UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RICARDO RODRIGUES LUCCA

MARO: Um modelo de emoções usando ontologia

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Rafael Heitor Bordini
Orientador

Porto Alegre, abril de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lucca, Ricardo Rodrigues

MARO: Um modelo de emoções usando ontologia / Ricardo Rodrigues Lucca. – Porto Alegre: PPGC da UFRGS, 2012.

105 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Rafael Heitor Bordini.

1. Agentes virtuais. 2. Programação de agentes. 3. Simulação. 4. Computação afetiva. 5. Modelo OCC. 6. Ontologias. I. Bordini, Rafael Heitor. II. Título.

RESUMO

Este trabalho apresenta um framework que permite a programação de agentes capazes de perceberem seus próprios estados emocionais. O framework foi desenvolvido em Java com base na plataforma multi-agente *Jason*, estendendo a base de crenças de agentes *Jason* a fim de utilizar a ontologia afetiva desenvolvida. Além disso, o ambiente foi construído a partir de uma base de conhecimento que descreve rotinas em ambientes simulados. Um mecanismo de avaliação das emoções baseando-se nas anotações dos objetos foi construído apoiado por uma ontologia de preferência sobre essas anotações. Dessa forma, aplicações de entretenimento poderiam utilizar o sistema ou as bases de conhecimento apresentadas para diferentes propósitos. A criação de um mapa onde os personagens atuam, e a criação da rotina de cada personagem e suas preferências são alguns exemplos de utilizações. Para validação do *framework* desenvolvido, dois exemplos foram construídos. O primeiro utilizou a maior parte dos grupos afetivos da ontologia proposta, com a finalidade principal de demonstrar o modelo implementado. Já o segundo usa apenas um grupo emotivo e serve para demonstrar a utilização conjunta de todas as ontologias apresentadas.

Palavras-chave: Agentes virtuais, programação de agentes, simulação, computação afetiva, modelo OCC, ontologias.

Maro: A model of emotions using ontology

ABSTRACT

This work presents a framework built on top of the Jason platform (BORDINI et al., 2004) to allow the development of software agents that have emotional states. The framework was developed in Java and extends the belief base of Jason agents so as to use an ontology for the OCC affective model (ORTONY; COLLINS; CLORE, 1988) that has been created as part of this work. The developed belief base allows an agent to perceive its own emotions throw inferring new beliefs based on the agent's appraisal of the state of the environment. In addition, a model of agents' routine tasks was defined, as was a model for agents' preferences about aspects of environment, helping automate the ascription of emotional states. Finally, in order to validate the developed framework, two applications were developed. The first demonstrates the use of various different emotions from the affective model and the second uses in a single application all the ontologies and models developed as part of this work.

Keywords: virtual agents, agent-oriented programming, simulation, affective computing, OCC's model, ontology.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AOP Agent Oriented Programming

ASL AgentSpeak Language

BDI Belief-Desire-Intention

IHC Interação Homem-Computador

OCC Ortony Clore e Collins

OWL Ontology Web Language

UEM Urban Environment Model

UML Unified Modeling Language

XML eXtensible Markup Language

LISTA DE FIGURAS

LISTA DE LISTAGENS

1 INTRODUÇÃO

Segundo Terzopoulos et al. (1998), a indústria cinematográfica possui excelentes exemplos do estudo do comportamento humano na área de animação. Essa área visa imitar as ações humanas computacionalmente. Entretanto, ainda há muito trabalho a ser feito na construção de um ator virtual que pareça realmente vivo. Um dos primeiros trabalhos envolvendo emoções, em personagens virtuais, é o de Bates (1994) visando melhorar a interpretação. Por exemplo, João pode ser agressivo quando estiver com medo e André fica retraído ao sentir medo.

As emoções podem ser categorizadas, segundo Damásio (2004), em primárias (não-cognitivas) e secundárias (cognitivas). As emoções primárias surgem a partir de reações a determinadas percepções vindas do meio ambiente e são simples e geradas rapidamente. Já as emoções secundárias são aprendidas ao longo da nossa vida, isto é, são lentas e geradas por uma avaliação de uma situação de acordo com nossos objetivos e valores morais. Por exemplo, André está com pena de Alberto por possuir determinada doença.

A Computação Afetiva estuda esse assunto dentro da Computação, dividindo-o em dois ramos principais. O primeiro estuda o reconhecimento e expressão de emoções dentro da Interação-Homem-Computador (IHC). O segundo estuda a síntese das mesmas visando estudar o comportamento humano por simulações e, dessa forma, contribuir para o aprimoramento de seres robóticos ou virtuais.

O presente trabalho visa desenvolver uma ferramenta que permita definir o comportamento afetivo de atores virtuais. Essa ferramenta é um *framework* que define as emoções dos atores utilizando ontologias. Uma ontologia é uma especificação explícita, fundamentada e bem formada de um conhecimento (GRUBER et al., 1993). Atualmente, elas são vistas como um entendimento comum e compartilhado que pode ser utilizado na comunicação entre máquinas ou pessoas (LÓPEZ et al., 2008).

Sendo assim, o principal resultado é a demonstração das três ontologias funcionando em conjunto. A ontologia de emoções foi definida conforme o modelo de Ortony, Collins e Clore (1988). A ontologia de rotinas, criada por Paiva, Vieira e Musse (2005), define a rotina dos personagens pelo perfil destes e permite descrever os locais que o mesmo pode ir visitar tornando possível criar uma visualização. Já, definir anotações em objetos

e preferências sobre essas anotações é a tarefa da última ontologia. Ela torna possível construir um mecanismo para emoções simples relacionadas com objetos.

Este trabalho está organizado da seguinte maneira. No capítulo 2 é feita a revisão bibliográfica das principais áreas desse trabalho: Agentes, Computação Afetiva e Ontologia. Os trabalhos relacionados também estão inclusos nesse capítulo em suas respectivas seções. A ideia da ferramenta é explicada no capítulo 3. Os estudos de caso são abordados no capítulo seguinte. No capítulo 5 é feita a avaliação do sistema juntamente com os trabalhos futuros e considerações finais.

2 ESTADO DA ARTE

2.1 Programação Orientada a Agentes

A presente seção tem como objetivo introduzir a Programação Orientada a Agentes (AOP). Com essa finalidade, o que se entende por Agente e por Personagem é discutido na seção 2.1.1. Uma introdução ao paradigma é dada na seção 2.1.2 e na seguinte a plataforma *Jason* é abordada porque será utilizada nos capítulos posteriores.

2.1.1 Visões Diferentes sobre o Conceito de Agente

Laird e VanLent (2001) afirmam que a pesquisa em Inteligência Artificial tem sido fragmentada em muitas áreas especializadas e, assim, algoritmos específicos e mais eficientes podem ser criados. Um exemplo disso é as inúmeras definições do conceito Agente. Shoham (1993) propõe que Agente é uma entidade definida por componentes mentais como crenças, habilidades, escolhas e comprometimentos.

Franklin e Graesser (1997) define um Agente através das características que a entidade deve apresentar. Atualmente, o menor conjunto comumente aceito dessas características é composto por três conceitos principais: (i) Autonomia; (ii) Sociabilidade; (iii) Situacionalidade. Jennings, Sycara e Wooldridge (1998) concordam que Autonomia é quando um Agente toma suas próprias decisões independente de qualquer outra entidade do sistema ou, ainda, da intervenção direta de seres humanos. A característica de Sociabilidade permite flexibilidade na execução das tarefas, através da interação com outros Agentes que estejam presentes no sistema. Por exemplo, é possível negociar que outro Agente faça a tarefa. A última característica permite que o agente situe-se em um ambiente dinâmico interagindo com o mesmo através de sensores e atuadores.

Ingrand, Georgeff e Rao (1992) indicam que entidades autônomas podem ter a capacidade de expor seus dados internos para que seja possível um usuário, possivelmente humano, dar dicas sobre a forma de resolução dos problemas sendo enfrentados. Esta visão incrementa a noção de Autonomia exposta acima, pois o agente permanece independente para tomar suas decisões podendo rejeitar as sugestões.

Ingrand, Georgeff e Rao também definem Agente em um ambiente com componentes

heterogêneos e com diferentes tempos de resposta para a execução de suas tarefas. Doyle e Hayes-Roth (1998) ampliam o conceito dizendo que os objetos no ambiente devem conter anotações que determinam a sua utilização pelos Agentes. Assim, não se sabe como todos os objetos funcionam e, sim, uma forma de aprender com o mesmo a sua forma de utilização. Shoham (1993) define Sociabilidade como uma habilidade cognitiva necessária para o agente desempenhar suas tarefas.

Essas inúmeras definições do termo Agente não permitem saber se ele é um ser físico ou abstrato. Dessa forma, Nareyek (2001), Damiano e Pizzo (2008) defendem que o Agente é o ser abstrato de um Ator físico. Em outras palavras, o ser que age ou atua no meio é chamado de Personagem ou Ator. Enquanto que a "mente" desse ser é chamada de Agente. Essa diferenciação é utilizada no presente texto, como forma de desambiguação.

2.1.2 Paradigma de Programação

Shoham (1993) propôs em seu trabalho a linguagem *Agent-0*, uma das primeiras a serem baseadas em atos de fala e uma nova visão para se programar. Esses atos de fala podem ser vistos como comandos falados entre atores com as mais diversas finalidades (informar, perguntar, requerer, aceitar, etc). Dessa forma, Shoham tentou promover a ideia de computação com uma interação mais social entre os membros de um sistema. Em outras palavras, esse novo paradigma de programação precisa que exista cooperação e competição entre os agentes para a realização das tarefas desejadas.

Assim, o agente encontra-se situado em um ambiente no qual pode receber de ou enviar para outros agentes informações. As ações são estabelecidas utilizando regras que descrevem comportamentos. Logo, a própria entidade decide qual ação deve ser tomada. Essas regras ou planos são construídas em fórmulas lógicas descrevendo-se o contexto que torna um curso de ação válido e sua consequência. Essa consequência pode ser uma ação ou um conjunto sequencial de ações que precisam ser feitos.

Atualmente, segundo Bordini et al. (2009), há diversas linguagens para a AOP. Por exemplo: *2APL*, *Agent-0*, *AgentSpeak(L)*, *GOAL*, *MetateM* e etc. Elas são baseadas em diversos formalismos diferentes. *MetateM*, por exemplo, é baseada em lógica temporal, permitindo que formulas temporais definam o comportamento do agente. *GOAL* tem a visão que o conhecimento pode ser estático (imutável) ou dinâmico e escolhem suas ações a partir das suas crenças e metas de uma maneira similar ao que acontece no *Jason*. *Agent-0*, como dito anteriormente, é a primeira linguagem da área de AOP. *2APL* é sucessora da *3APL* e dividiu a linguagem em dois formatos básicos (DASTANI, 2008). O primeiro define como a plataforma irá operar, isto é, quantos e em qual ambiente o agente se encontra. O segundo formato define a programação propriamente dita do indivíduo.

No presente trabalho, a plataforma *Jason* (BORDINI et al., 2004) será utilizada. Ela usa uma extensão da linguagem *AgentSpeak(L)*, definida por Rao (1996), para especificar as atitudes que o agente deve tomar frente as constantes modificações do ambiente. Assim,

essa linguagem é, normalmente, vista como reativa às crenças, metas ou percepções visto que trabalha com a ideia de eventos sobre esses conceitos.

2.1.3 Plataforma *Jason*

A presente seção visa explicar a plataforma *Jason* baseada na arquitetura BDI. Essa arquitetura é baseada no que o agente sabe ou guarda (crenças), as opções que ele possui de atuação baseado em suas próprias regras (desejos) e seus comprometimentos (intenções). Assim, na seção 2.1.3.1 uma visão geral da plataforma é apresentada e em seguida a engenharia da mesma é discutida.

2.1.3.1 Visão Geral

Para uma explicação mais didática será utilizado o exemplo *Room* que acompanha o *Jason*. Para isso será introduzido o arquivo de projeto da plataforma na Listagem 2.1 e, a partir desse, os demais arquivos. Na linha 2 da listagem, *room* é o nome do projeto e, por isso, pode ser qualquer identificador. A partir da linha 3 os valores antes dos dois-pontos (:) são palavras reservadas que o *Jason* entende para diferentes propósitos e o valor a seguir (depois dos dois-pontos) é o valor associado. Assim, *infrastructure*, na linha 3, pode assumir três valores possíveis: (i) *Centralised*, normalmente utilizada; (ii) *Jade*, utilizada quando se deseja integrar com agentes não *Jason* (jade) ou rodar distribuído na rede; (iii) *Saci*, utilizada quando deseja executar os agentes de maneira distribuída na rede.

Listagem 2.1: Arquivo de projeto do *Jason* para o exemplo *Room*

```
1 // Isso eh um comentario
  MAS room {
3   infrastructure: Centralised
    environment: RoomEnv
5   executionControl: jason.control.ExecutionControl
    agents: porter; claustrophobe; paranoid;
7 }
```

Continuando na Listagem 2.1, a entrada *environment* configura a classe de ambiente que será utilizada. A próxima entrada, normalmente não aparece, é a *executionControl* utilizada para mudar a forma com que os agentes são executados. O valor no exemplo é uma classe que obriga o próximo ciclo de deliberação acontecer somente quando todos os agentes terminarem o seu ciclo. O padrão é iniciar um novo após 500ms de o anterior ter sido concluído, porém, algumas vezes isso pode vir a apresentar problemas de sincronismo por ser assíncrono e, por isso, é interessante mostrar que há uma opção para controlar a forma de execução dos ciclos deliberativos. O usuário, inclusive, pode colocar sua própria classe como configuração.

Todas as entradas apresentadas até agora mapeiam para código Java. Os agentes são especificados na entrada *agents*. Como pode ser observado na Listagem 2.1, cada referên-

cia a um agente deve terminar com um ponto-e-vírgula (;). Há ainda como especializar cada um dos agentes mudando opções. Na seção 2.1.3.2 e no capítulo 4 são mostrados exemplos dessas opções.

Os agentes são desenvolvidos em arquivos com extensão *ASL* em formato de texto, porém antes de entrar em discussão sobre os agentes será explicado o exemplo sendo utilizado. Nesse cenário tem-se uma porta na sala e duas pessoas, uma claustrofóbica e outra paranoica. A pessoa claustrofóbica deseja que a porta da sala esteja aberta, enquanto que a paranoica deseja que a porta fique fechada.

Logo, há três agentes na linha 6 da Listagem 2.1: (i) *porter*, o agente responsável pela porta e o único que conhece como abri-lá ou fecha-lá; (ii) *claustrophobe*, o agente que deseja deixar a porta sempre aberta; (iii) *paranoid*, o agente que deseja deixar a porta sempre fechada. A implementação desses agentes faz uso de eventos.

Esses eventos podem ser de adição (+) ou de remoção (-) de crenças, metas ou consultas. Todas as estruturas são semelhantes a chamada de função. No exemplo "telefone(808080822)", telefone pode ser uma crença ou uma ação de ambiente que pode ser executada dependendo de onde a expressão aparece. Note que, se essa entrada for precedida pelo sinal de exclamação (!) ou de interrogação (?) então o significado é alterado para uma meta ou para uma consulta respectivamente. Ainda há a possibilidade de preceder uma crença ou meta com sinal de adição ou de subtração significando ou estar adicionando/removendo uma crença ou estar adicionando/removendo uma meta ou consulta.

Listagem 2.2: Agentes em ASL

```
1  // claustrophobe.asl
   +locked(door) : true
3    <- .send(porter,achieve,~locked(door)).

5  // paranoid.asl
   +~locked(door)
7    <- .send(porter,achieve,locked(door)).

9  // porter.asl
   +!locked(door)[source(paranoid)]
11   : ~locked(door)
     <- lock.
13
   +!~locked(door)[source(claustrophobe)]
15   : locked(door)
     <- unlock.
```

Na Listagem 2.2 tem-se a continuação do exemplo e, por simplicidade, todos os fontes dos agentes encontram-se reunidos. O agente denominado *claustrophobe* será o primeiro a ser detalhado e corresponde as linhas 1 até 3 da Listagem 2.2. A programação em *Jason* é guiada por reatividade às crenças e percepções do próprio agente, assim, é necessário uma forma de estruturar as ações à serem decididas. Essa forma é o plano. O plano pode ser ativado quando se deseja adicionar, consultar ou remover [1] uma crença ou meta. O plano da linha 2 até 3 da Listagem 2.2 será detalhado adiante.

[1] Uma remoção pode ser gerada devido a uma falha, porém isso não será abordado.

Em um plano há sempre três divisões: evento disparador, contexto e corpo. A primeira e única à ser explicita é o evento disparador, que deve ocorrer para o plano ser disparado. Ele está limitado do caractere inicial até o dois-pontos (:). A divisão de contexto é onde se coloca as ações, crenças ou regras que tem que ser válidas para o corpo do plano ser considerado válido. Dessa forma, é importante tomar cuidado no que vai no contexto em razão dele sempre ser executado previamente para definir quais planos devem ser descartados da lista de opções. A segunda divisão vai até a seta (<-) e não é obrigatória. A terceira divisão, também não é obrigatória, possui todas as ações a serem executadas quando o plano tornar-se ativo.

No exemplo o plano tem em seu corpo uma ação interna do *Jason* denominada *send* que envia determinada mensagem (terceiro parâmetro) para o agente especificado (primeiro parâmetro) com determinado propósito (segundo parâmetro). Uma ação interna possui o seguinte formato "tcp.send", assim essa ação está definida no pacote *tcp* pela classe *send* que deve especializar a classe *DefaultInternalAction* do *Jason*. Entretanto, uma ação interna definida pelo *Jason* não possui indicativo de pacote que é o caso do ".send" presente no código dos agentes.

No exemplo da linha 3 na Listagem 2.2, o agente *claustrophobe* está enviando uma mensagem para o agente denominado *porter* ter a meta (*achieve* no segundo parâmetro) de ter a crença que a porta não ($\sim$) está fechada. Analogamente, o agente denominado *paranoid*, definido na linha 5 à 7, envia como crença ter a porta fechada. Logo, o agente *porter* pode ser entendido em sua quase totalidade na Listagem 2.2. Um entendimento completo do exemplo vem com a compreensão das anotações que são informações que podem ser guardadas junto das crenças e essas anotações podem ter suas próprias anotações também. No agente *porter* essas anotações são utilizadas somente para dizer que determinado plano só é válido quando tiver como fonte (do inglês *source*) um determinado agente. O presente exemplo assume a hipótese do mundo aberto. Vale observar que essas anotações podem ser removidas sem nenhum problema adicional, entretanto, se isso for feito assume-se a hipótese do mundo fechado e não seria possível ter um novo agente que o *porter* ignora.

A execução dos agentes acontece de forma arbitrária e deve-se ter em conta que o primeiro agente que irá enviar a mensagem para o agente *porter* dependerá de como o mundo inicia. Logo, se o mundo iniciar com a porta fechada o primeiro a enviar a solicitação será o agente *claustrophobe*. Já se o mundo for iniciado com a porta aberta o primeiro a enviar solicitação será o agente *paranoid*. Depois disso, conforme a simulação vai correndo, os dois agentes ficam alternando mensagens com o agente *porter* por causa do compartilhamento do mesmo recurso (a porta).

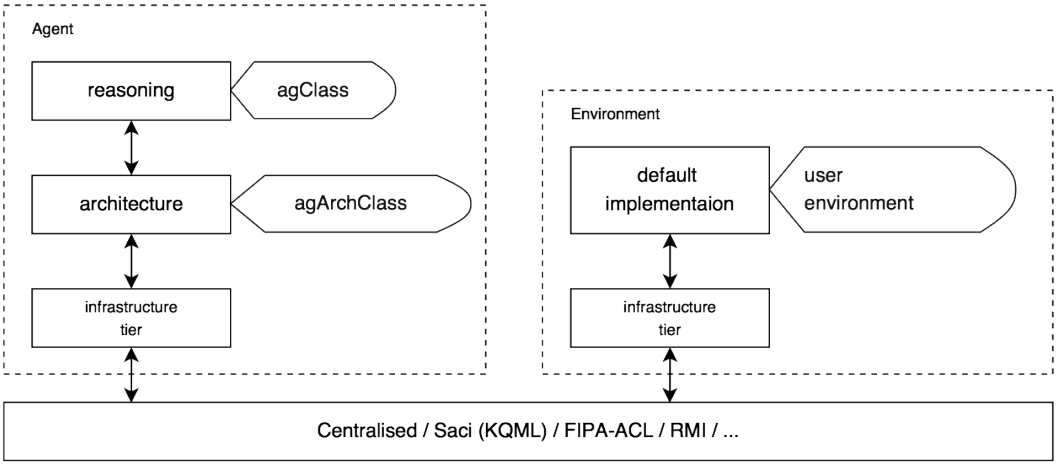

Figura 2.1: Modelo da infraestrutura do *Jason*.

2.1.3.2 Infra-estrutura do Jason

A plataforma *Jason* tem uma base de software completamente extensível como é possível observar mediante a Figura 2.1. Nessa figura, a infraestrutura encontra-se representada como um barramento na parte inferior. Ela pode ser configurada através da chave de configuração *infrastructure* no arquivo de projeto (ver Listagem 2.1).

A infraestrutura define todas as classes padrões que o usuário irá utilizar, por exemplo observe que o barramento liga-se a dois adaptadores: um do tipo agente e um de ambiente. Esses adaptadores permitem a compatibilidade entre implementações distintas permitindo que ocorram variações sem ter que alterar a estrutura do barramento.

Assim, há a possibilidade de informar para uma determinada simulação que um determinado agente utilizará uma arquitetura e/ou um raciocinador diferente dos demais agentes. Para se fazer isso no momento que se declara os agentes no projeto informa-se as chaves que encontram-se dentro das caixas com cantos arqueados na Figura 2.1. O exemplo “fb agArchClass KosMos.FireBrigadeArch agClass KosMos.Agent #1;” define um (#1) agente *fb* com a arquitetura usando a classe *FireBrigadeArch* do pacote *KosMos* (pode ser qualquer nome desejado) e configurando o raciocinador a partir da classe *Agent* do mesmo pacote. Note que, essas classes não existem no *Jason* e devem ser providas pelo usuário de alguma forma.

Como já explicado, a entrada *environment* no arquivo de projeto configura a classe de ambiente que será utilizada e somente um é permitido por simulação. Um dos motivos comuns de se implementar um ambiente é implementar as ações que o agente poderá realizar no mesmo. Aliás, também, é possível alterar como os agentes percebem o meio para, por exemplo, inserir percepções incorretas em alguns momentos.

Essa infraestrutura extensível permitiu a implementação do trabalho de Moreira et al. (2006) por Klapiscak e Bordini (2009) sem nenhuma alteração profunda nas classes da

plataforma. Esse trabalho permitiu que o *Jason* usa-se ontologias, inclusive definindo conceitos e compartilhando o conhecimento. Por exemplo, um agente pode dizer para outro que a informação que ele deseja encontra-se em determinada ontologia. Além disso, há outras extensões como não usar arquivos de projeto e sim uma base de conhecimento baseada em ontologia para definir a simulação.

O presente trabalho segue uma ideia parecida desses dois anteriores. A principal preocupação era ter as crenças que a ontologia considere relevantes em sua base de conhecimento e as outras guardadas na base padrão. Isso não é diferente no trabalho anterior, o que é diferente é o como a relevância é verificada. No trabalho anterior, o que era salvo na base era tudo que estivesse com a anotação usada para indicar a ontologia e no presente desenvolvimento isso é extraído da própria. Assim, se na ontologia contiver o conceito Pizza e inserimos uma crença "pizza(p1)". O mecanismo desenvolvido sabe que essa crença é relevante para a ontologia. Já o do trabalho anterior para saber que o mesmo é relevante precisa ser marcado com uma anotação, por exemplo "pizza(p1)[ontology(URL)]". Além disso, ele não lida com emoções e no presente trabalho os agentes podem perceber seus sentimentos.

2.2 Computação Afetiva

Picard (1998) definiu Computação Afetiva como uma "computação relacionada, surgida ou que influência as emoções". Além disso, computadores com emoções permitem aos mesmos um determinado nível de comportamento inteligente e criatividade que seria impossível sem as emoções e esse é o principal desafio dessa área. Logo, o seu entendimento pode explicar fenômenos como, por exemplo, atenção, memória e outros.

Essa área é normalmente dividida em duas subáreas. A primeira estuda o reconhecimento e a expressão de emoções dentro da IHC; a segunda, foca na síntese de emoções para aprimorar os seres robóticos e/ou para estudar o comportamento humano por meio de simulações. Há muita aplicabilidade dessas técnicas, por exemplo: reconhecer emoções pode ser utilizado para adaptar o sistema ao estado da pessoa permitindo ao mesmo instruí-la, questioná-la, encorajá-la ou ocultar informações irrelevantes.

O objetivo de Bickmore (2003) com o projeto *Relational Agents* é possibilitar aos usuários a criação de um relacionamento social e emocional de longa duração. Em Bickmore e Schulman (2009), a confiança no agente torna possível discutir tarefas mais importantes como melhoria da saúde ou até a compra de uma casa. Outro trabalho na área de IHC é o reconhecimento de emoções para aumentar a imersão em jogos, por exemplo permitindo ao próprio jogo adaptar eventos ou trechos tornando-o mais divertido.

O projeto *The Affective Tigger: a reactive expressive toy* de Kirsch (1999) é um brinquedo capaz de reconhecer e reagir às emoções exibidas pelas crianças. Por exemplo, quando a criança encontra-se feliz, o boneco expressa felicidade (ver Figura 2.2). Ao

Figura 2.2: Brinquedo que responde as emoções das crianças (KIRSCH, 1999).

todo existem 5 estados emocionais: muito feliz, feliz, neutro, triste e muito triste. Todos, com exceção do neutro, possuem alguma síntese vocal como um rosnado (tristeza) ou uma risada (muito feliz). Assim, esse brinquedo, por ser considerado um ser robótico que reage à criança com seus próprios estados emocionais, fica enquadrado na segunda área.

O projeto AIDA[2] (do inglês *Affective Intelligent Driving Agent*) pode ser entendido como enquadrado na área de IHC, pois o interesse é entender o estado afetivo da pessoa dirigindo. Além disso, interessa-se em ter um relacionamento com o usuário sugerindo alterações nas rotas baseando-se na rotina aprendida depois de um tempo de aprendizado. A pesquisa relatada em Dias e Paiva (2009) visou melhorar a simulação de agentes através do uso da emoção guiando o processo deliberativo e melhorando o entendimento e gerência das emoções, portanto, fica enquadrado na área de síntese de emoção. O presente trabalho se enquadra na área de síntese de emoção, pois o interesse é em entender e gerar estados emocionais e como esses podem afetar o comportamento de um personagem.

2.2.1 Modelo Emocional

Estudos neurológicos recentes (LEDOUX, 1998; DAMÁSIO, 2004) mostram a importância das emoções na tomada de decisão. Damásio (2004) definiu emoção como sendo um estado físico do corpo que se altera de forma contínua. Sendo assim, o sentimento foi definido como a percepção dessas alterações.

Na psicologia há diferentes modelos que tentam explicar a afetividade. Borod (2000) categorizou esses modelos afetivos em quatro categorias principais: modelos dimensionais, modelos discretos, modelos baseados em significados e modelos baseados em componentes. A primeira categoria visa descobrir variáveis que representam eixos das classes emotivas e estabelecem meios de se mover por esses eixos. A seguinte, especifica um conjunto básico de emoções e regras de evolução para esse conjunto. Já a categoria dos baseados em significados se preocupa com as situações em que o sentimento foi ocasionado e tenta descrever as estruturas semânticas dos mesmos. A última, entende que os sentimentos são aprendidos ao longo do tempo. Sendo assim, os modelos baseados em

[2]Mais detalhes, ver http://senseable.mit.edu/aida

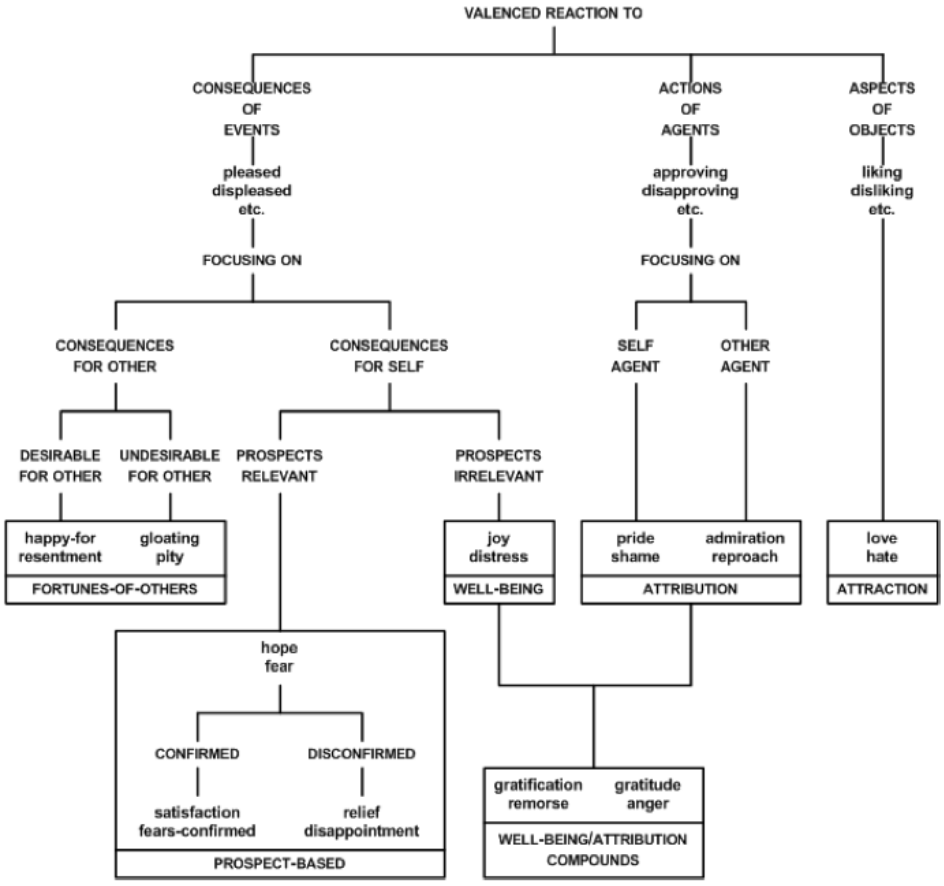

Figura 2.3: Estrutura de emoções (ORTONY; COLLINS; CLORE, 1988).

componentes estudam o elo entre os sentimentos e as suas situações. Esse elo é montado de diferentes formas e varia de pessoa para pessoa.

Um modelo baseado em significados, bastante conhecido na Inteligência Artificial, foi definido por Ortony, Collins e Clore (1988). Nesse modelo[3] são descritos 22 emoções com suas situações. Essas emoções são divididas em formas de se perceber o mundo a sua volta por consequências (importância das metas), ações (responsabilidade) e objetos (atração ou repulsa). Assim sendo, essas maneiras refletem diferentes jeitos de se analisar as situações que podem ser relativas aos objetivos, valores morais ou gostos da pessoa.

A Figura 2.3 resume esse modelo e mostra as percepções possíveis de um indivíduo. Partindo da direita para esquerda, o ramo mais básico, *Aspects Of Objects*, é ativado quando se avalia o gosto de alguém para algum objeto (inanimado ou não). Por exemplo, Millie gosta de rosas azuis. No seguinte, *Actions Of Agents*, o julgamento das ações exercidas por outro indivíduo ou por si mesmo é baseado nos valores morais da pessoa que está julgando. Exemplo: reprovar a atitude dos bancários que fazem greve a cada ano. Cabe salientar, que ao julgar ações, o modelo permite um grau de "empatia" chamado pelos autores de "força de unidade cognitiva"[4]. Dessa forma, é possível, por exemplo, ficar

[3] A partir daqui o modelo será referenciado por modelo *OCC*.
[4] Traduzido literalmente de *Strength of cognitive unit*.

com orgulho porque uma atleta ganhou uma medalha ou ficar envergonhado ao descobrir que o vizinho bate no(s) filho(s).

O último ramo da árvore, mais a esquerda na Figura 2.3, é o *Consequences Of Events* que representa as coisas que aconteceram (e foram consideradas importantes), acontecem ou podem acontecer (objetivos almejados). Essas emoções são avaliadas segundo as suas consequências para o alcance ou impedimento dos seus objetivos. Exemplos possíveis desse ramo do modelo são a emoção sentida ao receber uma boa nota em um teste, ao ser assaltado ou ao perceber seu voo ser cancelado por algum problema não esperado.

Todas as emoções do modelo trabalham com duas intensidades. A intensidade da emoção que representa o físico e a intensidade do sentimento que representa o quanto o agente esta percebendo daquela emoção. Dessa forma, um indivíduo só possui sentimento quando a intensidade da emoção ultrapassa um determinado limite. Essa intensidade é obtida por uma função matemática que utiliza variáveis de dois tipos: locais, que influenciam as emoções do ramo específico; e globais, que influenciam todas as emoções do modelo. Um exemplo de variável local é o desejo, enquanto um exemplo de variável global pode ser o senso de realidade de uma pessoa.

Bates (1994) foi um dos primeiros a trabalhar na utilização de emoções na área de animação. Nessa área, o estudo do comportamento humano é realizado visando imitar as ações humanas. Assim, a principal afirmativa do trabalho era que o comportamento emotivo de um personagem é um papel importante para que o mesmo pareça ter vida própria. Dessa forma, esse trabalho utilizou o modelo descrito visando melhorar a credibilidade de seus atores. Por exemplo, um dos agentes lida com o medo sendo agressivo com os outros enquanto outro agente lida com a mesma emoção sendo retraído.

Visando entender melhor o impacto da emoção na tomada de decisão, Zhang et al. (2009) desenvolveram uma aplicação que os sentimentos afetam o planejamento das ações à serem realizadas. Neto e Silva (2010) focaram no mesmo objetivo, porém visando estudar o impacto da memória no planejamento. Sendo assim, um meio para o agente "esquecer" determinadas crenças quando o estado emocional for diferente daquele guardado anteriormente foi realizado. Eles acreditam que essa característica torna o planejamento e as atitudes dos personagens virtuais mais realista.

Um dos trabalhos mais conhecidos baseado no modelo *OCC* é, sem dúvida, o de Kshirsagar (2002). Ele utilizou as emoções levantadas no modelo em conjunto com um modelo de personalidade baseado na psicologia que leva em consideração 5 fatores: extroversão, agradabilidade, conscientização, neurose e receptividade. O primeiro, descreve a preferência para o comportamento em situações sociais. O seguinte, a interação com os outros indivíduos. A conscientização é a organização e persistência das metas. A tendência de pensamentos negativos é a neurose ou fator neurótico. Por fim, o último, descreve se a pessoa tem interesse em cultura ou é "cabeça aberta" para novas ideias.

2.2.2 Ontologias Afetivas

De acordo com Gutiérrez et al. (2007), uma ontologia possui inúmeras utilidades tanto em pesquisa quanto na indústria. Na pesquisa o foco é uma melhor descrição do domínio propriamente dito, enquanto na indústria o foco é a melhor utilização dos recursos de seus colaboradores. As ontologias emocionais visam descrever emoções ou aspectos afetivos de um indivíduo se baseando ou não em estudos da psicologia.

Em Benta, Rarău e Cremene (2007) foi feita a construção de uma ontologia escrita em *OWL*. Nesse trabalho as emoções são divididas em primárias e secundárias, as secundárias se originam a partir das primárias. As emoções primárias, não cognitivas, são: *Angry*, *Disgust*, *Fear*, *Happy*, *Neutral*, *Sad* e *Surprise*. As emoções secundárias, cognitivas, descritas são ao todo 4. O interessante aqui é que essas 4 emoções são inferidas a partir das anteriores. Além disso, há o conceito de emoção ativa que é a emoção predominante naquele momento. O valor da emoção é calculado da seguinte forma: a sensibilidade (predisposição a emoção varia de 0 à 1) multiplicado pela intensidade da emoção. A emoção predominante é o maior valor entre as emoções.

No modelo *OCC* a distinção entre emoções primárias e secundárias não existe porque se pressupõe que toda emoção exige um certo nível de cognição. Não existe, também, um limite no que pode ou não ser percebido em quantidade de emoções no momento. Mas, existe um limite de perseguir uma meta por vez. Fora isso, se pode pensar que emoções opostas compartilham os mesmos atributos e, por isso, essas emoções não serão sentidas ao mesmo tempo.

Não tendo nenhuma informação de uma teoria de emoções específicas modeladas em sua ontologia, López et al. (2008) criaram uma ontologia de alto nível se aproveitando de outra ontologia de alto nível e de uma de analise léxica. Assim, o principal conceito desse trabalho é o de sensor que é um objeto físico no ambiente e que recebe as informações do meio e as "transportam" para o mundo mental do agente. Sendo assim, é possível reconhecer situações similares e descrever novas.

Hwang e Yang (2009) desenvolveram um motor de emoções que utiliza um modelo de mistura de emoções em conjunto com o modelo *OCC*. O trabalho deles utilizou o conceito de camadas onde cada camada tem uma responsabilidade distinta e que visa complementar a anterior. Essas são ao todo quatro: classificação, interação, mapeamento e expressão. A primeira visa determinar que categoria ou ramo será afetado e determina a intensidade da emoção. A de interação analisa os efeitos nas categorias emocionais do personagem. A próxima, mapeia as 22 emoções do modelo para pelo menos uma expressão. A expressão propriamente dita é feita pela última camada.

Esse trabalho ainda utilizou um modelo dimensional para misturar as emoções primárias. Assim, as emoções secundárias podem ser descobertas a partir do nível dos eixos afetados. O trabalho mostra quase todas as emoções do ramo de consequência de eventos com exceção das emoções de *Hope* e *Fear*. As emoções primárias são as emoções do

modelo *OCC* e como secundárias estão as emoções construídas a partir da mistura dessas emoções. Essa diferenciação, como dito anteriormente, não existe no modelo *OCC*.

Um modelo genérico que representa o ambiente e eventos que estão a volta de um personagem, sua personalidade e suas preferências foi feito por Lera et al. (2008). Nesse trabalho o foco eram emoções que podem ser representadas no rosto. Assim, a identificação do contexto é feita através de eventos e do retorno afetivo. Fora isso, as expressões faciais são modificadas dependendo dos eventos do ambiente e, também, da personalidade, metas e preferências dos atores virtuais.

Adam, Herzig e Longin (2009) formalizaram o modelo *OCC* de maneira lógica. Esse trabalho descreve a probabilidade de maneira comparativa, isto é, o evento A é mais provável que o evento B. Além disso, há uma ordem temporal nas ações sendo desempenhadas. Os autores propõem uma diferenciação entre ação e evento. O primeiro é causado intencionalmente pelo agente, enquanto o segundo o agente não tem controle da ação. Por exemplo, ir para aula é uma ação que pode ser julgada pela responsabilidade e espirrar é um evento que só pode ser julgado pela sua consequência. Outra formalização foi feita por Steunebrink, Dastani e Meyer (2010) para as emoções de esperança, medo, alegria e sofrimento. Esse modelo foi testado em linguagem *2APL* e restringe as escolhas dos planos[5] possíveis no momento que o agente esta decidindo que comportamento adotar. Por exemplo, o agente pode revisar seus objetivos quando estiver com medo.

2.2.3 Arquiteturas Emocionais

Elliott (1992) em sua tese "O raciocinador afetivo" foi um dos primeiros a trabalhar com emoções na área de sistemas multi-agentes. Em seu modelo, 24 tipos de emoções são usadas, isto é, duas à mais que o modelo *OCC* por causa que ele dividiu as emoções do ramo de objetos para diferenciar amor de gostar e ódio de não gostar. Nesse trabalho, um mesmo evento pode gerar mais de uma emoção ao mesmo tempo. Por exemplo, estar feliz por que recebe um bom salário e triste porque o trabalho é chato. Além disso, os agentes podem observar os outros e explicar situações usando regras emotivas e um sistema de classificação baseado em casos.

Já Gratch (2000) focou a sua arquitetura emocional diretamente nos planos e metas dos agentes. O modelo criado por ele permite agentes avaliarem o significado emocional de eventos que se relacionam com os planos e metas, modelando e predizendo os estados afetivos de outros agentes e alterando o comportamento de acordo. Para isso um sistema baseado em casos também foi utilizado para determinar a existência de relações e se existirem elas são reconhecidas por determinadas características. O trabalho estava restrito a eventos comunicados ou ações dos agentes e utilizou apenas cinco emoções: esperança, alegria, medo, sofrimento e raiva.

Gratch e Marsella (2004) utilizaram os dois primeiros trabalhos explicados anteriores

[5] Ver página 12

e ampliaram as emoções possíveis e a forma de avaliação que o agente realiza. Eles explicaram um processo de avaliação no qual observa-se o ambiente para configurar "variáveis de avaliação" e, após, o processo de imitação acontece. O processo de imitação pode ser de duas formas, ser focado no problema ou na emoção. Se for feita uma solução focada no problema o autor apontou várias estratégias possíveis: cópia ativa (tentar fazer), buscar suporte (procurar conselheiro, informação, etc) e planejar. Se for adotada uma solução emocional tem-se ao todo 11 estratégias possíveis. Além disso, o trabalho menciona que o agente mantém a história do que aconteceu, esta acontecendo e pode vir a acontecer para ser consultada durante seu planejamento. O trabalho usa como exemplo um médico que tem que decidir aplicar um medicamento agora e fazer o paciente morrer mais tarde ou deixa-lo morrer agora.

O trabalho *FearNot* foi desenvolvido visando reduzir o bullying nas escolas (DIAS; PAIVA, 2005). As crianças são expostas as cenas de bullying e fazem o papel do amigo imaginário da vitima que pode sugerir formas de ação para as situações. A arquitetura criada foi pensada tendo em mente: ter capacidades reativas e cognitivas, gerar credibilidade e empatia, possuir interação com o usuário (note que os agentes não seguem cegamente o que o usuário diz porque isso baixaria a credibilidade) e ser independente de domínio. O processo de deliberação do agente é focado na avaliação feita a partir do que se percebe do ambiente em níveis reativos e deliberativos (emoções com probabilidade). Depois, o estado emocional é gerado e a memória atualizada paralelamente. Esse trabalho também utiliza um processo de imitação que pode ser no nível reativo (tendências de ações) ou deliberativo (foco no problema ou emoção). Dessa forma, de maneira similar com o trabalho anterior, quando o processo de imitação é deliberativo uma reavaliação do problema é feita que pode afetar o estado emocional ou o entendimento do problema.

Todos os trabalhos anteriores descrevem um processo de imitação que é o responsável pela estrategia mental do ator. Por exemplo, fazer, replanejar ou reavaliar a prioridade. Quando essas estratégias são decididas emoções podem acontecer, todavia o presente trabalho não possui um modelo mental do ator. A arquitetura proposta (cap. 3) depende que o usuário defina como uma percepção se ligará ao agente e isso pode determinar uma emoção. Mesmo sendo isso sendo um ponto fraco do trabalho, isso aumenta o controle sobre o mesmo.

3 ARQUITETURA EMOCIONAL CONSTRUÍDA

3.1 Visão Geral

A presente arquitetura construída foi pensada conforme mostrado na Figura 3.1. Como é possível observar, o sistema é dividido em um ambiente e em agentes. O ambiente utiliza um modelo para descrever o comportamento urbano permitindo a descrição das rotinas para construir às rotinas dos atores no ambiente e uma visualização para o usuário. Assim, o que o usuário vê é exibido a partir da descrição ontológica. Mais detalhes ver seção 3.3 e o capítulo 4. Já a ontologia de preferências, explicada na seção 3.4, permite guardar junto dos objetos informações e especificar como o ator é atraído por essas. Além disso, o ambiente é o responsável por informar ao agente via percepção essas preferências e as rotinas que o agente possui para o dia sendo simulado.

A segunda parte do sistema estende os agentes focando em dois objetivos. Os agentes recebem um conjunto de percepções referente ao passo de simulação atual e conhecem seu conjunto de crenças do momento. O primeiro objetivo é permitir aos agentes raciocinarem sobre as anotações percebidas nos objetos. O segundo objetivo é alterar o local onde as crenças que o agente possui são armazenadas permitindo ao agente ter ou não crenças salvas dentro de uma base de conhecimento. Assim, quando as crenças são salvas dentro da ontologia proposta da seção 3.2 emoções podem ser geradas e percebidas.

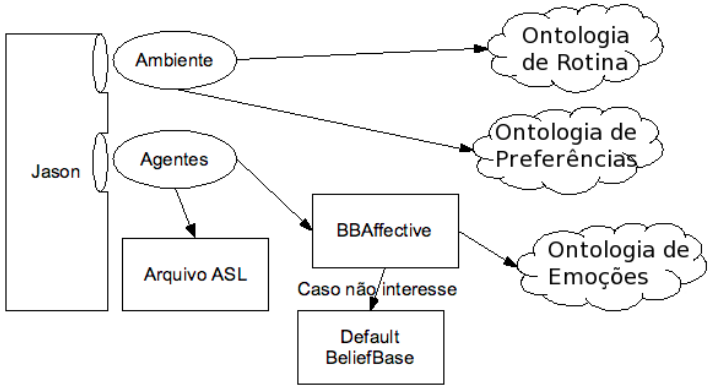

Figura 3.1: Visão abstrata do sistema construído.

Conforme definido nos capítulos anteriores, o agente é a mente de um ator ou personagem que atua em um mundo virtual. Entretanto, qualquer elemento do mundo virtual que não é um ator esta sendo considerado como um objeto. Uma avaliação foi definida como sendo o julgamento de um agente sobre algum ator conhecido ou sobre algum objeto que foi ou esta sendo percebido. Dessa forma, uma avaliação afetiva é uma avaliação que foi feita que ocasionou algum impacto emocional no agente. Cabe chamar atenção que emoção esta sendo considerada como sinônimo para avaliação emotiva.

3.2 Ontologia do Modelo Cognitivo Emocional

A fundamentação do modelo afetivo sendo utilizado aqui é o proposto por Ortony, Collins e Clore (1988) e encontra-se explicado na seção 2.2.1. A estrutura da ontologia pode ser visualizada na Figura 3.2 e é recomendável olhar a Figura 2.3 (pág. 19) durante o resto da discussão dessa seção. O conceito *Emotion* corresponde a uma avaliação emotiva do personagem e seus subconceitos correspondem aos três ramos do modelo original. O ramo *ActionsOfAgents* julga a responsabilidade e o quanto o agente que realizou uma ação se desviou do esperado, o de *ConsequencesOfEvents* julga a consequência de um evento e *AspectsOfObjects* julga a atração para com um objeto.

Figura 3.2: Taxonomia da ontologia proposta baseado no modelo *OCC*.

O primeiro ramo a ser abordado é o menos cognitivo *AspectsOfObjects*. As emoções desse tipo são relacionadas com atratividade e familiaridade. A emoção *Hate* é modelada como tendo a propriedade de familiaridade (*hasFamiliarity*) e de atração (*hasAttraction*) com valores negativos, enquanto a emoção *Love* tem valoração dessas mesmas propriedades positivas. Caso o valor seja zero em ambas ou uma tenha valor positivo e outra possua valor negativo, nada pode ser concluído.

Cabe notar que parece estranho uma emoção *Love* com um objeto, porém essa emoção foi escolhida por quem montou o modelo para representar seu tipo por ser a mais forte de sua categoria. Assim, níveis menores implicam em outros tipos de emoção. Além disso, agentes são vistos

como objetos quando se esta avaliando a sua atração. Logo, todo agente (*Agent*) é um objeto (*Object*). Por exemplo, John esta apaixonado pela Millie (Millie é avaliada como objeto) ou John tem repulsa por televisão.

O segundo conceito, chamado *ActionsOfAgents*, pode ser pensado como um ramo que julga a responsabilidade por uma determinada ação. Logo, esse ramo é capaz de gerar emoções de: Admiração (*Admiration*), Orgulho (*Pride*), Vergonha (*Shame*) e Reprovação (*Reproach*). Por exemplo, Jose possui orgulho por cozinhar ou Dilu reprova Jose porque ele come carne.

Na definição, as emoções de orgulho e vergonha podem acontecer mesmo quando se esta avaliando ações de outras pessoas. Por exemplo, Dolores tem vergonha de sua mãe que não cozinha. Essa conclusão é possível por causa de uma relação proposto no modelo *OCC* de empatia. Entretanto, como em nenhum outro momento eles dão mais detalhes sobre essa empatia foi resolvido considerar que vergonha e orgulho são emoções sentidas somente quando o agente esta avaliando a si mesmo e, dessa forma, o exemplo anterior não é possível no sistema desenvolvido.

As emoções que julgam responsabilidade são definidas como tendo uma relação de julgamento (*hasJudge*) e uma relação numérica que mapeia o valor (*hasJudgeness*) que representa o quanto o agente se desviou do comportamento esperado, isto é, em casos de aprovação é um valor positivo e em casos de reprovação é um valor negativo. Todavia, isso ainda não permite diferenciar a emoção de admiração da emoção de orgulho ou a reprovação da de vergonha. Essa distinção é possível ao se dividir a relação de julgamento com duas subrelações: tem auto julgamento (*hasJudgeMyself*) e tem julgamento sobre outro (*hasJudgeOther*).

A utilização de subpropriedade torna possível escrever a ontologia da maneira esperada suprimindo o problema. Entretanto, para o usuário pode se tornar complicado ter que lembrar quando utilizar uma subpropriedade ou outra. Assim, foi resolvido deixar o usuário sempre utilizar a relação de julgamento (*hasJudge*) e via 2 regras descobrir se é um julgamento sobre outra pessoa ou sobre si mesmo. Para essas regras funcionarem da maneira correta, o usuário deve declarar que os agentes ou objetos são diferentes uns dos outros. Caso isso não ocorra, o sistema considera que não há informação para verificar se um indivíduo é igual ou diferente e conclui que não conhece a resposta. Além disso, a relação de julgamento permite que tantos objetos quanto agentes sejam julgados porque possui como imagem o conceito *Object*.

Cabe salientar que toda avaliação tem pelo menos duas relações. A primeira relação serve para conhecer quem está avaliando (*isAppraisalOf*) e a seguinte serve para indicar quem ou o que esta sendo avaliado (*hasSomething*). Essa última pode não ser informada explicitamente porque pode ser concluída quando usada uma de suas subrelações. O último ramo, chamado de *ConsequencesOfEvents* é dividido em: *ConsequencesForSelf* e *ConsequencesForOthers*. Toda essa divisão foca na consequência de um evento reali-

zado por algum agente. Por exemplo, Dilu tem pena de Jose, Jose tem esperança de ser promovido, John tem satisfação por estar almoçando ou Millie esta alegre por cozinhar.

A *ConsequencesForOthers* expressa 4 emoções: *Gloating*, *HappyFor*, *Resentment* e *SorryFor*. Na definição, essas emoções dependem: do grau de desejabilidade do avaliador para com o outro; do grau de desejabilidade que se presume que o outro tenha; do grau de merecimento do evento; e, do tipo de relacionamento com a pessoa. Na ontologia proposta, a principal diferença com o modelo *OCC* é que foi considerado que o grau de desejabilidade do avaliador para com o outro e o grau de merecimento do evento são os mesmos. Dessa forma, se pode utilizar apenas três relações para descrever as 4 emoções.

A relação de merecimento (*hasDeserved*) e de desejabilidade presumida (*hasDesireOther*) são avaliadas de acordo com sua valoração positiva ou negativa. Entretanto, a relação *hasPerson* liga o indivíduo sendo avaliado com a avaliação. Para se ter o conhecimento do que esta sendo julgado ser amigo (*GoodKnowRelation*) ou inimigo (*BadKnowRelation*), esses conceitos foram criados e precisam ser configurados para cada um dos agentes em questão, porém quem precisa dessa informação é o conceito de avaliação quando as relações *hasPersonEnemy* e *hasPersonFriend* precisam ser descobertas. O agente pode declarar que só conhece uma pessoa, que conhece e é um amigo ou que conhece e não gosta dela (inimiga). Veja a Figura 3.4 (pág. 30) para mais detalhes.

ConsequencesForSelf se divide entre consequências de eventos com probabilidade (*ProspectRelevant*) ou sem probabilidade (*ProspectIrrelevant*). Note que esses dois conceitos se relacionam com probabilidade (*hasLikelihood*). O primeiro conceito se relaciona com a parte não nula, enquanto o outro se relaciona somente com valores nulos. Dessa forma, ambos os conceitos são disjuntos. A classe com probabilidade pode ser dividida ainda entre não realizada (*ProspectNotRealized*) e realizada (*ProspectRealized*).

As emoções *Fear* e *Hope* fazem parte do conceito *ProspectNotRealized*. Esse conceito usa as relações *hasLikelihood* e *hasDesireSelf*. Essa última é um número que representa o desejo de se obter ou repudiar o evento. Além disso, quando o evento ocorre, a emoção atual pode virar uma emoção do conceito *ProspectRealized*, isto é, *Fear* pode virar ou *FearsConfirmed* ou *Relief* e *Hope* pode virar ou *Satisfaction* ou *Disappointment*.

O conceito *ProspectRealized* não se relaciona em nenhum momento com a relação *hasLikelihood* porque o evento já aconteceu ou não vai mais acontecer. Assim, ele possui três relações distintas das anteriores, a primeira é o grau de realização do evento (*hasRealized*), isto é, a visão do agente sobre como a consequência do evento aconteceu. A segunda relação *hasPreviousIntensity* recebe a valoração da emoção *Fear* ou *Hope* do evento que tinha probabilidade e serve para saber se o evento era um evento bom (*Hope*) ou ruim (*Fear*). Já a terceira propriedade (*hasEffort*) tenta estimar o grau de esforço que foi despendido para a atração ou repulsa da consequência.

O conceito *ProspectIrrelevant* é parecido com o conceito *ProspectRelevant* com a diferença que a relação *hasLikelihood* vai somente para valores nulos. Fora isso, as emoções

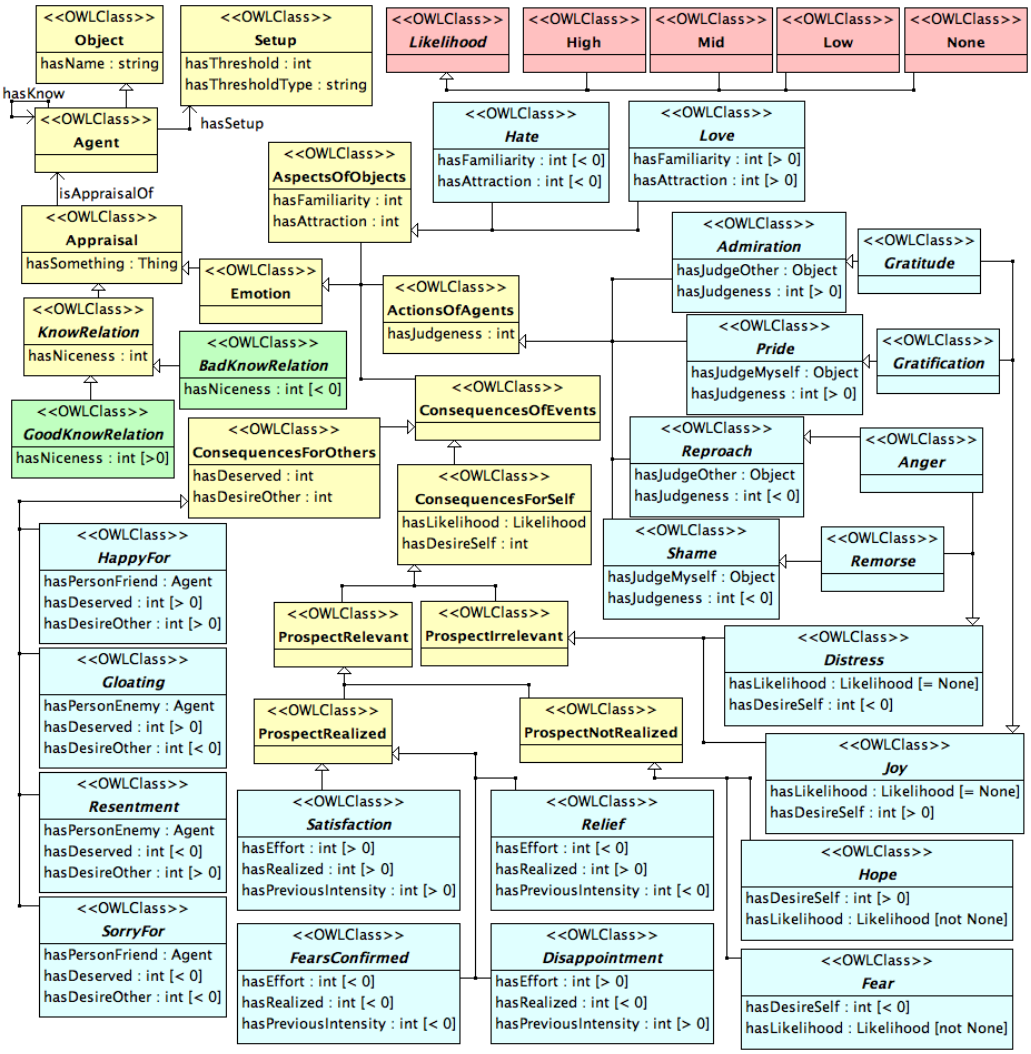

Figura 3.3: Ontologia afetiva proposta.

Tabela 3.1: Regras de classificação de indivíduos da classe de avaliação emotiva.

Nome	Regra
Fear	(hasLikelihood some (Likelihood and (not (None)))) and (hasDesireSelf some int[< "0"^^ int])
Hope	(hasLikelihood some (Likelihood and (not (None)))) and (hasDesireSelf some int[> "0"^^ int])
FearConfirmed	(hasEffort some int[< "0"^^int]) and (hasPreviousIntensity some int[< "0"^^int]) and (hasRealized some int[< "0"^^int])
Satisfaction	(hasEffort some int[> "0"^^int]) and (hasPreviousIntensity some int[> "0"^^int]) and (hasRealized some int[> "0"^^int])
Disappointment	(hasEffort some int[> "0"^^int]) and (hasPreviousIntensity some int[> "0"^^int]) and (hasRealized some int[< "0"^^int])
Relief	(hasEffort some int[< "0"^^int]) and (hasPreviousIntensity some int[< "0"^^int]) and (hasRealized some int[> "0"^^int])
Gloating	(hasPersonEnemy some Agent) and (hasDeserved some int[> "0"^^int]) and (hasDesireOther some int[< "0"^^int])
HappyFor	(hasPersonFriend some Agent) and (hasDeserved some int[> "0"^^int]) and (hasDesireOther some int[> "0"^^int])
Resentment	(hasPersonEnemy some Agent) and (hasDeserved some int[< "0"^^int]) and (hasDesireOther some int[> "0"^^int])
SorryFor	(hasPersonFriend some Agent) and (hasDeserved some int[< "0"^^int]) and (hasDesireOther some int[< "0"^^int])
Distress	(hasLikelihood some None) and (hasDesireSelf some int[< "0"^^int])
Joy	(hasLikelihood some None) and (hasDesireSelf some int[> "0"^^int])
Hate	hasFamiliarity some int[< "0"^^int]
Love	hasFamiliarity some int[> "0"^^int]
Admiration	(hasJudgeOther some Object) and (hasJudgeness some int[> "0"^^int])
Pride	(hasJudgeMyself some Object) and (hasJudgeness some int[> "0"^^int])
Reproach	(hasJudgeOther some Object) and (hasJudgeness some int[< "0"^^int])
Shame	(hasJudgeMyself some Object) and (hasJudgeness some int[< "0"^^int])
Anger	Distress and Reproach
Gratification	Joy and Pride
Gratitude	Joy and Admiration
Remorse	Distress and Shame

```
GoodKnowRelation(?ba), hasPerson(?ba, ?at), isAppraisalOf(?ba, ?ao) -> hasFriend(?ao, ?at)
hasJudge(?a, ?j), isAppraisalOf(?a, ?k), DifferentFrom (?j, ?k) -> hasJudgeOther(?a, ?j)
hasEnemy(?ag, ?at), hasPerson(?ap, ?at), isAppraisalOf(?ap, ?ag) -> hasPersonEnemy(?ap, ?at)
BadKnowRelation(?ba), hasPerson(?ba, ?at), isAppraisalOf(?ba, ?ao) -> hasEnemy(?ao, ?at)
hasJudge(?a, ?p), isAppraisalOf(?a, ?p) -> hasJudgeMyself(?a, ?p)
hasFriend(?ag, ?at), hasPerson(?ap, ?at), isAppraisalOf(?ap, ?ag) -> hasPersonFriend(?ap, ?at)
KnowRelation(?ba), hasPerson(?ba, ?at), isAppraisalOf(?ba, ?ao) -> hasKnow(?ao, ?at)
```

Figura 3.4: Regras da ontologia proposta.

desse conceito e do *ActionsOfAgents* podem ser misturadas formando conceitos compostos. A composição é quando uma emoção pode ser encaixada em mais de uma emoção como no caso de se estar alegre (*Joy*), orgulhoso (*Pride*) e gratificado (*Gratification*).

O conceito *Setup* foi introduzido para manter junto da ontologia criada o limite mínimo para uma emoção virar sentimento e é usado como um conceito de ligação com as demais ontologias. A Figura 3.3 (pág. 28) resume as definições utilizadas para classificar as avaliações como emoções, a Tabela 3.1 (pág. 29) mostra as regras suficientes para ser categorizado como uma avaliação emotiva e a Tabela 3.2 mostra as estatísticas da ontologia.

A Figura 3.4 mostra todas as regras existentes. Essas 7 regras visam facilitar o usuário e a maior parte delas operam no domínio do conceito *Appraisal*. As regras *hasKnow*, *hasFriend* e *hasEnemy* são uma exceção porque operam no domínio e na imagem do conceito *Agent*. Além disso, a finalidade dessas é permitir que um agente conheça o tipo de relacionamento que ele possui com outro agente a partir das avaliações feitas sobre o conceito *KnowRelation*.

Tabela 3.2: Ontologia proposta com expressividade: ALCHIN(D).

Descrição	Quantidade
Classes	45
Propriedade de Objetos	16
Propriedade de Dados	14
Indivíduos	0
Regras	7

Comparando essas regras mencionadas na figura anterior com as relações existentes na ontologia mostradas na Figura 3.5. A conclusão é que ao invés de 16 propriedades de objetos apenas 9 precisam ser conhecidas. Dessas 9, a propriedade menos utilizada é a *hasSomething* que serve para indicar genericamente o que esta sendo avaliado. Note que a relação *hasPerson* deve ser usada quando o indivíduo em avaliação for um membro da classe *Agent*. Já, a relação *hasJudge* serve para indicar que o membro da classe *Object* esta sendo avaliado pela sua responsabilidade. Por exemplo, Millie tem uma avaliação

julgando seu carro positivamente.

Toda a valoração numérica existente na ontologia é inteira. Isso foi feito com a finalidade de permitir que o usuário normalize o número obtido da maneira que desejar. Além disso, foi tomada a decisão de não especificar o domínio da maioria das propriedades porque isso forçaria um enquadramento em classes não desejadas. Por exemplo, se a relação *hasLikelihood* tiver o domínio *ConsequenceForSelf* e existir um indivíduo com somente essa relação então o mesmo seria enquadrado no conceito *ConsequenceForSelf*. Mas, o correto nesse caso seria não ser concluído nada, ou seja, pertencer à classe *Thing*.

3.3 Um Modelo de Comportamento Urbano

Um modelo de comportamento urbano descreve a vida normal que um personagem possui. De outro modo, cada personagem possui lugares regulares que costuma visitar frequentemente e lugares eventuais que visita com menos frequência. Dessa forma, a rotina que um ator possui é descrita através dos locais que ele deve ou pode visitar.

A Figura 3.6 demonstra um modelo baseado em ontologia, criada por Paiva, Vieira e Musse (2005), que será utilizada. As relações de generalização possuem a mesma semântica que na UML, as relações direcionais são relações binárias entre as instâncias das classes. Na figura, *Agent* se relaciona com o conceito *Profile* para determinar o tipo de agente. O *Profile* do agente pode variar entre alguns tipos especificados pelo usuário que podem possuir destinos fixos (usuais) ou destinos randômicos (eventuais).

Esses locais são definidos pelo conceito *Place* que contêm uma descrição de sua capacidade (quantidade máxima de atores), dimensão e horário de funcionamento. O conceito de *Dimension* guarda a posição e o tamanho nos eixos X e Y. Já o conceito de *Schedules* possui o horário de abertura e fechamento, intervalo de entrada e tempo médio de permanência.

Além disso, o conceito *Components_of_the_environments* esta sendo pensando como um subconceito de *Setup* na ontologia desenvolvida porque representa uma configuração.

(i) Relações de dados (ii) Relações de Objetos

Figura 3.5: As relações existentes na ontologia proposta.

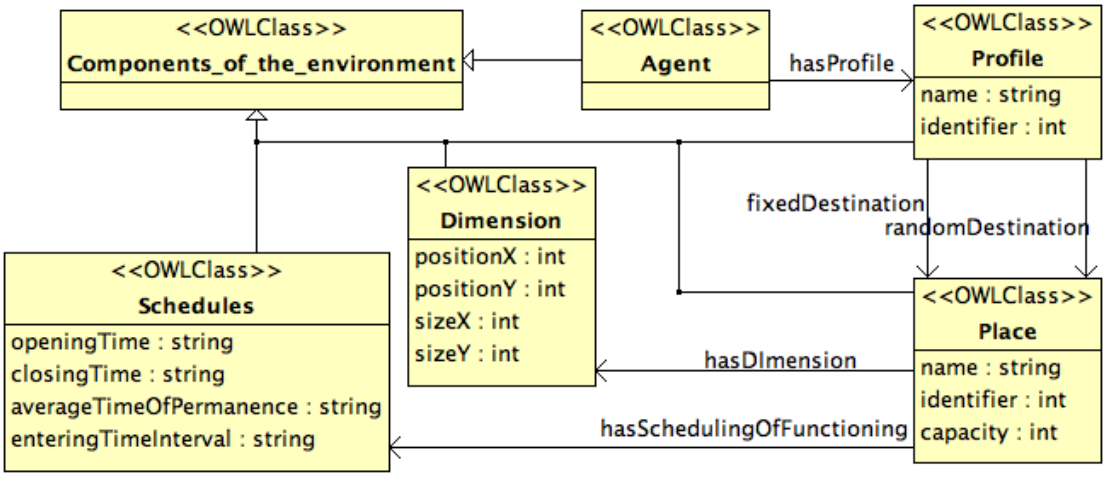

Figura 3.6: T-Box baseado no modelo de ambiente urbano (PAIVA; VIEIRA; MUSSE, 2005).

Fora isso, foi introduzida uma nova relação chamada *hasCharacter* que mapeia o conceito de agente da seção anterior para o conceito apresentado aqui. Dessa forma, um agente na ontologia afetiva pode descobrir seu perfil e seus locais visitados regularmente.

Esse modelo é utilizado no presente trabalho de duas formas. A primeira forma de utilização é para desenhar um mapa 2D do ambiente que os agentes atuam. Isso é permitido porque o conceito *Place* relaciona-se com o *Dimension*. Assim, todos os itens do mapa são retângulos. A outra forma de utilização é a que trouxe a ontologia para o trabalho e é definir que um agente tenha sua rotina descrita e disponibilizada pelo ambiente via percepção. No ambiente simulado construído como estudo de caso (ver Cap. 4), cada dia simulado equivale a aproximadamente 96 ciclos, portanto, cada passo simulado equivale a em torno de 15 minutos de um dia. Dessa forma, o ambiente é responsável por conhecer que dia é e informar para cada agente sobre sua rotina do dia.

3.4 Ontologia de Preferências

Doyle e Hayes-Roth (1998) propuseram que o mundo contivesse uma série de anotações nos objetos. Assim, o agente poderia conhecer apenas a forma de questionar os objetos sobre suas formas de usar, suas descrições e suas outras características. Esse conceito veio do conceito *affordance* que se refere a propriedade de um objeto que dita como o mesmo será utilizado. Dessa forma, uma cadeira tem a propriedade de ser sentada e uma porta tem as propriedades de ser aberta ou ser fechada.

Assim, eles utilizaram 5 tipos de anotações: (i) anotações emocionais, explicam como um agente responde “emocionalmente”; (ii) anotações de resposta, explicam como o agente deve reagir ao evento no ambiente que pode ser uma ação especifica ou uma sugestão de crença; (iii) anotações de resolução de problemas, descreve o estado do problema e permite anotar dicas que o ator talvez fale ou realize; (iv) anotações de papel, informam

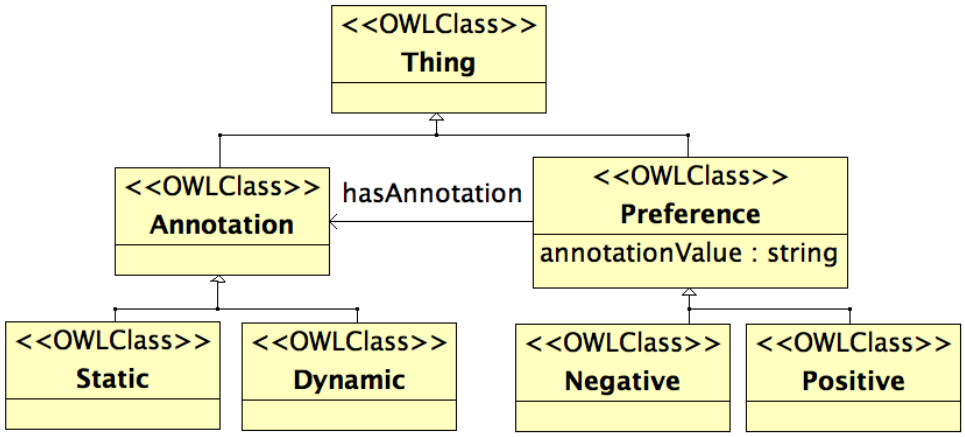

Figura 3.7: T-Box da ontologia de preferências proposta.

o agente sobre ações relevantes para determinados trabalhos no mundo; (v) anotações de jogo, descrevem o estado do jogo permitindo sugerir movimentos.

Kallmann e Thalmann (1998) propuseram uma ideia similar: os objetos no mundo são responsáveis por proverem o como ele deve ser usado. Por exemplo, durante a animação de o personagem estar abrindo uma porta, quem esta no controle do ator é o "agente" que controla a porta. Sendo assim, o agente do personagem delega a responsabilidade da animação para o agente do objeto já que o mesmo é o único que sabe como realizar a animação de abertura ou fechamento da porta. Esses objetos foram denominados objetos inteligentes e precisam ter um determinado nível de conhecimento sobre o ator.

De uma maneira similar, o conceito de artefatos que possuem propriedades observáveis e utilizáveis foi criado por Ricci, Viroli e Omicini (2006). Por exemplo, uma porta pode ter como propriedade observável seu estado (estar aberta ou estar fechada) e ação possível ou propriedade utilizável ser aberta ou ser fechada. Essas ações podem ficar disponíveis conforme o estado atual do objeto, mas o controle da disponibilidade e da ação é do próprio objeto por que é ele que sabe como realizar a ação propriamente dita. O agente unicamente diz de alguma forma que o objeto tem que realizar tal ação. Nesse trabalho, não é mencionado que o agente pode ser temporariamente controlado por objetos.

A ontologia proposta para as preferências pode ser vista na Figura 3.7. Como pode ser observado, foi criado dois conceitos principais: *Annotation* e *Preference*. O conceito *Annotation* é utilizado para representar características de objetos. Essas características podem ainda ser categorizadas entre as que mudam e não mudam durante a simulação. O subconceito *Dynamic* representa as características dinâmicas, isto é, as características que podem ser alteradas durante a "vida" do objeto. O subconceito *Static* representa as características que não podem ser alteradas depois da criação do objeto. O calor e o dono são exemplos de características dinâmicas, enquanto o nome e a capacidade total são exemplos de características estáticas.

O conceito *Preference* é usado para representar as preferências por determinadas ca-

racterísticas ou anotações. A preferência define como uma entidade ou agente é atraído por determinada característica, por exemplo algumas pessoas gostam de calor e outras não. Assim, as pessoas que gostam de calor são atraídas quando a característica de calor é positiva. Já, as pessoas que não gostam de calor são atraídas quando a característica é negativa. Dessa forma, os subconceitos da preferência representam sempre valores que fazem o agente ser atraído pelo objeto e repelido caso contrário.

Entretanto, esse exemplo não funciona para todos os casos. Por exemplo, um agente tem conhecimento de seu nível de energia e este é sempre positivo. Para resolver esse problema foi criada a relação *hasAnnotationValue* representada na Figura 3.7 pelo atributo *annotationValue*. Sendo assim, é possível descrever que um agente com nível de energia abaixo de um determinado limite é atraído pelo seu centro de controle. Voltando ao exemplo do calor do parágrafo anterior, uma pessoa gosta de calor quando a característica de calor ultrapassa o limite de 20 graus.

Essas anotações foram planejadas para serem colocadas pelo ambiente nos objetos e nos agentes. As anotações em agentes permitem que os outros saibam o que o agente esta fazendo e como ele parece de forma similar aos objetos. As preferências também são disponibilizadas em formas de percepção, mas o usuário que deve criar regras ou planos no *Jason* para montar as relações usadas pela afetividade.

3.5 Integrando a Plataforma *Jason* com as ontologias

As ontologias explicadas nas seções anteriores foram planejadas para serem usadas como módulos. Sendo assim, as ontologias explicadas anteriormente podem ser usadas sem implicar no uso das demais. Cabe salientar que elas podem ser usadas tanto pelo agente quanto pelo ambiente conforme poderá ser visto no capítulo 4.

Na Figura 3.8 pode ser observado as classes desenvolvidas para realizar a integração das ontologias com os agentes *Jason* e o suporte afetivo. Para essas classes serem usadas, o agente deve ser configurado para utilizar a classe *BBAffective* como base de crença. Essa configuração pode ser feita conforme explicado na seção 2.1.3.2 (pág. 16) e demostrado na seção 4.1 (pág. 41). A classe *UniqueAnnotation* é a responsável por garantir a não

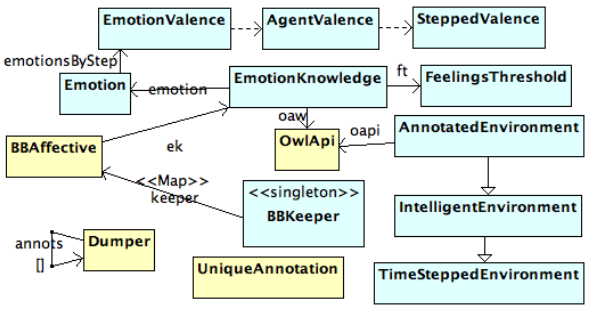

Figura 3.8: Diagrama de classes do sistema.

duplicidade de anotações em percepções.

A responsabilidade da classe *BBAffective* é manter as crenças dos agentes e atender as solicitações da plataforma *Jason* sobre a manipulação dessas em dois níveis. O primeiro nível é a ontologia e o segundo é a base de crença padrão que é utilizada quando o elemento não é considerado relevante para o primeiro nível. A relevância é descoberta consultando os conceitos e propriedades existentes na ontologia: instâncias de conceitos são crenças no formato "nomeDoConceito(nomeDaInstancia)" e relações são similares à "nomeDaRelacao(nomeDaInstanciaDominio, Imagem)". Conhecer o que é relevante para a ontologia é necessário para evitar a colocação de novas informações no domínio definido.

A base de crenças delega todo o assunto relacionado com as crenças da ontologia para a classe *EmotionKnowledge*. Essa é responsável por conhecer todo o assunto relacionado com as emoções e sentimentos, porém ela delega o conhecimento da história da potência das emoções para a classe *Emotion* e o conhecimento dos limites de ativação para a *FeelingsThreshold*. Da mesma forma, a manipulação e consulta da ontologia propriamente dita é de responsabilidade da classe *OwlApi*.

A classe *Emotion* é responsável por conhecer todas as emoções presentes e seus valores de potência. Para isso, no início da simulação, a ontologia é consultada procurando por todas as classes definidas[1] sobre o conceito "Emotion". Isso é utilizado pela classe *EmotionKnowledge* quando a mesma precisa consultar o nome de uma emoção por algum motivo. Por exemplo, na hora de calcular a valência da emoção em um determinado passo é pedido pelo nome de todas as emoções a essa classe. Essa ainda mantém a história da potência das emoções por agente para cada ciclo armazenado utilizando um objeto da classe *EmotionValence* que é uma extensão da classe *HashMap* do *Java*.

Para o suporte emotivo funcionar, a classe *FeelingsThreshold* precisa retornar que todos os limites mínimos de ativação dos sentimentos estão configurados. Assim, ela é responsável por fazer essa verificação e por disponibilizar esses limites para quando for realizado o cálculo da valência. Esse cálculo é coordenado pela *EmotionKnowledge* que consulta a ontologia, via instância da classe *OwlApi*, procurando as relações numéricas de cada indivíduo que representa uma avaliação. Dessa forma, a potência é a soma de todas as propriedades numéricas de um mesmo indivíduo.

As classes de ambiente, isto é, as classes que possuem *Environment* no nome são responsáveis por diversas coisas relacionadas com o ambiente. Por exemplo, manter os passos de simulação com um tempo máximo, conhecer a quantidade de agentes que tem que esperar responderem para ir ao passo de simulação seguinte, carregar as ações definidas pelo usuário a partir de um determinado pacote especificado, etc. Cabe salientar, as ações definidas no ambiente precisam estender a classe *EnvironmentAction* (não mostrada

[1] As classes definidas possuem condições necessárias e suficientes, isto é, qualquer indivíduo pode ser enquadrado nessa classe desde que obedeça as condições suficientes.

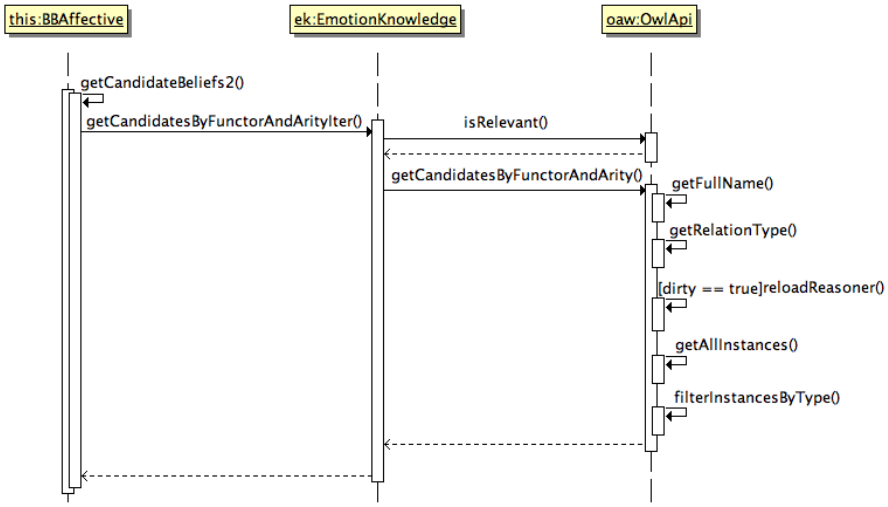

Figura 3.9: Diagrama de sequência para consultar uma crença.

na Figura 3.8) e estarem no pacote especificado na chamada a função “loadAllActions”.

A classe *Dumper* é responsável por conhecer o formato das crenças *Jason* e converte-lo para um formato interno. Isso é necessário para proteger o código feito de mudanças que a plataforma venha a ter. Assim, essa classe é utilizada por todas as classes internas do sistema e, principalmente, pela *BBAffective* após constatada a relevância da crença. Algumas funções são, por exemplo, devolver os parâmetros do literal como *string* ou número, retornar uma instância de seu tipo a partir de um literal, criar o literal a partir de uma *string* ou a partir de uma outra instância de seu tipo e, até mesmo, realizar a criação de um *XML* para transferir os dados para a ontologia.

Já a classe *BBKeeper* serve para facilitar o acesso a base de crenças. Essa classe é a única *singleton* do sistema e cada instância da *BBAffective* é conhecida por esta a partir do nome de seu respectivo agente. Isso é muito útil para as simulações porque torna possível acessar a base de crenças e exibir os valores das emoções e dos sentimentos ou conhecer os tipos emotivos disponíveis. Cabe salientar que os tipos emotivos disponíveis do simulador não levam em conta o agente e, por isso, mesmo que seja usado um agente que não esteja utilizando a base estendida os tipos serão retornados corretamente.

As seções seguintes, explicam os novos processos de inserção, remoção, consulta e listagem das crenças. Exemplos desses em código *AgentSpeak* são, respectivamente, *+crencca*, *-crencca*, *?crencca* e *?X*. A listagem também é utilizada pela tela de visualização de crenças quando realizando uma depuração no agente.

3.5.1 Processo de Listagem e Consulta de Crenças

O processo de listagem serve para exibir todas as crenças e percepções do agente. Isso é importante em momentos de depuração ou em momentos que se deseja consultar uma crença que é uma variável não unificada, isto é, sem valor definido. Dessa forma, a base de crenças implementa a interface *java.lang.Iterable* que permite a classe ser usada em

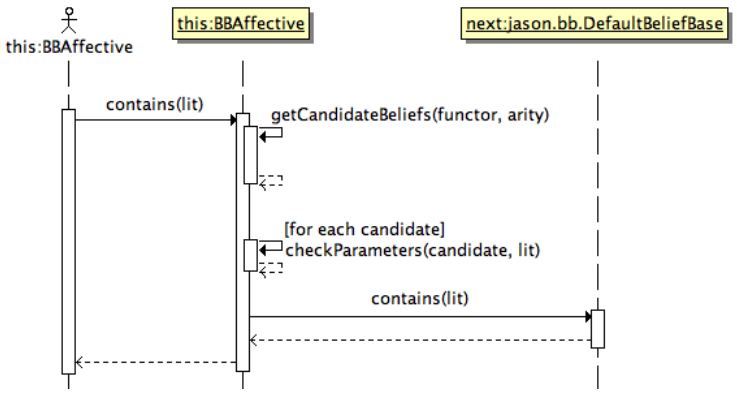

Figura 3.10: Diagrama de sequência para recuperar uma crença.

construções *for-each*. A implementação devolve todos os elementos dos dois níveis sendo que o nível da ontologia é recuperado primeiro.

De maneira similar, o processo de consulta de crenças foi alterado. Esse processo tem como entrada os métodos com nome *getCandidateBeliefs* que chamam um método denominado *getCandidateBeliefs2* conforme pode ser visto na Figura 3.9. O resultado dessa operação é um objeto *Iterator* com todas as crenças recuperadas da base que possuem o mesmo nome e mesma aridade que a informada na chamada. Isso é importante porque esses candidatos podem ter seus parâmetros verificados pela própria base de crenças ou por outros pontos de acesso do *Jason*.

3.5.2 Processo de Recuperação de Crenças

O processo de recuperação de uma crença é necessário para a compreensão das inserções e remoções de crenças. Ele serve para recuperar uma única crença quando ela já existir na base, e em sua busca leva em consideração toda a estrutura do literal excluindo as anotações. O procedimento é mostrado na Figura 3.10. Essa figura utiliza condições entre colchetes chamadas guardas para estabelecer que um código deve obedecer as condições impostas.

A recuperação é iniciada pela chamada do método *contains*. Essa função inicia com uma chamada à *getCandidateBeliefs* que foi explicada na seção anterior. Cada um dos candidatos retornados pela chamada são verificados para ver se os parâmetros conferem com a crença informada pelo cliente, se conferir então o candidato será retornado. Se nenhum candidato for retornado, o resultado da base de crenças padrão será retornado porque o mesmo pode estar localizado neste nível. Conforme foi falado anteriormente, tanto a inserção e remoção utilizam esse procedimento para recuperar o elemento armazenado na base de crença, porém os motivos serão vistos em suas respectivas seções.

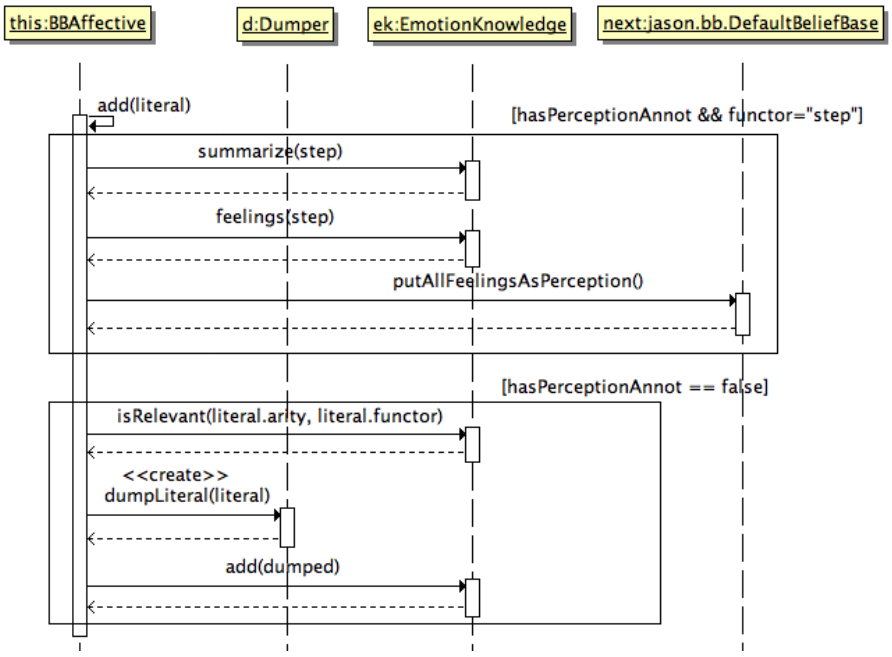

Figura 3.11: Diagrama de sequência para adicionar uma crença.

3.5.3 Processo de Adição de Crenças

O procedimento de inserção de uma crença pode ser visto na figura 3.11. Essa figura utiliza condições entre colchetes chamadas guardas perto de linhas que delimitam que todo um bloco de código deve obedecer aquela condição. Dessa forma, há na figura dois grandes blocos de código com suas condições próprias. Por exemplo, quando estiver sendo inserida uma crença de percepção com o termo diferente de "step" nenhum código será executado pelo diagrama e a mesma será inserida no segundo nível.

O processo de inserção de uma crença começa com o *Jason* chamando a função de inserção de crenças, essa função chamada *add* retorna verdadeiro para representar que o elemento foi inserido ou falso. Na Figura 3.11, o *Jason* não esta representado, porém quando a função executa ela chama um outro método *add* interno que retorna se o elemento foi ou não inserido na ontologia. No entanto, quando não foi feita a inserção na ontologia o valor retornado para o *Jason* dependerá da inserção na base de crenças padrão da plataforma. O processo que atualiza as emoções e guarda a potência das mesmas na classe *Emotion* é realizado quando uma percepção esta sendo inserida e o termo da crença é "step". Nesse momento os sentimentos atualizados são recuperados e é feita a reinserção deles no segundo nível da base de crenças.

Agora quando esta sendo feita uma inserção de uma crença que não é uma percepção, é feita a verificação se ela é relevante. Caso ela seja considerada irrelevante, a crença será inserida no segundo nível. Entretanto, se ela for considerado relevante então será feita a recuperação do elemento atualmente armazenado e, após isso, é feita a inserção das novas anotações nesse elemento. Se o elemento não existir na base de crenças então esse

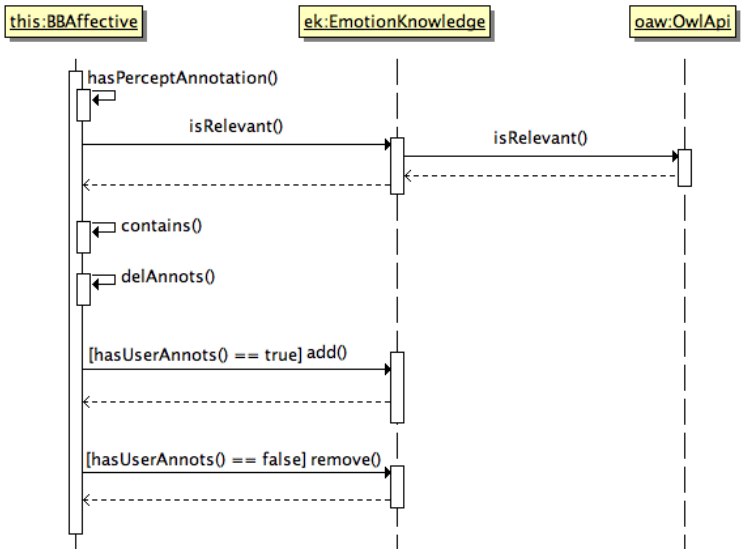

Figura 3.12: Diagrama de sequência para remover uma crença.

último procedimento é ignorado, porém ele é necessário para haver a correta atualização das anotações das crenças. Por fim, a crença é transformada no tipo interno e enviado para ser acrescentado na ontologia.

Assim, o processo de inserção serve para inserir os axiomas necessários para representar essa crença na ontologia quando a mesma for relevante. Para a criação na ontologia, é necessário ainda diferenciar se é uma instância de objeto ou propriedade de dados ou de objetos. Essa diferenciação é feita pela aridade e, em caso de ambiguidade, se pergunta o tipo à ontologia usando o nome da entidade. A chamada *Add* presente no final da Figura 3.11 é explicada adiante.

3.5.4 Processo de Remoção de Crenças

O processo de remoção de crenças permite remover uma crença da base de crenças ou, apenas, atualiza-la retirando alguma anotação que não devia existir. Dessa forma, o procedimento ao final do mesmo remove os axiomas postos anteriormente e quando for uma atualização os insere novamente. A Figura 3.12 mostra a rotina modificada da base de crenças.

A rotina de remoção desenvolvida inicia verificando se o termo é considerado relevante. Se ele for uma percepção então ele é irrelevante para esse procedimento. Caso não seja uma percepção, a relevância é verificada e a sua relevância é constatada então o elemento armazenado é recuperado. Se não for recuperado nenhum elemento ou se ele for irrelevante então é realizada a chamada a remoção da base de crenças padrão e retornado o resultado.

Nesse momento o que se sabe é qual é a crença e que a mesma é relevante. Assim, as anotações informadas são removidas da crença obtida e a nova crença é testada para ver se ainda possui anotações. Se possuir e não for uma anotação de fonte e não é um

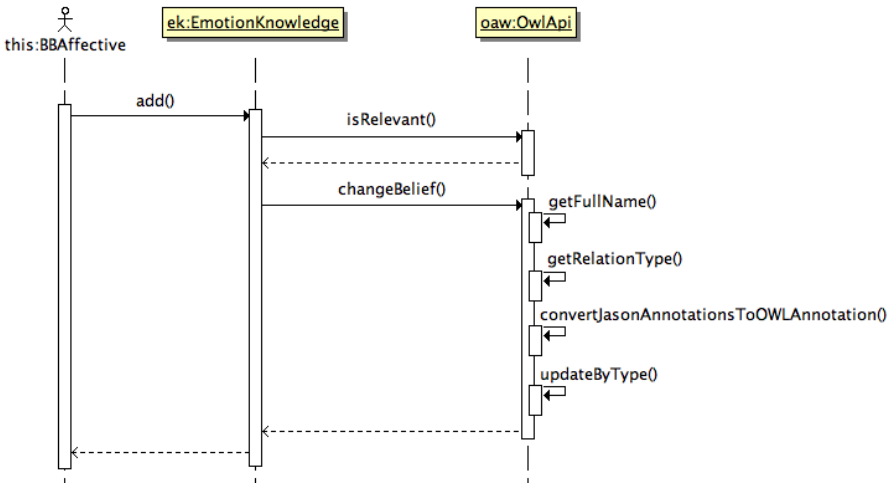

Figura 3.13: Diagrama de sequência para adicionar uma crença.

indicativo de estar sendo armazenada em determinada ontologia então é feita a atualização do valor atualmente armazenado. No caso de não possuir anotação ou possuir somente as anotações mencionadas é feita a remoção do elemento que esta na base de crenças. Tanto as funções *Add* e *Remove* da Figura 3.12 compartilham o mesmo código e estão explicadas na seção seguinte.

3.5.5 Processo Compartilhado de Remoção e Adição de Crenças

O processo de adição e remoção de crenças compartilham pedaços de códigos. A classe *EmotionKnowledge* possui dois pontos de entrada para adicionar ou remover crenças, um pelo método *Add* e outro pelo método *Remove*. Tanto um método quanto outro fazem uma chamada para uma função denominada *changeBelief* informando se o procedimento é de inserção ou remoção. A partir dai a sequência de chamadas mostradas na Figura 3.13 segue de maneira igual. Isso foi feito porque muito código é igual para ambos os casos.

A classe *OwlApi* é a única que consulta a ontologia diretamente via biblioteca. No inicio da aplicação são extraídas as informações de quais conceitos existem, quais relações e de quais tipos são. Sendo assim, consultar o nome completo de uma determinada entidade ou qual é seu tipo não precisa ir na ontologia. A função *updateByType* é a responsável por realizar a inserção e remoção dos axiomas. Uma alteração é feita removendo os axiomas anteriores, acrescentando os novos e marcando a ontologia como “suja” para quando uma consulta for feita a mesma ter o raciocinador atualizado.

4 ESTUDOS DE CASO

O presente capítulo visa explicar os estudos de caso utilizados para validar o sistema desenvolvido. Todo experimento foca no agente que possui uma parte escrita em *AgentSpeak* para decidir como as percepções se ligam com a ontologia afetiva, os limites da emoção virar sentimento deve ser decidido inicialmente e o agente deve receber uma percepção "step" quando for um novo ciclo de simulação para recalcular os sentimentos.

Os experimentos são simulações de parte da vida dos personagens. No primeiro, eles encontram-se observando um jogo de futebol e cada um torce por um time. No segundo, uma casa é simulada com quatro atores e suas rotinas. Entretanto, antes desses estudos serem abordados na seção 4.2 e na 4.3, será descrito como a ontologia foi testada juntamente com a plataforma *Jason*. A seção 4.1 explica as ferramentas desenvolvidas para o teste da base de crenças desenvolvida e são duas: uma interativa e outra não interativa que serve como uma espécie de teste unitário dos artefatos explicados no capítulo 3.

4.1 Teste da Base de Crenças

Os testes da base de crenças têm como finalidade checar se a utilização normal está acontecendo da maneira esperada. Assim, eles fazem testes de inserção, recuperação, remoção e listagem. Os testes foram escritos na linguagem *AgentSpeak*.

A primeira aplicação de teste desenvolvida foi de maneira interativa conforme pode

Figura 4.1: Interface para mostrar os sentimentos dos agentes.

Figura 4.2: Exemplo de utilização criando uma emoção de orgulho.

ser observado na Figura 4.1. A aplicação permite que o usuário escreva as crenças do agente em um campo texto (área 2 na figura) e esse é enviado diretamente para a base de crenças do agente. Na área 1 da mesma figura a valência das emoções é mostrada. Essas informações podem aparecer em: preto, se não houve alteração com o ciclo anterior; azul, se houve um aumento; vermelho, se houve uma diminuição do valor.

Na Figura 4.2 estão representados dois momentos da criação de uma emoção de orgulho pelo usuário. Na parte de cima, o agente tem configurado uma avaliação de e sobre si próprio. Além disso, a avaliação tem probabilidade nula ou irrelevante e o desejo para si mesmo foi considerado com o valor 10. Esse valor poderia ser qualquer número inteiro, mas se fosse negativo ao invés de alegria (*joy*) o resultado atual seria sofrimento (*distress*).

Na parte de baixo da Figura 4.2 está o resultado obtido após a inserção da última

crença para criação da emoção de orgulho. Conforme pode ser visto, foi necessário três ciclos deliberativos para o agente ter a percepção do sentimento. Note que essa percepção do sentimento, não veio do simulador e sim de o agente erroneamente acreditar que existe um passo de simulação novo e então disparar o processo emotivo para criar as emoções conforme configuração. Se fosse um passo normal da simulação, não se teria na base de crenças duas percepções *feeling* sobre uma emoção porque a todo novo ciclo as percepções são apagadas. Cabe chamar atenção que no sistema desenvolvido não existe o decaimento dos valores com o passar do tempo, para isso ser feito o agente deve reavaliar o problema e atualizar os valores nas suas crenças.

Na Listagem 4.1 pode ser observada uma configuração para rodar o presente aplicativo[1]. O ambiente na linha 3 espera que todos os agentes (quantidade informada no parâmetro 3) mandem uma ação para prosseguir. O simulador espera por essas ações pelo tempo em milissegundos configurado no primeiro parâmetro, caso não venha ignora o agente e segue para o próximo ciclo. O segundo parâmetro permite informar um número que representa o último passo de simulação e o quarto parâmetro permite configurar a atitude a ser tomada quando um agente tentar fazer duas ações no mesmo passo. Essa atitude pode ser ou o enfileiramento da ação ou a falha imediata da mesma que é a utilizada no exemplo.

Listagem 4.1: Arquivo de projeto do *Jason* para a aplicação interativa de teste.

```
MAS eoaus {
2     infrastructure: Centralised //(pool, 2)
      environment: maro.example.console.Environment(30000, 1000, 1, FALSE)
4     agents:
          millie
6           beliefBaseClass maro.wrapper.BBAffective("ontology/occ-tbox.owl")
            agentArchClass maro.example.console.AddGui
8           agentClass maro.wrapper.UniqueAnnotation
          ;
10    aslSourcePath: "asl";
}
```

O agente configurado da linha 5 à 9 utiliza opções do *Jason* para alterar a base de crença sendo utilizada (*beliefBaseClass*), arquitetura (*agentArchClass*) e o raciocinador (*agentClass*). As classes dessas alterações foram explicadas anteriormente na seção 3.5 (pág. 34). Assim, para o presente exemplo a mudança da arquitetura na linha 7, única classe não mencionada até agora, foi realizada para criar e atualizar a janela mostrada na Figura 4.2 que exibe os dados emotivos.

Na linha 6 da Listagem 4.1 foi incluída somente a ontologia afetiva desenvolvida. Assim, para o agente utilizar as emoções ainda é necessário incluir as configurações de ativação dos sentimentos via código. Uma amostra do código utilizado para se fazer isso pode ser vista na Listagem 4.2 que mostra algumas crenças iniciais do agente.

[1] Veja a seção 2.1.3.1 na página 13 para ver uma introdução sobre esse tipo de arquivo.

As crenças iniciais são carregadas no agente no inicio da simulação. O processo de carga segue conforme determinado na classe da linha 6 da Listagem 4.1. Sendo assim, a crença “step” como não consta na ontologia será carregada na base de crenças padrão e as demais serão criadas na base de conhecimento. Esse processo fica transparente para o código *AgentSpeak*, porém o usuário precisa definir para as 22 emoções o limite mínimo para as mesmas serem percebidas como sentimento.

Listagem 4.2: Parte do código do agente para aplicação interativa de teste.

```
1  step(0)[source(percept),source(self)].

3  agent(millie).

5  hasSetup(millie, setup1).
   hasThreshold(setup1, 0).
7  hasThresholdType(setup1, "Joy").

9  hasSetup(millie, setup2).
   hasThreshold(setup2, 0).
11 hasThresholdType(setup2, "Distress").
```

Nesse exemplo, os valores de limites estão sendo configurados com valoração zero para que a potência e a valência sejam iguais. O agente somente conhece a valência de uma emoção que é o valor da potência menos o limite de ativação. Entretanto, tanto o limite de ativação quanto a potência não são conhecidos pelo agente. Por exemplo, se o limite da pena (*sorryFor*) é 6 e o valor atual é 8 então o sentimento de pena terá valor 2 sendo expresso por uma crença da seguinte forma “feeling("sorryFor",2).”.

A aplicação não interativa utiliza a configuração do arquivo de projeto mostrado no Apêndice A (pág. 63). O ambiente especificado na linha 4 possui os mesmos parâmetros do anterior com o acréscimo de um novo que indica uma ontologia. Essa serve para o ambiente conhecer as rotinas dos agentes, como deve ser desenhado o mapa a ser exibido e as posições iniciais de cada agente. O agente millie aqui é o utilizado para os testes e os limiares de ativação de emoção estão configurados para valores diferentes. Existe uma variedade de testes para as propriedades de objeto ou de dados e para as instâncias de classes, além de testes das conclusões esperadas.

4.2 Assistindo um jogo de futebol

O caso de uso apresenta dois personagens que estão no mesmo lugar e assistindo ao mesmo jogo de futebol. O jogo tem ao todo 60 turnos que representam os 90 minutos de um jogo regulamentar. Em cada turno é sorteado um número randômico com probabilidade de 1.67% para os times marcarem gol, isto é, há 98.33% de chance a cada turno de nada acontecer e o placar permanecer inalterado.

Esse exemplo aparentemente simples, propicia um cenário bastante rico para as emoções porque um personagem pode ter diferentes emoções acontecendo. Essas emoções são muitas vezes do ramo de consequências de eventos por estar ligada com a desejabilidade do evento ou do ramo de ações de agentes que tem a ver com a responsabilidade das

ações. Assim, as mentes dos atores que estão assistindo ao jogo de futebol decidem no inicio da simulação os nomes de seus times e ambos acreditam que vencerão.

O arquivo de projeto dessa aplicação pode ser consultada no Apêndice C (pág. 70) e cria dois agentes: “watch1” e “watch2”. Cada um com seu próprio código fonte e, dessa forma, cada um deles inclui em seus códigos as configurações necessárias para utilizar o sistema. Um exemplo dessas configurações é o limite de ativação para uma emoção virar sentimento. Outro exemplo é que os times que participam do jogo são considerados pelos personagens como outros agentes, assim o time pode ser considerado “amigo” e o outro “inimigo”.

Cabe salientar que cada um dos agentes realizam as mesmas avaliações sobre os mesmos eventos. Esses eventos foram pensados como a marcação de um gol (Listagem 4.3), o estado atual do jogo (Listagem 4.4) e o estado final do jogo (Listagem 4.5 e Listagem 4.6). Esses são alguns exemplos escritos na plataforma *Jason* para fins de validação.

Listagem 4.3: Parte do código do agente referente à avaliação de gol.

```
1  +?appraisalGoal
       :  goal(TEAM) & myPoints(TEAM, PF, _, _)
3      <- ?fib(8-(PF+1), VAL);
          ?updateEmotion(prospectIrrelevant, "goal_myteam_pi", VAL);
5         +removeEmotion("goal_myteam_pi").

7  +?appraisalGoal
       :  goal(TEAM) & myPoints(_, _, TEAM, PE)
9      <- ?fib(PE+1, VAL);
          ?updateEmotion(prospectIrrelevant, "goal_enemyteam_pi", -VAL);
11        +removeEmotion("goal_enemyteam_pi").

13 +?appraisalGoal
       : removeEmotion(INDIVIDIDUAL)
15    <- ?removeEmotion(prospectIrrelevant, INDIVIDIDUAL);
         -removeEmotion(INDIVIDIDUAL).
17
   +?appraisalGoal.
```

A Listagem 4.3 realiza o processamento das emoções sentidas por um personagem quando um time marca um gol. Cada um dos planos criados visam atender as 4 condições possíveis: gol do meu time, gol do adversário, gol que não é mais percebido e não houve gols. O plano da linha 4 (*updateEmotion*) e o da linha 15 (*removeEmotion*, não confundir o plano com a crença) serão explicados adiante, por hora basta saber que eles atualizam ou removem as relações necessárias para a afetividade.

Na última listagem mencionada, as linhas 3 e 9 chamam um plano que consulta os números de *fibonacci*. Esse plano tem no primeiro parâmetro a posição do número na sequência e o retorno correspondente. Os times quando jogam no simulador dificilmente marcam mais que 5 gols então foi decidido limitar os números de *fibonacci* até a sétima posição. Valores de posição negativos ou superiores à 7 terão retorno um. Quando um gol é marcado a favor do time que o agente torce o *fibonacci* é usado de maneira decrescente,

isto é, o primeiro gol retornará a sétima posição, o segundo à sexta e etc. No caso de gols do time adversário é utilizado o cálculo de *fibonacci* de maneira crescente. Logo, o valor 1 é retornado para os dois primeiros gols, o terceiro valor 2 e assim por diante.

A Listagem 4.4 é similar à anterior. Ela é responsável por inserir uma emoção de esperança no agente com valoração 50 e inicia o cálculo de performance do evento assumindo o valor positivo 1. Cada gol percebido do próprio time incrementa esse valor em 10 e quando o gol é do adversário é feito um decremento no mesmo valor. A emoção de esperança não sofre alteração depois que é criada porque esta sendo considerado que as emoções são baseadas na percepção dos eventos ou ações. Dessa forma, a esperança do jogo ser ganho só termina quando há a percepção do encerramento do mesmo e a performance do evento que foi calculada é utilizada nesse momento.

Listagem 4.4: Parte do código do agente referente ao andamento do jogo.

```
+?appraisalTeam
    :  not(hasLikelihood("team_hope_prn", PROB)) & step(STEP) & STEP < 10
    <- ?updateEmotion(prospectNotRealized, "team_hope_prn", 50);
       +realizedCalculation(1).

+?appraisalTeam
    :  goal(TEAM) & myPoints(_,_, TEAM, _) // ponto que nao eh meu
    &  realizedCalculation(CALC)
    <- -realizedCalculation(CALC);
       +realizedCalculation(CALC-10).

+?appraisalTeam
    :  goal(TEAM) & myPoints(TEAM, _, _,_) // ponto que eh meu
    &  realizedCalculation(CALC)
    <- -realizedCalculation(CALC);
       +realizedCalculation(CALC+10).

+?appraisalTeam.
```

Listagem 4.5: Parte do código do agente referente à avaliação do final do jogo para as emoções de probabilidade.

```
+?appraisalEndMatch
    :  realizedCalculation(CALC) & match(_,_)
    &  hasLikelihood("team_hope_prn", PROB)
    &  feeling(hope, HOPEVAL)
    <- ?removeEmotion(prospectNotRealized, "team_hope_prn");
       ?updateEmotion(prospectRealized, "team_satisfaction_pr", HOPEVAL, CALC);
       -realizedCalculation(CALC).

+?appraisalEndMatch
    :  realizedCalculation(CALC) & match(_,_)
    &  hasLikelihood("team_hope_prn", PROB)
    <- ?removeEmotion(prospectNotRealized, "team_hope_prn");
       -realizedCalculation(CALC).

+?appraisalEndMatch.
```

Nas Listagens 4.5 e 4.6 são realizadas avaliações depois do termino do jogo. O jogo pode terminar empatado ou um dos dois times ganhar. Entretanto, para a primeira listagem basta que o jogo tenha terminado para uma atitude ser tomada. Essa atitude é a remoção da emoção de esperança e a criação de uma nova emoção baseado em como foi o andamento do jogo e no valor atual da esperança. Sendo assim, um personagem pode ficar satisfeito ou desapontado.

Já, a Listagem 4.6 permite um personagem sentir-se feliz por ou com pena do resultado de seu time. Quando o time do personagem tiver ganho o jogo ele pode sentir-se feliz ou neutro e quando o time perde ele pode sentir-se com pena ou neutro da mesma forma. Esse valor de neutralidade é porque o modelo de emoções só diz que a felicidade por alguém ou a pena serão sentidas quando o merecimento e o desejo presumido sobre o outro forem ambas positivas ou negativas.

Listagem 4.6: Parte do código do agente referente à avaliação do final do jogo para as emoções relacionadas com a consequência de eventos para outros.

```
+?appraisalEndMatchHappy
     : match(win, TEAM) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10; DT=20;
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).

+?appraisalEndMatchHappy
     : match(lose, TEAM) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10;
       DT=-20; // eh negativo para implicar na emocao de pena
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).

+?appraisalEndMatchHappy
     : match(draw, _) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10; DT=10;
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).

+?appraisalEndMatchHappy.
```

As consultas de “removeEmotion”, presentes nas listagens anteriores, recebem o grupo emotivo e o nome do indivíduo que deve ser removido. Com base nessas informações, as relações que foram outrora inseridas são removidas da ontologia e, consequentemente, da base do *Jason* normal. Um exemplo é mostrado na Listagem 4.7 no qual são removidas as relações: de merecimento, de desejo por outros, de qual pessoa está sendo avaliada e de quem está avaliando.

As consultas de “updateEmotion” funcionam de maneira parecida à “removeEmotion” e fazem a inserção de relações. Caso já exista o indivíduo, removem e acrescentam novas relações para alterar o valor guardado. Dessa forma, para a construção do exemplo tanto as atualizações como as remoções foram agrupadas por grupo de emoções visto que

essa é a forma que o próprio modelo *OCC* agrupa as emoções com regras similares. A Listagem 4.8 mostra as regras para o grupo emotivo que tem a ver com a consequência dos destinos dos outros, isto é, as relações de merecimento, de nível de desejo presumido e de quem está avaliando e de qual pessoa está sendo avaliada.

Listagem 4.7: Amostra de remoção de emoção do tipo destino de outros.

```
+?removeEmotion(fortunesOfOthers, INDIVIDUAL)
     : hasDeserved(INDIVIDUAL, OLDDE) & hasDesireOther(INDIVIDUAL, OLDDO)
     & hasPerson(INDIVIDUAL, TEAM)
    <- ?myName(NAME);
       -hasPerson(INDIVIDUAL, TEAM);
       -hasDeserved(INDIVIDUAL, OLDDE);
       -hasDesireOther(INDIVIDUAL, OLDDO);
       -hasAppraisal(NAME, INDIVIDUAL).

+?removeEmotion(fortunesOfOthers, INDIVIDUAL).
```

Listagem 4.8: Amostra de código referente as atualizações de emoções do tipo destino de outros.

```
+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DESERVED, DESIREOTHER)
     : team(TEAM) & not(hasPerson(INDIVIDUAL, TEAM))
    <- ?myName(NAME);
       +hasAppraisal(NAME, INDIVIDUAL);
       +hasPerson(INDIVIDUAL, TEAM);
       +hasDeserved(INDIVIDUAL, DESERVED);
       +hasDesireOther(INDIVIDUAL, DESIREOTHER).

+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DESERVED, DESIREOTHER)
     : hasDeserved(INDIVIDUAL, OLDDE) & hasDesireOther(INDIVIDUAL, OLDDO)
    <- -hasDeserved(INDIVIDUAL, OLDDE);
       +hasDeserved(INDIVIDUAL, DESERVED);
       -hasDesireOther(INDIVIDUAL, OLDDO);
       +hasDesireOther(INDIVIDUAL, DESIREOTHER).

+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DE, DO).
```

4.3 Simulando uma casa

O presente estudo de caso tem por finalidade demonstrar as ontologias de rotinas e preferências trabalhando junto da afetividade. Ele utiliza o ambiente e os agentes de maneira diferente do caso anterior. O agente aqui não carrega os limites de ativação dos sentimentos de crenças iniciais e, sim, da própria ontologia. Todos seus valores estão configurados para a emoção virar sentimento a partir do valor 20 por simplicidade, mas poderiam ter qualquer valor configurado pelo usuário.

O ambiente utilizado nesse exemplo foi explicado brevemente quando foi introduzido o aplicativo de teste não-interativo. Esse ambiente carrega uma ontologia que tem por finalidade conhecer as preferências e as rotinas dos agentes e disponibilizá-las na forma de percepções aos agentes, além de construir uma visualização do ambiente onde é possível

visualizar os atores desses agentes atuando. As percepções nesse ambiente são apagadas e criadas novamente em cada passo de simulação.

Para a construção da visualização, cada um dos indivíduos do conceito *Place* são considerados como objetos que devem ser desenhados se estiverem associados à uma dimensão. Assim, todos os objetos na tela são desenhados a partir de retângulos. Além disso, esse conceito pode usar a relação *hasSetup* visando determinar que um objeto tem determinada anotação ou, ainda, determinar que objetos fazem parte de um objeto maior. Para o ambiente é importante conhecer as anotações dos objetos porque quando esses objetos são percebidos pelo agente vai junto as anotações que esses objetos possuem. Por exemplo, o objeto "camaSolteiro1" tem as anotações de posição nos eixos x e y, tamanhos nos dois eixos, utilidade "dormir" e dono um ou mais personagens.

Cabe chamar atenção que o personagem em nenhum momento recebe a informação do ambiente de onde ele se encontra. Essa informação o agente descobre baseado na percepção do ambiente. Por exemplo, o agente sabe que o "quarto1" possui um armário, dois criados mudos, duas camas e duas portas. O agente sabe o nome correto de cada um dos itens que estão no quarto então ele pode a partir da percepção buscar em seu mapa onde os objetos que percebe estão e então concluir que ele se encontra em uma determinada sala. Assim, usando o conhecimento extraído de um objeto que possui outros objetos, ele é capaz de construir um mapa mental com a finalidade principalmente de se localizar e saber quais objetos ele deve alcançar para chegar em um destino. Mas, esse mapa não carrega consigo as informações de anotações que só são obtidas via percepção. Dessa forma, um ator precisa a primeira vez descobrir qual é a sua cama e onde ela se encontra quando quiser descansar.

A Figura 4.3 possui quatro atores diferentes chamados: Millie, Albert, Nina e John. Esses atores estão localizados inicialmente em diferentes pontos da casa e cada um possui uma orientação própria. No inicio da simulação eles devem averiguar os objetos próximos para perceberem em qual sala se encontram e depois decidem um objetivo próprio baseado em seu estado interno atual. Esse estado interno encontra-se representado por uma característica fundamental que é a energia que o ator possui para gastar e todos os agentes iniciam com ela no valor 40 podendo a mesma ir de 0 a 100. Sendo assim, muito provavelmente no inicio da simulação os agentes tentarão ir descansar em suas respectivas camas para obter energia para o dia seguinte.

O valor de energia mencionando antes é informado pelo ambiente ao agente. Cada um dos agentes recebe uma percepção sobre seu estado interno que foi denominada "myself". O agente usa essa informação junto de sua preferência pelas anotações para determinar que emoção ele irá sentir. Quando o valor de energia está abaixo de um determinado limite estipulado pela preferência da anotação de energia o agente começa a ter um determinado nível de sofrimento. No caso de ele estar com a energia acima do limite, o mesmo possuirá a emoção que corresponde à alegria.

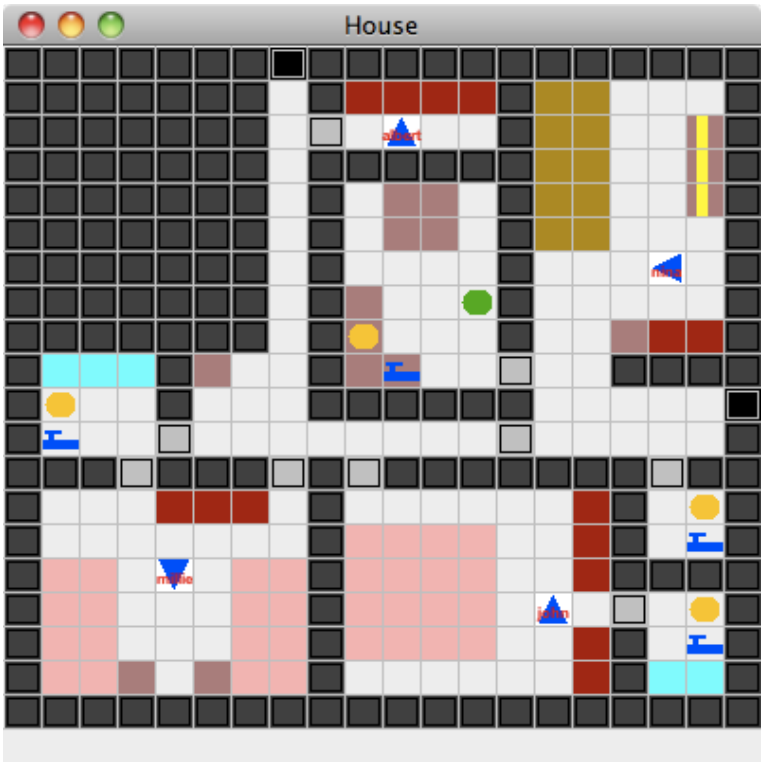

Figura 4.3: Interface de visualização da casa sendo simulada.

Esse exemplo simples com duas emoções serve para demonstrar o uso de todas as ontologias desenvolvidas. Sendo assim, o ambiente possui 206 indivíduos com aproximadamente 70% deles sendo necessários para a criação dos 60 objetos presentes na tela (paredes, mesa, camas, etc). A Figura 4.4 mostra na esquerda o mapa abstrato que o agente pode consultar via ação interna e a direita o mapa da visualização.

A Figura 4.4 é útil para entender a forma como o agente conhece a casa. Esse grafo é utilizado pelo mesmo para ir de um lugar para outro. Assim, este é construído antes da simulação iniciar, a partir dos dados da ontologia informada para o ambiente. Entretanto, esse mapa possui somente os itens (Figura 4.5 adiante) de cada lugar e não possui nenhum conhecimento das anotações. Além disso, essas informações são disponibilizadas por ações internas: retornar todos os itens, retornar todos os cômodos, retornar o cômodo de determinado item ou retornar todos os itens de um cômodo.

Conforme dito anteriormente, a primeira coisa que o agente faz é descobrir onde ele se encontra. Assim, quando o agente entra em uma sala, ele busca por todos os itens que ele percebe por duas razões. Primeira, é para manter uma lista dizendo a orientação e o que foi percebido lá. A segunda razão é que quando o agente não sabe onde se encontra, o mesmo pergunta ao ambiente a localização de todos os itens percebidos e a localização mais citada é a considerada atual. Isso é necessário porque mesmo havendo obstáculos que o agente não consegue passar ou ver, algumas vezes objetos de outros lugares são percebidos. Por exemplo, o agente que inicia no norte (depósito) quando olha para o sul

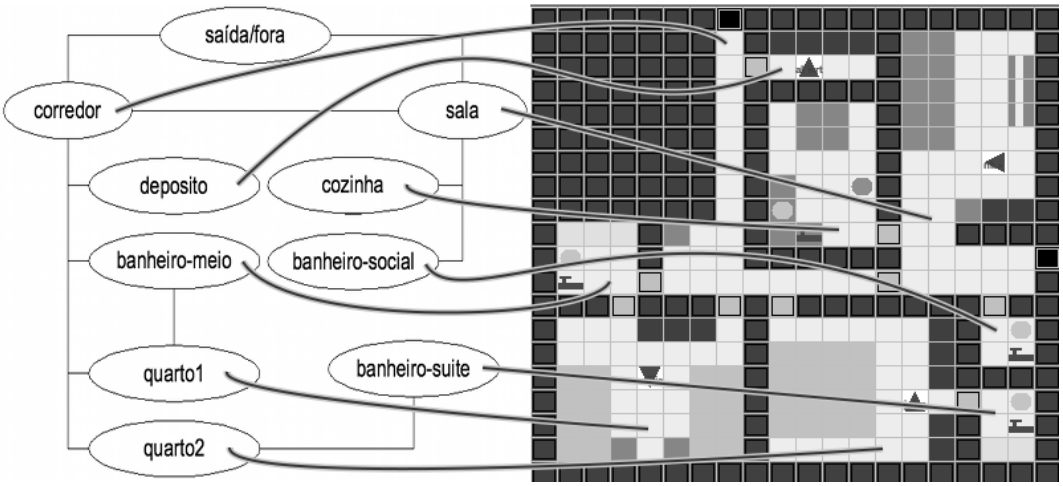

Figura 4.4: Mapa do agente (esquerda) e visualização (direita).

pode perceber a mesa (cozinha).

O mapa abstrato mostrado na Figura 4.4 é apresentado com os respectivos itens em suas respectivas localizações na Figura 4.5. Novamente, esse mapa é montado antes do inicio da simulação conforme configuração da ontologia informada no ambiente. Além disso, as posições iniciais dos personagens também encontram-se configurados nesta juntamente com seus perfis.

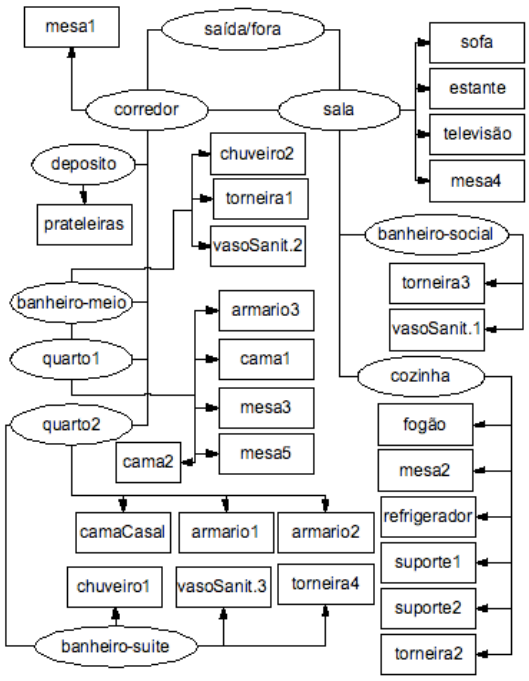

Figura 4.5: Grafo usado para descobrir os itens.

Esses perfis são usados para definir a rotina diária de um ator porque eles permitem relacionar estabelecimentos como destinos possíveis ou fixos. Assim, ao todo existem 4 perfis configurados, dois para estudantes e outros dois para trabalhadores. Os estudantes vão para escola pela manhã e após possuem destinos eventuais. Enquanto um dos perfis vai eventualmente ao mercado, o outro vai ao fliperama. O perfil "trabalhador" tem o local de trabalho como fixo e o mercado como eventual. Já o último perfil está sendo considerado como um trabalhador caseiro. Dessa forma, ele não possui nenhum destino fixo, somente destinos eventuais.

Os locais existentes são, basicamente, a casa, a escola, o fliperama, o mercado e o trabalho. Desses locais, a casa é a única visualizada por ser o local que os agentes passam a maior parte de seu tempo e tempos de abertura e fechamento não são aplicáveis. A Tabela 4.1 mostra os horários de funcionamento dos estabelecimentos. Os horários estão no formato 0-23 horas, sendo que "-" significa que não abre naquele dia. Além disso, os horários são em formato texto porque são passados os horários para todos os dias da semana. A semana esta sendo considerada como iniciando em uma segunda-feira e terminando no domingo. Fora isso, os locais marcados como fixos são locais que o agente deve permanecer até que o horário de funcionamento termine. Já os considerados eventuais pode se ir algumas vezes e ficar por um tempo indeterminado.

Tabela 4.1: Horário de funcionamento dos estabelecimentos.

Local	Abertura	Encerramento
Casa	X	X
Escola	7,7,7,7,7,-,-	11,11,11,11,11,-,-
Fliperama	15,15,15,15,15,9,8	21,21,21,21,21,21,21
Mercado	7,7,7,7,7,7,8	21,21,21,21,21,20,19
Trabalho	8,8,8,8,8,-,-	17,17,17,17,17,-,-

As preferências do agente sobre as anotações estão sendo usadas para ajudar o agente a decidir o que fazer. Elas permitem o agente sentir-se ou alegre ou sofrer. Basicamente, cada um dos agentes tem seu próprio conjunto de preferências. Entretanto, o presente exemplo irá focar somente na anotação de energia. Cada um dos agentes é atraído por uma valoração positiva desse valor. Por exemplo, o limite de Albert é 50. John é 55. Millie é 30 e Nina é 53.

Esses valores de limite permite a aplicação determinar quando uma emoção será sentida. Para Albert o valor 50 é o ponto médio de energia no qual ele não sente nenhuma emoção e qualquer valor acima ou abaixo disso ocasiona uma emoção que pode virar sentimento. Dessa forma, quando Albert passar a ter 71 de energia ele passará a experimentar alegria. De forma igual, quando Millie atingir o valor 9 de energia estará sentindo sofrimento. O valor de 71 e 9 são justificados porque o processo de avaliação os cria com base nas preferências as emoções. Albert, quando a energia está em 71, possui a emoção de alegria no valor 21. Sendo assim, o agente experimenta o sentimento de alegria com valor 1 porque o limite para uma emoção virar sentimento esta configurada com o valor 20 (pág. 48). A mesma coisa acontece com o agente Millie, sua energia está no valor 9 que garante a emoção de sofrimento com valor 21 e a percepção da emoção com valor 1.

O presente exemplo possui 4 agentes que compartilham o mesmo código *AgentSpeak*. Entretanto, as suas bases de crenças estão configuradas com ontologias diferentes. Assim, eles têm diferentes crenças iniciais sem precisar escrever código. O processo de delibera-

Listagem 4.9: Amostra de código referente ao processo de avaliação considerando as preferências.

```
+?appraisal
    <- .findall(prRe(A,P), priority(repulse, A, P), LRE);
       .findall(prAt(A,P), priority(attract, A, P), LAT);
       .findall(OBJ, OBJ[source(percept)], PERCEPTS);
       ?appraisalPercept(PERCEPTS, LRE, LAT).

+?appraisalPercept([], LRE, LAT).
+?appraisalPercept([PERCEPT|R], LRE, LAT)
     : PERCEPT =.. [FUNCTOR,TERMS,_]
    <- ?appraisalPercept(R, LRE, LAT);
       .findall(X, PERCEPT[X], ANNOTS);
       ?appraisalOnePercept(PERCEPT, ANNOTS, LRE, VALR);
       ?appraisalOnePercept(PERCEPT, ANNOTS, LAT, VALP).

+?appraisalOnePercept(_, _, [], 0).
+?appraisalOnePercept(PERCEPT, ANNOTS, [PR|L], NEWVAL)
    <- ?appraisalOnePercept(PERCEPT, ANNOTS, L, VAL);
       ?appraisalOneAnnotation(ANNOTS, PR, VAL, NEWVAL);
       ?evalAppraisal(PERCEPT, PR, VAL, NEWVAL).

+?appraisalOneAnnotation([], _, VAL, VAL).
+?appraisalOneAnnotation([H|R], PR, OLDVAL, NEWVAL)
     :  H =.. [NAMEPRI,          [VALTER|_], _]
     & PR =.. [FUNCPRI, [NAMEPRI,PARTPRI|_], _]
    <- ?appraisalOneAnnotationValue(VALTER, PARTPRI, FUNCPRI, OLDVAL, VAL);
       ?appraisalOneAnnotation(R, PR, VAL, NEWVAL).
+?appraisalOneAnnotation([H|R], PR, OLDVAL, NEWVAL)
    <- ?appraisalOneAnnotation(R, PR, OLDVAL, NEWVAL).
```

ção inicia avaliando a percepção de energia, se ela chegou em zero então o personagem é removido da visualização. Depois, é feita a avaliação das emoções e um comportamento é escolhido.

No inicio da simulação, a base de crenças converte os itens guardados na ontologia como preferências para crenças no formato “priority(Type, Annotation, Partition)”. Assim, o processo pega todas essas crenças em duas listas e cria uma lista de objetos percebidos e as envia para a próxima etapa que recupera todas as anotações de um objeto e realiza o cálculo baseado nas prioridades do que é importante. A Listagem 4.9 mostra esse procedimento, o operador “=..” é usado para desmanchar a variável em uma lista com três elementos: o termo, os parâmetros e as anotações.

O cálculo que dispara as emoções baseando-se nas preferências do indivíduo é realizado pela consulta “appraisalOneAnnotationValue”. Essa consulta recebe por parâmetro: o valor da anotação; o valor de limite da mesma; seu tipo (atração por números abaixo ou acima do limite); valor atual das anotações do objeto; e, o resultado do cálculo. Esse cálculo pode ser visto na Listagem 4.10.

Esse processo leva em consideração o tipo da anotação sendo analisada. Por exemplo, elas podem ser números ou átomos. O cálculo verifica se os dois valores informados nos dois primeiros parâmetros são iguais, se for então o resultado é a devolução do valor antigo. No caso de ser uma preferência por negativa então o segundo parâmetro que é o valor informado na ontologia é subtraído do primeiro parâmetro que é o valor atual ou

Listagem 4.10: Amostra de código referente ao cálculo de uma anotação.

```
// NUMBER
+?appraisalOneAnnotationValue(VALT, VALP, _, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT = VALP
    <- NEWV = OLDV.
+?appraisalOneAnnotationValue(VALT, VALP, prRe, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT < VALP
    <- NEWV = OLDV+(VALP-VALT).
+?appraisalOneAnnotationValue(VALT, VALP, prRe, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT > VALP
    <- NEWV = OLDV+(VALP-VALT).
+?appraisalOneAnnotationValue(VALT, VALP, prAt, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT > VALP
    <- NEWV = OLDV+(VALT-VALP).
+?appraisalOneAnnotationValue(VALT, VALP, prAt, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT < VALP
    <- NEWV = OLDV+(VALT-VALP).

// ATOM
+?appraisalOneAnnotationValue(VALT, VALT, FUNCTOR, OLDV, OLDV+1)
     : .atom(VALT).
+?appraisalOneAnnotationValue(VALT, VALP, FUNCTOR, OLDV, OLDV)
     : .atom(VALT) & .atom(VALP).
```

Listagem 4.11: Amostra de código referente à atualização das emoções.

```
+?evalAppraisal(_, _, OLDVAL, OLDVAL).
+?evalAppraisal(PCP, PREF, OLDVAL, NEWVAL)
     : PCP =..  [FUNCP,          [], _]
     & PREF =.. [FUNCF, [TERM1F|_], _]
     & NEWVAL > 0
    <- .concat("", FUNCP, "_", TERM1F, INDIVIDUAL);
       ?updateAppraisal(joy, INDIVIDUAL, NEWVAL).
+?evalAppraisal(PCP, PREF, OLDVAL, NEWVAL)
     : PCP =..  [FUNCP,          [], _]
     & PREF =.. [FUNCF, [TERM1F|_], _]
     & NEWVAL < 0
    <- .concat("", FUNCP, "_", TERM1F, INDIVIDUAL);
       ?updateAppraisal(distress, INDIVIDUAL, NEWVAL).
```

observado da anotação. No caso de ser uma preferência positiva então o valor atual da anotação é subtraído do valor informado.

O processo de atualização de emoções é disparado pela consulta “evalAppraisal” presente na Listagem 4.9 na linha 19. Essa consulta recebe como parâmetro: a percepção; a crença de prioridade; o valor anterior; e, o valor calculado das anotações presentes na percepção. Assim, a consulta “evalAppraisal” é utilizada para averiguar os dados antes de ocorrer a chamada à consulta “updateEmotion” que é similar a explicada anteriormente. O código dessa consulta está apresentado na Listagem 4.11 e é responsável simplesmente por construir o nome do indivíduo usando o termo da percepção e o valor informado na ontologia.

5 CONCLUSÃO

O presente trabalho apresentou um mecanismo desenvolvido para a plataforma *Jason* que permite aos agentes da simulação possuírem sentimentos. Todo o código do sistema, incluído os exemplos e o presente texto, pode ser acessado pelo repositório *online* hospedado via *GitHub*[1]. Os apêndices presentes visam facilitar a consulta de alguns dos artefatos lá presentes e debatidos aqui. O presente capítulo esta dividido em duas seções. A primeira serve para avaliar o sistema desenvolvido e discutir melhorias futuras. A segunda seção serve para apresentar as considerações finais fazendo um pequeno retrospecto sobre o que se esperava do trabalho inicialmente e o que foi alcançado.

5.1 Avaliação e Trabalhos Futuros

Para avaliar o sistema criado, os critérios levados em consideração são de simplicidade de uso e configuração. Uma coisa é considerada simples de usar quando requer pouco esforço para ser usada, enquanto simplicidade de configuração quer dizer que se o usuário desejar alterar determinados parâmetros ele deve ser capaz disso sem um grande esforço. Alguns exemplos de parâmetros podem ser as próprias emoções porque o usuário deseja utilizar apenas um subconjunto delas, ou ser a alteração do processo de avaliação que resulta no valor da emoção.

Conforme dito anteriormente no capítulo 4, a configuração mínima exigida é que o usuário especifique na configuração do *Jason* a base de crenças desenvolvida. Fora isso, o sistema ainda precisa conhecer os limites mínimos para uma emoção virar sentimento que pode ser informada de duas formas. A primeira é como crenças iniciais dos agentes, conforme demonstrado na Listagem 4.2 (pág. 44), porém essa forma de uso é extremamente cansativa porque se precisa informar várias coisas. Cada uma das emoções precisa ser configurada, e ao todo existem 22 delas. Para cada uma são necessárias 3 relações que dizem de quem é a configuração, qual emoção está sendo configurada e qual é o valor mínimo para sua ativação. O segundo é o formato utilizado pela seção 4.3 (pág. 48). Esse armazena esses valores de configuração na própria ontologia. Sendo assim, basta infor-

[1] http://github.com/rlucca/Maro

mar para a base de crenças a ontologia correta e essa configuração não precisa ser feita via código *AgentSpeak*.

Agora, a plataforma *Jason* será capaz de criar as percepções das emoções quando for adicionada uma percepção "step". Sendo assim, o ambiente ou algum agente deve informar essa crença para os demais. Essas duas formas foram usadas na seção 4.1 (pág. 41). Entretanto, cabe salientar novamente que antes de o agente acreditar que essa crença foi recebida ele deveria limpar as crenças existentes sobre seus sentimentos para evitar duplicações (ver figura da pág. 42).

Logo, a necessidade de uma percepção "step" ser adicionada à base de crenças sempre que se deseja fazer uma avaliação das emoções atuais afeta o projeto do usuário. Essa é uma necessidade do mecanismo construído, e é invasiva porque torna o sistema do usuário descontínuo nessa parte das emoções. Além disso, ele deve se preocupar com quem irá informar essa percepção. Eliminar essa necessidade é um plano futuro pois torna o desenvolvimento mais transparente permitindo-o ser melhor utilizado. Essa atualização pode ser colocada no momento da consulta da percepção "feeling" ou em algum outro ponto.

Atualmente, se o usuário não quiser escrever código para ligar percepções com as emoções então elas não podem ser criadas e percebidas. Esse é um ponto altamente invasivo e crítico. Todavia, um modelo de percepção padrão baseado nas preferências do personagem é simplista e não pode alcançar todas as emoções existentes. A necessidade de sempre ter os limites mínimos para todas as 22 emoções precisa ser revista porque em determinadas simulações somente um conjunto ou algumas emoções selecionadas poderiam ser desejadas para virarem sentimentos. Esse comportamento é alcançado na atual implementação simplesmente não inserindo as relações necessárias para aquela emoção.

Além disso, remover a necessidade de o cálculo da potência da emoção ser feita em *Java* para ser interessante. A vantagem de fazer isso em código *AgentSpeak* é uma maior transparência no que está sendo feito e como se está fazendo quando for necessário fugir da forma proposta. Entretanto, alterar o cálculo hoje é possível estendendo a classe *OwlApi* de leitura da ontologia com a finalidade de sobrecarregar as funções *summaryOf* que realizam esses cálculos.

De maneira similar, as consultas de crenças feitas em código *AgentSpeak* são transparentes para o usuário quanto ao local de armazenamento. Entretanto, se o usuário precisar saber onde a crença está sendo salva, e pode verificar a existência da anotação "ontology". Se ela existir então a crença esta sendo armazenada na ontologia especificada. Caso contrário, ela está sendo guardada na base de crenças da plataforma.

Infelizmente, o mecanismo de consulta de crenças não esta ainda perfeito e exige que o usuário quando faça uma consulta utilize sempre *strings* ou números inteiros. Além disso, como foi visto nos exemplos do capítulo 4, a responsabilidade de ligar as percepções as relações necessárias à ontologia afetiva é do usuário. Mesmo sendo capaz de fazer uso

da ontologia de preferência, todas as emoções não podem ainda ser concluídas só com base no que se encontra configurado. Isso se deve principalmente ao fato de que algumas emoções são complexas exigindo conhecimento de que um personagem esta interferindo com um determinado objetivo ou que uma meta afeta outra ou que uma ação é fora dos padrões do ator.

Sobre a ontologia afetiva, ela foi testada de diferentes formas na integração com o *Jason*. Os pontos de melhora dessa ontologia são três. O primeiro seria a inclusão do conceito de empatia (pág. 26) como sendo um número a ser informado pelo usuário. O segundo é a remoção do conhecimento de amigo e inimigo. Esse conhecimento pode ou deveria ser deduzido a partir das emoções *Love* e *Hate*. Terceiro, transformar o conceito de probabilidade em uma relação numérica. Essas alterações não melhoram a facilidade do usuário, mas aumentam o controle sobre o ator.

5.2 Considerações Finais

Um sistema que estende a base de crenças do agente *Jason* para permitir que o mesmo tenha emoções segundo o modelo desenvolvido por Ortony, Collins e Clore (1988) foi apresentado. Assim, o desenvolvimento se baseou em utilizar uma ontologia para classificar as emoções sendo sentidas com base nas crenças e percepções do agente. Dessa forma, a base de crenças da plataforma foi alterada para manipular ontologias permitindo, de maneira transparente, realizar inserções, remoções e consultas.

O objetivo principal foi a construção de uma ferramenta que permita definir o comportamento emocional de personagens virtuais. Por isso foram utilizadas as ontologias como seus principais elementos por serem artefatos reutilizáveis. O objetivo de ter uma configuração mínima para criar emoções não foi alcançado porque exige do usuário um certo nível de conhecimento da mesma e um refinamento manual dos ajustes criados. Considero também que o objetivo de não usar regras foi alcançado visto que todas elas visam facilitar o usuário e como elas são parte integrante da versão 2 da *OWL* não há razão para não se usá-las.

Entretanto, os artefatos desenvolvidos baseados em ontologia são todas contribuições importantes. A primeira, a ontologia de preferências, descreve como os agentes irão interpretar as anotações existentes em crenças ou em percepções usando o conceito de *affordance*. A segunda contribuição é a ontologia afetiva possuindo as 22 emoções do modelo original descritas e validadas pelas aplicações desenvolvidas.

Inicialmente, o objetivo do trabalho era simular a construção do comportamento afetivo usando os agentes da plataforma *Jason* de maneira integrada com alguma aplicação em outra linguagem de programação. Entretanto, essa integração não envolvia adaptar nenhuma aplicação e para focar na criação dos artefatos do projeto esse objetivo foi removido visando um melhor foco no almejado. Além disso, a detecção do que é relevante

funcionou conforme esperado.

Os principais resultados são as aplicações construídas com o sistema. Elas servem, principalmente, para validar o sistema desenvolvido e possuem diferentes propósitos conforme foi visto no capítulo anterior. Ao todo foram feitas quatro aplicações e demonstram que mesmo sem um conceito de decaimento automático do valor da emoção os artefatos desenvolvidos funcionam de uma maneira adequada.

REFERÊNCIAS

ADAM, C.; HERZIG, A.; LONGIN, D. A logical formalization of the occ theory of emotions. *Synthese*, Springer, v. 168, p. 201–248, 2009.

BATES, J. The role of emotion in believable agents. *Commun. ACM*, ACM, New York, NY, USA, v. 37, n. 7, p. 122–125, 1994.

BENTA, K.; RARĂU, A.; CREMENE, M. Ontology Based Affective Context Representation. In: EURO AMERICAN CONFERENCE ON TELEMATICS AND INFORMATION SYSTEMS, EATIS, 2007, Faro, Portugal. ***Proceedings...*** New York, USA: ACM New York, 2007.

BICKMORE, T. *Relational Agents: Effecting Change through Human-Computer Relationships*. Tese (Doutorado) — Massachusetts Institute of Technology, 2003. Disponível em: <http://affect.media.mit.edu/projectpages/relational/>. Acesso em: 28 oct. 2009.

BICKMORE, T.; SCHULMAN, D. A virtual laboratory for studying long-term relationships between humans and virtual agents. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, AAMAS, 8th., 2009, Budapeste, Hungria. ***Proceedings...*** [S.l.]: International Foundation for Autonomous Agents and Multiagent Systems, 2009. v. 1, p. 297–304.

BORDINI, R. et al. *Multi-Agent Tools: Languages, Platforms and Applications*. [S.l.]: Springer, 2009.

BORDINI, R. et al. *Jason: A java-based agentspeak interpreter used with saci for multi-agent distribution over the net, manual, first release edition*. Jan 2004. Disponível em: <http://jason.sourceforge.net>. Acesso em: 28 oct. 2009.

BOROD, J. C. Psychological models of emotion. SCHERER, K. R. In: ____. *The neuropsychology of emotion*. [S.l.]: Oxford University Press, 2000. cap. 6.

DAMÁSIO, A. *O erro de Descartes: emoção, razão e o cérebro humano*. [S.l.]: Companhia das Letras, 2004.

DAMIANO, R.; PIZZO, A. Emotions in drama characters and virtual agents. *AAAI Sprint Symposium on Emotion, Personality, and Social Behavior*, v. 296, 2008.

DASTANI, M. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 16, p. 214–248, 2008.

DIAS, J.; PAIVA, A. Feeling and reasoning: A computational model for emotional characters. *Progress in artificial intelligence*, Springer, p. 127–140, 2005.

DIAS, J.; PAIVA, A. Agents with Emotional Intelligence for Storytelling. *Doctoral Mentoring Program*, 2009.

DOYLE, P.; HAYES-ROTH, B. Agents in annotated worlds. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2nd., 1998, Melbourne, Austrália. ***Proceedings...*** New York, NY, USA: ACM, 1998. p. 173–180.

ELLIOTT, C. D. *The affective reasoner: a process model of emotions in a multi-agent system*. 135 f. Tese (Doutorado em Ciência da Computação) — Northwestern University, Evanston, IL, USA, 1992.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: WORKSHOP ON INTELLIGENT AGENTS III, AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, ECAI '96, 1996, Budapeste, Hungria. ***Proceedings...*** London, UK: Springer-Verlag, 1997. p. 21–35.

GRATCH, J. Émile: Marshalling passions in training and education. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 4th, Barcelona, Spain. ***Proceedings...*** New York, NY, USA: ACM, 2000. p. 325–332.

GRATCH, J.; MARSELLA, S. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, Elsevier, v. 5, p. 269–306, 2004.

GRUBER, T. et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, v. 5, p. 199–220, 1993.

GUTIÉRREZ, M. et al. An ontology of virtual humans: Incorporating semantics into human shapes. *The Visual Computer*, Springer-Verlag, Secaucus, NJ, USA, v. 23, p. 207–218, 2007. Disponível em: <http://infoscience.epfl.ch/record/100019/files/Gutierrez_Thalmann_Vexo_SVE_05.pdf>. Acesso em: 18 jan. 2012.

HWANG, W.; YANG, J.-J. Ontology-based emotion system for digital environment. *Agent Computing and Multi-Agent Systems*, Springer, v. 5044, p. 464–472, 2009.

INGRAND, F.; GEORGEFF, M.; RAO, A. An architecture for real-time reasoning and system control. *IEEE EXPERT INTELLIGENT SYSTEMS and THEIR APPLICATIONS*, IEEE Computer Society, p. 34–44, 1992.

JENNINGS, N. R.; SYCARA, K.; WOOLDRIDGE, M. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, p. 7–38, 1998.

KALLMANN, M.; THALMANN, D. Modeling objects for interaction tasks. In: WORKSHOP ON COMPUTER ANIMATION AND SIMULATION, EGCAS'97, 8th, 1997, Budapeste, Hungria. ***Proceedings...*** Lisbon, Portugal: Springer Verlag Wien, 1998.

KIRSCH, D. *The Affective Tigger*. Dissertação (Mestrado) — Massachusetts Institute of Technology, 1999. Disponível em: <http://affect.media.mit.edu/projectpages/archived/projects/Atigger.html>.

KLAPISCAK, T.; BORDINI, R. Jasdl: A practical programming approach combining agent and semantic web technologies. *Declarative Agent Languages and Technologies VI*, p. 91–110, 2009.

KSHIRSAGAR, S. A multilayer personality model. In: INTERNATIONAL SYMPOSIUM ON SMART GRAPHICS, 2nd., 2002, Hawthorne, USA. ***Proceedings...*** [S.l.]: ACM, 2002. p. 107–115.

LAIRD, J.; VANLENT, M. Human-level ai's killer application: Interactive computer games. *AI magazine*, v. 22, n. 2, 2001.

LEDOUX, J. *The emotional brain: The mysterious underpinnings of emotional life*. [S.l.]: Simon & Schuster, 1998.

LERA, I. et al. Semantic model for facial emotion to improve the human computer interaction in ami. In: SYMPOSIUM OF UBIQUITOUS COMPUTING AND AMBIENT INTELLIGENCE, 3rd., 2008, Salamanca, Espanha. ***Proceedings...*** [S.l.]: Springer, 2008. p. 139–148.

LÓPEZ, J. M. et al. Towards an ontology for describing emotions. In: WORLD SUMMIT ON THE KNOWLEDGE SOCIETY, 1st., 2008, Athens, Greece. ***Proceedings...*** Berlin, Heidelberg: Springer-Verlag, 2008. p. 96–104.

MOREIRA, Á. et al. Agent-oriented programming with underlying ontological reasoning. *Declarative Agent Languages and Technologies III*, Springer, p. 155–170, 2006.

NAREYEK, A. Review: Intelligent agents for computer games. *Computers and Games*, Springer, p. 414–422, 2001.

NETO, A. F. B.; SILVA, F. S. C. da. On the Construction of Synthetic Characters with Personality and Emotion. *Advances in Artificial Intelligence* — SBIA 2010: 20th Brazilian Symposium on Artificial Intelligence, Springer-Verlag New York Inc, São Bernardo Do Campo, Brazil, 2010.

ORTONY, A.; COLLINS, A.; CLORE, G. *The cognitive structure of emotions*. [S.l.]: Cambridge university press, 1988.

PAIVA, D. C.; VIEIRA, R.; MUSSE, S. R. Ontology-based crowd simulation for normal life situations. In: THE COMPUTER GRAPHICS INTERNATIONAL, CGI2005, 2005, Stony Brook, New York, USA. ***Proceedings...*** [S.l.]: IEEE Computer Society, 2005. p. 221–226.

PICARD, R. W. Affective computing. Cambridge: MIT, 1998.

RAO, A. AgentSpeak (L): BDI agents speak out in a logical computable language. *Agents Breaking Away*, Springer, p. 42–55, 1996.

RICCI, A.; VIROLI, M.; OMICINI, A. CArtAgO: An infrastructure for engineering computational environments in MAS. *Workshop E4MAS*, p. 102–119, 2006.

SHOHAM, Y. Agent-oriented programming. *Artificial intelligence*, Elsevier, v. 60, n. 1, p. 51–92, 1993.

STEUNEBRINK, B. R.; DASTANI, M.; MEYER, J. J. C. Emotions to control agent deliberation. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, AAMAS, 9th., 2010, Toronto, Canada. ***Proceedings...*** [S.l.], 2010. v. 1, p. 973–980.

TERZOPOULOS, D. et al. Behavioral modeling and animation (panel): past, present, and future. In: INTERNATIONAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES. ***Painel...*** New York, USA: ACM New York, 1998. p. 209–211.

ZHANG, h. et al. Emotional agent in serious game (dino). In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, AAMAS, 8th., 2009, Budapeste, Hungria. ***Proceedings...*** [S.l.]: International Foundation for Autonomous Agents and Multiagent Systems, 2009. v. 2, p. 1385–1386.

APÊNDICE A FONTES DA APLICAÇÃO DE TESTE NÃO INTERATIVA

Arquivo do projeto da plataforma - eoaus.mas2j

```
MAS eoaus {
    infrastructure: Centralised
    // timeout = 2 minutes
    environment: maro.example.sims.House(120000, 1000, 4, FALSE, "sims.owl")
    agents:
        millie
          beliefBaseClass maro.wrapper.BBAffective("ontology/occ-tbox.owl")
          agentClass maro.wrapper.UniqueAnnotation ;
        nina nope.asl #1;
        john nope.asl #1;
        albert nope.asl #1;
    aslSourcePath: "asl";
}
```

Arquivo do agente Millie - millie.asl

```
agent(john).
agent(jose).
agent(dilu).
agent(millie).
object(television).
~sameAs(john, jose, dilu, millie, television, milliesCar).
high(lhigh).
mid(lmid).
low(llow).
none(lnone).
hasSetup(millie, setup1).           hasSetup(millie, setup2).
hasSetup(millie, setup3).           hasSetup(millie, setup4).
hasSetup(millie, setup5).           hasSetup(millie, setup6).
hasSetup(millie, setup7).           hasSetup(millie, setup8).
hasSetup(millie, setup9).           hasSetup(millie, setup10).
hasSetup(millie, setup11).          hasSetup(millie, setup12).
hasSetup(millie, setup13).          hasSetup(millie, setup14).
hasSetup(millie, setup15).          hasSetup(millie, setup16).
hasSetup(millie, setup17).          hasSetup(millie, setup18).
hasSetup(millie, setup19).          hasSetup(millie, setup20).
hasSetup(millie, setup21).          hasSetup(millie, setup22).
hasThreshold(setup1, 10).           hasThresholdType(setup1, "Joy").
hasThreshold(setup2, 11).           hasThresholdType(setup2, "Distress").
hasThreshold(setup3, 12).           hasThresholdType(setup3, "Hope").
hasThreshold(setup4, 13).           hasThresholdType(setup4, "Fear").
hasThreshold(setup5, 14).           hasThresholdType(setup5, "FearsConfirmed").
hasThreshold(setup6, 15).           hasThresholdType(setup6, "Satisfaction").
hasThreshold(setup7, 16).           hasThresholdType(setup7, "Relief").
hasThreshold(setup8, 17).           hasThresholdType(setup8, "Disappointment").
hasThreshold(setup9, 18).           hasThresholdType(setup9, "HappyFor").
hasThreshold(setup10, 19).          hasThresholdType(setup10, "SorryFor").
hasThreshold(setup11, 20).          hasThresholdType(setup11, "Gloating").
hasThreshold(setup12, 21).          hasThresholdType(setup12, "Resentment").
hasThreshold(setup13, 22).          hasThresholdType(setup13, "Love").
hasThreshold(setup14, 23).          hasThresholdType(setup14, "Hate").
```

```
hasThreshold(setup15, 24).          hasThresholdType(setup15, "Admiration").
hasThreshold(setup16, 25).          hasThresholdType(setup16, "Pride").
hasThreshold(setup17, 26).          hasThresholdType(setup17, "Shame").
hasThreshold(setup18, 27).          hasThresholdType(setup18, "Reproach").
hasThreshold(setup19, 28).          hasThresholdType(setup19, "Gratitude").
hasThreshold(setup20, 29).          hasThresholdType(setup20, "Gratification").
hasThreshold(setup21, 30).          hasThresholdType(setup21, "Anger").
hasThreshold(setup22, 31).          hasThresholdType(setup22, "Remorse").
////////////////////////////////////////////////////////////
hasAppraisal(john, john_relationsWith_millie).
hasAppraisal(john, john_relationsWith_jose).
hasNiceness(john_relationsWith_millie, 20).
hasPerson(john_relationsWith_millie, millie).
hasNiceness(john_relationsWith_jose, 5).
hasPerson(john_relationsWith_jose, jose).
hasAppraisal(john, john_inlove_millie).
hasAppraisal(john, john_repulse_television).
hasFamiliarity(john_inlove_millie, 55).
hasPerson(john_inlove_millie, millie).
hasFamiliarity(john_repulse_television, -11).
hasAppraisal(john, john_happyFor_millie).
hasPerson(john_happyFor_millie, millie).
hasDesireOther(john_happyFor_millie, 11).
hasDeserved(john_happyFor_millie, 30).
hasAppraisal(john, john_fears_demission).
hasDesireSelf(john_fears_demission, -11).
hasLikelihood(john_fears_demission, lmid).
hasAppraisal(john, john_satisfaction_lunch).
hasEffort(john_satisfaction_lunch, 3).
hasPreviousIntensity(john_satisfaction_lunch, 19).
hasRealized(john_satisfaction_lunch, 40).
hasAppraisal(john, john_remorse_itself).
hasLikelihood(john_remorse_itself, lnone).
hasDesireSelf(john_remorse_itself, -1).
hasJudge(john_remorse_itself, john).
hasJudgeness(john_remorse_itself, -7).
////////////////////////////////////////////////////////////
hasAppraisal(jose, jose_relationsWith_john).
hasAppraisal(jose, jose_relationsWith_millie).
hasAppraisal(jose, jose_relationsWith_dilu).
hasNiceness(jose_relationsWith_john, -33).
hasPerson(jose_relationsWith_john, john).
hasNiceness(jose_relationsWith_millie, 0).
hasPerson(jose_relationsWith_millie, millie).
hasNiceness(jose_relationsWith_dilu, 77).
hasPerson(jose_relationsWith_dilu, dilu).
isAppraisalOf(jose_admire_dilu, jose).
hasJudge(jose_admire_dilu, dilu).
hasJudgeness(jose_admire_dilu, 22).
hasAppraisal(jose, jose_resentment_john).
hasPerson(jose_resentment_john, john).
hasDesireOther(jose_resentment_john, 61).
hasDeserved(jose_resentment_john, -36).
hasAppraisal(jose, jose_frustation_lunch).
hasEffort(jose_frustation_lunch, 12).
hasPreviousIntensity(jose_frustation_lunch, 40).
hasRealized(jose_frustation_lunch, -7).
hasAppraisal(jose, jose_hopes_promotion).
hasDesireSelf(jose_hopes_promotion, 44).
hasLikelihood(jose_hopes_promotion, lhigh).
hasAppraisal(jose, jose_gratification_itself).
hasJudge(jose_gratification_itself, jose).
hasJudgeness(jose_gratification_itself, 33).
hasLikelihood(jose_gratification_itself, lnone).
hasDesireSelf(jose_gratification_itself, 88).
////////////////////////////////////////////////////////////
hasAppraisal(dilu, dilu_relationsWith_jose).
hasNiceness(dilu_relationsWith_jose, 17).
hasPerson(dilu_relationsWith_jose, jose).
hasFamiliarity(dilu_attracted_television, 30).
hasSomething(dilu_attracted_television, television).
isAppraisalOf(dilu_attracted_television, dilu).
hasAppraisal(dilu, dilu_sorryFor_jose).
```

```
hasPerson(dilu_sorryFor_jose, jose).
hasDesireOther(dilu_sorryFor_jose, -11).
hasDeserved(dilu_sorryFor_jose, -30).
hasAppraisal(dilu, dilu_fearsc_lunch).
fearsConfirmed(dilu_fearsc_lunch).
hasAppraisal(dilu, dilu_anger_jose).
hasLikelihood(dilu_anger_jose, lnone).
hasDesireSelf(dilu_anger_jose, -7).
hasJudge(dilu_anger_jose, jose).
hasJudgeness(dilu_anger_jose, -8).
////////////////////////////////////////////////////////
hasAppraisal(millie, millie_relationsWith_dilu).
hasNiceness(millie_relationsWith_dilu, -4).
hasPerson(millie_relationsWith_dilu, dilu).
isAppraisalOf(millie_reproach_john, millie).
hasJudge(millie_reproach_john, john).
hasJudgeness(millie_reproach_john, -14).
hasAppraisal(millie, millie_gloating_dilu).
hasPerson(millie_gloating_dilu, dilu).
hasDesireOther(millie_gloating_dilu, -29).
hasDeserved(millie_gloating_dilu, 29).
relief(millie_relief_gas).
isAppraisalOf(millie_relief_gas, millie).
isAppraisalOf(millie_enjoy_cooking, millie).
hasLikelihood(millie_enjoy_cooking, lnone).
hasDesireSelf(millie_enjoy_cooking, 11).
hasAppraisal(millie, millie_gratitudes_car).
hasLikelihood(millie_gratitudes_car, lnone).
hasDesireSelf(millie_gratitudes_car, 68).
hasJudgeness(millie_gratitudes_car, 3).
hasJudge(millie_gratitudes_car, milliesCar).
////////////////////////////////////////////////////////
!start.
+!start
    <-  !functions;
        !!emocoes.
+!relacoes
    <-  .findall(friend(X,Y), hasFriend(X,Y), LF); ?printBR(LF, "Background:␣");
        LF=[friend("jose","dilu"), friend("dilu","jose"), friend("john","millie"),
            friend("john","jose")]; nope;
        .findall(enemy(X,Y), hasEnemy(X,Y), LE); ?printBR(LE, "Background:␣");
        LE=[enemy("millie","dilu"), enemy("jose","john")]; nope;
        .findall(know(X,Y), hasKnow(X,Y), LB); ?printBR(LB, "Background:␣");
        LB=[know("jose","john"), know("john","millie"), know("jose","dilu"), know("john"
            ,"jose"), know("millie","john"), know("millie","jose"), know("jose","millie"
            ), know("dilu","jose"), know("dilu","millie"), know("millie","dilu")]; nope.
+?printBR([], _).
+?printBR([H|R], PROMPT)
    <-  .println(PROMPT, H);
        ?printBR(R, PROMPT).
+!emocoes
    <-  !relacoes;
        !testeEmocoes(john); !testeEmocoes(jose);
        !testeEmocoes(dilu); !testeEmocoes(millie);
        !!perguntas.
+!testeEmocoes(john)
    <-  ?love("john_inlove_millie"); nope;
        ?hate("john_repulse_television"); nope;
        ?happyFor("john_happyFor_millie"); nope;
        ?fear("john_fears_demission"); nope;
        ?satisfaction("john_satisfaction_lunch"); nope;
        ?distress("john_remorse_itself"); nope;
        ?shame("john_remorse_itself"); nope;
        ?remorse("john_remorse_itself"); nope;
        .eval(false, feeling(remorse, _)); nope;
        .println("john␣tudo␣certo!").
+!testeEmocoes(jose)
    <-  ?admiration("jose_admire_dilu"); nope;
        ?resentment("jose_resentment_john"); nope;
        ?disappointment("jose_frustation_lunch"); nope;
        ?hope("jose_hopes_promotion"); nope;
        ?pride("jose_gratification_itself"); nope;
        ?joy("jose_gratification_itself"); nope;
```

```
        ?gratification("jose_gratification_itself"); nope;
        .eval(false, feeling(gratification, _)); nope;
        .println("jose␣tudo␣certo!").
+!testeEmocoes(dilu)
    <-  ?love("dilu_attracted_television"); nope;
        ?sorryFor("dilu_sorryFor_jose"); nope;
        ?fearsConfirmed("dilu_fearsc_lunch"); nope;
        ?distress("dilu_anger_jose"); nope;
        ?reproach("dilu_anger_jose"); nope;
        ?anger("dilu_anger_jose"); nope;
        .eval(false, feeling(anger, _)); nope;
        .println("dilu␣tudo␣certo!").
+!testeEmocoes(millie)
    <-  ?reproach("millie_reproach_john"); nope;
        .eval(false, feeling(reproach,_)); nope;
        ?gloating("millie_gloating_dilu"); nope;
        ?feeling(gloating, 38); nope;
        ?relief("millie_relief_gas"); nope;
        .eval(false, feeling(relief, _)); nope;
        ?joy("millie_enjoy_cooking"); nope;
        ?joy("millie_gratitudes_car"); nope;
        ?feeling(joy, 72); nope;
        ?admiration("millie_gratitudes_car"); nope;
        ?feeling(admiration, 47); nope;
        ?gratitude("millie_gratitudes_car"); nope;
        ?feeling(gratitude, 43); nope;
        .println("millie␣tudo␣certo!").
+!perguntas
    <-  .findall(X, hasKnow("jose", X), AJ);
        .println("Quem␣eh␣conhecido␣por␣jose?␣", AJ);
         AJ=["john","dilu","millie"]; nope;
        .findall(X, hasKnow(X, "dilu"), AD);
        .println("Quem␣conhece␣a␣dilu?", AD);
        AD=["jose","millie"]; nope;
        .findall(X, love(Y) & isAppraisalOf(Y, X) & hasPerson(Y, "millie"), AL);
        .println("Quem␣ama␣a␣millie?", AL);
        AL=["john"]; nope;
        .findall(X, feeling(X, _), FL);
        .println("Que␣sentimentos␣millie␣tem?", FL);
        FL=[joy,gratitude,gloating,admiration]; nope;
        .findall(diff("millie",X), ~sameAs("millie", X), SMMX);
        .findall(diff("millie",X), ~sameAs(X, "millie"), SMXM);
        .difference(SMMX, SMXM, []);
        .difference(SMXM, SMMX, []); nope;
        !!summary.
+!functions
    <- !function(object); !function(data); !function(instance).
+!function(object)
    <- +hasLikelihood("test1", lnone);
       ?hasLikelihood("test1", "lnone");
       .println("insertion␣of␣relation␣ok"); nope;
       +hasLikelihood("test1", lnone)[foo];
       ?hasLikelihood("test1", "lnone")[foo];
       .println("changing␣the␣relation␣to␣add␣a␣annotation␣ok"); nope;
       +hasLikelihood("test1", lnone)[bar];
       ?hasLikelihood("test1", "lnone")[bar];
       .println("changing␣the␣relation␣to␣add␣another␣annotation␣ok"); nope;
       ?hasLikelihood("test1", "lnone")[foo,bar];
       .println("testing␣the␣relation␣to␣get␣the␣two␣annotation␣ok"); nope;
       .eval(false, hasLikelihood(test1, lnone)[foo,bar]);
       .println("testing␣the␣relation␣as␣string␣to␣get␣the␣two␣annotation␣ok"); nope;
       -hasLikelihood("test1", "lnone")[bar];
       ?hasLikelihood("test1", "lnone")[foo];
       .eval(false, hasLikelihood("test1", "lnone")[bar]);
       .println("removing␣annot␣from␣relation␣ok"); nope;
       -hasLikelihood("test1", "lnone");
       ?hasLikelihood("test1", "lnone")[foo];
       -hasLikelihood("test1", "lnone")[foo];
       .eval(false, hasLikelihood("test1", "lnone"));
       .println("removing␣another␣annot␣from␣relation␣ok"); nope.
+!function(data)
    <- +hasDesireSelf("test2", 10);
       ?hasDesireSelf("test2", 10);
```

```
        .println("insertion␣of␣a␣data␣relation␣ok"); nope;
        +hasDesireSelf("test2", 10)[foo];
        ?hasDesireSelf("test2", 10)[foo];
        .println("changing␣the␣data␣relation␣to␣add␣a␣annotation␣ok"); nope;
        +hasDesireSelf("test2", 10)[bar];
        ?hasDesireSelf("test2", 10)[bar];
        .println("changing␣the␣data␣relation␣to␣add␣another␣annotation␣ok"); nope;
        ?hasDesireSelf("test2", 10)[foo,bar];
        .println("testing␣the␣data␣relation␣to␣get␣the␣two␣annotation␣ok"); nope;
        .eval(false, hasDesireSelf(test2, 10)[foo,bar]);
        .println("testing␣the␣relation␣as␣string␣to␣get␣the␣two␣annotation␣ok"); nope;
        -hasDesireSelf("test2", 10)[bar];
        ?hasDesireSelf("test2", 10)[foo];
        .eval(false, hasDesireSelf("test2", 10)[bar]);
        .println("removing␣annot␣from␣relation␣ok"); nope;
        -hasDesireSelf("test2", 10);
        ?hasDesireSelf("test2", 10)[foo];
        -hasDesireSelf("test2", 10)[foo];
        .eval(false, hasDesireSelf("test2", 10));
        .println("removing␣another␣annot␣from␣relation␣ok"); nope;
        +hasDesireSelf("test4", -10);
        ?hasDesireSelf("test4", -10);
        .println("insertion␣of␣negative␣integers␣ok"); nope;
        +hasDesireSelf("test4", -10)[foo];
        ?hasDesireSelf("test4", -10);
        .println("insertion␣of␣negative␣integers␣with␣annot␣ok"); nope;
        -hasDesireSelf("test4", -10)[foo];
        .eval(false, hasDesireSelf("test4", -10));
        .println("removing␣negative␣integers␣with␣annot␣ok"); nope;
        +hasDesireSelf("test4", -10);
        ?hasDesireSelf("test4", -10);
        -hasDesireSelf("test4", -10);
        .eval(false, hasDesireSelf("test4", -10));
        .println("removing␣negative␣integers␣ok"); nope.
+!function(instance)
    <- +low("test3");
        ?low("test3");
        .println("insertion␣of␣concept␣ok"); nope;
        +low("test3")[foo];
        ?low("test3")[foo];
        .println("changing␣the␣concept␣to␣add␣a␣annotation␣ok"); nope;
        +low("test3")[bar];
        ?low("test3")[bar];
        .println("changing␣the␣concept␣to␣add␣another␣annotation␣ok"); nope;
        ?low("test3")[foo,bar];
        .println("testing␣the␣concept␣to␣get␣the␣two␣annotation␣ok"); nope;
        .eval(false, low(test3)[foo,bar]);
        .println("testing␣the␣concept␣as␣string␣to␣get␣the␣two␣annotation␣ok"); nope;
        -low("test3")[bar];
        ?low("test3")[foo];
        .eval(false, low("test3")[bar]);
        .println("removing␣annot␣from␣concept␣ok"); nope;
        -low("test3");
        ?low("test3")[foo];
        -low("test3")[foo];
        .eval(false, low("test3"));
        .println("removing␣another␣annot␣from␣concept␣ok"); nope.
+!summary <- +startUpOK; nope.
+step(X) : startUpOK <- .println("Tudo␣ok"); .stopMAS.
-!X <- .println("teste␣falhou:␣", X); .stopMAS.
```

Arquivo dos demais agentes - nope.asl

```
+step(_) <- nope.
```

APÊNDICE B FONTES DA APLICAÇÃO DE TESTE INTERATIVA

Arquivo do projeto da plataforma - eoaus.mas2j

```
MAS eoaus {
    infrastructure: Centralised
    environment: maro.example.console.Environment(30000, 1000, 1, FALSE)
    agents:
        millie
          beliefBaseClass maro.wrapper.BBAffective("ontology/occ-tbox.owl")
          agentArchClass maro.example.console.AddGui
          agentClass maro.wrapper.UniqueAnnotation;
    aslSourcePath: "asl";
}
```

Arquivo do agente Millie - millie.asl

```
step(0)[source(percept), source(self)].
//------------------------------------------------------------------------------
agent(millie).
high(lhigh).
mid(lmid).
low(llow).
none(lnone).
hasSetup(millie, setup1).   hasSetup(millie, setup2).
hasSetup(millie, setup3).   hasSetup(millie, setup4).
hasSetup(millie, setup5).   hasSetup(millie, setup6).
hasSetup(millie, setup7).   hasSetup(millie, setup8).
hasSetup(millie, setup9).   hasSetup(millie, setup10).
hasSetup(millie, setup11).  hasSetup(millie, setup12).
hasSetup(millie, setup13).  hasSetup(millie, setup14).
hasSetup(millie, setup15).  hasSetup(millie, setup16).
hasSetup(millie, setup17).  hasSetup(millie, setup18).
hasSetup(millie, setup19).  hasSetup(millie, setup20).
hasSetup(millie, setup21).  hasSetup(millie, setup22).
hasThreshold(setup1, 0).    hasThresholdType(setup1, "Joy").
hasThreshold(setup2, 0).    hasThresholdType(setup2, "Distress").
hasThreshold(setup3, 0).    hasThresholdType(setup3, "Hope").
hasThreshold(setup4, 0).    hasThresholdType(setup4, "Fear").
hasThreshold(setup5, 0).    hasThresholdType(setup5, "FearsConfirmed").
hasThreshold(setup6, 0).    hasThresholdType(setup6, "Satisfaction").
hasThreshold(setup7, 0).    hasThresholdType(setup7, "Relief").
hasThreshold(setup8, 0).    hasThresholdType(setup8, "Disappointment").
hasThreshold(setup9, 0).    hasThresholdType(setup9, "HappyFor").
hasThreshold(setup10, 0).   hasThresholdType(setup10, "SorryFor").
hasThreshold(setup11, 0).   hasThresholdType(setup11, "Gloating").
hasThreshold(setup12, 0).   hasThresholdType(setup12, "Resentment").
hasThreshold(setup13, 0).   hasThresholdType(setup13, "Love").
hasThreshold(setup14, 0).   hasThresholdType(setup14, "Hate").
hasThreshold(setup15, 0).   hasThresholdType(setup15, "Admiration").
hasThreshold(setup16, 0).   hasThresholdType(setup16, "Pride").
hasThreshold(setup17, 0).   hasThresholdType(setup17, "Shame").
hasThreshold(setup18, 0).   hasThresholdType(setup18, "Reproach").
hasThreshold(setup19, 0).   hasThresholdType(setup19, "Gratitude").
hasThreshold(setup20, 0).   hasThresholdType(setup20, "Gratification").
```

```
hasThreshold(setup21, 0).   hasThresholdType(setup21, "Anger").
hasThreshold(setup22, 0).   hasThresholdType(setup22, "Remorse").
//------------------------------------------------------------------------------
!start.
+!start <- .at("now␣+1␣s", {+!step}).
+!step : step(X)
    <- .abolish(step(_)); +step(X + 1)[source(percept), source(self)];
       .println("new␣step␣",X+1);
       .at("now␣+1␣s", {+!step}).
-!X <- .println(X).
```

Arquivo de controle da interface interativa - AddGui.java

```
package maro.example.console;
import jason.architecture.AgArch;
import jason.asSyntax.Literal;
import java.util.List;
public class AddGui extends AgArch {
    private boolean initialized = false;
    private CharacterInspectorView civ = null;
    public synchronized void myInit() {
        if (initialized == true) return ;
        civ = new CharacterInspectorView(getAgName());
        civ.setController(this);
        civ.setVisible(true);
        initialized = true;
    }
    @Override public List<Literal> perceive() {
        if (initialized == false) myInit();
        civ.update();
        return super.perceive();
    }
}
```

Arquivo do ambiente - Environment.java

```
package maro.example.console;
import maro.core.IntelligentEnvironment;
public class Environment extends IntelligentEnvironment {
    @Override protected void stepStarted(int step) {
        clearAllPercepts();
        super.stepStarted(step);
    }
    @Override protected void stepFinished(int step, long elapsedTime, boolean byTimeout)
    { tryStop(lastStep); }
}
```

APÊNDICE C FONTES DA APLICAÇÃO DE FUTEBOL

Arquivo de Projeto - soccer.mas2j

```
MAS soccer {
    infrastructure: Centralised
    environment: maro.example.soccer.Environment(30000, 70, 3, FALSE)
    agents:
        watch1 #1
          beliefBaseClass maro.wrapper.BBAffective("ontology/occ-tbox.owl")
          agentClass maro.wrapper.UniqueAnnotation ;
        watch2 #1
          beliefBaseClass maro.wrapper.BBAffective("ontology/occ-tbox.owl")
          agentClass maro.wrapper.UniqueAnnotation ;
        stadium;
        aslSourcePath: "asl";
}
```

Arquivo de ambiente - Environment.java

```
package maro.example.soccer;
import maro.core.IntelligentEnvironment;
import maro.core.ActionLoader;
import jason.asSyntax.ASSyntax;
public class Environment extends IntelligentEnvironment {
        public Environment () {
                super();
                actionLoader.loadAllActions("maro.example.soccer.ea");
        }
        @Override
        protected void stepStarted(int step) {
                getLogger().info("Started␣step␣" + step);
                addPercept(ASSyntax.createLiteral("step", ASSyntax.createNumber(step)));
                super.stepStarted(step);
        }
    @Override
    protected void stepFinished(int step, long elapsedTime, boolean byTimeout) {
        super.stepFinished(step, elapsedTime, byTimeout);
        clearAllPercepts();
    }
}
```

Arquivo do Agente Stadium - stadium.asl

```
!start.
//------------------------------------------------------------------------------
+!start
    : step(N) & timeR(P1, N1) & timeL(P2,N2)
    <- !step(N); ok;
       maro.example.soccer.ia.addPerception(points(N1,P1,N2,P2));
       !!start.
+!start <- !!start.
//------------------------------------------------------------------------------
+!step(0).
+!step(1) <- .print("inicia␣a␣partida!").
+!step(33)
     : timeL(C1, N1) & timeR(C2, N2)
```

```
    <- .print("intervalo");
       -timeL(C1, N1); -timeR(C2, N2);
       +timeL(C2, N2); +timeR(C1, N1);
       -timeAL(_); -timeAR(_).
+!step(34) <- .print("recomeca␣a␣partida!").
+!step(66)
     : timeL(C1, N1) & timeR(C2, N2)
    <- .print("final␣de␣partida.");
       !sendWinner.
//------------------------------------------------------------------------------
+!step(X) : X > 66.
+!step(X)
     : timeL(C1, N1) & timeR(C2, N2) & timeAL(C1) & timeAR(C2)
    <- .random(GOAL); G=math.round(GOAL*3000); !goal(G).
+!step(X)
     : timeL(C1, N1) & timeR(C2, N2)
    <- .random(GOAL); G=math.round(GOAL*3000);
       !goal(G);
       .print(N1, "␣", C1, "␣X␣", C2, "␣", N2);
       +timeAL(C1); +timeAR(C2).
//------------------------------------------------------------------------------
+!goal(G) : G >= 100.
+!goal(G) : G >= 50 & timeL(C1, NAME1)
    <- -timeL(C1, NAME1); +timeL(C1 + 1, NAME1);
       maro.example.soccer.ia.addPerception(goal(NAME1)).
+!goal(G) : G >= 0 & timeR(C2, NAME2)
    <- -timeR(C2, NAME2); +timeR(C2 + 1, NAME2);
       maro.example.soccer.ia.addPerception(goal(NAME2)).
//------------------------------------------------------------------------------
+name(NAME) : timeL(_, _) & timeR(_, _).
+name(NAME) : timeL(_,_) <- +timeR(0, NAME).
+name(NAME) : timeR(_,_) <- +timeL(0, NAME).
+name(NAME) <- +timeL(0, NAME).
//------------------------------------------------------------------------------
+!sendWinner
     : timeL(DRAW, N1) & timeR(DRAW, N2)
    <- maro.example.soccer.ia.addPerception(match(draw,"")).
//------------------------------------------------------------------------------
+!sendWinner
     : timeL(C1, N1) & timeR(C2, N2) & C1 > C2
    <- maro.example.soccer.ia.addPerception(match(win, N1));
       maro.example.soccer.ia.addPerception(match(lose, N2)).
//------------------------------------------------------------------------------
+!sendWinner
     : timeL(C1, N1) & timeR(C2, N2) & C1 < C2
    <- maro.example.soccer.ia.addPerception(match(win,N2));
       maro.example.soccer.ia.addPerception(match(lose,N1)).
```

Arquivo do Agente Watch1 - watch1.asl

```
{ include("names.asl") }
{ include("emotionsWatch1.asl") }
{ include("common.asl") }
{ include("appraisal.asl") }
```

Arquivo do Agente Watch2 - watch2.asl

```
{ include("names.asl") }
{ include("emotionsWatch2.asl") }
{ include("common.asl") }
{ include("appraisal.asl") }
```

Arquivo *AgentSpeak* - emotionsWatch1.asl

```
agent("watch1").
//------------------------------------------------------------------------------
high("lhigh").                      mid("lmid").
low("llow").                        none("lnone").
//------------------------------------------------------------------------------
hasSetup("watch1", "setup1").       hasSetup("watch1", "setup2").
hasSetup("watch1", "setup3").       hasSetup("watch1", "setup4").
hasSetup("watch1", "setup5").       hasSetup("watch1", "setup6").
```

```
hasSetup("watch1", "setup7").       hasSetup("watch1", "setup8").
hasSetup("watch1", "setup9").       hasSetup("watch1", "setup10").
hasSetup("watch1", "setup11").      hasSetup("watch1", "setup12").
hasSetup("watch1", "setup13").      hasSetup("watch1", "setup14").
hasSetup("watch1", "setup15").      hasSetup("watch1", "setup16").
hasSetup("watch1", "setup17").      hasSetup("watch1", "setup18").
hasSetup("watch1", "setup19").      hasSetup("watch1", "setup20").
hasSetup("watch1", "setup21").      hasSetup("watch1", "setup22").
//------------------------------------------------------------------------------
hasThreshold("setup1", 0).          hasThresholdType("setup1", "Joy").
hasThreshold("setup2", 0).          hasThresholdType("setup2", "Distress").
hasThreshold("setup3", 0).          hasThresholdType("setup3", "Hope").
hasThreshold("setup4", 0).          hasThresholdType("setup4", "Fear").
hasThreshold("setup5", 0).          hasThresholdType("setup5", "FearsConfirmed").
hasThreshold("setup6", 0).          hasThresholdType("setup6", "Satisfaction").
hasThreshold("setup7", 0).          hasThresholdType("setup7", "Relief").
hasThreshold("setup8", 0).          hasThresholdType("setup8", "Disappointment").
hasThreshold("setup9", 0).          hasThresholdType("setup9", "HappyFor").
hasThreshold("setup10", 0).         hasThresholdType("setup10", "SorryFor").
hasThreshold("setup11", 0).         hasThresholdType("setup11", "Gloating").
hasThreshold("setup12", 0).         hasThresholdType("setup12", "Resentment").
hasThreshold("setup13", 0).         hasThresholdType("setup13", "Love").
hasThreshold("setup14", 0).         hasThresholdType("setup14", "Hate").
hasThreshold("setup15", 0).         hasThresholdType("setup15", "Admiration").
hasThreshold("setup16", 0).         hasThresholdType("setup16", "Pride").
hasThreshold("setup17", 0).         hasThresholdType("setup17", "Shame").
hasThreshold("setup18", 0).         hasThresholdType("setup18", "Reproach").
hasThreshold("setup19", 0).         hasThresholdType("setup19", "Gratitude").
hasThreshold("setup20", 0).         hasThresholdType("setup20", "Gratification").
hasThreshold("setup21", 0).         hasThresholdType("setup21", "Anger").
hasThreshold("setup22", 0).         hasThresholdType("setup22", "Remorse").
```

Arquivo *AgentSpeak* - emotionsWatch2.asl

```
agent("watch2").
//------------------------------------------------------------------------------
high("lhigh").                      mid("lmid").
low("llow").                        none("lnone").
//------------------------------------------------------------------------------
hasSetup("watch2", "setup1").       hasSetup("watch2", "setup2").
hasSetup("watch2", "setup3").       hasSetup("watch2", "setup4").
hasSetup("watch2", "setup5").       hasSetup("watch2", "setup6").
hasSetup("watch2", "setup7").       hasSetup("watch2", "setup8").
hasSetup("watch2", "setup9").       hasSetup("watch2", "setup10").
hasSetup("watch2", "setup11").      hasSetup("watch2", "setup12").
hasSetup("watch2", "setup13").      hasSetup("watch2", "setup14").
hasSetup("watch2", "setup15").      hasSetup("watch2", "setup16").
hasSetup("watch2", "setup17").      hasSetup("watch2", "setup18").
hasSetup("watch2", "setup19").      hasSetup("watch2", "setup20").
hasSetup("watch2", "setup21").      hasSetup("watch2", "setup22").
//------------------------------------------------------------------------------
hasThreshold("setup1", 0).          hasThresholdType("setup1", "Joy").
hasThreshold("setup2", 0).          hasThresholdType("setup2", "Distress").
hasThreshold("setup3", 0).          hasThresholdType("setup3", "Hope").
hasThreshold("setup4", 0).          hasThresholdType("setup4", "Fear").
hasThreshold("setup5", 0).          hasThresholdType("setup5", "FearsConfirmed").
hasThreshold("setup6", 0).          hasThresholdType("setup6", "Satisfaction").
hasThreshold("setup7", 0).          hasThresholdType("setup7", "Relief").
hasThreshold("setup8", 0).          hasThresholdType("setup8", "Disappointment").
hasThreshold("setup9", 0).          hasThresholdType("setup9", "HappyFor").
hasThreshold("setup10", 0).         hasThresholdType("setup10", "SorryFor").
hasThreshold("setup11", 0).         hasThresholdType("setup11", "Gloating").
hasThreshold("setup12", 0).         hasThresholdType("setup12", "Resentment").
hasThreshold("setup13", 0).         hasThresholdType("setup13", "Love").
hasThreshold("setup14", 0).         hasThresholdType("setup14", "Hate").
hasThreshold("setup15", 0).         hasThresholdType("setup15", "Admiration").
hasThreshold("setup16", 0).         hasThresholdType("setup16", "Pride").
hasThreshold("setup17", 0).         hasThresholdType("setup17", "Shame").
hasThreshold("setup18", 0).         hasThresholdType("setup18", "Reproach").
hasThreshold("setup19", 0).         hasThresholdType("setup19", "Gratitude").
hasThreshold("setup20", 0).         hasThresholdType("setup20", "Gratification").
hasThreshold("setup21", 0).         hasThresholdType("setup21", "Anger").
hasThreshold("setup22", 0).         hasThresholdType("setup22", "Remorse").
```

Arquivo *AgentSpeak* - common.asl

```
!start.
//------------------------------------------------------------------------------
+!start <- !buildTestName("", NAME); +team(NAME);
           .send(stadium, tell, name(NAME));
           !!deliberation.
//------------------------------------------------------------------------------
+!deliberation <- ?appraisal; ok; !!deliberation.
//------------------------------------------------------------------------------
+!buildTestName("", NAME) <- !buildName(N); !buildTestName(N, NAME).
+!buildTestName(NAME, NAME).
```

Arquivo *AgentSpeak* - appraisal.asl

```
fib(3, 2). fib(4, 3). fib(5, 5). fib(6, 8). fib(7, 13). fib(_, 1).
myPoints(N,VAL,M,V) :- team(N) & (points(N,VAL,M,V) | points(M,V,N,VAL)).
myName(NAME) :- .my_name(ATOM) & .term2string(ATOM, NAME).
//------------------------------------------------------------------------------
+?appraisal
    <- ?background;
       ?appraisalGoal;
       ?appraisalTeam;
       ?appraisalEndMatch;
       ?appraisalEndMatchHappy;
       ?printEmotions.
//-- people will know your team with a niceness positive -----------------------
+?background :  not(myPoints(_,_,_,_)).
+?background
    :  myPoints(MYTEAM,_, ENEMYTEAM, _) & not(hasNiceness(_,_))
    <- ?myName(NAME); INDIVIDUALMT="myteam_background";
       +hasAppraisal(NAME, INDIVIDUALMT);
       +hasPerson(INDIVIDUALMT, MYTEAM);
       +hasNiceness(INDIVIDUALMT, 10);
       INDIVIDUALET = "enemyteam_background";
       +hasAppraisal(NAME, INDIVIDUALET);
       +hasPerson(INDIVIDUALET, ENEMYTEAM);
       +hasNiceness(INDIVIDUALET, -10).
+?background.
//-- people will be joy when your team does a point ----------------------------
+?appraisalGoal
    :  goal(TEAM) & myPoints(TEAM, PF, _, _)
    <- ?fib(8-(PF+1), VAL);
       ?updateEmotion(prospectIrrelevant, "goal_myteam_pi", VAL);
       +removeEmotion("goal_myteam_pi").
+?appraisalGoal
    :  goal(TEAM) & myPoints(_, _, TEAM, PE)
    <- ?fib(PE+1, VAL);
       ?updateEmotion(prospectIrrelevant, "goal_enemyteam_pi", -VAL);
       +removeEmotion("goal_enemyteam_pi").
+?appraisalGoal
    : removeEmotion(INDIVIDIDUAL)
   <- ?removeEmotion(prospectIrrelevant, INDIVIDIDUAL);
      -removeEmotion(INDIVIDIDUAL).
+?appraisalGoal.
//-- All team have hope that they will win the match ---------------------------
+?appraisalTeam
    :  not(hasLikelihood("team_hope_prn", PROB)) & step(STEP) & STEP < 10
    <- ?updateEmotion(prospectNotRealized, "team_hope_prn", 50);
       +realizedCalculation(1).
+?appraisalTeam
    :  goal(TEAM) & myPoints(_,_, TEAM, _) // ponto que nao eh meu
    &  realizedCalculation(CALC)
    <- -realizedCalculation(CALC);
       +realizedCalculation(CALC-10).
+?appraisalTeam
    :  goal(TEAM) & myPoints(TEAM, _, _,_) // ponto que eh meu
    &  realizedCalculation(CALC)
    <- -realizedCalculation(CALC);
       +realizedCalculation(CALC+10).
+?appraisalTeam.
//-- In the end of the match, the satisfaction or disappointment appear --------
+?appraisalEndMatch
```

```
    :  realizedCalculation(CALC) & match(_,_)
    &  hasLikelihood("team_hope_prn", PROB)
    &  feeling(hope, HOPEVAL)
    <- ?removeEmotion(prospectNotRealized, "team_hope_prn");
       ?updateEmotion(prospectRealized, "team_satisfaction_pr", HOPEVAL, CALC);
       -realizedCalculation(CALC).
+?appraisalEndMatch
    :  realizedCalculation(CALC) & match(_,_)
    &  hasLikelihood("team_hope_prn", PROB)
    <- ?removeEmotion(prospectNotRealized, "team_hope_prn");
       -realizedCalculation(CALC).
+?appraisalEndMatch.
//-- In the end of the match, we can felt happy because my team win ------------
+?appraisalEndMatchHappy
     : match(win, TEAM) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10; DT=20;
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).
+?appraisalEndMatchHappy
     : match(lose, TEAM) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10;
       DT=-20; // is negative to try implicate the sorryFor
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).
+?appraisalEndMatchHappy
     : match(draw, _) & myPoints(TEAM, _, _, _)
     & not(hasAppraisal(NAME, "team_happy_foo"))
    <- .random(N); DE=math.round(N*20)-10; DT=10;
       ?updateEmotion(fortunesOfOthers, "team_happy_foo", DE, DT).
+?appraisalEndMatchHappy.
//-- update JOY or DISTRESS ----------------------------------------------------
+?updateEmotion(prospectIrrelevant, INDIVIDUAL, VAL)
     : not(hasDesireSelf(INDIVIDUAL, ANY))
    <- ?myName(NAME);
       +hasAppraisal(NAME, INDIVIDUAL);
       +hasLikelihood(INDIVIDUAL, "lnone");
       +hasDesireSelf(INDIVIDUAL, VAL).
+?updateEmotion(prospectIrrelevant, INDIVIDUAL, VAL)
     : hasDesireSelf(INDIVIDUAL, OLDVAL)
    <- -hasDesireSelf(INDIVIDUAL, OLDVAL);
       +hasDesireSelf(INDIVIDUAL, OLDVAL+VAL).
+?removeEmotion(prospectIrrelevant, INDIVIDUAL)
     : hasDesireSelf(INDIVIDUAL, OLDVAL)
    <- ?myName(NAME);
       -hasLikelihood(INDIVIDUAL, "lnone");
       -hasDesireSelf(INDIVIDUAL, OLDVAL);
       -hasAppraisal(NAME, INDIVIDUAL).
+?removeEmotion(prospectIrrelevant, INDIVIDUAL).
//-- update HOPE or FEAR -------------------------------------------------------
+?updateEmotion(prospectNotRealized, INDIVIDUAL, VAL)
     : not(hasDesireSelf(INDIVIDUAL, ANY))
    <- ?myName(NAME);
       +hasAppraisal(NAME, INDIVIDUAL);
       +hasLikelihood(INDIVIDUAL, "lmid");
       +hasDesireSelf(INDIVIDUAL, VAL).
+?updateEmotion(prospectNotRealized, INDIVIDUAL, VAL)
     : hasDesireSelf(INDIVIDUAL, OLDVAL)
    <- -hasDesireSelf(INDIVIDUAL, OLDVAL);
       +hasDesireSelf(INDIVIDUAL, OLDVAL+VAL).
+?removeEmotion(prospectNotRealized, INDIVIDUAL)
     : hasDesireSelf(INDIVIDUAL, OLDVAL) & hasLikelihood(INDIVIDUAL, PROBABILITY)
    <- ?myName(NAME);
       -hasLikelihood(INDIVIDUAL, PROBABILITY);
       -hasDesireSelf(INDIVIDUAL, OLDVAL);
       -hasAppraisal(NAME, INDIVIDUAL).
+?removeEmotion(prospectNotRealized, INDIVIDUAL).
//-- update SATISFACTION or DISAPPOINTMENT or RELIEF or FEARCONFIRMED ----------
+?updateEmotion(prospectRealized, INDIVIDUAL, PREVFEEL, VAL)
     : not(hasEffort(INDIVIDUAL, ANY))
    <- ?myName(NAME);
       +hasAppraisal(NAME, INDIVIDUAL);
       +hasEffort(INDIVIDUAL, PREVFEEL); // yes, this is correct
       +hasPreviousIntensity(INDIVIDUAL, PREVFEEL);
```

```
       +hasRealized(INDIVIDUAL, VAL).
//-- dont have the update of a prospect realized emotion -----------------------
+?updateEmotion(prospectRealized, INDIVIDUAL, PREVFEEL, VAL).
+?removeEmotion(prospectRealized, INDIVIDUAL)
    :  hasEffort(INDIVIDUAL, ANY) & hasRealized(INDIVIDUAL, ANOTHER)
    <- ?myName(NAME);
       -hasEffort(INDIVIDUAL, ANY);
       -hasPreviousIntensity(INDIVIDUAL, ANY);
       -hasRealized(INDIVIDUAL, ANOTHER);
       -hasAppraisal(NAME, INDIVIDUAL).
+?removeEmotion(prospectRealized, INDIVIDUAL).
//-- update HAPPYFOR or SORRYFOR or GLOATING or RESENTMENT ---------------------
+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DESERVED, DESIREOTHER)
     : team(TEAM) & not(hasPerson(INDIVIDUAL, TEAM))
    <- ?myName(NAME);
       +hasAppraisal(NAME, INDIVIDUAL);
       +hasPerson(INDIVIDUAL, TEAM);
       +hasDeserved(INDIVIDUAL, DESERVED);
       +hasDesireOther(INDIVIDUAL, DESIREOTHER).
+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DESERVED, DESIREOTHER)
     : hasDeserved(INDIVIDUAL, OLDDE) & hasDesireOther(INDIVIDUAL, OLDDO)
    <- -hasDeserved(INDIVIDUAL, OLDDE);
       +hasDeserved(INDIVIDUAL, DESERVED);
       -hasDesireOther(INDIVIDUAL, OLDDO);
       +hasDesireOther(INDIVIDUAL, DESIREOTHER).
+?updateEmotion(fortunesOfOthers, INDIVIDUAL, DE, DO).
+?removeEmotion(fortunesOfOthers, INDIVIDUAL)
     : hasDeserved(INDIVIDUAL, OLDDE) & hasDesireOther(INDIVIDUAL, OLDDO)
     & hasPerson(INDIVIDUAL, TEAM)
    <- ?myName(NAME);
       -hasPerson(INDIVIDUAL, TEAM);
       -hasDeserved(INDIVIDUAL, OLDDE);
       -hasDesireOther(INDIVIDUAL, OLDDO);
       -hasAppraisal(NAME, INDIVIDUAL).
+?removeEmotion(fortunesOfOthers, INDIVIDUAL).
+?printEmotions
    <- .findall(feel(L,V), feeling(L,V), FS);
       ?myPoints(N,P,M,O); ?team(T);
       .print("Emotion:␣", FS, "team␣", T, "==",N, "␣", P, "␣x␣", O, "␣", M).
-?X <- .print("Error␣",X).
-!X <- .print("Error␣",X).
```

APÊNDICE D FONTES DA APLICAÇÃO DE CASA VIRTUAL

Arquivo de Projeto - sims.mas2j

```
MAS sims {
    infrastructure: Centralised
    environment: maro.example.sims.House(30000, 1000, 4, FALSE, "onto/sims.owl")
    agents:
        millie basicAgent3.asl #1
            beliefBaseClass maro.wrapper.BBAffective("onto/millie.owl")
            agentClass maro.wrapper.UniqueAnnotation ;
        nina basicAgent3.asl #1
            beliefBaseClass maro.wrapper.BBAffective("onto/nina.owl")
            agentClass maro.wrapper.UniqueAnnotation ;
        john basicAgent3.asl #1
            beliefBaseClass maro.wrapper.BBAffective("onto/john.owl")
            agentClass maro.wrapper.UniqueAnnotation ;
        albert basicAgent3.asl #1
            beliefBaseClass maro.wrapper.BBAffective("onto/albert.owl")
            agentClass maro.wrapper.UniqueAnnotation ;
    aslSourcePath: "asl";
}
```

Arquivo *Java* - House.java

```
package maro.example.sims;
import maro.core.AnnotatedEnvironment;
public class House extends AnnotatedEnvironment {
    protected HouseModel model = null;
    protected HouseView view = null;
    protected HouseController controller = null;
    @Override
    public void init(String[] args) {
        initOptions(args);
                super.initOntology();
        model = new HouseModel(20, 20, getNumberAgentsSettings());
        view = new HouseView(model, "House", 400);
        controller = new HouseController(this, oapi);
                oapi = null;
        initDefault();
    }
        @Override
        protected void stepStarted(int step) {
                if (step > 0) updateNumberOfAgents();
                if (getNbAgs() == 0) {
                        try {
                                getLogger().info("All␣agents␣died...␣exiting");
                                getEnvironmentInfraTier().getRuntimeServices().stopMAS()
                                    ;
                        } catch (Exception e) { }
                }
                getLogger().info("Started␣step␣" + step + "(day="+controller.day+";hour=
                    "+
                                controller.hour+";minute="+controller.mins+";shift="+
                                    controller.shift
```

```
                                +")");
                controller.newStep(step, this);
        }
        public HouseModel getModel() { return model; }
}
```

Arquivo *Java* - HouseController.java

```
package maro.example.sims;
import jason.environment.grid.GridWorldModel;
import maro.wrapper.OwlApi;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import java.util.ArrayList;
public class HouseController {
    final protected House parent;
    final protected ArrayList<CharacterInspectorController> characters;
    private int factorTime = 870;
    protected int step;
        protected int day;
        protected int hour;
        protected int lsth;
        protected int mins;
        protected int secs;
        protected String shift;
        static protected String[] shiftDay = {
                        "Dawn", "Noon", "Afternoon", "Night"
                };
        protected String today;
        static protected String[] weekDays = {
                        "Monday", "Tuesday", "Wednesday", "Thursday",
                        "Friday", "Saturday", "Sunday"
                };
    public int getStep() { return this.step; }
    public int getSimulationTime() { return this.step * factorTime; }
    public GridWorldModel getModel() { return parent.model; }
    public HouseController(House owner, OwlApi oapi) {
        parent = owner;
        characters = new ArrayList<CharacterInspectorController> ();
        step = -1; lsth = -1;
        parent.model.setView( parent.view );
        parent.model.loadPlacesFromOntology(oapi); // It doesn't have annottion
        parent.model.loadAgentsFromOntology(oapi); // It does have annotation
        parent.model.loadObjectsFromOntology(oapi); // It does have annotation
        parent.view.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        parent.view.addWindowListener( new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    parent.tryStop(0);
                }
            });
        parent.view.setVisible(true);
        parent.view.getCanvas().setFocusable(true);
                parent.view.getCanvas().addKeyListener(new KeyAdapter() {
                                public void keyReleased(KeyEvent e) {
                                        if (e.getID() == KeyEvent.KEY_RELEASED) {
                                handler_keyboard(e.getKeyCode());
                            }
                                }
                            });
        parent.view.getCanvas().requestFocusInWindow();
        parent.view.repaint();
    }
    protected void handler_keyboard(int keyCode) {
        CharacterInspectorController ch;
        String agName = null;
        int agId;
        if (keyCode < 48 || keyCode > 57) return ;
        agId = (keyCode - 49) % 10;
        agName = parent.model.getAgentName(agId);
        if (agName == null) return ;
```

```
    ch = null;
    for (CharacterInspectorController cic : characters) {
        HouseModel.Agent a = cic.getAgent();
        if (a.getName().equals(agName)) {
            ch = cic;
            break;
        }
    }
    if (ch == null) return ;
    ch.repaint();
    ch.setVisible(true);
}
public void newStep(int step, House h) {
    this.step = step;
            secs = getSimulationTime();
            // secs minus days
            day  = (secs / 86400);
            secs = secs % 86400;
            // secs minus hours
            hour = (secs / 3600);
            secs = (secs % 3600);
            // secs minus minutes
            mins = (secs / 60);
            secs = (secs % 60);
            updateDay();
            if (step == 1) {
                    for (HouseModel.Agent a : parent.model.referAgent.values()) {
                            CharacterInspectorController ch
                                    = new CharacterInspectorController(a.getID(), a.
                                        getName());
                            ch.setParent(this);
                            characters.add(ch);
                    }
            }
            if (lsth != hour) {
                    lsth = hour;
                    for (HouseModel.Place p : parent.model.referPlace.values())
            p.update(day % 7, hour, step);
            }
    updateInspector();
            updatePercepts(h);
}
protected void updateDay() {
            today = weekDays[day % 7];
            if (hour < 5) shift = shiftDay[0]; // (M) adrugada
            else if (hour < 12) {
                    if (hour == 5 && mins <= 30)
                            shift = shiftDay[0];
                    else
                            shift = shiftDay[1]; // M (a) nha
            } else if (hour < 18) {
                    if (hour == 12 && mins == 0)
                            shift = shiftDay[1];
                    else
                            shift = shiftDay[2]; // (T) arde
            } else { // 18 - 23h59
                    if (hour == 18 && mins == 0)
                            shift = shiftDay[2];
                    else
                            shift = shiftDay[3]; // (N) oite
            }
    }
public void updateInspector() {
    for (CharacterInspectorController cic : characters) cic.update();
}
    public void updatePercepts(House h) {
            for (HouseModel.Agent a : parent.model.referAgent.values()) {
                    if (a == null) continue;
                    a.updatePerception(h);
            }
    }
public int convertToStep(int hour) {
    int offset = hour * 3600;
```

```
        int time = day * 86400 + offset;
        int fix = time % factorTime;
        fix = (fix == 0) ? 0 : factorTime - fix;
        return 1 + ((time + fix) / factorTime);
    }
}
```

Arquivo *Java* - HouseModel.java

```
package maro.example.sims;
import jason.asSyntax.Term;
import jason.asSyntax.ListTerm;
import jason.asSyntax.Literal;
import jason.asSyntax.ASSyntax;
import jason.environment.grid.GridWorldModel;
import jason.environment.grid.Location;
import maro.wrapper.OwlApi;
import maro.wrapper.Dumper;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
public class HouseModel extends GridWorldModel {
        public static final int CLOSED_DOOR = 1 <<  3;
        public static final int OPENED_DOOR = 1 <<  4;
        public static final int TABLE       = 1 <<  5;
        public static final int FAUCET      = 1 <<  6;
        public static final int SHOWER      = 1 <<  7;
        public static final int BED         = 1 <<  8;
        public static final int TOILET      = 1 <<  9; // can be a toilet or a gas stove
             to cook
        public static final int BOOKCASE    = 1 << 10; // Maybe change to wardrobe?
        public static final int SOFA        = 1 << 11;
        public static final int TV          = 1 << 12;
        public static final int REFRIGERATOR= 1 << 13;
    public static final int ALL         = (1 << 14) - 1;
    public HouseModel(int width, int height, int numberOfAgents) {
        super(width, height, numberOfAgents);
                referPlace = new HashMap<String, Place>(); // placeName X Place
                referAgent = new HashMap<Integer, Agent>(); // integerID X Agent
    }
        private String getString(Set<Dumper> names, String base, boolean unquote) {
                if (names == null) return null;
                for (Dumper name: names) {
                        String source = name.getTerms()[0];
                        if (source.equals(base) == false) continue;
                        if (unquote == true) return name.getTermAsString(1);
                        return name.getTerms()[1];
                }
                return null;
        }
        private String getStringUnquoted(Set<Dumper> names, String base) {
                return getString(names, base, true);
        }
        private String getStringQuoted(Set<Dumper> names, String base) {
                return getString(names, base, false);
        }
        private Integer getInteger(Set<Dumper> ids, String base) {
                if (ids == null) return null;
                for (Dumper id : ids) {
                        String source = id.getTerms()[0];
                        if (source.equals(base) == false) continue;
                        return id.getTermAsInteger(1);
                }
                return null;
        }
        private Dumper getDumper(Set<Dumper> ids, String base) {
                if (ids == null) return null;
                for (Dumper id : ids) {
                        String source = id.getTerms()[0];
                        if (source.equals(base) == false) continue;
                        return id;
```

```
        }
        return null;
}
public void loadPlacesFromOntology(OwlApi oapi) {
        Set<Dumper> places
                = oapi.getCandidatesByFunctorAndArity(1, "place"); // place(
                    placeName)
        Set<Dumper> ids
                = oapi.getCandidatesByFunctorAndArity(2, "hasIdentifier"); //
                    hasIdentifier(placeName, id)
        Set<Dumper> names
                = oapi.getCandidatesByFunctorAndArity(2, "hasName"); //hasName(
                    placeName, name)
        Set<Dumper> capacities
                = oapi.getCandidatesByFunctorAndArity(2, "hasCapacity");
        Set<Dumper> schedules
                = oapi.getCandidatesByFunctorAndArity(2, "
                    hasScheduleOfFunctioning");
        Set<Dumper> averageTime // hasAverageTimeOfPermanence(schedule, string)
                = oapi.getCandidatesByFunctorAndArity(2, "
                    hasAverageTimeOfPermanence");
        Set<Dumper> closingTime
                = oapi.getCandidatesByFunctorAndArity(2, "hasClosingTime");
        Set<Dumper> openingTime
                = oapi.getCandidatesByFunctorAndArity(2, "hasOpeningTime");
        Set<Dumper> enteringTime
                = oapi.getCandidatesByFunctorAndArity(2, "
                    hasEnteringTimeInterval");
        Set<Dumper> dimensions
                = oapi.getCandidatesByFunctorAndArity(2, "hasDimension"); //
                    hasDimension(placeName, dim)
        Set<Dumper> positionsX
                = oapi.getCandidatesByFunctorAndArity(2, "hasPositionX"); //
                    hasPositionX(dim, intVal)
        Set<Dumper> positionsY
                = oapi.getCandidatesByFunctorAndArity(2, "hasPositionY");
        Set<Dumper> sizesX
                = oapi.getCandidatesByFunctorAndArity(2, "hasSizeX");
        Set<Dumper> sizesY
                = oapi.getCandidatesByFunctorAndArity(2, "hasSizeY");
        if (places == null) return ;
        for (Dumper place: places) {
                Place placeData = new Place();
                String base;
                base = place.getTerms()[0];
                placeData.setName(getStringUnquoted(names, base));
                placeData.setType(getInteger(ids, base));
                placeData.setCapacity(getInteger(capacities, base));
                Dumper dimensionRelevant = getDumper(dimensions, base);
                if (dimensionRelevant != null) {
                                String target = dimensionRelevant.getTerms()[1];
                                placeData.setPX(getInteger(positionsX, target));
                                placeData.setPY(getInteger(positionsY, target));
                                placeData.setWidth(getInteger(sizesX, target));
                                placeData.setHeight(getInteger(sizesY, target));
                }
                Dumper scheduleRelevant = getDumper(schedules, base);
                if (scheduleRelevant != null) {
                        String target = scheduleRelevant.getTerms()[1];
                        placeData.setAverageTime(getStringUnquoted(averageTime,
                            target));
                        placeData.setTimeClosing(getStringUnquoted(closingTime,
                            target));
                        placeData.setTimeOpening(getStringUnquoted(openingTime,
                            target));
                        placeData.setArriveInterval(getStringUnquoted(
                            enteringTime, target));
                }
                if (placeData.getName() == null) continue;
                referPlace.put(placeData.getName(), placeData);
                if (placeData.haveDimension() && placeData.getName().startsWith(
                    "initial") == false) {
                        int minX = placeData.getPX();
```

```
                        int minY = placeData.getPY();
                        int maxX = minX + placeData.getWidth();
                        int maxY = minY + placeData.getHeight();
                        int type = placeData.getType();
                        for (int x = minX; x < maxX; x++) {
                                for (int y = minY; y < maxY; y++) data[x][y] =
                                    type;
                        }
                }
        }
}
public void loadAgentsFromOntology(OwlApi oapi) {
        Set<Dumper> agents
                = oapi.getCandidatesByFunctorAndArity(1, "agent");
        Set<Dumper> characters
                = oapi.getCandidatesByFunctorAndArity(2, "hasCharacter");
        Set<Dumper> profiles
                = oapi.getCandidatesByFunctorAndArity(2, "hasProfile");
        Set<Dumper> names
                = oapi.getCandidatesByFunctorAndArity(2, "hasName");
        Set<Dumper> fixedDestinations
                = oapi.getCandidatesByFunctorAndArity(2, "hasFixedDestination");
        Set<Dumper> randomDestinations
                = oapi.getCandidatesByFunctorAndArity(2, "hasRandomDestination")
                    ;
        int curr = 0;
        if (agents == null) {
                // no agents....
                if (getNumberOfAgents() > 0) {
                        String waitAgents = agents.toString();
                }
                return ;
        }
        for (Dumper agent : agents) {
                Agent agentData;
                Profile profileData;
                String character;
                String profile;
                String base;
                base = agent.getTerms()[0];
                character = getStringQuoted(characters, base);
                if (character == null) continue;
                profile = getStringQuoted(profiles, character);
                if (profile == null) continue;
                profileData = new Profile(profile);
                agentData = new Agent(curr, agent.getTermAsString(0));
                referAgent.put(curr, agentData);
                for (Dumper d : fixedDestinations) {
                        if (profile.equals(d.getTerms()[0]) == false) continue;
                        String placeNameIndividual = d.getTerms()[1];
                        String placeName = getStringUnquoted(names,
                            placeNameIndividual);
                        Place place = referPlace.get(placeName);
                        if (place == null) continue;
                        String name = place.getName();
                        if (name.startsWith("initial")) {
                                setAgPos(agentData.getID(), place.getPX(), place
                                    .getPY());
                        } else {
                                profileData.addFixedDestination(name);
                        }
                }
                for (Dumper d : randomDestinations) {
                        if (profile.equals(d.getTerms()[0]) == false) {
                                continue;
                        }
                        String placeNameIndividual = d.getTerms()[1];
                        String placeName = getStringUnquoted(names,
                            placeNameIndividual);
                        Place place = referPlace.get(placeName);
                        if (place == null) continue;
                        String name = place.getName();
                        profileData.addRandomDestination(name);
```

```
                }
                agentData.randomOrientation();
                agentData.setProfile(profileData);
                curr += 1;
        }
}
public void loadObjectsFromOntology(OwlApi oapi) {
        Set<Dumper> setups    // hasSetup(placeName1, placeName2) and others
                = oapi.getCandidatesByFunctorAndArity(2, "hasSetup");
        Set<Dumper> names
                = oapi.getCandidatesByFunctorAndArity(2, "hasName");
        Set<Dumper> values // placeName2 to string
                = oapi.getCandidatesByFunctorAndArity(2, "hasAnnotationValue");
        Set<Dumper> annots // placeName2 to annotation
                = oapi.getCandidatesByFunctorAndArity(2, "hasAnnotation");
        itemView = new HashMap<Place, ArrayList<Place> > ();
        placeView = new HashMap<Place, ArrayList<Place> > ();
        for (Dumper dumper : setups) {
                String source = dumper.getTerms()[0];
                String target = dumper.getTerms()[1];
                String sname = getStringUnquoted(names, source);
                String tname = getStringUnquoted(names, target);
                Place ps = referPlace.get(sname);
                Place pt = referPlace.get(tname);
                if (ps == null) continue;
                if (pt != null) {
                        ArrayList<Place> rooms, items;
                        items = placeView.get(ps);
                        if (items == null) {
                                items = new ArrayList<Place> ();
                                placeView.put(ps, items);
                        }
                        rooms = itemView.get(pt);
                        if (rooms == null) {
                                rooms = new ArrayList<Place> ();
                                itemView.put(pt, rooms);
                        }
                        items.add(pt);
                        rooms.add(ps);
                } else {
                        String aname = getStringUnquoted(annots, target);
                        String vname = getStringUnquoted(values, target);
                        if (aname == null || vname == null) continue;
                        ps.setAnnot(aname, vname);
                }
        }
        boolean excluded = true;
        while (excluded == true) {
                excluded = false;
                for (String key : referPlace.keySet()) {
                        if (key.startsWith("initial")) {
                                referPlace.remove(key);
                                excluded = true;
                                break;
                        }
                }
        }
}
public int nextInt(int limit) { return random.nextInt(limit); }
public int nextNormal(int average, int range) {
        double offset = average;
        double r = range;
        return (int) (offset + random.nextGaussian() * r);
}
public String getAgentName(int agentID) {
        Agent a = referAgent.get(agentID);
        if (a == null) return null;
        return new String (a.getName());
}
protected Agent getAgentFromName(String ag) {
        if (ag == null) return null;
        for (Agent a : referAgent.values()) {
                if (a != null && a.getName().equals(ag)) return a;
```

```
        }
        return null;
}
public int getNumberOfAgents() { return agPos.length; }
public boolean changeOrientation(String ag, char newOrientation) {
        Agent a = getAgentFromName(ag);
        if (a == null) return false;
        Location l = getAgPos(a.getID());
        synchronized (data) {
                a.setOrientation(newOrientation);
        }
if (view != null && l != null) view.update(l.x,l.y);
        a.addHungry(-1);
        return true;
}
protected boolean positionByOrientation(Location l, Character c, boolean flag) {
        switch (c) {
                case 'N':
                        if (l.y > 0) l.y -= 1;
                        else if (flag && l.y == 0) return false;
                        break;
                case 'E':
                        if (l.x < 19) l.x += 1;
                        else if (flag && l.x == 19) return false;
                        break;
                case 'S':
                        if (l.y < 19) l.y += 1;
                        else if (flag && l.y == 19) return false;
                        break;
                case 'W':
                        if (l.x > 0) l.x -= 1;
                        else if (flag && l.x == 0) return false;
                        break;
                default:
                        return false;
        }
        return true;
}
protected boolean forward2(Location l, Character c, Agent a) {
        if ( a == null || a.getID() == null || agPos[a.getID()] == null) {
                return false;
        }
        if ( positionByOrientation(l, c, false) == false ) return false;
boolean ret = true;
        int e = nextNormal(4,2);
        synchronized (data) {
                if (data[l.x][l.y] != 0) {
        ret = false;
    } else {
        setAgPos(a.getID(), l);
    }
        }
        if (ret == false) return false;
        if (e > 8) e = 8;
        if (e < 0) e = 1;
        if (e > 0) e = -e;
        a.addEnergy(e);
        a.addHungry(-2);
        a.addCleaning(-1);
        return true;
}
public boolean forward(String ag) {
        Agent a = getAgentFromName(ag);
        if (a == null) return false;
        Location l = getAgPos(a.getID());
        Character c = a.getOrientation();
        if (l == null || c == null) return false;
        return forward2(l, c, a);
}
public boolean nope(String ag) {
        Agent a = getAgentFromName(ag);
        if (a == null) return false;
        a.addHungry(-1);
```

```
        a.addEnergy(-1);
        return true;
}
public boolean hideMe(String ag) { // death
        Agent a = getAgentFromName(ag);
        if (a == null) return false;
        referAgent.put(a.getID(), null);
        Location l = agPos[a.getID()];
        remove(AGENT, l.x, l.y);
        agPos[a.getID()] = new Location(-1, -1);
        return true;
}
public boolean tryUseObject(String ag) {
        Agent a = getAgentFromName(ag);
        if (a == null) return false;
        Location l = getAgPos(a.getID());
        Character c = a.getOrientation();
        if (l == null || c == null) return false;
        if ( positionByOrientation(l, c, true) == false ) return false;
        int d = data[l.x][l.y];
        if ((d & ALL) == 0) return false;
        if ((d & CLOSED_DOOR) == CLOSED_DOOR) {
                return false;
        } else if ((d & OPENED_DOOR) == OPENED_DOOR) {
                return forward2(l, c, a);
        } else if ((d & BED) == BED) {
                a.addEnergy(3);
                a.addHungry(1);
                return true;
        }
        return false;
}
public String [] planRoute(String ag, String place) {
        Agent a = getAgentFromName(ag);
        if (a == null) return null;
        Place p = referPlace.get(place);
        if (p == null) return null;
        Location l = getAgPos(a.getID());
        if (l == null) return null;
        AStar astar = new AStar();
        return astar.astar(l.x, l.y, p.getPX(), p.getPY(),
                                p.getWidth(), p.getHeight(), data);
}
protected HashMap<Place, ArrayList<Place> > placeView;
public String[] getPlacesFromPlaceView() {
        ArrayList<String> als = new ArrayList<String>();
        if (placeView == null) return als.toArray(new String [0]);
        for (Place p : placeView.keySet()) als.add(p.getName());
        return als.toArray(new String[0]);
}
public String[] getItemsAtPlace(String name) {
        ArrayList<String> als = new ArrayList<String>();
        if (placeView == null || name == null) return als.toArray(new String
            [0]);
        Place base = referPlace.get(name);
        if (base == null) return als.toArray(new String [0]);
ArrayList<Place> alp = placeView.get(base);
        if (alp == null) return als.toArray(new String [0]);
        for (Place p : placeView.get(base)) als.add(p.getName());
        return als.toArray(new String[0]);
}
protected HashMap<Place, ArrayList<Place> > itemView;
public String[] getItemsFromItemView() {
        ArrayList<String> als = new ArrayList<String>();
        if (itemView == null) return als.toArray(new String [0]);
        for (Place p : itemView.keySet()) als.add(p.getName());
        return als.toArray(new String[0]);
}
public String[] getPlacesByItem(String name) {
        ArrayList<String> als = new ArrayList<String>();
        if (itemView == null || name == null) return als.toArray(new String [0])
            ;
        Place base = referPlace.get(name);
```

```
        if (base == null) return als.toArray(new String [0]);
ArrayList<Place> alp = itemView.get(base);
        if (alp == null) return als.toArray(new String [0]);
        for (Place p : itemView.get(base)) als.add(p.getName());
        return als.toArray(new String[0]);
}
protected HashMap<String, Place> referPlace;
protected class Place {
        private String placeName;
        private Integer px; // can be out of screen or null
        private Integer py; // can be out of screen or null
        private Integer width;
        private Integer height;
        private Integer type;
        private Integer totalCapacity;
        private Integer[] timeOpening;
        private Integer[] timeClosing;
        private Integer[] averageTime;
        private Integer[] arriveInterval;
        private HashMap<String,String> annots; // static annotations
        private ArrayList<Integer> capacity;
        private boolean opened = false;
        public Place () {
                annots = new HashMap<String,String>();
                debugPlace = false;
                if (System.getenv("debugPlace") != null) {
                        debugPlace = true;
                }
        }
        public boolean isOpen() { return opened; }
        public void setName(String i) { placeName = i;}
        public String getName() { return placeName; }
        public void setPX(Integer i) { px = i; }
        public Integer getPX() { return px; }
        public void setPY(Integer i) { py = i; }
        public Integer getPY() { return py; }
        public void setWidth(Integer i) { width = i; }
        public Integer getWidth() { return width; }
        public void setHeight(Integer i) { height = i; }
        public Integer getHeight() { return height; }
        public void setType(Integer i) { type = i; }
        public Integer getType() { return type; }
        public void setCapacity(Integer i) { totalCapacity = i; }
        public Integer getCapacity() { return totalCapacity; }
        public void setAnnot(String key, String value) {
                String checkAppend = getAnnot(key);
                if (checkAppend != null) checkAppend += "," + value;
                else checkAppend = value;
                annots.put(key, checkAppend);
        }
        public String getAnnot(String key) { return annots.get(key); }
        public void setTimeOpening(String line) {
                String [] days = line.split(",");
                timeOpening = new Integer[7];
                if (days[0].equals("-")) {
                        for (int i=0; i<7; i++) timeOpening[i] = -1;
                        return ;
                }
                for (int i=0; i<7 && i < days.length; i++) {
                        try {
                                timeOpening[i] = Integer.parseInt(days[i]);
                        } catch (Exception e) {
                                timeOpening[i] = -2;
                        }
                }
        }
        public void setTimeClosing(String line) {
                String [] days = line.split(",");
                timeClosing = new Integer[7];
                if (days[0].equals("-")) {
                        for (int i=0; i<7; i++) timeClosing[i] = -1;
                        return ;
                }
```

```
        for (int i=0; i<7 && i < days.length; i++) {
                try {
                        timeClosing[i] = Integer.parseInt(days[i]);
                } catch (Exception e) {
                        timeClosing[i] = -2;
                }
        }
}
public void setAverageTime(String line) {
        String [] days = line.split(",");
        averageTime = new Integer[7];
        if (days[0].equals("-")) {
                for (int i=0; i<7; i++) averageTime[i] = -1;
                return ;
        }
        for (int i=0; i<7 && i < days.length; i++) {
                try {
                        averageTime[i] = Integer.parseInt(days[i]);
                } catch (Exception e) {
                        averageTime[i] = -2;
                }
        }
}
public void setArriveInterval(String line) {
        String [] days = line.split(",");
        arriveInterval = new Integer[7];
        if (days[0].equals("-")) {
                for (int i=0; i<7; i++) arriveInterval[i] = -1;
                return ;
        }
        for (int i=0; i<7 && i < days.length; i++) {
                try {
                        arriveInterval[i] = Integer.parseInt(days[i]);
                } catch (Exception e) {
                        arriveInterval[i] = -2;
                }
        }
}
public Integer getTimeOpening(int idx) { return timeOpening[idx] - 1; }
public Integer getTimeClosing(int idx) { return timeClosing[idx] - 1; }
public Integer getAverageTime(int idx) { return averageTime[idx]; }
public Integer getArriveInterval(int idx) { return arriveInterval[idx];
    }
public boolean haveDimension() {
        return !(px == null || py == null || width == null || height ==
            null || type == null);
}
private void consume(int step) {
        if (capacity == null) return ;
        Iterator<Integer> ii = capacity.iterator();
        if (pw != null) {
                pw.print("␣␣"+capacity.size()+"␣-");
                for (Integer i : capacity) pw.print("␣"+i);
                pw.println("␣BEFORE␣CONSUME");
        }
        while (ii.hasNext()) {
                Integer a = ii.next();
                if (a <= step) ii.remove();
        }
        if (pw != null) {
                pw.print("␣␣"+capacity.size()+"␣-");
                for (Integer i : capacity) pw.print("␣"+i);
                pw.println("␣AFTER␣CONSUME");
        }
        if (capacity.isEmpty()) capacity = null;
}
private void respawn(int weekday, int step) {
        int avgAgent = (int) (0.5 * totalCapacity);
        int rangeAgent = (int)(0.1 * avgAgent);
        int c = nextNormal(avgAgent, rangeAgent);
        if (c <= 0) return ;
        if (capacity == null) capacity = new ArrayList<Integer> ();
        int aHour = 5; // 3600s is 4,14 steps
```

```
        int news = 0;
        for (int i = 0; i < (int)c; i++) {
                int forward = nextNormal(averageTime[weekday],
                    arriveInterval[weekday]);
                if (forward > 0) {
                        capacity.add(step+forward*aHour);
                        news += 1;
                }
        }
        if (pw != null) pw.println("respawn␣-␣new␣person␣in␣place␣"+news
            );
}
// between 0 to 100
public int getRelativeCapacity() {
        int ret = 0;
        if (capacity == null || totalCapacity == null || totalCapacity
            <= 0)
                return ret;
        ret = (int) ((capacity.size() * 100.0) / totalCapacity);
        return ret;
}
public boolean itsOpen(int weekday) {
        if (timeOpening == null || timeClosing == null) return false;
        int to = getTimeOpening(weekday);
        int tc = getTimeClosing(weekday);
        if (to >= 0 && tc >= 0) { // It's can be -1 to indicate "not-
            applicable"
                return true;
        }
        return false;
}
// today,
//      0-monday                3-thursday              6-sunday
//      1-tuesday               4-friday
//      2-wednesday             5-saturday
public void update (int weekday, int hour, int step) {
        if (itsOpen(weekday) == false) {
                // When the locations dont open do nothing!
                capacity = null;
                return ;
        }
        // it's between 0 and 100
        int relativeCapacity;
        int cap = 0;
        int to = getTimeOpening(weekday);
        int tc = getTimeClosing(weekday);
        int mtc = (int)(1.1*tc);
        this.consume(step);
        if (opened == false) {
                if (hour > to && hour <= tc) {
                        this.respawn(weekday, step);
                        opened = true;
                }
        } else {
                if (hour > tc) {
                        opened = false;
                } else {
                        if (arriveInterval[weekday] > 0 && (hour %
                            arriveInterval[weekday]) == 0)
                                this.respawn(weekday, step);
                }
        }
        if ((mtc == 0 && hour <= mtc) || ((mtc > 0) && hour >= mtc)) {
                capacity = null;
        }
        if (capacity != null) cap = capacity.size();
        relativeCapacity = getRelativeCapacity();
        if (debugPlace) {
                if (pw == null) {
                        try {
                                pw = new java.io.PrintWriter(
                                        new java.io.FileWriter("
                                            /tmp/place-"+
```

```
                                                placeName+".txt")
                                        );
                        } catch (Exception e) {}
                }
                pw.println("placeName␣"+placeName
                                +"␣avgT␣"+ averageTime[weekday]
                                +"␣arvT␣"+ arriveInterval[weekday]
                                +"␣to␣"+ timeOpening[weekday]
                                +"␣tc␣"+ timeClosing[weekday]
                                +"␣mtc␣"+ mtc
                                +"␣capacity␣"+ relativeCapacity +"%"
                                +"␣capacity␣"+ cap
                                +"␣step␣"+step
                                +"␣opened␣"+opened
                                +"␣weekday␣"+weekday+"␣"+hour);
                pw.flush();
        }
}
public Literal getLiteral(String functor, int today, House h) {
        Literal ret = ASSyntax.createLiteral(functor,
                                ASSyntax.createString(getName()));
        if (today >= 0 && today < 7) {
                if (timeOpening != null) {
    int open = getTimeOpening(today);
    int openStep = h.controller.convertToStep(open);
                        ret.addAnnot(ASSyntax.createLiteral("opening",
                                                ASSyntax.createNumber(
                                                    openStep)));
                }
                if (timeClosing != null) {
    int close = getTimeClosing(today);
    int closeStep = h.controller.convertToStep(close);
                        ret.addAnnot(ASSyntax.createLiteral("closing",
                                                ASSyntax.createNumber(
                                                    closeStep)));
                }
        }
        if (capacity != null) {
                ret.addAnnot(ASSyntax.createLiteral("capacity",
                                        ASSyntax.createNumber(
                                            getRelativeCapacity())));
        }
        if (px != null) {
                ret.addAnnot(ASSyntax.createLiteral("positionX",
                                        ASSyntax.createNumber(getPX()))
                                            ;
                ret.addAnnot(ASSyntax.createLiteral("positionY",
                                        ASSyntax.createNumber(getPY()))
                                            ;
                if (width != null) {
                        ret.addAnnot(ASSyntax.createLiteral("sizeX",
                                                ASSyntax.createNumber(
                                                    getWidth())));
                }
                if (height != null) {
                        ret.addAnnot(ASSyntax.createLiteral("sizeY",
                                                ASSyntax.createNumber(
                                                    getHeight())));
                }
        }
        if (placeName != null) {
                ret.addAnnot(ASSyntax.createLiteral("name",
                                        ASSyntax.createString(getName())
                                            ));
        }
        // LATER verify if need send annotation as Number
        for (String key : annots.keySet()) {
                String[] value = annots.get(key).split(",");
                Term v;
                if (value.length > 1) {
                        ListTerm lt = ASSyntax.createList();
                        for (String s : value) {
                                lt.append(ASSyntax.createAtom(s));
```

```
                                }
                                v = lt;
                        } else {
                                v = ASSyntax.createAtom(value[0]);
                        }
                        ret.addAnnot(ASSyntax.createLiteral(key, v));
                }
                return ret;
        }
        private boolean debugPlace;
        private java.io.PrintWriter pw = null;
}
protected class Profile {
        private String profileName;
        private ArrayList<String> fixedDestinies;
        private ArrayList<String> randomDestinies;
        public Profile(String name) {
                profileName = name;
                fixedDestinies = new ArrayList<String> ();
                randomDestinies = new ArrayList<String> ();
        }
        public void addFixedDestination(String place) { fixedDestinies.add(place
            ); }
        public void addRandomDestination(String place) { randomDestinies.add(
            place); }
        public Set<Place> getFixedDestinations() {
                Set<Place> places = new HashSet<Place>();
                for (String s : fixedDestinies) {
                        Place p = referPlace.get(s);
                        if (p == null) continue;
                        places.add(p);
                }
                return places;
        }
        public Set<Place> getRandomDestinations() {
                Set<Place> places = new HashSet<Place>();
                for (String s : randomDestinies) {
                        Place p = referPlace.get(s);
                        if (p == null) continue;
                        places.add(p);
                }
                return places;
        }
}
protected HashMap<Integer, Agent> referAgent;
protected class Agent {
        private Integer id;
        private String name;
        private Profile profile;
        public Agent(Integer identification, String name) {
                this.name = name.toLowerCase();
                id = identification;
        }
        public Integer getID() { return id; }
        public String getName() { return name; }
        public void setProfile(Profile p) { profile = p; }
        public Profile getProfile() { return profile; }
        // Anotations?
        private Integer hungry = 50; // 0-100, 0 death
        private Integer social = 50; // 0-100, 0 death
        private Integer cleaning = 50; // 0-100
        private Integer energy = 40; // 0-105, 0 death
        private Character orientation = '␣';
        public Integer getHungry() { return hungry; }
        public void addHungry(Integer e) {
                hungry += e;
                if (hungry < 0) hungry = 0;
                if (hungry > 105) hungry = 105;
        }
        public Integer getSocial() { return social; }
        public void addSocial(Integer e) {
                social += e;
                if (social < 0) social = 0;
```

```
        if (social > 105) social = 105;
}
public Integer getCleaning() { return cleaning; }
public void addCleaning(Integer e) {
        cleaning += e;
        if (cleaning < 0) cleaning = 0;
        if (cleaning > 100) cleaning = 100;
}
public Integer getEnergy() { return energy; }
public void addEnergy(Integer e) {
        energy += e;
        if (energy < 0) energy = 0;
        if (energy > 105) energy = 105;
}
public String getOrientationText() {
        if (orientation == 'N') return "Nort";
        else if (orientation == 'E') return "East";
        else if (orientation == 'S') return "South";
        else if (orientation == 'W') return "West";
        return "Unknow";
}
public void randomOrientation() {
        String os = "NESW";
        orientation = os.charAt(nextInt(os.length()));
}
public Character getOrientation() { return orientation; }
public void setOrientation(char o) { orientation = o; }
public void updatePerception(House h) {
        h.clearPercepts(getName());
        Literal step    = ASSyntax.createLiteral("step",
                          ASSyntax.createNumber(h.controller.getStep()));
        Literal myself  = ASSyntax.createLiteral("myself");
        step.addAnnot(ASSyntax.createLiteral("day", ASSyntax.
            createNumber(h.controller.day)));
        step.addAnnot(ASSyntax.createLiteral("hour", ASSyntax.
            createNumber(h.controller.hour)));
        step.addAnnot(ASSyntax.createLiteral("minute", ASSyntax.
            createNumber(h.controller.mins)));
        // shift --> {"Dawn", "Noon", "Afternoon", "Night"}
        step.addAnnot(ASSyntax.createLiteral("shift", ASSyntax.
            createString(h.controller.shift)));
        myself.addAnnot(ASSyntax.createLiteral("hungry", ASSyntax.
            createNumber(getHungry())));
        myself.addAnnot(ASSyntax.createLiteral("social", ASSyntax.
            createNumber(getSocial())));
        myself.addAnnot(ASSyntax.createLiteral("cleaning", ASSyntax.
            createNumber(getCleaning())));
        myself.addAnnot(ASSyntax.createLiteral("energy", ASSyntax.
            createNumber(getEnergy())));
        // lookFor --> {"N", "E", "S", "W"}
        myself.addAnnot(ASSyntax.createLiteral("lookFor", ASSyntax.
            createString(getOrientation())));
        if (name != null) {
                myself.addAnnot(ASSyntax.createLiteral("name",
                                        ASSyntax.createString(getName())
                                            ));
        }
        Location l = getAgPos(getID());
        if (l != null) {
                myself.addAnnot(ASSyntax.createLiteral("positionX",
                                        ASSyntax.createNumber(l.x)));
                myself.addAnnot(ASSyntax.createLiteral("positionY",
                                        ASSyntax.createNumber(l.y)));
        }
        int today = h.controller.day % 7;
        perceptionPlace(getProfile().getFixedDestinations(), h, today, "
            fixed");
        perceptionPlace(getProfile().getRandomDestinations(), h, today,
            "random");
        updateBasedOnVision(h);
        h.addPercept(getName(), myself);
        h.addPercept(getName(), step); // this is the last
}
```

```
private void perceptionPlace(Set<Place> places, House h, int today,
    String prefix) {
        for (Place p : places) {
                Literal place;
                if (p.itsOpen(today) == false) continue;
                place = p.getLiteral(prefix+"Place", today, h);
                h.addPercept(getName(), place);
        }
}
private void updateBasedOnVision(House h) {
        int dx = 0, dy = 0, xrange = 0, yrange = 0;
        switch (orientation) {
                case 'N':
                        dy = -1;
                        dx = -1;
                        xrange = 2;
                        break;
                case 'E':
                        dx = 1;
                        dy = -1;
                        yrange = 2;
                        break;
                case 'S':
                        dy = 1;
                        dx = -1;
                        xrange = 2;
                        break;
                case 'W':
                        dx = -1;
                        dy = -1;
                        yrange = 2;
                        break;
                default:
                        return ; // fail??
        }
        Location l = agPos[getID()];
        boolean occluded = false;
        int x = l.x, y = l.y;
        int mx, my;
        for (int k = 1; k < 10; k++) { // 4,77m
                Place it = null;
                x = x + dx;
                y = y + dy;
                mx = x + xrange;
                my = y + yrange;
                if (x < 0 || y < 0 || x > 20 || y > 20)
                        break;
                for (Agent a : referAgent.values()) {
                        int id;
                        Location ll;
                        // When is doing a self annotation is used
                            myself
                        if (a == null || a.getID() == null || a.getID()
                            < 0 || a.getID() == getID())
                                continue;
                        id = a.getID();
                        ll = getAgPos(id);
                        if (ll == null) continue;
                        if (ll.x <= mx && ll.x >= x && ll.y <= my && ll.
                            y >= y) {
                                Literal literal = a.getLiteral();
                                String laction = h.getLastAction(getName
                                    ());
                                if (laction != null) {
                                        literal.addAnnot(
                                                ASSyntax.createLiteral("
                                                    lastAction",
                                                        ASSyntax.
                                                            createString
                                                            (laction),
                                    ASSyntax.createNumber(h.getStep()-1)
                                                                )
                                                );
```

```
                                        }
                                        h.addPercept(getName(), literal);
                                }
                        }
                        for (Place p : referPlace.values()) {
                                int px, py, pmx, pmy;
                                if (p.haveDimension() == false)
                                        continue;
                                px = p.getPX();
                                py = p.getPY();
                                pmx = p.getWidth() + px;
                                pmy = p.getHeight() + py;
                                if (px <= mx && pmx >= x && py <= my && pmy >= y
                                    ) {
                                        Literal place = p.getLiteral("object",
                                            -1, h);
                                        h.addPercept(getName(), place);
                                        if (p.getName().startsWith("wall")) {
                                                occluded = true;
                                        }
                                }
                        }
                        if (occluded == true) break;
                        if (xrange > 0) xrange += 2;
                        if (yrange > 0) yrange += 2;
                }
        }
        public Literal getLiteral() {
                Literal ret = ASSyntax.createLiteral("agent",
                                        ASSyntax.createString(getName())
                                );
                if (name != null) {
                        ret.addAnnot(ASSyntax.createLiteral("name",
                                                ASSyntax.createString(getName())
                                                    ));
                }
                Location l = getAgPos(getID());
                if (l != null) {
                        ret.addAnnot(ASSyntax.createLiteral("positionX",
                                                ASSyntax.createNumber(l.x)));
                        ret.addAnnot(ASSyntax.createLiteral("positionY",
                                                ASSyntax.createNumber(l.y)));
                }
                ret.addAnnot(ASSyntax.createLiteral("lookFor",
                        ASSyntax.createString(getOrientation())));
                return ret;
        }
    }
}
```

Arquivo *Java* - HouseView.java

```
package maro.example.sims;
import jason.environment.grid.GridWorldView;
import java.awt.Font;
import java.awt.Color;
import java.awt.Graphics;
public class HouseView extends GridWorldView {
    private HouseModel hm;
        private Font myFont = new Font("Arial", Font.BOLD, 8);
    public HouseView(HouseModel model, String windowTitle, int windowSize) {
        super(model, windowTitle, windowSize);
        hm = model;
    }
    @Override
    public void initComponents(int width) { super.initComponents(width); }
        @Override
    public void draw(Graphics g, int x, int y, int object) {
                switch (object) {
                        case HouseModel.OPENED_DOOR:
                                g.setColor(Color.lightGray);
                                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                                    -1, cellSizeH-1);
```

```
                g.setColor(Color.black);
                g.drawRect(x * cellSizeW + 2, y * cellSizeH+2, cellSizeW
                    -4, cellSizeH-4);
                break;
            case HouseModel.CLOSED_DOOR:
                g.setColor(Color.black);
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                g.setColor(Color.lightGray);
                g.drawRect(x * cellSizeW + 2, y * cellSizeH+2, cellSizeW
                    -4, cellSizeH-4);
                break;
            case HouseModel.TOILET: // vaso sanitario
                drawAgent(g, x, y, Color.orange, -1);
                break;
            case HouseModel.FAUCET: // vaso sanitario
                g.setColor(Color.blue);
                g.fillRect(
                                x * cellSizeW + 1,
                                (int) (y * cellSizeH + cellSizeH * 0.6),
                                cellSizeW - 1,
                                (int) (cellSizeH * 0.38 - 1)
                );
                g.fillRect(
                                (int) (x * cellSizeW + cellSizeW * 0.1),
                                (int) (y * cellSizeH + cellSizeH * 0.3),
                                (int) (cellSizeW * 0.5 - 1),
                                (int) (cellSizeH * 0.2 - 1)
                );
                g.fillRect(
                                (int) (x * cellSizeW + cellSizeW * 0.2),
                                (int) (y * cellSizeH + cellSizeH * 0.4 -
                                    1),
                                (int) (cellSizeW * 0.2 - 1),
                                (int) (cellSizeH * 0.3 - 1)
                );
                break;
            case HouseModel.SHOWER:
                g.setColor(Color.cyan);
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                break;
            case HouseModel.BED:
                g.setColor(Color.pink);
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                break;
            case HouseModel.TABLE:
                g.setColor(Color.pink.darker());
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                break;
            case HouseModel.BOOKCASE:
                g.setColor(Color.red.darker());
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                break;
            case HouseModel.SOFA:
                g.setColor(Color.orange.darker());
                g.fillRect(x * cellSizeW + 1, y * cellSizeH+1, cellSizeW
                    -1, cellSizeH-1);
                break;
            case HouseModel.TV:
                g.setColor(Color.yellow);
                g.fillRect(
                        x * cellSizeW + (int) (0.3 * cellSizeW),
                        y * cellSizeH + 1,
                        (int) (cellSizeW * 0.3),
                        cellSizeH-1);
                break;
            case HouseModel.REFRIGERATOR:
                drawAgent(g, x, y, Color.green.darker(), -1);
                break;
```

```
                        default:
                                System.out.println(object+"␣don't␣have␣a␣representation␣
                                    to␣draw");
                                break;
                }
    }
        @Override
        public void drawAgent(Graphics g, int x, int y, Color c, int id) {
                if (id < 0) {
                        super.drawAgent(g, x, y, c, id);
                        return ;
                }
        HouseModel.Agent a = hm.referAgent.get(id);
        if (a == null || id < 0) return ;
                g.setColor(c);
                drawTriangle(g, x, y, a.getOrientation());
        String agName = a.getName();
        if (agName != null) {
            g.setColor(Color.red);
            drawString(g, x, y, myFont, agName);
        }
    }
        private void drawTriangle(Graphics g, int x, int y, char orientation) {
                int cx = x * cellSizeW + (int) (0.5 * cellSizeW);
                int cy = y * cellSizeH + (int) (0.5 * cellSizeH);
                int [] xs = new int [] { cx, cx, cx };
                int [] ys = new int [] { cy, cy, cy };
                int [] base = new int [] {
                                                                (int) (0.4 * cellSizeW),
                                                                (int) (0 * cellSizeW),
                                                                (int) (-0.4 * cellSizeW)
                                                        };
                int [] top = new int [] {
                                                                (int) (-0.4 * cellSizeW)
                                                                    ,
                                                                (int) (0.4 * cellSizeW),
                                                                (int) (-0.4 * cellSizeW)
                                                        };
                switch (orientation) {
                        case 'N': // cima
                                xs[0] += base[0];       xs[1] += base[1];
                                xs[2] += base[2];       ys[0] += -top[0];
                                ys[1] += -top[1];       ys[2] += -top[2];
                                break;
                        case 'E': // direita
                                xs[0] += top[0];        xs[1] += top[1];
                                xs[2] += top[2];        ys[0] += base[0];
                                ys[1] += base[1];       ys[2] += base[2];
                                break;
                        case 'S':
                                xs[0] += base[0];       xs[1] += base[1];
                                xs[2] += base[2];       ys[0] += top[0];
                                ys[1] += top[1];        ys[2] += top[2];
                                break;
                        case 'W':
                                xs[0] += -top[0];       xs[1] += -top[1];
                                xs[2] += -top[2];       ys[0] += base[0];
                                ys[1] += base[1];       ys[2] += base[2];
                                break;
                        default:
                                break;
                }
                g.fillPolygon(xs, ys, xs.length);
        }
}
```

Arquivo *Java* - AStar.java

```
package maro.example.sims;
import java.util.PriorityQueue;
import java.util.Collections;
import java.util.LinkedList;
import java.util.Comparator;
```

```
import java.util.Iterator;
import java.util.HashMap;
import java.util.Arrays;
import java.util.Stack;
import java.util.List;
import java.util.Date;
import java.util.Map;
public class AStar {
        class Point implements java.lang.Comparable {
                private int x;
                private int y;
                private int f;
                private int g;
                private int h;
                private Point parent;
                public Point(int px, int py, int tx, int ty) {
                        x = px;
                        y = py;
                        g = 0;
                        h = Math.abs(tx-px) + Math.abs(ty-py);
                        f = g+h;
                        parent = null;
                }
                public Point getParent() { return parent; }
                public void setParent(Point p) {
                        parent = p;
                        g = p.getF();
                        f = g + h;
                }
                public int getX() { return x; }
                public int getY() { return y; }
                public int getF() { return f; }
                public int getG() { return g; }
                public int getH() { return h; }
                @Override
                public int compareTo(Object o) {
                        Point point = (Point) o;
                        if (getF() < point.getF()) return -1;
                        else if (getF() > point.getF()) return 1;
                        return 0;
                }
        }
        private int gx, gy;
        private int gsx, gsy;
        public String[] astar(int mx, int my, int tx, int ty, int tsx, int tsy, int[][]
            matrix) {
                PriorityQueue<Point> open = new PriorityQueue<Point>();
                List<Point> close = new LinkedList<Point>();
                Point next;
                boolean first = true;
                gx = tx; gy = ty;
                gsx = tsx - 1; gsy = tsy - 1;
                open.add(new Point(mx, my, gx, gy));
                while (open.isEmpty() == false) {
                        next = (Point) open.poll();
                        if (next == null) break;
                        if (checkGoal(next)) {
                                Stack<Point> path = new Stack<Point>();
                                Point current = next;
                                do {
                                        path.push(current);
                                        current = (Point)current.getParent();
                                } while (current!=null);
                                String[] result = new String[path.size()-1];
                                current = (Point)path.pop();
                                for (int i=0;i<result.length;++i) {
                                        next = (Point)path.pop();
                                        if (next.getX() > current.getX()) {
                                                result[i] = "E";
                                        } else if (next.getX() < current.getX()) {
                                                result[i] = "W";
                                        } else if (next.getX() == current.getX()) {
                                                if (next.getY() > current.getY()) {
```

```
                                        result[i] = "S";
                                    } else if (next.getY() < current.getY())
                                        {
                                        result[i] = "N";
                                    } else {
                                        // throw exception nulled
                                        String str = null;
                                        str.compareTo("EH␣POSSIVEL?");
                                    }
                                }
                                current = next;
                            }
                            return result;
                        }
                        if (first == false && matrix[next.getX()][next.getY()] > 0)
                            continue;
                        close.add(next);
                        first = false;
                        Point [] points = findNeighbours(next, matrix);
                        for (Point point : points) {
                                boolean exists = false;
                                for (Iterator<Point> iter = close.iterator(); iter.
                                    hasNext(); ) {
                                        Point p = (Point) iter.next();
                                        if (p.getX() == point.getX() && p.getY() ==
                                            point.getY()) {
                                                if ( point.getG() < p.getG() ) {
                                                        p.setParent( point.getParent() )
                                                            ; // this change G and F
                                                }
                                                exists = true;
                                                break;
                                        }
                                }
                                if (exists == true) continue;
                                for (Iterator<Point> iter = open.iterator(); iter.
                                    hasNext(); ) {
                                        Point p = (Point) iter.next();
                                        if (p.getX() == point.getX() && p.getY() ==
                                            point.getY()) {
                                                if ( point.getG() < p.getG() ) {
                                                        p.setParent( point.getParent() )
                                                            ; // this change G and F
                                                }
                                                exists = true;
                                                break;
                                        }
                                }
                                if (exists == true) continue;
                                open.add(point);
                        }
                }
                return null;
        }
        public Point[] findNeighbours(Point p, int [][]matrix) {
                List<Point> points = new LinkedList<Point> ();
                int[] xs = { 0, 0,-1, 1};
                int[] ys = {-1, 1, 0, 0};
                int size = matrix.length;
                for (int pos = 0; pos < xs.length; pos++) {
                        int cx = xs[pos] + p.getX();
                        int cy = ys[pos] + p.getY();
                        if (cx >= size || cx < 0 || cy >= size || cy < 0)
                                continue;
            Point point = new Point(cx, cy, gx, gy);
            point.setParent(p);
            points.add(point);
                }
                return points.toArray(new Point[0]);
        }
        protected boolean checkGoal(Point p) {
                int x = p.getX();
                int y = p.getY();
```

```
                if (gy <= y && y <= (gy+gsy) && (x == gx || x == (gx+gsx)) ) return true
                    ;
                else if ( gx <= x && x <= (gx+gsx) && (y == gy || y == (gy+gsy))) return
                     true;
                return false;
        }
}
```

Arquivo de Ação - ChangeOrientationAction.java

```
package maro.example.sims.ea;
import jason.asSyntax.Structure;
import jason.asSyntax.StringTerm;
import maro.example.sims.House;
import maro.core.EnvironmentAction;
import maro.example.sims.HouseModel;
import maro.core.IntelligentEnvironment;
public class ChangeOrientationAction extends EnvironmentAction {
        @Override
        public String getName() { return "changeOrientation"; }
        @Override
        public boolean execute(String agName, Structure action, IntelligentEnvironment
            ie) {
                House h = (House) ie;
                if (h == null) return false;
                HouseModel hm = h.getModel();
                if (hm == null) return false;
                StringTerm nt = (StringTerm) action.getTerm(0);
                String val = nt.getString();
                return hm.changeOrientation(agName, val.charAt(0));
        }
}
```

Arquivo de Ação - ForwardAction.java

```
package maro.example.sims.ea;
import maro.example.sims.House;
import maro.core.EnvironmentAction;
import maro.example.sims.HouseModel;
import maro.core.IntelligentEnvironment;
import jason.asSyntax.Structure;
public class ForwardAction extends EnvironmentAction {
        @Override
        public String getName() { return "forward"; }
        @Override
        public boolean execute(String agName, Structure action, IntelligentEnvironment
            ie) {
                House h = (House) ie;
                if (h == null) return false;
                HouseModel hm = h.getModel();
                if (hm == null) return false;
                return hm.forward(agName);
        }
}
```

Arquivo de Ação - HideMeAction.java

```
package maro.example.sims.ea;
import maro.example.sims.House;
import maro.core.EnvironmentAction;
import maro.example.sims.HouseModel;
import maro.core.IntelligentEnvironment;
import jason.asSyntax.Structure;
public class HideMeAction extends EnvironmentAction {
        @Override
        public String getName() { return "hide"; }
        @Override
        public boolean execute(String agName, Structure action, IntelligentEnvironment
            ie) {
                House h = (House) ie;
                if (h == null) return false;
                HouseModel hm = h.getModel();
```

```
                if (hm == null) return false;
                return hm.hideMe(agName);
        }
}
```

Arquivo de Ação - NopeAction.java

```
package maro.example.sims.ea;
import maro.example.sims.House;
import maro.core.EnvironmentAction;
import maro.example.sims.HouseModel;
import maro.core.IntelligentEnvironment;
import jason.asSyntax.Structure;
public class NopeAction extends EnvironmentAction {
        @Override
        public String getName() { return "nope"; }
        @Override
        public boolean execute(String agName, Structure action, IntelligentEnvironment
            ie) {
                House h = (House) ie;
                if (h == null) return false;
                HouseModel hm = h.getModel();
                if (hm == null) return false;
                return hm.nope(agName);
        }
}
```

Arquivo de Ação - TryUseObjectAction.java

```
package maro.example.sims.ea;
import jason.asSyntax.Structure;
import jason.asSyntax.StringTerm;
import maro.example.sims.House;
import maro.core.EnvironmentAction;
import maro.example.sims.HouseModel;
import maro.core.IntelligentEnvironment;
public class TryUseObjectAction extends EnvironmentAction {
        @Override public String getName() { return "tryUseObject"; }
        @Override
        public boolean execute(String agName, Structure action, IntelligentEnvironment
            ie) {
                House h = (House) ie;
                if (h == null) return false;
                HouseModel hm = h.getModel();
                if (hm == null) return false;
                return hm.tryUseObject(agName);
        }
}
```

Arquivo *AgentSpeak* - appraisal.asl

```
high("lhigh").                      mid("lmid").
low("llow").                        none("lnone").

+?appraisal
    <- // startin' get the priorities!
       .findall(prRe(A,P), priority(repulse, A, P), LRE);
       .findall(prAt(A,P), priority(attract, A, P), LAT);
       .findall(OBJ, OBJ[source(percept)], PERCEPTS);
       ?appraisalPercept(PERCEPTS, LRE, LAT).

+?appraisalPercept([], LRE, LAT).
+?appraisalPercept([PERCEPT|R], LRE, LAT)
     : PERCEPT =.. [FUNCTOR,TERMS,_]
    <- ?appraisalPercept(R, LRE, LAT);
       .findall(X, PERCEPT[X], ANNOTS);
       ?appraisalOnePercept(PERCEPT, ANNOTS, LRE, VALR);
       ?appraisalOnePercept(PERCEPT, ANNOTS, LAT, VALP).

+?appraisalOnePercept(_, _, [], 0).
+?appraisalOnePercept(PERCEPT, ANNOTS, [PR|L], NEWVAL)
    <- ?appraisalOnePercept(PERCEPT, ANNOTS, L, VAL);
```

```
       ?appraisalOneAnnotation(ANNOTS, PR, VAL, NEWVAL);
       ?evalAppraisal(PERCEPT, PR, VAL, NEWVAL).

+?appraisalOneAnnotation([], _, VAL, VAL).
+?appraisalOneAnnotation([H|R], PR, OLDVAL, NEWVAL)
     :  H =.. [NAMEPRI,          [VALTER|_], _]
     & PR =.. [FUNCPRI, [NAMEPRI,PARTPRI|_], _]
    <- ?appraisalOneAnnotationValue(VALTER, PARTPRI, FUNCPRI, OLDVAL, VAL);
       ?appraisalOneAnnotation(R, PR, VAL, NEWVAL).
+?appraisalOneAnnotation([H|R], PR, OLDVAL, NEWVAL)
    <- ?appraisalOneAnnotation(R, PR, OLDVAL, NEWVAL).
       //.println("       OneAnnot ", H, " priority ", PR).


//appraisalOneAnnotationValue----NUMBER-----------------------------------------
+?appraisalOneAnnotationValue(VALT, VALP, _, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT = VALP
    <- NEWV = OLDV.
+?appraisalOneAnnotationValue(VALT, VALP, prRe, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT < VALP
    <- NEWV = OLDV+(VALP-VALT).
+?appraisalOneAnnotationValue(VALT, VALP, prRe, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT > VALP
    <- NEWV = OLDV+(VALP-VALT).
+?appraisalOneAnnotationValue(VALT, VALP, prAt, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT > VALP
    <- NEWV = OLDV+(VALT-VALP).
+?appraisalOneAnnotationValue(VALT, VALP, prAt, OLDV, NEWV)
     : .number(VALT) & .number(VALP) & VALT < VALP
    <- NEWV = OLDV+(VALT-VALP).
//appraisalOneAnnotationValue----ATOM-------------------------------------------
+?appraisalOneAnnotationValue(VALT, VALT, prAt, OLDV, OLDV+1)
     : .atom(VALT).
+?appraisalOneAnnotationValue(VALT, VALT, prRe, OLDV, OLDV-1)
     : .atom(VALT).
+?appraisalOneAnnotationValue(VALT, VALP, FUNCTOR, OLDV, OLDV)
     : .atom(VALT) & .atom(VALP).

//evalAppraisal-----------------------------------------------------------------
+?evalAppraisal(_, _, OLDVAL, OLDVAL).
+?evalAppraisal(PCP, PREF, OLDVAL, NEWVAL)
     : PCP =..  [FUNCP,         [], _]
     & PREF =.. [FUNCF, [TERM1F|_], _]
     & NEWVAL > 0
    <- .concat("", FUNCP, "_", TERM1F, INDIVIDUAL);
       ?updateAppraisal(joy, INDIVIDUAL, NEWVAL).
+?evalAppraisal(PCP, PREF, OLDVAL, NEWVAL)
     : PCP =..  [FUNCP,         [], _]
     & PREF =.. [FUNCF, [TERM1F|_], _]
     & NEWVAL < 0
    <- .concat("", FUNCP, "_", TERM1F, INDIVIDUAL);
       ?updateAppraisal(distress, INDIVIDUAL, NEWVAL).
+?evalAppraisal(PCP, PREF, OLDVAL, NEWVAL). // ignoring others

//updateAppraisal-Joy-----------------------------------------------------------
+?updateAppraisal(joy, INDIVIDUAL, VALUE)
     : not(hasLikelihood(INDIVIDUAL, "lnone")) & .my_name(NAME)
    <- +isAppraisalOf(INDIVIDUAL, NAME);
       +hasLikelihood(INDIVIDUAL, "lnone");
       +hasDesireSelf(INDIVIDUAL, VALUE).
+?updateAppraisal(joy, INDIVIDUAL, VALUE)
     : hasDesireSelf(INDIVIDUAL, OLDVAL)
    <- -hasDesireSelf(INDIVIDUAL, OLDVAL);
       +hasDesireSelf(INDIVIDUAL, VALUE).
//updateAppraisal-Distress------------------------------------------------------
+?updateAppraisal(distress, INDIVIDUAL, VALUE)
     : not(hasLikelihood(INDIVIDUAL, "lnone")) & .my_name(NAME)
    <- +isAppraisalOf(INDIVIDUAL, NAME);
       +hasLikelihood(INDIVIDUAL, "lnone");
       +hasDesireSelf(INDIVIDUAL, VALUE).
+?updateAppraisal(distress, INDIVIDUAL, VALUE)
     : hasDesireSelf(INDIVIDUAL, OLDVAL)
    <- -hasDesireSelf(INDIVIDUAL, OLDVAL);
```

```
       +hasDesireSelf(INDIVIDUAL, VALUE).
//updateAppraisal-Love-------------------------------------------------------
+?updateAppraisal(love, INDIVIDUAL, VALUE)
     : not(hasFamiliarity(INDIVIDUAL, _)) & .my_name(NAME)
    <- +isAppraisalOf(INDIVIDUAL, NAME);
       +hasFamiliarity(INDIVIDUAL, VALUE).
+?updateAppraisal(love, INDIVIDUAL, VALUE)
     : hasFamiliarity(INDIVIDUAL, OLDVAL)
    <- -hasFamiliarity(INDIVIDUAL, OLDVAL);
       +hasFamiliarity(INDIVIDUAL, VALUE).
//updateAppraisal-Hate-------------------------------------------------------
+?updateAppraisal(hate, INDIVIDUAL, VALUE)
     : not(hasFamiliarity(INDIVIDUAL, _)) & .my_name(NAME)
    <- +isAppraisalOf(INDIVIDUAL, NAME);
       +hasFamiliarity(INDIVIDUAL, VALUE).
+?updateAppraisal(hate, INDIVIDUAL, VALUE)
     : hasFamiliarity(INDIVIDUAL, OLDVAL)
    <- -hasFamiliarity(INDIVIDUAL, OLDVAL);
       +hasFamiliarity(INDIVIDUAL, VALUE).
```

Arquivo Comum dos Agentes - basicAgent3.asl

```
{ include("appraisal.asl") }
{ include("supportDiscoverLocation.asl") }
{ include("util.asl") }
//------------------------------------------------------------------------------
!start.
+!start
    <- nope; // if have some preprocess do here!
       !!deliberation.
//------------------------------------------------------------------------------
+!deliberation
     : .random(R) & NRANDOM=math.round(R*100000)
    <- ?appraisal;
       !behaviour(NRANDOM). // 0 to 100000
//------------------------------------------------------------------------------
+!behaviour(RANDOM)
     : not(room(_))
    <- .println("doing␣discover␣location");
       !planDiscoverLocation(RANDOM);
       ?room(ROOM);
       .println("I␣am␣on␣room␣", ROOM);
       !!deliberation.
//------------------------------------------------------------------------------
+!behaviour(N)
     : feeling(distress, DV) & DV > 0
    <- .println("doing␣rest");
       ?step(STEP);
       !!planRest(N, STEP+15).
//------------------------------------------------------------------------------
+!behaviour(N)
    : not(feeling(distress, PV))
    & fixedPlace(_)[opening(OP),closing(CL),name(LOCATION)]
    & step(STEP) & beforeGo(BEFORE) & GO = OP - BEFORE
    & STEP >= GO & STEP <= CL
    <- .println("going␣to␣", LOCATION);
       !!planGoOut(LOCATION, N, STEP, OP).
//------------------------------------------------------------------------------
+!behaviour(RANDOM)
    : myself[energy(E)]
    & ARRAY=[changeOrientation("N"), changeOrientation("S"), //0 and 1
             changeOrientation("E"), changeOrientation("W"), //2 and 3
             forward, forward, forward, forward, forward, nope]
    & .nth(RANDOM mod 10, ARRAY, ACTION)
    <- .println("doing␣", ACTION, "␣(energy=",E,")");
       ACTION;
       !!deliberation.
//------------------------------------------------------------------------------
+!behaviour(_) // probably not reach here...
    : myself[energy(E)]
    <- .println("doing␣nope␣(energy=",E,")");
       nope;
       !!deliberation.
```

```
//------------------------------------------------------------------------------
+!planDiscoverLocation(RANDOM)
    : myself[lookFor(ORIENTATION)] & not(perceived(ORIENTATION, _))
   <- .println("perceiving␣the␣around␣environment");
      .findall(OBJ, object(OBJ)[source(percept)], LISTA);
      +perceived(ORIENTATION, LISTA);
      !planDiscoverLocation(RANDOM).
//------------------------------------------------------------------------------
+!planDiscoverLocation(RANDOM)
     : .findall(O, perceived(O, _), OS)
     & .difference(["N","E","S","W"], OS, LS)
     & .length(LS, LENGTH) & LENGTH > 0
    <- .println("look␣for␣a␣new␣orientation");
       POSITION=RANDOM mod LENGTH;
       .nth(POSITION, LS, NEWO);
       changeOrientation(NEWO);
       !planDiscoverLocation(RANDOM).
//------------------------------------------------------------------------------
+!planDiscoverLocation(RANDOM)
     : .findall(O, perceived(_, O), OS)
     & .length(OS, LENGTH) & LENGTH > 2
    <- .println("calculate␣position");
       !calculatePosition(OS);
       .abolish(perceived(_,_)).
//------------------------------------------------------------------------------
+!planRest(RANDOM, LastStep)
    : step(LastStep) | (myself[energy(E)] & E >= 100)
    <- .println("planning␣rest:␣forget␣plan");
       .abolish(target(_,_));
       !!deliberation.
//------------------------------------------------------------------------------
+!planRest(RANDOM, LastStep)
    :  not(target(_,_)) & room(ROOM)
    <- .println("planning␣rest:␣find␣a␣goal␣from␣room␣", ROOM);
       if (not rest(_)) {
          !planRoute("bed", ROOM, ROUTE);
          !nearRouteOf(ROUTE, _, _, route(O, DISTANCE, PLAN));
        } else {
          ?rest(O);
          !planRoute(O, ROOM, ROUTE);
          .member(route(O, _, PLAN), ROUTE);
        }
       +target(O, PLAN);
       .println("planning␣rest:␣found␣object␣", O, "␣and␣route␣", PLAN);
       !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    : (step(LastStep) & NEWRANDOM=RANDOM)
    | (feeling(joy, X) & X > 5 & (RANDOM mod 10) >= 7
        & .random(R) & NEWRANDOM=math.round(R*100000))
    <- !!planRest(NEWRANDOM, LastStep). // only one place
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
     : target(O, _) & ~rest(O)
    <- .println("planning␣reach␣object␣", O, "␣canceled␣because␣is␣not␣my␣bed");
       .abolish(target(_,_));
        !planDiscoverLocation(RANDOM);
       !!planRest(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [])
    &  fixMeOrientation(O, NEWO, true) // perceived object and is near (true)
    &  myself[lookFor(NEWO), energy(E)]
    <- .println("planning␣reach␣object␣", O, ":␣try␣use␣(energy=",E,")");
        ?appraisal;
        tryUseObject;
        !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [])
    &  fixMeOrientation(O, NEWO, _)
    &  myself[lookFor(NEWO)]
    <- .println("planning␣reach␣object␣", O, ":␣approach");
```

```
        ?appraisal;
        forward;
        !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [])
    &  fixMeOrientation(O, NEWO, _)
    <- .println("planning␣reach␣object␣", O, ":␣fix␣orientation");
        ?appraisal;
        changeOrientation(NEWO);
        !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [P|R]) & fixMeOrientation(P, NEWO, true) & myself[lookFor(NEWO)]
    <- .println("planning␣reach␣object␣", P, ":␣use␣gate");
       ?appraisal;
       tryUseObject;
       .abolish(target(_,_)); +target(O, R);
       !changeRoom(P); !planDiscoverLocation(RANDOM);
       !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [P|R]) & fixMeOrientation(P, NEWO, _)
    &  myself[lookFor(NEWO)]
    <- .println("planning␣reach␣object␣", P, ":␣approach␣to␣gate");
       ?appraisal;
       forward;
       !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
    :  target(O, [P|R]) & fixMeOrientation(P, NEWO, _)
    <- .println("planning␣reach␣object␣", P, ":␣fix␣orientation␣to␣gate");
       ?appraisal;
       changeOrientation(NEWO);
       !!planReach(RANDOM, LastStep).
//------------------------------------------------------------------------------
+!planReach(RANDOM, LastStep)
     : target(O, []) | target(_,[O|_])
    <- .println("planning␣reach␣object␣", O, ":␣meet␣object");
       sims.ia.planRoute(O, PLAN);
       //.println("plan to reach object ", O, " = ", PLAN);
       !!planReachObject(RANDOM, LastStep, PLAN).
//------------------------------------------------------------------------------
+!planReachObject(R, L, _)
    : step(L)
    <- !!planReach(R, L).
//------------------------------------------------------------------------------
+!planReachObject(R, L, [STEP|[]])
     : myself[lookFor(STEP)]
    <- .println("planning␣reach␣object␣finish");
       !!planReach(R, L).
//------------------------------------------------------------------------------
+!planReachObject(R, L, [STEP|PLAN])
     : myself[lookFor(STEP)]
    <- .println("planning␣reach␣object:␣approach");
       forward;
       !!planReachObject(R, L, PLAN).
//------------------------------------------------------------------------------
+!planReachObject(R, L, [STEP|PLAN])
    <- .println("planning␣reach␣object:␣fix␣orientation");
       changeOrientation(STEP);
       !!planReachObject(R, L, [STEP|PLAN]).
//------------------------------------------------------------------------------
-!planReachObject(R, L, _)
    <- .println("plan␣reach␣object:␣fail␣on␣route");
       .abolish(target(_,_)); nope;
       !!deliberation.
//------------------------------------------------------------------------------
-!planReach(RANDOM,LASTSTEP)[code(sims.ia.planRoute(_,_))]
    <- .println("plan␣reach␣object:␣fail␣on␣locate␣a␣route");
       .abolish(target(_,_));
       ?room(ROOM);
       .broadcast(achieve, freedomMy(ROOM)); nope;
```

```
       !!deliberation.
//------------------------------------------------------------------------------
-!planReach(RANDOM,LASTSTEP)
    <- .println("plan␣reach␣object:␣retry");
       nope; // stop now and try again
       !!planReach(RANDOM,LASTSTEP).
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
     : step(FINISH)
    <- .println("plan␣go␣out:␣failed␣to␣reach␣point");
       ?appraisal;
       ?updateBeforeGo(STEP, FINISH, false);
       .abolish(goOutPlan(_,_));
       .abolish(goOutSubPlan(_));
       nope;
       !!deliberation.
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
     : goOutPlan(OBJ,[_|R]) & goOutSubPlan([ELEM|[]]) & myself[lookFor(ELEM)]
    <- .println("plan␣go␣out:␣using␣object");
       ?appraisal;
       tryUseObject;
       .abolish(room(_)); !planDiscoverLocation(N);
       .abolish(goOutSubPlan(_));
       .abolish(goOutPlan(_,_)); +goOutPlan(OBJ,R);
       !!planGoOut(LOCATION, N, STEP, FINISH).
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
     : goOutPlan(_,_) & goOutSubPlan([ELEM|R]) & myself[lookFor(ELEM)]
    <- .println("plan␣go␣out:␣approach");
       ?appraisal;
       forward;
       .abolish(goOutSubPlan(_)); +goOutSubPlan(R);
       !!planGoOut(LOCATION, N, STEP, FINISH).
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
     : goOutPlan(_,_) & goOutSubPlan([ELEM|R])
    <- .println("plan␣go␣out:␣fixing␣orientation␣to␣", ELEM);
       ?appraisal;
       changeOrientation(ELEM);
       !!planGoOut(LOCATION, N, STEP, FINISH).
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
     : goOutPlan(ELEM, []) | goOutPlan(_, [ELEM|R])
    <- .println("plan␣go␣out:␣reach␣", ELEM);
       ?appraisal;
       sims.ia.planRoute(ELEM, PLAN);
       +goOutSubPlan(PLAN);
       .println("plan␣go␣out:␣reach␣", ELEM, "␣using␣route␣", PLAN);
       !!planGoOut(LOCATION, N, STEP, FINISH).
//------------------------------------------------------------------------------
+!planGoOut(LOCATION, N, STEP, FINISH)
    <- .println("plan␣go␣out:␣planning␣route");
       ARRAY = ["doorToHome1", "doorToHome2"];
       POS = (N mod 2);
       .nth(POS, ARRAY, ELEM);
       ?room(ROOM);
       ?appraisal;
       !planRoute(ELEM, ROOM, ROUTE);
       .member(route(ELEM, _, PLAN), ROUTE);
       +goOutPlan(ELEM, PLAN);
       !!planGoOut(LOCATION, N, STEP, FINISH).
//------------------------------------------------------------------------------
-!planGoOut(LOCATION, N, STEP, FINISH)[code(sims.ia.planRoute(_,_))]
    <- .println("plan␣go␣out:␣no␣route␣found");
       ?room(ROOM);
       .broadcast(achieve, freedomMy(ROOM));
       .abolish(goOutPlan(_,_)); .abolish(goOutSubPlan(_));
       !!deliberation.
//------------------------------------------------------------------------------
+!freedomMy(ROOM)
    :  room(ROOM)
    <- .findall(D, .desire(D), DS);
```

```
       .findall(CI, .current_intention(CI), IS);
       .drop_all_desires; .drop_all_intentions;
       .abolish(goOutSubPlan(_)); .abolish(goOutPlan(_, _));
       .abolish(perceived(_,_)); .abolish(target(_,_));
       !!freedomRetry(0).
//------------------------------------------------------------------------------
+!freedomMy(ROOM).
//------------------------------------------------------------------------------
+!freedomRetry(QTD)
    <- ?room(ROOM);
       sims.ia.getItemsAtPlace(ROOM, LIST);
       .length(LIST, LISTLEN);
       .random(N); RANDOM = math.round(N * 100000);
       .nth(RANDOM mod LISTLEN, LIST, POS);
       sims.ia.planRoute(POS, PLAN);
       +goOutPlan(POS, []); +goOutSubPlan(PLAN);
       !!planGoOut("freedom", RANDOM, -1, -1).
//------------------------------------------------------------------------------
-!freedomRetry(QTD)
    : room(ROOM) & sims.ia.getItemsAtPlace(ROOM, LIST)
    & .length(LIST, LISTLEN) & QTD > (2*LISTLEN)
    <- .abolish(goOutSubPlan(_)); .abolish(goOutPlan(_, _));
       nope;
       !!deliberation.
//------------------------------------------------------------------------------
-!freedomRetry(QTD)
    <- !!freedomRetry(RANDOM, LASTSTEP, QTD+1).
//------------------------------------------------------------------------------
-!X <- .println("Handler␣failure:␣", X);
       !!deliberation.
//------------------------------------------------------------------------------
+object(O)[utility(sleep),owner(K)]
    : not(rest(O)) & not(~rest(O))
    & .my_name(NAME)
    & (K=NAME | (.list(K) & .sublist([NAME], K)))
    <- .println("object␣",O,"␣is␣my␣bed");
       +rest(O).
//------------------------------------------------------------------------------
+object(O)[utility(sleep),owner(K)]
    : not(rest(O)) & not(~rest(O))
    <- +~rest(O).
//------------------------------------------------------------------------------
```

Arquivo *AgentSpeak* - supportDiscoverLocation.asl

```
{ include("routes.asl") }
//------------------------------------------------------------------------------
+!calculatePosition(_) : room(_).
+!calculatePosition([]) : .findall(L, roomP(L), []).
+!calculatePosition([])
    <- .findall(L, roomP(_)[likelihood(L)], RRL);
       .max(RRL, AR);
       .findall(R, roomP(R)[likelihood(AR)], RR);
       .abolish(roomP(_));
       .nth(0, RR, VAL);
       +room(VAL).
+!calculatePosition([H|R])
    <- !calculatePositionByItems(H);
       !calculatePosition(R).
//------------------------------------------------------------------------------
+!calculatePositionByItems([]).
+!calculatePositionByItems([H|R])
    <- sims.ia.getPlacesByItem(H, Places);
       !computatePlaces(Places);
       !calculatePositionByItems(R).
//------------------------------------------------------------------------------
+!computatePlaces([]).
+!computatePlaces([H|R])
     : roomP(H)[likelihood(N)]
    <- +roomP(H)[likelihood(N+1)];
       !computatePlaces(R).
+!computatePlaces([H|R])
    <- +roomP(H)[likelihood(1)];
```

```
       !computatePlaces(R).
//------------------------------------------------------------------------------
+!planRoute(PREFIX, SOURCE, ROUTE)
    <- sims.ia.getItems(ITEMS);
       !filterListByPrefix(PREFIX, ITEMS, ITEMSFILTERED);
       !planRouteByItem(ITEMSFILTERED, SOURCE, ROUTE).
//------------------------------------------------------------------------------
+!filterListByPrefix(PrefixName, [], []).
+!filterListByPrefix(PrefixName, [Item|Items], NewItemsFiltered)
     : .substring(PrefixName, Item, 0)
    <- !filterListByPrefix(PrefixName, Items, ItemsFiltered);
       .concat([Item], ItemsFiltered, NewItemsFiltered).
+!filterListByPrefix(PrefixName, [Item|Items], ItemsFiltered)
    <- !filterListByPrefix(PrefixName, Items, ItemsFiltered).
//------------------------------------------------------------------------------
+!planRouteByItem([], _, []).
+!planRouteByItem([ITEM|ITEMS], SOURCE, ROUTE)
     : ~rest(ITEM) // exist the information that I cannot rest on item
    <- !planRouteByItem(ITEMS, SOURCE, ROUTE).
+!planRouteByItem([ITEM|ITEMS], SOURCE, [route(ITEM,RL,R)|ROUTE])
    <- !planRouteByItem(ITEMS, SOURCE, ROUTE);
       sims.ia.getPlacesByItem(ITEM, [TARGET]);
       ?routeCompleteTo(SOURCE, TARGET, R);
       .length(R, RL).
//------------------------------------------------------------------------------
+!nearRouteOf([], L, R, R) : .ground(L).
+!nearRouteOf([H|R], LESSER, _, ROUTE)
     : H =.. [route, [_, LESSER, _], _]
    <- !nearRouteOf(R, LESSER, H, ROUTE).
+!nearRouteOf([H|R], LESSER, _, ROUTE)
     : H =.. [route, [_, DISTANCE, _], _] & DISTANCE < LESSER
    <- !nearRouteOf(R, DISTANCE, H, ROUTE).
+!nearRouteOf([H|R], LESSER, K, ROUTE)
     : H =.. [route, [_, DISTANCE, _], _]
    <- !nearRouteOf(R, LESSER, K, ROUTE).
//------------------------------------------------------------------------------
+!changeRoom(OBJECT) : room(R)
    <- sims.ia.getPlacesByItem(OBJECT, LIST);
       ?anotherSource(LIST, R, [NEWSOURCE]);
       -room(R); +room(NEWSOURCE).
```

Arquivo *AgentSpeak* - util.asl

```
beforeGo(5). // initial value of steps before go to a fixed place
//------------------------------------------------------------------------------
myNth(POS, ARRAY, OUT) :- index(ARRAY, POS, 0, OUT).
//------------------------------------------------------------------------------
index([], _, _, _) :- .fail.
index([H|R], IDX, IDX, H).
index([H|R], IDX, POS, OUT) :- index(R, IDX, POS+1, OUT).
//------------------------------------------------------------------------------
+?updateBeforeGo(STEPINITIAL, STEPFINAL, true)
     : DIFF=(STEPFINAL-STEPINITIAL) & beforeGo(DIFF).
+?updateBeforeGo(STEPINITIAL, STEPFINAL, true)
     : DIFF=(STEPFINAL-STEPINITIAL) & beforeGo(VAL)
    <- NEWVAL=VAL-(DIFF div 2); -beforeGo(VAL); +beforeGo(NEWVAL).
+?updateBeforeGo(_, _, false)
    <- -beforeGo(VAL); +beforeGo(VAL*VAL).
+?updateBeforeGo(STEPINITIAL, STEPFINAL, REACH).
```