maneiras bem distintas. O cérebro humano somente lida com memórias em formato de padrões, realizando o armazenamento e a recuperação de informação neste formato. Quando um indivíduo estica os braços para segurar uma bola jogada para ele, seu cérebro não calcula a trajetória da bola, muito menos integra as forças para realizar a cinemática inversa, ele simplesmente consulta a memória e traz a resposta. O cérebro sempre consulta as respostas dos problemas a serem resolvidos. Por isso que técnicas de raciocínio são indispensáveis para lidar com problemas complexos. Os seres humanos usam estas técnicas para organizar as variáveis do problema na memória, recuperar os padrões necessários para reconhecer estas variáveis através de associações e, por fim, alcançar a solução (que será armazenada para consultas futuras). Problemas desconhecidos são resolvidos através de analogias a problemas conhecidos já armazenados em nossa memória.

Para finalizar, segundo as teorias de Hawkins & Blakeslee e Mountcastle, inteligência é a capacidade de antecipar as coisas. Os testes de QI exploram justamente esta capacidade. Portanto, segundo esta teoria, é possível que o cérebro seja treinado para se tornar inteligente (inclusive para tirar notas altas em testes de QI). Quanto mais invariantes e padrões genéricos, mais eficiente o cérebro será na antecipação de situações desconhecidas ou consideradas complicadas para a maioria das pessoas.

Apêndice B

Exemplos de códigos

O protótipo utiliza diversos arquivos de dados para a simulação de uma cena. Abaixo, estão relacionados fragmentos de arquivos utilizados pelo protótipo.

Listing B.1. Descritor XML para criar uma cena., utilizando Resources

```
<?xml version="1.0"?>
<Scene name="room">
 <Dimension>100 100</Dimension>
 <Position>0 0</Position>

 <Entities>

  <Entity name="Sarah" type="Agent">
   <Position>55.01 65.00</Position>
   <Orientation>-120</Orientation>
   <Dimension>1.75 1.75</Dimension>
   <Frame>sarah</Frame>
   ...
  </Entity>

  <Entity name="YellowKey" type="Item">
   <Position>35.00 50.00</Position>
   <Orientation>0</Orientation>
   <Dimension>1.00 0.50</Dimension>
   <Frame>yellowKey</Frame>
  </Entity>

 </Entities>
</Scene>
```

O descritor apresentado na listagem B.1 configura toda a cena que será carregada e simulada. Uma cena é, de forma geral, parte da narrativa que se passa em um local específico. Este local é todo configurado com entidades estáticas e móveis. Entidades estáticas são paredes, mobília, etc. e móveis são itens, os agentes, avatares, etc.. Cada objeto presente em uma cena é configurado a partir de um elemento Entity. Todo

Entity possui um tipo. O tipo define quais comportamentos são esperados daquele objeto. As entidades obedecem uma hierarquia, conforme mostrado na Figura 4.4.

Listing B.2. Mapeia entidades da cena com Frames.

```
<?xml version="1.0"?>
<SceneDescriptor name="room">
 <Frames>
  <Frame name="yellowKey" instanceOf="Key">
   <Description>A normal key description</Description>
   <Slot name="color" type="Color">Yellow()</Slot>
  </Frame>
  <Frame name="sarah" instanceOf="Human">
   <Description>Johns wife</Description>
   <Slot name="color" type="Color">DarkRed()</Slot>
  </Frame>
 </Frames>
</SceneDescriptor>
```

Cada elemento presente na cena possui uma instância de *Frame* correspondente, que descreve o objeto dentro da mente do agente. A listagem B.2 apresenta o descritor do arquivo XML com os *Frames* que são referenciados pelas entidades através da propriedade "<Frame>nome frame</Frame>" conforme listagem B.1.

Listing B.3. Descritor de fragmentos de memória do agente.

```
<?xml version="1.0"?>
<Memory name="Memory1">
 <Resources>
  <Frames>
   <!-- concepts -->
   <Frame name="Thing">
    <Description>Thing is the "universal collection": the collection which, by
        definition, contains everything there is. Every individual object, every other
         collection. Everything that is represented in the Knowledge Base ("KB") and
        everything that could be represented in the KB. </Description>
    <Slot name="Thing" type="Function">
     <![CDATA[
       -- do nothing
     ]]>
    </Slot>
    <Slot name="name" type="Text"/>
   </Frame>
  ...
   <Frame name="Grab" parent="Action">
    <Description></Description>
    <Slot name="Grab" type="Function">
     <![CDATA[
      logger:log( Logger.FINE_MSG, "Grab - constructor called" )
      local param1, param2 = ...
      if not isObject( param1 ) or not isObject( param2 ) then
       error( "Grab::Grab - parameters must be objects" )
      end
      self.actor:setValue( DataType( param1 ) )
```

```
      self.target:setValue( DataType( param2 ) )
      logger:log( Logger.FINE_MSG, "Grab - initialized" )
     ]]>
    </Slot>
    <Slot name="actor" type="Human"/>
    <Slot name="target" type="Thing"/>
    <Slot name="preconditions" type="AssertionCollection">AssertionCollection( Near(
        val(self.actor), val(self.target) ), NotHoldingSomething( val(val(val(val(val(
        self.actor).body).torso).leftArm).hand) ) )</Slot>
    <Slot name="update" type="Function">
    <![CDATA[
     logger:log( Logger.DEBUG_MSG, "Grab::update - calling update" )
     if not agent:wasEntityPerceivedByFrameName( val(self.target):getName( ) ) then
      logger:log( Logger.ERROR_MSG, "Grab::update - entity '" .. val(self.target)
          :getName( ) .. "' wasn't perceived by the agent" )
      self.state:setValue( DataType( Failed( ) ) )
     elseif agent:isHoldingSomething( ) then
      logger:log( Logger.ERROR_MSG, "Grab::update - Agent cannot hold another thing" )
      self.state:setValue( DataType( Failed( ) ) )
     else
      if agent:wasEntityPerceivedByFrameName( val(self.target):getName( ) ) then
       local entity = agent:getPerceivedEntityByFrameName( val(self.target):getName( )
           )
       logger:log( Logger.DEBUG_MSG, "Grab::update - Trying to grab '" .. val(self.
           target):getName( ) .. "' id '" .. entity:getId( ) .. "'" )
       if agent:hold( entity:getName( ) ) then
        self.state:setValue( DataType( Succeed( ) ) )
        local agentsHand = val(val(val(val(self.actor).body).torso).leftArm).hand)
        agentsHand.holds:setValue( DataType( val(self.target) ) )
        logger:log( Logger.DEBUG_MSG, "Grab::update - Hold was successfully executed"
            )
       else
        logger:log( Logger.ERROR_MSG, "Grab::update - Real Body cannot hold the item.
            Maybe there is some fault in synchronization" )
        self.state:setValue( DataType( Failed( ) ) )
       end
      else
       logger:log( Logger.ERROR_MSG, "Grab::update - The object disapeared..." )
       self.state:setValue( DataType( Unknown( ) ) )
      end
     end
     return self.state:getValue( )
    ]]>
    </Slot>
   </Frame>
  </Frames>
 </Resources>

 <SimilarityLinks>
  <SimilarityLink frameA="Locked" frameB="Unlock" strength="1.0" bidirectional="false"
      />
  <SimilarityLink frameA="Door" frameB="Key" strength="1.0" bidirectional="true"/>
 </SimilarityLinks>
</Memory>
```

Para cada agente, um arquivo descritor de sua memória, como o apresentado na listagem B.3, deve ser preparado. É nesse arquivo que todas os fragmentos de memórias do agente, representados por *Frames* devem ser descritas. Ou seja, este descritor modela o comportamento do agente através de conhecimento que lhe é modelado e atribuído.

Listing B.4. Função membro de um Slot escrita em Lua.

```
<Slot name="status" type="Function">
 <![CDATA[
  logger:log( Logger.FINE_MSG, "Near - Calling status" )
  local object1 = val(self.thing1)
  local object2 = val(self.thing2)

  if not o.currentScene then
   logger:log( Logger.ERROR_MSG, "Near - currentScene wasnt created yet" )
   return DataType( false )
  end

  local relations = toTypeListIt( o.currentScene.relations:getValue( ) )

  local id = hashCombine( val(self.thing1):getId( ), val(self.thing2):getId( ) )

  for r in relations do
   local relation = toFrame( r )
   if relation:isA( s.Near ) then
    local relationId = hashCombine( val(relation.thing1):getId( ), val(relation.thing2
        ):getId( ) )
    if relationId == id then
     return DataType( true )
    end
   end
  end

  return DataType( false )
 ]]>
</Slot>
```

Todo *Frame* que realiza algum tipo processamento em tempo de execução deve ser escrita como o exemplo da listagem B.4. O ScriptManager interpreta este código e gera uma função que é então associada ao *Slot*. O *Slot* pode então ser invocado por qualquer parte do código, e, no caso acima, o status do *Frame* será então computado.

Referências Bibliográficas

Araújo, S. & Chaimowicz, L. (2009). A synthetic mind model for virtual actors in interactive storytelling systems. In Young, R. M. & Laird, J. E., editores, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, October 14-16, 2009, Stanford, California, USA*. AAAI Press.

Aylett, R.; Louchart, S.; Dias, J.; Paiva, A. & Vala, M. (2005). Fearnot! - an experiment in emergent narrative. In *IVA*, pp. 305–316.

Barros, L. M. & Musse, S. R. (2008). Towards consistency in interactive storytelling: Tension arcs and dead-ends. *Comput. Entertain.*, 6(3):1--17.

Bourg, D. M. & Seeman, G. (2004). *AI for Game Developers*. O Reilly, primeira edição.

Brachman, R. & Levesque, H. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufmann.

Brakeen, D.; Barker, B. & Vanhelsuwé, L. (2003). *Developing Games in Java*. New Riders, primeira edição.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA.

Cavazza, M.; Charles, F. & Mead, S. J. (2001). Characters in search of an author: AI-based virtual storytelling. *Lecture Notes in Computer Science*, 2197:145--??

Cavazza, M.; Charles, F. & Mead, S. J. (2002). Character-based interactive storytelling. *IEEE Intelligent Systems*, 17(4):17--24.

Charles, F.; Lozano, M.; Mead, S. J.; Bisquerra, A. F. & Cavazza, M. (2003). Planning formalisms and authoring in interactive storytelling. In *Proceedings of the 1 st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pp. 216--225. Fraunhofer IRB Verlag.

Ciarlini, A. E. M.; Pozzer, C. T.; Furtado, A. L. & Feijó, B. (2005). A logic-based tool for interactive generation and dramatization of stories. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pp. 133--140, New York, NY, USA. ACM.

Crawford, C. (1999). Assumptions underlying the erasmatron interactive storytelling engine. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, pp. 189–197.

Crawford, C. (2005). *On Interactive Storytelling*. New Riders, primeira edição.

Eaters, T. D. (1998). Player 1 stage 1: Bits from the primordial ooze.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. MIT Press, primeira edição.

Fikes, R. E. & Nilsson, N. J. (1990). Strips: A new approach to the application of theorem proving to problem solving. In Allen, J.; Hendler, J. & Tate, A., editores, *Readings in Planning*, pp. 88–97. Kaufmann, San Mateo, CA.

Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley Professional.

Glassner, A. (2004). *Interactive Storytelling: Techniques for 21st Century Fiction*. AK Peters, Ltd., primeira edição.

Goertzel, B. & Pennachin, C. (2007). *Artificial General Intelligence*. Springer, primeira edição.

Hawkins, J. & Blakeslee, S. (2004). *On Intelligence*. Times Books, primeira edição.

Iklé, M.; Goertzel, B. & Goertzel, I. (2006). Indefinite probabilities for general intelligence. *AGIRI Workshop*.

Kurzweil, R. (2006). *The Singularity Is Near: When Humans Transcend Biology*. Penguin (Non-Classics).

Kutluhan Erol, James A. Hendler, D. S. N. (1994). Htn planning: Complexity and expressivity. In *AAAI*, pp. 1123–1128.

Laurel, B. (1993). *Computers as Theatre*. Addison-Wesley Professional, 1 edição.

Lenat, D. B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):32--38.

Levi, G. & Sirovich, F. (1976). Generalized AND/OR graphs. *Artificial Intelligence*, 7(3):243--259.

Lieberman, H.; Liu, H.; Singh, P. & Barry, B. (2004). Beating common sense into interactive applications. *AI Magazine*, 25(4):63--76.

Magerko, B. & Laird, J. (2002). Towards building an interactive, scenario-based training simulator. In *Proceedings of 11th Conference on Computer Generated Forces and Behavioral Representation*, p. 9.

Magerko, B.; Laird, J. E.; Assanie, M.; Kerfoot, A. & Stokes, D. (2004). Ai characters and directors for interactive computer games. In *In Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*, pp. 877--883. AAAI Press.

Magerko, B. S. (2006). Player modeling in the interactive drama architecture. In *Fall AAAI Symposium: Socially Intelligent Agents: The Human in the*, pp. 139--150.

Mateas, M. & Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama.

Mateas, M. & Stern, A. (2004a). A behavior language: Joint action and behavioral idioms. In *Life-like Characters: Tools, Affective Functions and Applications*. Springer.

Mateas, M. & Stern, A. (2004b). Natural language understanding in façade: Surface-text processing. In Göbel, S.; Spierling, U.; Hoffmann, A.; Iurgel, I.; Schneider, O.; Dechau, J. & Feix, A., editores, *Technologies for Interactive Digital Storytelling and Entertainment, Second International Conference, TIDSE 2004, Darmstadt, Germany, June 24-26, 2004, Proceedings*, volume 3105 of *Lecture Notes in Computer Science*, pp. 3--13. Springer.

Mateas, M. & Stern, A. (2005). Structuring content in the façade interactive drama architecture. In Young, R. M. & Laird, J. E., editores, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pp. 93--98. AAAI Press.

Meehan, J. R. (1977). Tale-spin, an interactive program that writes stories. In *In Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 91--98.

Microsoft (2006). *Xbox 360 Technical Specifications*.

Miller, C. H. (2004). *Digital Storytelling A Creator's Guide to Interactive Entertainment*. Focal Press, primeira edição.

Miller, G. A. (1998). Nouns in WordNet. In Fellbaum, C., editor, *WordNet: An Electronic Lexical Database*, pp. 23--46. The MIT Press, Cambridge, Massachusetts.

Minsky, M. (1985). *Society of Mind*. Simon & Schuster.

Minsky, M. (2006). *The Emotion Machine*. Simon & Schuster, primeira edição.

Mitchel, T. M. (1997). *Machine Learning*. McGraw-Hill, primeira edição.

Mountcastle, V. (1978). An organizing principle for cerebral function: The unit model and the distributed system. *Mindful Brain*, pp. 7--50.

Nareyek, A. (2007). Game ai is dead. long live game ai! *IEEE Intelligent Systems*.

Ortony, A.; Clore, G. L. & Collins, A. (1988). *The Cognitive Structure of Emotions*. Cambridge University Press.

Picard, R. W. (1997). *Affective Computing*. The MIT Press, Cambridge, Massachusetts.

Quillian, R. M. (1968). Semantic memory. In Minsky, M., editor, *Semantic Information Processing*, pp. 227--270. MIT Press, Cambridge, MA.

Quillian, R. M. (1991). Word concepts: a theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410--430.

Reilly, W. S. N. & Bates, J. (1996). *Believable Social and Emotional Agents*. PhD thesis, CMU.

Richards, D. (2006). Is interactivity actually important? In *IE '06: Proceedings of the 3rd Australasian conference on Interactive entertainment*, pp. 59--66, Murdoch University, Australia, Australia. Murdoch University.

Riedl, M. O.; Saretto, C. J. & Young, R. M. (2003). Managing interaction between users and agents in a multi-agent storytelling environment. In *AAMAS*, pp. 741--748. ACM.

Riedl, M. O. & Young, R. M. (2003). Character-focused narrative generation for execution in virtual worlds. In Balet, O.; Subsol, G. & Torguet, P., editores, *Virtual Storytelling; Using Virtual Reality Technologies for Storytelling, Second International Conference, ICVS 2003, Toulouse, France, November 20-21, 2003, Proceedings*, volume 2897 of *Lecture Notes in Computer Science*, pp. 47--56. Springer.

Russel, S. & Norvig, P. (2003). *Artificial Intelligence: a Modern Approach*. Prentice-Hall, segunda edição.

Ruth, A.; Louchart, S.; Tychsen, A.; Hitchens, M.; Figueiredo, R. & Mata, C. D. (2007). Managing emergent character-based narrative. In *INTETAIN '08: Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, pp. 1--8, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Schank, R. C. & Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding: An Inquiry Into Human Knowledge Structures (Artificial Intelligence)*. Lawrence Erlbaum, 1 edição.

Silva, A.; Vala, M. & Paiva, A. (2001). Papous: The virtual storyteller. In *IVA*, pp. 171–180.

Stroustrup, B. (2000). *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Sun, R. (1991). Neural network models for rule-based reasoning. In *IEEE International Joint Conference on Neural Networks (5th IJCNN'91)*, volume 1, pp. 503--508, Singapore. IEEE. Brandeis U.

Szilas, N. (1999). Interactive drama on computer : Beyond linear narrative.

Szilas, N. (2008). IDtension - highly interactive drama. In Darken, C. & Mateas, M., editores, *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA*. The AAAI Press.

Taha, S. & Löfstedt, T. (2004). Student papers in computer architecture. *Umea's Ninth Student Workshop in Computer Architecture*.

Thomas, M.; Weyhrauch, K. P. & Bates, J. (1992). Dramatic presence.

Tobias, R. B. (2003). *20 Master Plots and how to build them*. Writers Digest Books, primeira edição.

Tychsen, A. (2006). Role playing games: comparative analysis across two media platforms. In *IE '06: Proceedings of the 3rd Australasian conference on Interactive entertainment*, pp. 75--82, Murdoch University, Australia, Australia. Murdoch University.

Wang, P. (1995). On the working definition of intelligence. Technical report, Temple University.

Wang, P. (2006). From nars to a thinking machine. *AGIRI Workshop*.

Weyhrauch, P. W. (1997). *Guiding interactive drama*. PhD thesis, Carnegie Mellon University.

Wikipedia (2009a). Massive multiplayer online game. Wikipedia, The free Encyclopedia.

Wikipedia (2009b). Physics engine. Wikipedia, The free Encyclopedia.

Wikipedia (2009c). Reality television. Wikipedia, The free Encyclopedia.

Woodcock, S. (1998). Game ai: The state of the industry. *Game Developer Magazine*.

Wright, W.; Schroh, D.; Proulx, P.; Skaburskis, A. W. & Cort, B. (2006). The sandbox for analysis: concepts and eevaluation. In Grinter, R. E.; Rodden, T.; Aoki, P. M.; Cutrell, E.; Jeffries, R. & Olson, G. M., editores, *Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, pp. 801--810. ACM.

Young, R. M. (2001). An overview of the mimesis architecture: Integrating intelligent narrative control into an existing gaming environment.