

Programação da Comunicação entre Agentes BDI em um Ambiente SMA

Paulo Roberto Pasqualotti
PIPCA- Unisinos
ppasqua@terra.com.br

Resumo

Este artigo apresenta uma arquitetura baseada em agentes com o uso de uma linguagem do paradigma de lógica, a linguagem AgentSpeak(L). Além da formalização da comunicação e demais aspectos que ocorrem no ambiente e entre os agentes, interferindo nas percepções dos indivíduos e/ou da coletividade.

O interpretador dessa linguagem é o Jason, responsável pela execução dos aspectos computacionais do ambiente, escrito em Java, rodando em uma JVM (Máquina Virtual Java).

É demonstrado um estudo de caso, com as interações entre dois agentes em um ambiente virtual, suas percepções e as alterações advindas da comunicação com o envio e o recebimento de mensagens. Dessa forma é possível conhecer a sintaxe da linguagem, a programação das ações bem como os mecanismos de cada parte desse mundo virtual, quais sejam: os agentes, o ambiente, a forma como são organizados e a interação/comunicação entre eles.

Finalmente é demonstrada uma ocorrência de situação no ambiente proposto, apresentando-se a comunicação como uma solução viável.

O estudo de caso ocorre em um ambiente social de troca, de interação entre agentes, expondo os recursos da programação e de mecanismos que especificam os comportamentos sociais da comunicação por meio de uma Linguagem de Comunicação de Agentes: LCA.

1. Introdução.

Agentes são entidades imersas em ambientes virtuais onde, a partir de sistemas computacionais complexos e técnicas da inteligência artificial, interagem com outros agentes e com o próprio meio, possuindo capacidade de perceber alterações, agir de acordo com regras pré-definidas ou criadas a partir dessas percepções, comunicar-se e interagir, explicitar e representar entidades, objetos e grandezas físicas com personalidade e autonomia, raciocinar, deliberar e tomar decisão, além de apresentar uma capacidade para aprender.

Sua capacidade em perceber ocorre a partir de sensores e de agir por intermédio de atuadores[4].

Como uma entidade, agentes usam a comunicação como mecanismos de troca de informações com o ambiente e entre os próprios agentes. Essa troca, intencional, ocorre a partir de mudanças percebidas no ambiente, realimentando o estado e a capacidade de um

agente, em específico no modelo BDI tratado nesse trabalho.

Esse artigo aborda também os aspectos sociais dos agentes, interagindo coletivamente em um sistema Multiagentes, denominado SMA, onde mecanismos de raciocínio sociais ocorrem a partir das relações com outros agentes, sendo abordados as duas grandes arquiteturas de agentes SMA: os reativos e os cognitivos.

A linguagem AgentSpeak(L), inspirada na arquitetura BDI é uma extensão do paradigma de programação em lógica, tendo sido a abordagem predominante na implementação de agentes inteligentes ou “racionais”[3].

O Jason é um interpretador da linguagem AgentSpeak(L) e possui capacidade de implementar as facilidades esperadas e previstas para agentes em um ambiente social, em especial a comunicação. Será dada atenção a esse tópico, pela sua familiaridade com a área da computação, pois é onde todo o ambiente, bem como as estruturas dos agentes são interpretadas e executadas de acordo com as formalidades da linguagem.

Jason é escrito em Java e apresenta a portabilidade necessária e esperada de um sistema computacional, características das aplicações interpretadas por uma JVM.

Continuando com o foco nas estruturas algorítmicas da programação, é apresentado um estudo de caso de um ambiente SMA, onde ocorre a comunicação entre agentes sendo possível perceber e demonstrar a lógica por trás desses mecanismos que dão significado e consistência à programação de agentes. O ambiente é um cenário onde o agente (personagem) deve percorrer e atravessar, percebendo obstáculos (dificuldades), contornando-os e encontrando caminhos alternativos, comunicando à coletividade (ao segundo personagem: p2) quando encontrou os obstáculos, bem como quando finalizou a travessia.

Finalmente são apresentadas algumas telas do funcionamento do sistema, com a identificação de um problema aqui denominado o “problema da caverna”, ou seja, quando o cenário, criado aleatoriamente, apresentar uma disposição dos obstáculos que formam um ambiente “fechado”, onde não há possibilidades do personagem avançar, mas somente retornar.

A análise de caminhos alternativos por um personagem é utilizada nos sistemas de agentes cognitivos, onde a capacidade de escolha do melhor caminho exige a capacidade de perceber, analisar, decidir e agir, conforme a complexidade do algoritmo implementado.

Uma sociedade de agentes (personagens) será capaz de perceber, aprimorar e comunicar aos demais integrantes, caminhos e rotas alternativas no caso de bloqueios, obstáculos ou dificuldades encontradas.

Dentro da metodologia de SMA (Sistemas Multiagentes), a comunicação entre os diversos

personagens, no caso da análise dos caminhos seguidos, acontecerá por meio de um sistema de filas, ou no caso da programação recursiva, por meio de uma pilha de células transpostas.

2. Arquitetura de agentes BDI.

Uma arquitetura BDI é baseada em um modelo de cognição fundamentada nas atitudes mentais que são: crenças, desejos e intenções [2], que caracterizam e dão significado para a lógica BDI (sigla de beliefs, desires e intentions). Essas atitudes são explicitadas por meio da identificação pelo agente da situação do ambiente. Sua capacidade sensorial de perceber e identificar mudanças ocorre pela troca com outros agentes e com o ambiente. Ao perceber alterações, ocorre a revisão daquilo que o agente acredita ser verdadeiro sobre o ambiente, ou seja, suas *crenças*.

A função de revisão de crenças (FRC na figura 1) recebe a informação percebida, consulta as bases de crenças do agente, atualiza essas crenças para que elas reflitam o novo estado do ambiente.

A partir dessa redefinição de suas bases de crenças, o agente é capaz de analisar e redefinir seus objetivos a fim de atingir um estado esperado e desejado de mundo, ou seja, seus *desejos*. Na arquitetura apresentada na figura 1, essa função é denominada de Gera opções.

Reconstruindo esses objetivos, o agente revisa e adequa suas bases de *intenções* para assim atingi-los. Dessa forma faz-se o ciclo esquemático da arquitetura BDI.

Uma vez atualizado o conhecimento de mundo e a motivação do agente, é preciso, em seguida, decidir qual curso de ações específico será usado para alcançar os objetivos atuais do agente (para isso é preciso levar em conta os outros cursos de ações com os quais o agente já se comprometeu, para evitar ações incoerentes, bem como eliminar intenções que já foram atingidas ou que se tornaram impossíveis de ser atingidas). Esse é o papel da função Filtro, que atualiza o conjunto de intenções do agente, com base nas crenças e desejos atualizados e nas intenções já existentes. Esse processo realizado pela função Filtro para determinar como atualizar o conjunto de intenções do agente é normalmente chamado de deliberação¹ [3].

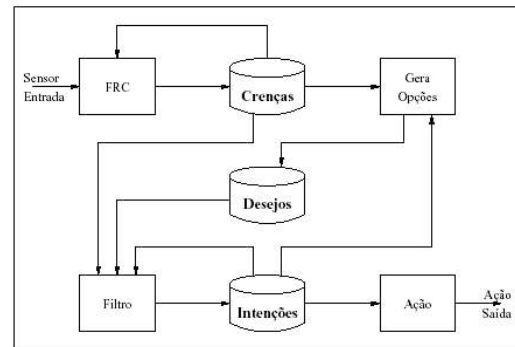


Figura 1: Arquitetura BDI.

3. Sistemas Multiagentes (SMA).

Um sistema Multiagentes possui características e vantagens que determinam sua viabilidade e funcionalidade como ambiente virtual de uma coletividade baseada em recursos computacionais.

Sistemas SMA possuem a capacidade de adaptar-se a novas situações, mudando sua organização e/ou determinando a formação da coletividade de agentes, permitindo um alto nível de abstração por apresentar-se na forma de uma metáfora natural para a modelagem de sistemas complexos de forma distribuída, onde o conhecimento, o controle e os recursos podem estar distribuídos entre várias e diferentes plataformas, arquiteturas e sistemas[5].

O presente trabalho fundamenta-se no formalismo da lógica, empregada para planejar, projetar e implementar o funcionamento de um SMA.

Arquiteturas SMA baseiam-se em lógica e utilizam mecanismos de raciocínio de um agente em relação a sua percepção de mundo, com e para os outros agentes e também para o ambiente. Por isso por definição, essa capacidade é denominada de raciocínio social.

Existem dois grandes tipos de arquitetura para os agentes de um SMA: os reativos e os cognitivos

3.1 SMA reativos

Os agentes agem e comportam-se baseados em estruturas e definições pré-definidas e agem sob um esquema estímulo-resposta [3].

Da mesma forma selecionam ações com base na percepção atual, ignorando o restante do histórico de percepções [4].

3.2 SMA cognitivos

Nos modelos cognitivos, normalmente se considera que os agentes possuem um estado mental (suas bases de crenças, desejos e intenções, entre outras) e funcionam racionalmente, isto é, raciocinam para construir um plano de ações que leva a um objetivo pretendido.

Estes agentes apresentam, portanto, características particulares que os diferenciam de programas dos agentes reativos, dentre elas a autonomia funcional, ou seja, a capacidade de alterar seu funcionamento a fim de adaptar-se às condições e situações do ambiente, além de

¹ Essa é a origem do termo *deliberativo* muitas vezes utilizado para se referir a agentes cognitivos, já que a habilidade de realizar essa deliberação é uma característica muito importante para esse tipo de agente.

comunicar-se, foco desse trabalho e também de raciocinar sobre as ações a serem realizadas [5].

4. Linguagem Agent Speak.

A linguagem AgentSpeak(L) [2][3][6] foi projetada para a programação de agentes BDI na forma de sistemas de planejamento reativos (reactive planning systems) que são sistemas que estão permanentemente em execução, reagindo a eventos que acontecem no ambiente em que estão situados através da execução de planos que se encontram em uma biblioteca de planos parcialmente instanciados.

A linguagem é uma extensão natural e elegante de programação em lógica para a arquitetura de agentes BDI, que representa um modelo abstrato para a programação de agentes e tem sido a abordagem predominante na implementação de agentes inteligentes ou “racionais”. Um agente AgentSpeak(L) corresponde à especificação de um conjunto de crenças que formarão a base de crenças inicial e um conjunto de planos. Um átomo de crença é um predicado de primeira ordem na notação lógica usual, e literais de crença são átomos de crenças ou suas negações. A base de crenças de um agente é uma coleção de átomos de crença.

AgentSpeak(L) distingue dois tipos de objetivos: objetivos de realização (achievement goals) e objetivos de teste (test goals). Objetivos de realização e teste são predicados, tais como crenças, porém com operadores prefixados ‘!’ e ‘?’, respectivamente. Objetivos de realização expressam que o agente quer alcançar um estado no ambiente onde o predicado associado ao objetivo é verdadeiro. Na prática, esses objetivos iniciam a execução de subplanos. Um objetivo de teste retorna a unificação do predicado de teste com uma crença do agente, ou falha caso não seja possível a unificação com nenhuma crença do agente. Um evento ativador (triggering event) define quais eventos podem iniciar a execução de um plano.

Um evento pode ser interno, quando gerado pela execução de um plano em que um sub-objetivo precisa ser alcançado, ou externo, quando gerado pelas atualizações de crenças que resultam da percepção do ambiente. Eventos ativadores são relacionados com a adição e remoção de atitudes mentais (crenças ou objetivos). Adição e remoção de atitudes mentais são representadas pelos operadores prefixados (+) e (-).

Os exemplos abaixo demonstram essa notação e a lógica da linguagem.

Planos fazem referência a ações básicas que um agente é capaz de executar em seu ambiente. Essas ações são definidas por predicados com símbolos predicativos especiais (chamados símbolos de ação) usados para distinguir ações de outros predicados. Um plano é formado por um evento ativador (denotando o propósito do plano), seguido de uma conjunção 16 RITA ² Volume 10 ² Número 1 ² 2003 Linguagens de Programação Orientadas a Agentes: AgentSpeak(L) de literais de crença representando um contexto. O contexto deve ser consequência lógica do conjunto de crenças do agente no momento em que o evento é selecionado pelo agente para

o plano ser considerado aplicável. O resto do plano é uma sequência de ações básicas ou sub-objetivos que o agente deve atingir ou testar quando uma instância do plano é selecionada para execução.

Em AgentSpeak(L), um agente é especificado por um conjunto de crenças bs (beliefs) correspondendo à base de crenças inicial do agente, e um conjunto de planos ps que forma a biblioteca de planos do agente.

Um exemplo da especificação de uma crença em AgentSpeak(L) para um agente denominado p1, que irá percorrer um cenário com a crença inicial de que é necessário percorrer os slots (células do cenário) e que o local de chegada está na posição x=12 e y=12 (c2,12,12).

Agent p1

Beliefs

pos(c2,10,10).
checking(slots).

A **crença** inicial do agente é sobre a posição do local de chegada c2 na grade que define o território e que sua tarefa inicial será verificar os diversos pontos desse território (checking slots)

Plans

+pos(p1,X1,Y1) : checking(slots) & (obstacle(r1))
<- next(slot).

+notobstacle(p1) : checking(slots)
<- !stop(check);
.send(p2,tell, pos(X1,Y1));
!continue(check).

O **plano**, nesse exemplo, é ativado quando o personagem percebe que ele está numa nova posição e está procurando por caminhos alternativos de travessia. Se há obstáculos percebidos naquele ponto, só o que deve ser feito é a ação básica next(slot) que move o personagem para o próximo ponto na grade, sempre considerando a lógica da matriz periférica, implementada no ambiente.

Quando p1 percebe uma célula livre em sua posição, a crença notobstacle (p1) é adicionada à base de crenças de tal maneira que o próximo plano (comunicar e avançar) pode ser então usado.

Nesse caso o personagem interrompe a verificação do cenário (!stop(check)), envia mensagem ao personagem p2 sobre a posição livre de obstáculo (.send(p2,tell, pos(X1,Y1));) e continua sua “caminhada” para a completa travessia.

Para um melhor entendimento dos métodos de movimento e comunicação, abaixo estão relacionados as duas chamadas (ações) para movimento e envio de mensagens, respectivamente, que geram no receptor percepções que alteram suas bases de crenças e incluem, ou não, novos objetivos:

- moveTowards(X1,Y1); move o personagem para as posições X1 e Y1 e
- .send(p2, tell, free(X1,Y1)); enviar uma mensagem ao personagem p2

No item 5, Jason, será apresentado como que internamente a ação de movimento é executada.

A ação de comunicação de mensagens, `.send`, é apresentada no item 6.4, sobre a linguagem KQML.

5. Interpretador Jason.

Jason² (do inglês, A Java-based AgentSpeak Interpreter Used with Saci For Multi-Agent Distribution Over the Net) é um interpretador para Agent Speak [2][9]. Inclui o tratamento de mecanismos de comunicação, de funções de confiança além da arquitetura baseada no modelo BDI (crenças, desejos e intenções). Uma aplicação sob o Jason pode ser distribuída para rodar em múltiplas máquinas. Como característica fundamental, Jason é portátil, ou seja, multi-plataforma, pois roda em Java e pode ser executado em qualquer sistema e máquina que possui a JVM instalada [6]

Está disponível como Open Source, sob a licença GNU LGPL.

Além de ser um interpretador da linguagem AgentSpeak(L), o Jason apresenta outros recursos interessantes, podendo ser citado:

- tratamento de falhas em planos;
- comunicação baseada em atos de fala (incluindo informações de fontes como anotações de crenças);
- tratamento de ações de percepção nas mudanças do ambiente, gerando novos significados na elaboração e seleção de planos;
- suporte para o desenvolvimento de ambientes, pela sua característica de ser programado em Java;
- a possibilidade de executar o SMA distribuídamente em uma rede;
- possibilidade de especializar (em Java) as funções de seleção de planos bem com toda a arquitetura que compões um agente, em especial nesse trabalho, a comunicação, além da percepção, revisão de crenças e sua atuação no ambiente;
- possui uma biblioteca básica de “ações internas”, e por ser escrito, em Java, possibilita a extensão dessas funcionalidades;

Um exemplo de definição de um ambiente e seus agentes, em Jason:

```
/* Jason Project */
```

```
MAS scene {  
    infrastructure: Centralised  
    environment: sceneEnv  
    agents: p1; p2;
```

Para cada recurso, deve haver um arquivo `.asl` para os agentes e `.mars2j` para o ambiente.

No código abaixo está listado parte do programa em Java que controla o funcionamento e comportamento do ambiente. As definições das ações são vistas no exemplo como ações de movimento do personagem (next slot):

```
..  
public static final Term ns = Term.parse("next(slot)");
```

```
..  
public boolean executeAction(String ag, Term action) {  
    logger.fine("Agent "+ag+" doing  
"+action+" in the environment");  
    if (action.equals(ns)) {  
        crl[X]++;  
        if (crl[X] == GSize) {  
            crl[X] = Min;  
            crl[Y]++;  
        }  
        if (crl[Y] == GSize)  
            return true;  
        if (crl[X] == Mid && crl[Y]  
== Mid)  
            crl[X]++;  
    } else ...
```

O método `Term.parse` faz parte das classes do interpretador. Ele passa um valor de variável (no caso “*ns*”) que é recebida em `executeAction` como parâmetro de entrada (*ag* é o personagem e *action* é a ação a ser executada).

No caso acima, *ns* define a ação como sendo “next slot”, movendo o personagem para as coordenadas definidas em `crl[X]` e `crl[Y]`, controlando também se os limites do cenário não foram extrapolados (`Gsize`).

6. Comunicação entre agentes.

A comunicação permite que os agentes em um sistema SMA troquem informações que servirão de base para coordenar suas ações e realizar a cooperação[8].

Um agente é parte integrante de uma sociedade. Ele, ao fazer parte do ambiente, de seu mundo virtual, possui capacidades específicas de comunicar-se e interagir com os demais agentes, atendendo dessa forma suas necessidades e objetivos. É por meio dessa troca em um SMA que agentes coordenam suas ações e objetivos. A troca, intencional de envio e recepção de mensagens, ocorre a partir de percepções de mundo que o agente possui.

As Linguagens de Comunicação para Agentes (LCA) têm o papel de fornecer mecanismos para troca de informação e conhecimento entre agentes.[8]

Muitas são as ferramentas, modelos, arquiteturas, interfaces e linguagens que incorporam essa capacidade de comunicar, outras criadas para essa finalidade.

O objetivo desse trabalho é apresentar a programação da comunicação entre agentes, especificamente na linguagem AgentSpeak(L), mas com o objetivo de complementar e trazer mais informações sobre esse assunto.

Em um SMA os agentes não são desenvolvidos visando resolver um problema específico, pelo contrário, dado um problema, um grupo de agentes se une para resolvê-lo [7][11]. Assim, os agentes não precisam necessariamente ter sido desenvolvidos utilizando as mesmas ferramentas e modelos. Este grupo de agentes, contudo, deve concordar no uso de uma linguagem comum. Além disso, como nas sociedades humanas, a

² Na WEB em <http://jason.sourceforge.net/>

comunicação é mais eficiente quando seus usuários seguem certas regras, chamadas de protocolos de comunicação. Algumas especificações de Linguagem de Comunicação de Agentes (LCA) foram propostas com o objetivo de viabilizar a comunicação entre agentes desenvolvidos em projetos diferentes e possibilitar a descrição das regras que regem o fluxo de comunicação entre eles. Dentre as propostas de LCA existentes, duas se destacam: KQML e FIPA ACL.

6.1 Comunicação em Agent Speak

Conforme definiu Fayard [1], “uma informação não estratégica seria uma banalidade, contrária até mesmo à definição de informação!”. Isso remete à idéia do uso e dos resultados obtidos a partir de uma informação recebida, da alimentação que tal informação gera em relação aos personagens e ambientes, visando à ação, diante da percepção e da visão mais ampla e complexa do que ocorre.

Dessa forma remete-se à modelagem de agentes interagido em um contexto social. Cada agente envia e recebe informações com a intenção de gerar conhecimentos, informações propriamente ditas sobre ações realizadas, situações percebidas, alterações comportamentais ocasionadas e propostas de atuação.

Os estudos baseados em agentes permitem a compreensão da forma de interação entre os personagens, ou seja as entidades de um sistema e dos mecanismos de comunicação, sendo essa o instrumento mediador dessa troca de percepções, gerando novas concepções de mundo a partir do recebimento da mensagem.

A arquitetura que possibilita a comunicação, proposta nesse artigo apresenta-se organizada na seguinte forma:

- personagens: agentes cognitivos, formados por sistemas de percepção de situações, raciocínio baseado no modelo BDI;
- cenários: ambiente apresentado na forma de matriz contendo caminhos livres por onde o personagem se deslocará e obstáculos que possuem níveis de dificuldades para a travessia;
- **comunicação**: baseado na troca de mensagens entre o personagem e a coletividade,

6.2 SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes

O Saci provê a parte social de um ambiente multiagentes, quando os agentes adquirem capacidade e recursos para trocarem informações com os demais, comunicando-se, interagindo e conhecendo-se mutuamente[7].

Conhecer outro agente é ter informações que vão além de simplesmente saber quem ele é, sua identificação.

É necessário conhecer também suas qualidades, características, e dessa forma, o funcionamento e a capacidade de toda a coletividade.

Ao conhecer nessa amplitude do conhecimento mútuo, os agentes incorporam o conhecimento de toda sua estrutura, que pode variar com o tempo devido a eventos, designados como eventos sociais, que por sua vez

refletem o ciclo de vida do agente[7]: ingresso na sociedade, com sua identificação sendo conhecida pela coletividade, o anúncio das habilidades do agente, o envio e recebimento de mensagens e sua saída da sociedade, quando ele perde sua identificação e conseqüentemente sua identidade junto ao coletivo.

6.3 FIPA - Foundation for Intelligent Physical Agents (FIPA) ACL

A especificação de LCA (Linguagem de Comunicação de Agentes) da FIPA está baseada nos mesmos princípios que o KQML (teoria dos atos de fala) e tem a mesma sintaxe para as mensagens. Esta especificação consiste de um conjunto de tipos de mensagens e a descrição formal de sua pragmática — o efeito das mensagens na mente do emissor e do receptor. Neste aspecto, a FIPA ACL difere do KQML, a semântica dos tipos de mensagens não é exatamente a mesma. Além da LCA, a FIPA também especifica uma notação para descrição de protocolos de comunicação. Esta notação é exemplificada com protocolos para requisição de serviços, rede contratual e outros.

6.4 KQML - Knowledge Query and Manipulation Language

KQML é uma especificação de linguagem e protocolo de comunicação entre Agentes [7] que tem por objetivo ser um meio comum de troca de informação e conhecimento entre agentes. Esta especificação foi amplamente adotada por ter algumas características importantes numa LCA:

- Qualquer linguagem pode ser usada para escrever o conteúdo das mensagens
(por exemplo: LISP, Prolog, SQL, Português);
- As informações necessárias para compreender o conteúdo das mensagens estão incluídas na própria comunicação;
- Quando os agentes trocam mensagens KQML, o mecanismo de transporte é transparente, i.e., como a mensagem sai do agente emissor e chega no receptor.
- O formato das mensagens é simples, fácil de ler por pessoas e de ser analisada por um *parser*.

Geralmente, arquiteturas que utilizam KQML adicionam um agente facilitador à sociedade. Este agente sabe quem são os agentes da sociedade e que requisições cada um é capaz de responder. Desta forma o procedimento de apresentação entre os agentes é simplificado.

A KQML além de ser utilizada como uma linguagem que facilita a interação entre agentes, também pode ser utilizada no compartilhamento de conhecimento entre dois ou mais sistemas inteligentes para encontrar soluções na resolução de problemas. Isto demonstra que aqueles que estão em conformidade com a linguagem KQML podem responder a estas mensagens de maneira adequada e independentemente da estrutura do seu emissor.[8]

Em KQML, as mensagens são expressas por meio de performativas e elas definem as operações que um agente pode conduzir. A lista completa das performativas reservados e seus significados pode ser encontrada em [8]. No item 4, está descrita a estrutura do comando de envio de mensagens entre dois agentes. Nesse caso a performativa *tell* é utilizada para o agente emissor informar ao receptor sobre uma sentença que está ou não na sua base de conhecimento.

Uma capacidade viabilizada por essa linguagem é o envio para um ou mais agentes do ambiente, podendo enviar/comunicar a todos, de forma a tornar uma percepção, ou conhecimento sobre determinada ação, coletiva e, dessa forma, socializada.

7. Um estudo de caso: navegação colaborativa

Este trabalho baseia-se e uma implementação de um cenário onde o agente (personagem p1) possui crenças iniciais que existe um caminho possível para travessia e outras que são incorporadas a medida que avança pelo “terreno”. O objetivo é a travessia, saindo de uma posição [0 0] até chegar a [Gsize Gsize] onde Gsize é o tamanho da grade quadrática onde o cenário está definido.

A figura 2 apresenta um quadro gerado a partir de um tamanho pré-definido, onde o caminho mais escuro é a trajetória do personagem, com seu deslocamento ocorrendo nas mais diversas direções, conforme análise dos caminhos possíveis dentro de uma visão limitada a 1 quadro periférico.

Com a crença da possibilidade de travessia o personagem desloca-se desviando dos obstáculos, gerados aleatoriamente e dispostos no terreno em quantidade também pré-definida. Estão representados no quadro pelos quadrados cinzas. Os espaços vazios (em branco) são os caminhos livres por onde o personagem pode se deslocar.

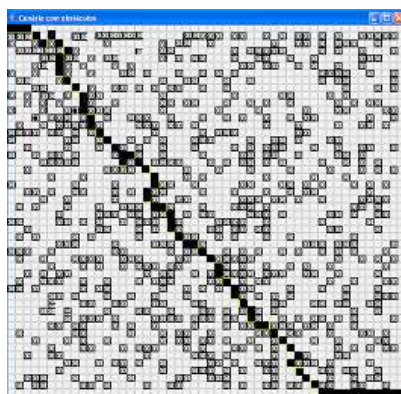


Figura 2. Cenário de navegação do personagem

7.1. Proposta de uma técnica de programação

O código abaixo apresenta uma parte da programação Java onde o personagem verifica, pela sua visão periférica com capacidade de alcance 1, a existência de obstáculos para, se estiver livre, optar pela melhor alternativa.

```
if (pX < GSize-1 && pY < GSize-1)
// Pode validar posição 0
{
    if (!(scene[pX+1][pY+1]))
// Na direção (0) não tem obstáculo.
    {
        area[0]=true;
        controle = 0;
    }
}
```

onde:

pX e pY: posição do personagem nos eixos;

Gsize tamanho do cenário (matriz);

Scene[x][y] matriz booleana que define a existência (true) ou não (false) de obstáculos;

area[n] é um vetor de 8 posições que armazenará como verdadeiro (true) a melhor direção livre. Esse vetor será utilizado para repassar para a pilha de controle o endereço do melhor caminho, possibilitando o retorno do personagem bem como para ele comunicar aos demais personagens (comunidade) de um caminho livre de obstáculos.

O código abaixo, por sua vez, decide o avanço do personagem pelo cenário, de acordo com as análises feitas anteriormente, dando-lhe sempre a melhor opção e assim o menor caminho (menor esforço) para a travessia.

De acordo com a figura 3, cada nova cena (célula) dentro da visão periférica do personagem, recebe um peso, de 0 a 7, onde o menor valor define o menor esforço. A variável controle é utilizada apenas para diferenciar o valor (um inteiro) do conteúdo do vetor (booleano).

```
if (pX < GSize-1 && pY < GSize-1)
// Pode validar posição 0
{
    if (!(scene[pX+1][pY+1]))
// Na direção (0) não tem obstáculo.
    {
        area[0]=true;
        controle = 0;
    }
}
```

No código acima, por exemplo, o melhor caminho é a posição 0, na diagonal do personagem, onde as variáveis que controlam os eixos (x e y) são incrementadas com uma posição (x++ e y++).

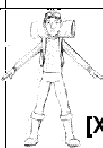
7 X -- Y --	6 Y --	4 X++ Y --
5 X --	 [X][Y]	2 X++
3 X -- Y++	1 Y++	0 X++ Y++

Figura 3. Proposta algorítmica de navegação pelo cenário.

7.2 A recursividade no algoritmo

A partir do pressuposto que todo caminho possui em sua matriz periférica³ uma possibilidade de caminhada (pesos de 0 a 7, conforme figura 3), a função que analisa e percorre essa matriz pode ser chamada a partir de uma função recursiva, conforme esquema apresentado na figura 4.

As coordenadas [x y] sempre refletirão uma posição válida (livre) no cenário.

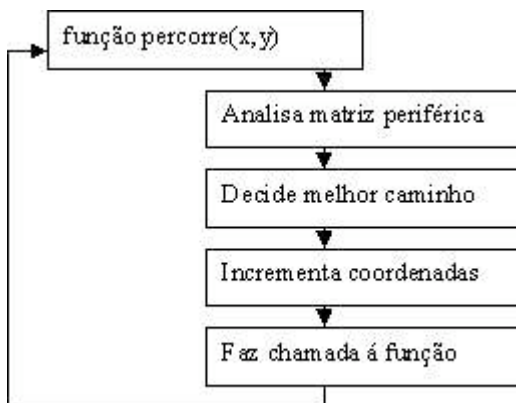


Figura 4. Modelo da recursividade na chamada à função percorrer

7.3. O problema da “caverna”

Dentro da proposta de análise do melhor caminho, durante as fases de vários testes com as mais variadas formas de cenário, apresentou-se um problema de organização, não prevista nas primeiras concepções do sistema. A esse problema denominou-se “o problema da caverna”, pois dentro da proposta da navegação de um personagem, ocorre a entrada em uma célula, obviamente vazia, mas que a partir dela não há caminho possível, a não ser o retorno para a célula anterior.

Nesse caso ocorre sempre a escolha da célula que deu origem ao último passo, pois ele sempre será o primeiro melhor caminho encontrado, gerando um loop infinito.

Na figura 5, percebe-se as células onde o problema foi detectado., marcada nos quadros por três pontos (“...”).

Os quadros com a letra P é o personagem e o caminho feito durante sua caminhada pelo cenário. Os quadros marcados com a letra “O” representam os obstáculos.

Esse quadro é semelhante ao da figura 2, porém com zoom para detalhar a situação.

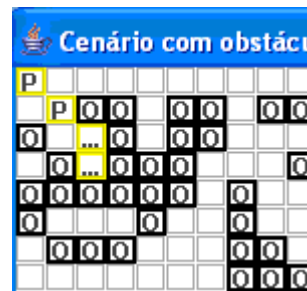


Figura 5. Tela parcial com o problema da caverna

7.4 Uma alternativa de saída

Uma proposta de solução para o problema da caverna, onde o personagem está em uma situação de conflito com relação aos objetivos de atravessar o cenário, e suas crenças de que existe um caminho possível para isso, pode se resolvido com a formalização da comunicação entre esse personagem e a coletividade.

Pode-se considerar que nessa situação, ocorre a interferência de outro personagem (p2) que recebe uma mensagem sobre a ocorrência dos obstáculos e o problema enfrentado pelo personagem p1, incorporado a sua base de crenças que deve tentar a travessia, tomando outro caminho diferente daquele informado por p1, pois insere uma crença que iniciando por uma posição diferente da posição inicial de p1, a possibilidade de ocorrer a mesma situação diminui ou inexistente.

A proposta de solução é marcar o caminho de p1 como caminho a não ser percorrido, implementado a recuperação desses slots, utilizando a retirada das coordenadas feitas por p1 (retira na forma de pilha) para que o mesmo retorne para sua posição inicial, ou pode-se optar por mantê-lo na caverna até p2 completar a travessia.

na forma de lista. Usar a recursividade par p1 retornar a posição inicial e marcar a “entrada da caverna” como ponto a não ser utilizado.

Portanto, uma opção é retornar com p1 a partir da “entrada da caverna”, marcando o local como obstáculo.

O código em AgentSpeak para o personagem p1 perceber a situação e comunicar ao personagem 2 sua situação pode ser conforme abaixo:

```

+pos(p1,X1,Y1): iscave(true)
.send (p2;tell;pos(p1));
!stop checking(slot);
  
```

³ Nesse trabalho denomina-se matriz periférico as células, ou slots, ao redor da posição onde o personagem se encontra, perfazendo sempre uma matriz lógica de 3 x 3 com o personagem ocupando a posição [2 2] dessa matriz.

Ao verificar como verdadeira sua crença de estar na caverna (iscave), p1 envia mensagem para p2 iniciar sua travessia e para a verificação de um caminho válido, ficando no aguardo da mensagem de p2 sobre seu sucesso, ou não.

E em p2;
+ iscave(p1):
! checking(slot);

p2 ao receber a mensagem torna verdadeira sua crença em iniciar a travessia, em posição diferente do início de p1.

8. Conclusões

Uma implementação de SMA possui diversas tecnologias que viabilizam e permitem construir um modelo que atenda à especificação do projeto de ambientes sociais e agentes com características cognitivas e com capacidade de interagir e fazer parte de uma coletividade.

Essas especificações muitas vezes requerem que agentes possuam capacidades e características próximas à realidade e dependendo da situação, exigem tal comportamento.

Comunicar, raciocinar, tomar decisões, aprender, perceber e adaptar-se são verbalizações que cabem a seres humanos, mas possibilitados de modelagem por meio dessas facilidades e interfaces aqui apresentadas.

Nesse trabalho foram citados linguagens de programação, interpretadores, recursos, especificações e padrões que, juntamente com técnicas de programação eficazes, viabilizam e tornam possível a tarefa de construir uma realidade virtual aceitável e otimizada com relação ao real.

Pesquisas e estudos envolvendo agentes, ambientes, técnicas e tudo o que foi apresentado, tem-se constituído como um fértil campo de trabalho, onde grupos de pesquisa desenvolvem cada vez mais facilidades e novas descobertas e tem tornado possível a integração dos recursos computacionais com a área de IA.

9. Referências

- [1] Fayard, P. O jogo da interação: informação e comunicação em estratégia, Caxias do Sul: EDUCRS, 2000
- [2] Hubner, J.F., Bordini, R.H., Vieira, R. Introdução ao Desenvolvimento de Sistemas Multiagentes com Jason. Disponível em <http://www.inf.furb.br/~jomi/pubs/>
- [3] Bordini, R.H., Vieira, R. Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L). Disponível em <http://www.inf.furb.br/~jomi/pubs/>
- [4] Russel, R. Norvig, P. Inteligência Artificial. Segunda edição. Rio de Janeiro: Elsevier, 2004.
- [5] Hubner, J.F. Sichman, J.S. Organização de Sistemas Multiagentes. Disponível em <http://www.inf.furb.br/~jomi/pubs/>
- [6] Alisson Rafael Appio, A.R. Hubner, J.F. Sistema Multiagentes utilizando a linguagem AgentSpeak(L) para criar estratégias de armadilha e cooperação em um jogo tipo PacMan.

Disponível em <http://www.inf.furb.br/~jomi/pubs/>

[7] Hubner, J.F. Sichman, J.S. SACI - Simple Agent Communication Infrastructure

Disponível em <http://www.lti.pcs.usp.br/saci/>

[8] WEB. Comunicação entre personagens. Disponível em [www2.dbd.puc-](http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0210488_04_cap_06.pdf)

[rio.br/pergamum/tesesabertas/0210488_04_cap_06.pdf](http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0210488_04_cap_06.pdf)

[9] Bordini, R.H. Multi-Agent Programming with Jason. University of Durham, U.K.

Disponível em <http://www.dur.ac.uk/r.bordini>

[10] Montesco, C. A. E. UCL - Uma linguagem de comunicação para agentes de

software. Dissertação de Mestrado. São carlos, 2001.

Disponível

em http://java.icmc.usp.br/research/master/Carlos_Montesco/tese.pdf

[11] Hubner, J.F. Um modelo de reorganização de um sistema multiagentes.

Disponível em <http://www.inf.furb.br/~jomi/pubs/>